applied sciences

MDPI

Article

DroidPortrait: Android Malware Portrait Construction Based on Multidimensional Behavior Analysis

Xin Su [1,2], Lijun Xiao [3], Wenjia Li [4], Xuchong Liu [1,2], Kuan-Ching Li [5,*] and Wei Liang [6,*]

1 Hunan Provincial Key Laboratory of Network Investigational Technology, Hunan Police Academy, Changsha 410000, China; suxin@hnu.edu.cn (X.S.); liuxuchong@163.com (X.L.)
2 Big Data Intelligence Police Hunan Provincial Engineering Research Center, Hunan Police Academy, Changsha 410000, China
3 Big Data Development and Research Center, Guangzhou College of Technology and Business, Guangzhou 510006, China; 2013112101@xmut.edu.cn
4 Department of Computer Sciences, New York Institute of Technology, New York, NY 10023, USA; wli20@nyit.edu
5 Department of Computer Science and Information Engineering, Providence University, Taichung 43301, Taiwan
6 College of Information Science and Engineering, Hunan University, Changsha 41000, China
* Correspondence: kuancli@pu.edu.tw (K.-C.L.); weiliang99@hnu.edu.cn (W.L.)

Received: 19 April 2020; Accepted: 3 June 2020; Published: 8 June 2020

Abstract: Recently, security incidents such as sensitive data leakage and video/audio hardware control caused by Android malware have raised severe security issues that threaten Android users, so thus behavior analysis and detection research researches of malicious Android applications have become a hot topic. However, the behavioral portrait of Android malware that can depict the behavior of Android malware is not approached in previous literature. To fill this gap, we propose DroidPortrait, an Android malware multi-dimensional behavioral portrait construction approach. We take the behavior of Android malware as an entry point and extract an informative behavior dataset that includes static and dynamic behavior from Android malware. Next, aiming at Android malware that contains different kinds of behaviors, a behavioral tag is defined then combined with a machine learning (ML) algorithm to implement the correlation of these behavioral tags. Android malware behavioral portrait architecture based on behavior analysis and its design is investigated, as also an optimized random forest algorithm is conceived then combined with Android malware behavioral portrait to detect Android malware. The evaluation findings indicate the DroidPortrait can depict behavioral characteristics of Android malware comprehensive and detect them with high performance.

Keywords: Android malware; behavioral portrait; behavioral tag; machine learning

1. Introduction

According to the statistical report released by IDC, the Android platform's market share first breaks through 80% in the smartphone market in Q3/2019[1]. There are almost three million applications (or apps) that can be downloaded from the official Android market, Google Play and more than 84 billion downloads to date[2]. The popularity of Android platform attracts many cyber-criminals to attack this platform by designing various Android malware that could launch several types of malicious activities, such as stealing sensitive information, controlling audio hardware to record user's conversation with others, obtaining the real-time geolocation of user and sharing it with others and others. These malicious behaviors impose threats to the platform as well as user privacy, permitting malicious

Android applications to launch various types of cyberattacks. Moreover, certain malware even has a large number of variants, which makes it very hard to detect all of them.

In order to cope with the increasing security concerns caused by Android malware, there have been many research efforts to develop Android malware analysis and detection approaches [1–11]. For example, in recent research works such as SIGPID [1], DroidAPIMiner [2], APK Auditor [3], Drebin [4], the researchers first extract static features from Android applications, including permissions, API calls and others, so then apply them to various machine learning algorithms as feature vectors to identify Android malware. However, these static-based solutions cannot identify malware with code obfuscation. In addition, they generally assume that some behaviors occur more frequently than they would be in practice, which generally causes a considerable number of false positives.

To address these limitations, some researchers propose to rely on dynamic app characteristics to capture run-time execution context and apply them to classification algorithms for Android malware detection [12–14]. For example, TaintDroid [12] identifies some privacy information sources by conducting a dynamic tainting analysis. In [13], the authors propose to capture HTTP traffic during an Android app's execution to detect Android malware. The main issues with these dynamic-based approaches are in the following two aspects: (1) they have to modify Android apps when they get updated [14] and (2) They require sufficient input suites to sufficiently exercise execution paths. In summary, both static and dynamic-based approaches only focus on one aspect of Android malware behaviors that cannot depict Android malware behaviors in a comprehensive and precise manner.

In this study, we propose a new malware detection approach for Android system named DroidPortrait, which first extracts both static and dynamic behaviors from Android applications (or apps), then utilizes data portrait techniques to build multi-dimensional behavioral portrait for Android malware and finally uses the behavioral portrait to identify malware. The DroidPortrait primarily aims at accurately depicting behaviors of Android malware, and then detecting them efficiently and precisely. As the first step, both static and dynamic behavioral features are extracted from numerous Android apps. Several dimensions of features are then defined to depict Android malware. Second, we define a behavioral tag which generalizes meta-data of the behavioral characteristic of Android malware and design a behavioral tag for Android malware based on the tag extraction, definition and identification. Third, we analyze the correlation among these behavioral tags to build a behavioral portrait of Android malware. Last, we design a random forest-based algorithm, which uses a behavioral portrait for Android malware detection. The results of extensive experimentation performed to show that the proposed approach depicts Android malware behavior comprehensively and accurately—and can achieve 0.971 in terms of detection accuracy, 0.986 for precision, 0.97 for recall and 0.973 in terms of F1-measure.

The main technical contributions of this study are summarized as follows:

— To introduce a behavioral portrait technique to depict Android malware behavior for detection purposes. This approach relies on characteristic extraction, selection, behavioral tag construction and it builds a behavioral portrait that could consider both static and dynamic behavioral characteristic;
— To propose a random forest-based malware detection algorithm, which can dynamically select decision trees with higher classification performance. This algorithm overall improves decision tree fusion effectiveness and ensures the generality of the algorithm;
— To evaluate the efficiency and effectiveness of the proposed approach by using three real-world Android app datasets. Based on the experimental results, we find that the DroidPortrait approach can achieve 99% accuracy for making a comprehensive portrait for an Android app and it can achieve more than 97% accuracy for the detection of malware;

The remaining of this study is organized as follows. Section 2 introduces related works and compares them with the proposed approach, Section 3 illustrates the detailed design of DroidPortrait, Section 4 explains the optimized random forest algorithm for Android malware

detection, the discussions of results and experimentations in Section 5 and finally, concluding remarks and future directions in Section 6.

2. Related Work

In recent years, many research efforts were made focusing on Android malware detection and behavioral characteristics analysis. We summarize these research efforts in two types, namely static analysis and dynamic analysis-based approaches.

2.1. Static Analysis Based Malware Detection Approaches

The first type of approach for detecting Android malware stemmed from the traditional static program analysis. Static analysis is a common approach to detect malware in personal computers, servers, Android platform. For instance, Nath et al. propose a static analysis approach to detect malware that can launch the advanced persistent threat [15]. Meanwhile, there have been various approaches in this direction that could statically inspect Android apps by disassembling the mobile apps. For example, some researchers analyzed permission requests for app installation [16], while others proposed a signature-based detection approach [17].

The primary basis of this type of approach is to extract static behavioral characteristics by disassembling the application installation file (.apk file) into smaller pieces, including Configuration files and source Codes and building different types of detection models step. As shown in DroidDet [18], the rotational forest algorithm can effectively detect Android malware using static features such as permission, sensitive API. In [19], a malware detection approach was studied based on Bagging-SVM and it mainly relies on static features extracted from AndroidManifest.xml. Park et al. first divide mobile applications into three categories according to their APIs and permissions and then construct a classification system based on the YARA Rule [20]. This system could make the users aware of security risks by providing them insights on the mobile application behaviors, which allows the users to make an informed decision on whether to install the application on their devices or not. Dexteroid [21] is a framework of static analysis that employs a reverse-engineered life cycle model to capture Android component behavior, as this approach import event sequence and use them to detect attacks launched by specific sequence. DroidMat [22] extracts permission, Intent, API, employ clustering algorithm to cluster these kinds of static behavior characteristics, and detect Android malware by classification algorithms.

2.2. Dynamic Analysis-Based Malware Detection

Unlike static analysis-based approaches, dynamic analysis-based approaches aim at capturing run-time behaviors from Android apps during the execution process of the apps on Android emulator or real devices. The examples in this category include taint analysis based approaches [12,23] and behavior-based detection strategies [24,25].

Yin et al. proposed a malware detection solution named DroidScope, where both the operating system level and programming language level semantics were rebuilt and allowed the instrumentation of the Dalvik and native instructions [20]. Saracino et al. proposed a host-based malware detection system that analyzed Android app features at the levels of the kernel, application, user and package [25]. In [26], a framework named AdDroid was studied to analyze malicious behaviors in Android applications based on different combinations of mobile app behaviors such as network access information, file uploading to a server or package installation on the device, among others. Hou et al. proposed a malware detection approach named component traversal that automatically executes the code routine of the individual application effectively [27]. Ali et al. applied fuzzy C-means clustering to the generated network traffic for Android malware detection [28], and lastly, DL-Droid is a deep learning system to detect malicious Android applications through dynamic analysis using stateful input generation [29].

The proposed approach has the following significant innovations when compared to the existing static and dynamic analysis-based approaches, and they are twofold. The former is, we extract far more behavioral characteristics, which include both static and dynamic behaviors, while as the latter, we build a multidimensional behavioral portrait for Android malware to depict them accurately and systematically.

3. The Overall, Design of DroidPortrait

The main goals of DroidPortrait are twofold. The former is to depict and analyze the behavior of Android malware comprehensive, while the latter is to achieve high Android malware detection accuracy. For such, the proposed approach first extracts static and dynamic behavioral characteristics from Android apps and their corresponding running time and then divides them into six dimensions to build a behavioral model. Next, we extract behavioral characteristics, define a behavioral tag to depict Android malware, and employ data portrait technology to build a multidimensional behavioral portrait for Android malware. The complete approach architecture of DroidPortrait is shown in Figure 1. The critical components are described in the remainder of this section.

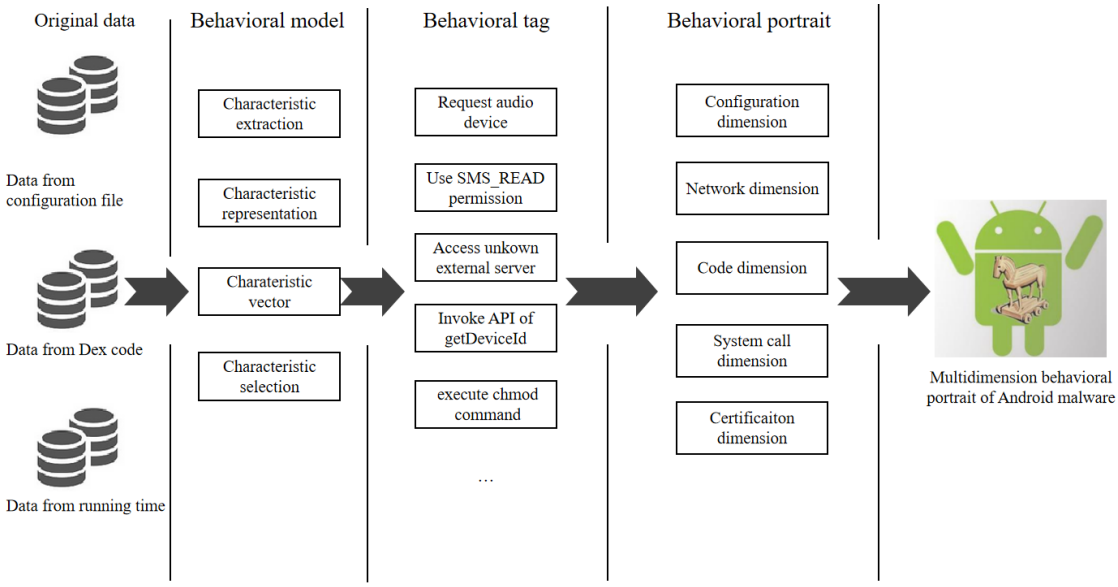


Figure 1. Architecture of DroidPortrait.

3.1. Behavioral Model Construction

The behavioral characteristics of the Android app are discussed in two categories, namely static features and dynamic features and we extract several different kinds from each of them. As the next step, we use a uniform data standard to represent these kinds of static characteristics due to their dispersion and a characteristic selection algorithm to reduce the common characteristic used by the benign app is designed.

3.1.1. Behavioral Characteristic Extraction

The main target for static feature extraction is the AndroidManifest.xml and class.dex files, and we profile the Android apps in six types of features: requested permission, used permission, sensitive API call, action, app component and intent, and extract corresponding features from the Android apps, all of which could serve as a comprehensive indicator for malicious activities. The detailed extraction process is illustrated in Figure 2.

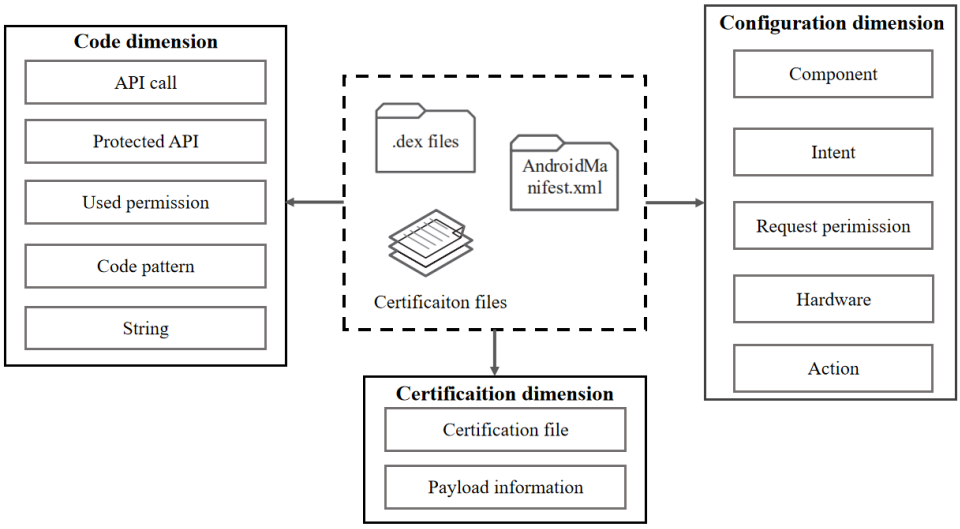


Figure 2. Static behavioral characteristic extraction.

An essential step in obtaining AndroidManifest.xml, *class.dex* and Certification *file* is to decompress the .apk files by using a tool named apktool to dissect Android apk files. After the decompression, the static behavioral features are extracted from these three types of files. The AndroidManifest.xml file contains some important information regarding the Android app, including permission and app component and we define features extracted from this file as Configuration dimension. We will then parse the manifest file by utilizing tools named *XMLPrinter2* and *TinyXml*, and extract behavioral features of permission, intent and app component. The class.dex file is used to store Dalvik byte code that can be converted to smali code for better behavioral feature extraction and we define this kind of characteristic as code *dimension*. Used permission, sensitive API call can be extracted from smali code. We define several customized extraction rules in xml files. The certification file contains certification and payload information and we define this kind of characteristic as a certification dimension. The app developers typically use their secret keys to sign the apk files when they are released. The certificate information contains several essential developer information such as country, email, organization, which can be used to distinguish among different developers. Listing 1 gives an example of extracting static behavioral characteristic discussed below.

Listing 1: An example of extracting API

```
<rule>
  <id>5</id>
  <category>FrameworkAPI</category>
  <description>Extract APIs of Framework</description>
  <regex>Landroid(/\w+)+;->\w+|Landroid(/\w+)+\$\w+</regex>
  <targetfile>smali</targetfile>
  <multiMatch>true</multiMatch>
</rule>
```

Unlike static behavioral features, dynamic behavioral features are considered to depict the dynamic behaviors of Android apps. To extracting the dynamic features, we first install Android apps and execute them on a real phone or emulator. Notably, we design an automatic approach to first install and execute the Android apps, and then extract corresponding dynamic features. Figure 3 shows the process of characteristic extraction.

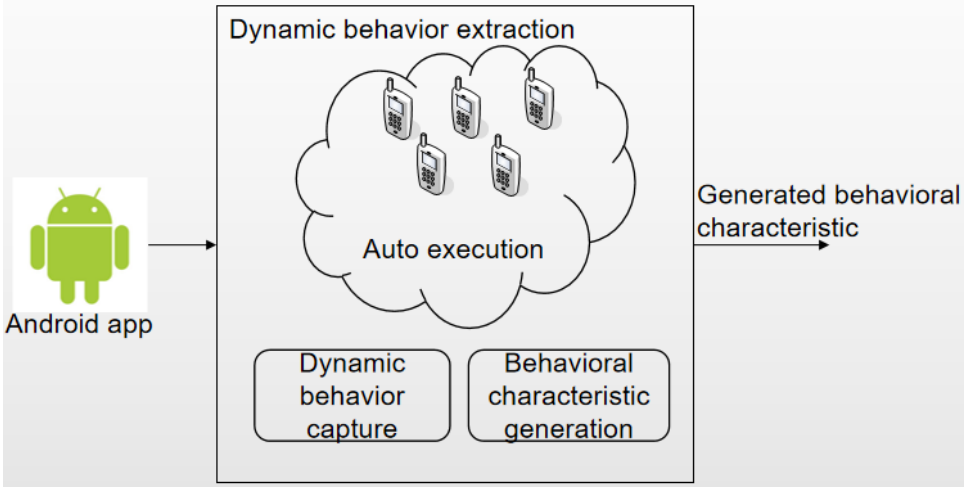


Figure 3. Dynamic behavioral characteristic extraction.

In Figure 3, we can see that the proposed approach is composed of three modules. The first module is app execution that executes Android apps on either Android phones or emulators in an automated fashion, and then outputs the captured dynamic behavioral features, such as network traffic, system call. This module could also be tuned by applying different conditions, including the composition of Android apps, execution duration or execution behavior. The second module collects dynamic behavioral features and then extracts network packets, flow and system calls. Last, the third module generates behavioral features read from a configurable file and generates dynamic behavioral features, which will be used in malware detection later. Moreover, we use tcpdump to capture network flows generated by the Android app running and use strace to collect system calls. The details about extracting dynamic behavioral features are discussed in [13,30].

In a nutshell, Table 1 shows the details of extracted behavioral characteristics from Android apps.

Table 1. Detail information of the extracted behavioral characteristic.

Dimension Category	Behavioral Characteristic	Instances
Configuration dimension	Requested permission	ACCESS_GPS, WAKE_LOCK
	App component	com.google.ssearch, com.eguan.state, com.google.update
	Intent	PHONE_STATE, MAIN, SIG_STR
	Hardware	CAMERA, NPC, AUDIO
code dimension	API call	util.log.w, Dialog.show, Uri.prase
	Protected API	getDeviceId, sendSMS, getWififiState
	Used permission	INTERNET, SEND_SMS, READ_CONTACT
	code pattern	MessageDigest, loadLibrary, pathClassLoader
	String	map.google.com, www.umeng.com, media. admob.com
certification dimension	certification information	2b7172a335 b66873dc793af3fe5c3fc6d8 . . . . 5fb16d12bc8a36b9071907bc6e042840c2 . . .
	Payload information	.MF,.RSA,.jpg
Network dimension	Quantitative	Number of bytes, number of packets
	Time	Flow duration
	Semantic	Length of URI, length of the page
System call dimension	System call	Chmod, fork, kill

3.1.2. Behavioral Characteristic Transformation

Malicious activity can be depicted by specific modes and incorporations of the static behavioral characteristics (the value of the dynamic behavioral characteristic is numeric). However, it is not easy to transform real-world data to Boolean expressions. To solve this problem, we plan to obtain a dependency relationship of static behavioral characteristics by employing a machine learning (ML) model. Due to most such type of model executes on numerical vectors; we need to build a mapping relationship between obtained behavioral characteristic sets and vector. For such, we employ a

joint set FS consisting of all strings in the extracted static behavioral characteristics and calculated by Equation (1).

$$FS := \{FS_1 \cup FS_2 \cup \ldots \cup FS_n\} \in FS_i \tag{1}$$

The set FS contains more than 50,000 different behavioral characteristics. Using FS, we define an $|FS|$-dimensional vector space, where each dimension is a binary value. An Android app A is mapped to this space by constructing a vector $Vector$.

$$Vector = \{ve1, ve2, \ldots, ven\}, vem = \begin{cases} 1 & if \quad f \sin \in FSA \\ 0 & if \quad f \sin \notin FSA \end{cases} \tag{2}$$

Based on Equation (2), a characteristic behavioral vector can be translated into $Vector = \{1, 1, 0, \ldots, 1\}$, the value of one represents that the behavioral characteristic contained by this Android app. The value of zero represents that this Android app does not contain behavioral characteristic. Based on such observation, we observed that the $Vector$ is sparse, many elements are not meaningful for later analysis, so it may increase storage overhead. Therefore, we compress $Vector$, translated it into $Vector^*$ and show as presented in Equation (3):

$$Vector^* = \{1, 3, 6, \ldots\} \tag{3}$$

The positions of non-zero elements in $Vector$ are stored in $Vector^*$, which saves a considerable amount of memory space.

3.1.3. Behavioral Characteristic Selection

After behavioral characteristic extraction and transformation, we retrieve more than 50,000 characteristics from an Android app. Such a large number of characteristics include many common characteristics used in both Android benign apps and malware, which cannot depict Android malware uniquely. Moreover, redundant and irrelevant behavioral characteristic would cause extensive computation resources during model construction and make detection accuracy decline. To address these problems, we plan to select intrinsic behavioral characteristics from the extracted characteristics. Because of these behavioral characteristics that consist of static and dynamic characteristic, their value types are different. Therefore, we propose two kinds of characteristic selection approaches for each kind of characteristic.

Static characteristic selection. Given that the value of the static behavioral characteristic is discrete, not every app contain all of the behavioral characteristic. From previous works, there are several well-known feature selection approaches have been proposed, such as chi-squared [31] and information gain [32]. However, Zhao et al. [33] illustrated that both selection approaches have distribution bias and long-tail effect limitations. To address these limitations, we use the feature selection approach, as illustrated in [33].

The characteristic selection approach assumes that T_m and T_b are the number of malware m and benign apps b, and then we examine two conditions:

$$\begin{aligned} &Cond.1 : r_c \geq \alpha_c ? r_c = \tfrac{N_c}{(N_m + N_b)}, c \in \{m, b\} \\ &Cond.2 : \tfrac{N_c}{T_c} \geq \beta_c ? \end{aligned}$$

where N_m is the number of malware that contains a specific feature, N_b is the number of benign apps that contain particular features, r_c means the ratio of a specific feature contains by benign app or malware. N_c means a certain feature contains in benign app and malware, α and β are threshold, $0.5 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$, respectively. Thus, *Cond.* 1 implies that a feature is used more frequently in malware than benign apps, while *Cond.* 2 means that the occurrence times of a feature in all malware exceeds the threshold β_m. A feature should be selected as a typical feature once it satisfies the two conditions. In this way, the typical features we collect are not only frequently used by malware, but also

have a particular coverage in the feature dataset, Distribution Bias and Long Tail Effect are well solved. Since the detailed algorithm is discussed in [33], we do not go further with details.

Dynamic characteristic selection. Dynamic characteristic includes network and system call behavioral characteristic. The value of both kinds of characteristics is of numeric type, and each app contains all behavioral characteristics on them. Therefore, we use a behavioral comparison approach to observe the difference between the Android benign app and malware. In previous research works [13], we extracted network traffic and system call behavioral characteristics and then compared the difference between the Android benign apps and malware. After behavioral characteristics comparison, we select 11 types of behavioral characteristics from network traffic and 15 types of behavioral characteristics from the system call, as shown in Table 2 the detailed information.

Table 2. Detail information about selected dynamic behavioral characteristics.

Behavioral Characteristic Category	Characteristic Name	Characteristic Description
Network traffic	Number of packets	Number of packets transmitted between app and server
	Number of bytes	Number of bytes transmitted between app and server
	Number of received packets	Number of packets received by the app
	Average bytes of received packets	Average bytes of packets received by the app
	Average size of packets	Average size of packets transmitted between app and server
	In/out ratio	Ration of traffic size between sent and received of app
	Flow duration	TCP session length
	Number of bytes per second	Number of bytes transmitted between app and server per second
	Length of URI per GET/POST request	The number of resources requested by app
	Length of page per GET/POST request	The length of paths visited by the app to obtain the resources
	Length of parameter per GET/POST request	The length of parameter contained in each request
System call	sys_chmod, sys_chown, sys_mount, sys_access, sys_open, sys_clone, sys_getpriority, sys_mmap, sys_read, sys_exit, sys_kill, sys_brk, sys_execve, sys_kill, sys_times and sys_nice	Number of functions listed in left row was called by app during running time

3.2. Behavioral Tag

We already extracted several types of behavioral characteristics from five dimensions that are described in Section 3.1.1. In this subsection, behavioral tags are defined. Given that Android malware contains various types of behavior for better analysis, and detect Android malware, describe multidimensional behavioral characteristics for Android malware correlation analysis ability. Figure 4 shows an example tag of Android malware behavior.

From Figure 4, we list several behavioral tags of Android malware that can describe Android malware's behavior in five dimensions. However, Android malware may contain many behaviors corresponding to different tags. We define them by three different manners, namely behavioral tag based on meta-data, behavioral tag based on a statistical, behavioral tag based on correlation analysis. The behavioral tag based on meta-data are easy to define and obtain by comparing with meta-data, such as IP address or domain name of accessing the unknown server, permission requested by Android malware. However, this kind of tag is too frequent that is not intrinsic to distinguish Android benign app and malware. The second behavioral tag is based on statistical data, which describes Android malware dynamic behavior in a specific time, such as the number of bytes transition between app and server. The third behavioral tag is based on correlation analysis, which can describe the correlation among these kinds of behavioral tags. For example, most Android apps would use the permission to access the Internet, read device data simultaneously. Some of them send device data to well-known servers for app usage statistics; some of them send these data to unknown servers for malicious goals.

Using a single of these behavioral tags cannot distinguish the Android benign apps and malware, use their correlation can find a difference between them.

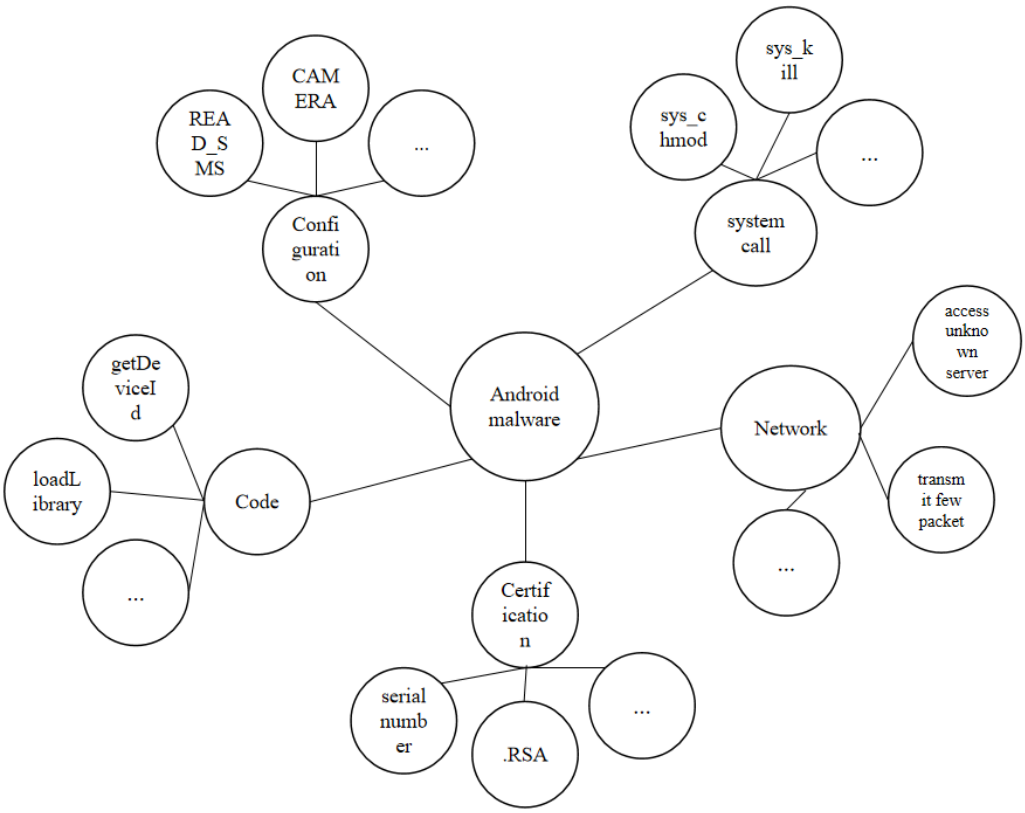


Figure 4. An example of Android malware behavioral tag.

3.3. Behavioral Portrait

We divided behavioral characteristics into five dimensions and defined behavioral tags for each dimension. In these dimensions of behavioral characteristics, configuration, code and certification dimension can be obtained by corresponding behavioral tags. network and system call dimensions should use machine learning algorithms to learn the difference between Android benign and malware to build a behavioral portrait. Figure 5 shows the process of Android malware behavioral portrait construction.

In Figure 5, we first establish the relationship between behavioral tag and dimension based on an unsupervised learning algorithm. Next, we use a supervised learning algorithm to train labeled Android malware behavior and results of the portrait and obtain a parameter set of Android malware behavioral portrait, so finally, we adjust the machine learning (ML) model based on the behavioral portrait's error value to improve the behavioral portrait's performance.

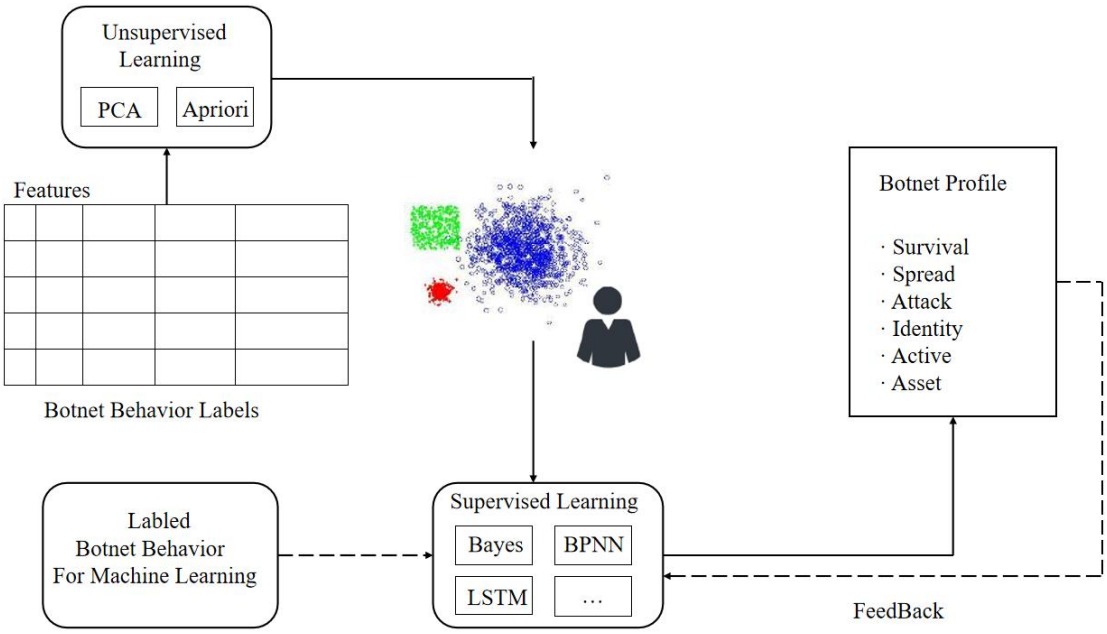


Figure 5. Process of Android malware behavioral portrait.

The unsupervised learning module is responsible for correlation analysis. Since Android malware contains various behaviors, there are correlations between behaviors. Therefore, find such kind of correlation would be helpful for Android malware behavioral portrait construction. In this study, we employ the Apriori algorithm [34] to find a correlation between each tag. This algorithm use *Support*, *Confidence*, *Lift* as metrics to judge frequent itemset. The three metrics can be calculated as follows:

$$Support(X->Y) = \frac{P(X,Y)}{P(I)} = \frac{Count(X \cup Y)}{|I|} \tag{4}$$

$$Confidence(X->Y) = \frac{P(X,Y)}{P(X)} = \frac{Support(X \cup Y)}{Support(Y)} \tag{5}$$

$$Lift(X->Y) = \frac{P(Y|X)}{P(Y)} \tag{6}$$

In Equations (4)–(6), I is an itemset that contains several behavioral tags. X, Y represent antecedent and consequent of the correlation rule. *Support* represents the probability of occurrence of X, Y in I, *Confidence* represents the possibility of behavior Y occurrences when behavior X occurrences, *Lift* represents the correlation of behaviors of X, Y.

The core concept of the Apriori algorithm is that find K frequent itemsets. This algorithm first searches candidate 1-itemset and corresponding *Support* value, prunes 1-itemset with a value of less than *Support* value, obtain 1-frequent itemset. Then, we join the rest of 1 frequent itemset and obtain candidate 2-frequent itemset. We filter out the 2-frequent itemset which value is less than *Support* value and obtain real 2-frequent itemset. This algorithm would iterate as the previous steps until obtaining K-frequent itemset.

The supervised learning module is responsible for training Android malware behavioral portraits with behavioral tags. Based on different value types of five characteristic behavioral dimensions, we use different machine learning algorithms. For example, in the dimension of network and system call, the measurement of their characteristic behavioral value may reflect disparity. Therefore, we need to normalize these values, use unified standards to measure these kinds of behavioral characteristics. To achieve this, we employ the Back Propagation Neural Network algorithm (BBNP) [35] that is discussed in Section 4.1 in detail. Regarding the dimension of configuration, code and certification,

the measurement of their behavioral characteristic is transformed into a binary value, so thus, we use a binary classification algorithm to train them, which will discuss in Section 4.2.

4. Android Malware Behavioral Portrait Training

We employ two types of machine learning algorithms to train the Android malware behavioral portrait based on the different measurement standards of behavioral characteristics.

4.1. Back Propagation Neural Network

The Back Propagation Neural Network algorithm is a standard method for training a neural network. In this section, we use this algorithm to measure behavioral characteristics from the dimension of network and system call. We first normalize all of the behavioral characteristics values and map the normalization value into the range of (0,1) to build a mapping relation between behavioral tags and characteristics. In this section, we use Min−Max Normalization to achieve this transformation, as shown in Equation (7).

$$x^* = \frac{x - \min}{\max - \min} \tag{7}$$

where x represents the value of system call and network traffic features, min represents the minimum value and max represents the maximum value.

Then, we use the *BPPN* algorithm to train weight and bias of hidden node and mapping the training results into a range of (0,1) to measure a behavioral characteristic of network and system call. The algorithm is shown in Algorithm 1.

Algorithm 1 Algorithm of BPPN
Input: training dataset $D = \{(x_k, y_k)\}_{k=1}^{m}$, learning rate η
Output: BPPN model with established weight and corresponding bias

1 Random initialized weight and bias of BPPN in range of (0,1)
2 for all $(x_k, y_k) \in D$ do
3 $\widehat{y}_j^k = f(\beta_j - \theta_j)$; // Output results of BPPN
4 $g_j = \widehat{y}_j^k(1 - \widehat{y}_j^k)(y_j^k - \widehat{y}_j^k)$; // Neuronal gradient of the output layer
5 $e_h = b_h(1 - b_h)\sum_{j=1}^{l} w_{hj}g_j$; // Neuronal gradient of hidden layer
6 $w_{hj} = \eta g_j b_h$; // Weight update
7 $\theta_j = -\eta g_j$; // Bias update
8 $v_{ih} = \eta e_h x_i$;// Input layer update
9 $\gamma_h = -\eta e_h$; // Error rate update
10 end for
11 Repeat the above steps, until γ is less than the threshold;

4.2. Optimized Random Forest

In Section 3.1.2, we already transformed behavioral characteristics from the configuration, code and certification dimension into a binary value. Therefore, we select the classification algorithm to train the three kinds of behavioral characteristics. Based on our previous works [34], we found that compare with these well-known classification algorithms and the random forest algorithm can achieve better performance. However, due to the complexity of the behavioral characteristic dataset, which may contain redundancy to disturb the process of training and reduce the accuracy of classification.

To address these limitations, we propose an optimized random forest algorithm. This algorithm adds guidance fusion technique during the process of decision tree fusion, which could improve the probability of keep decision trees with excellent classification performance and reduce effects of fusion caused by decision trees with bad classification performance, and ensure the training model contains

good generalization ability. The architecture of this algorithm is shown in Figure 6, which consists of two parts, the first part is decision tree construction, and the second part is decision tree selection and ensemble. We will discuss these parts in the following section.

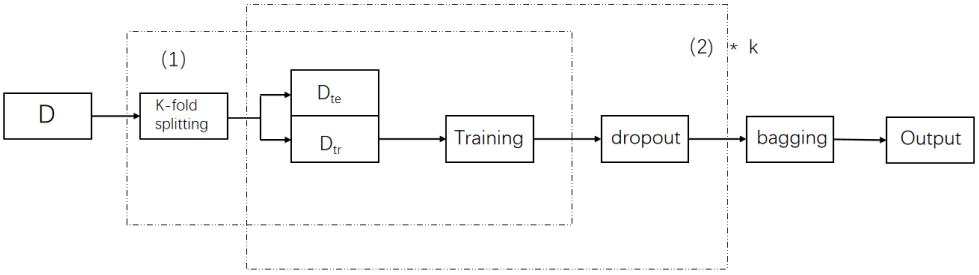


Figure 6. The architecture of optimized random forest.

4.2.1. Decision Tree Construction

To build a decision tree in random forest, we first use K-fold cross-validation to split the behavioral characteristic dataset *D* into K portions. This kind of dataset comes from behavioral characteristic model construction (describe in Section 3.1), which includes 3986 Android benign apps and 3986 malicious apps (the details describe in Section 5.1). The process of splitting is shown in Figure 7.

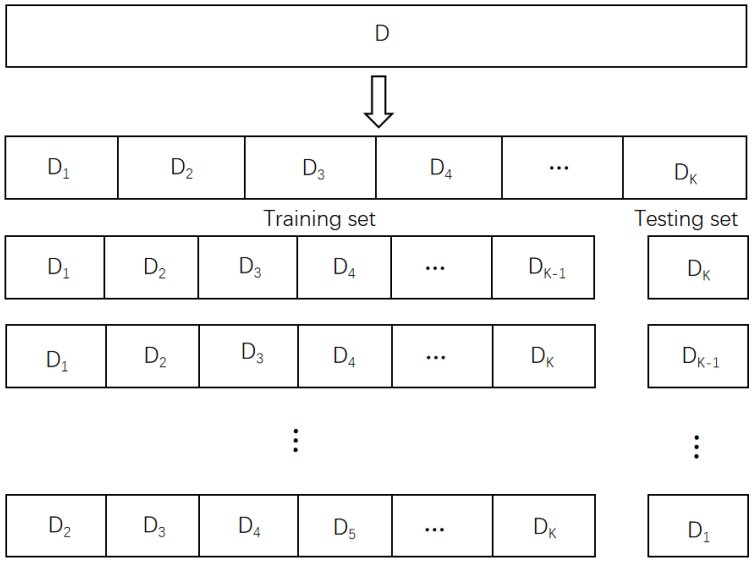


Figure 7. Dataset splitting.

From Figure 7, we divide the dataset *D* into K groups in randomly and evenly way and we select K-1 portions at random as the training set D_{tr} and 1 portion as the testing set D_{te}. Then, we follow the decision tree construction way of traditional random forest to sampling and build n decision trees. Third, we use the rest of the data as a testing dataset D_{te} for evaluation and obtain precision P_t of n decision trees in the testing dataset, P_t which can be represented as $P_t = \{p_1, p_2, \ldots, p_k\}$. Repeat the above steps for K times until each portion in the dataset *D* is used as a testing set for one time.

4.2.2. Decision Tree Selection and Ensemble

After all of the decision trees construction, we start to select and an ensemble of the decision tree. Traditional random forest algorithm ensemble all decision trees directly without considering the performance of their integrated decision tree, which may affect ensemble model performance because of the dataset contains redundancy behavioral characteristic or noise data. To improve the

classification performance of random forest and ensure satisfactory generalization ability, we combine the dropout technique [36,37] with the roulette wheel selection approach [38,39] to increase the probability of decision tree with excellent performance could be selected in each iteration of random forest construction.

Dropout is a technique used in deep learning, which aims to prevent over-fit during classification model construction [40]. Therefore, we reference this technique to select a decision tree with a certain precision P_d to ensure a classification model with satisfactory generalization ability. During each iteration, we first calculate the weight $W_i (i = 1, 2, \ldots, n)$ of every decision tree by Equation (8) as showing below:

$$W_i = p_i - \min(P_t) \tag{8}$$

We then calculate the cumulative weight $q_i = \sum_{j=1}^{i} W_i$ of each decision tree for selection purposes by referencing the roulette wheel selection approach. The main idea of this selection approach is that the probability of a decision tree could be selected proportional to its weight. After obtaining the cumulative weight of each decision tree, this approach generates an equally distributed pseudo-random number r in the range of $[0, \sum_{i=1}^{n} W_i]$. If $r < q_1$, we select the 1st decision tree, otherwise, we select ith decision tree which satisfies $q_{i-1} < r < q_i$. Following previous steps, we select $n \times P_d$ decision trees and abandon rest decision trees and finish one iteration.

Finally, we ensemble decision trees obtain by K iterations by a voting approach based on Equation (9) and build an optimized random forest model.

$$H(x) = c_{\text{argmax}\sum_j h_i^j(x)} \tag{9}$$

where $H(x)$ represents the output model after ensemble, $c = \{c_1, c_2, \ldots, c_M\}$ represents the classification category set, h_i^j represents the output of decision tree h_i on category c_j. The voting approach predicts classification results based on which category obtains the most votes.

5. Evaluation

In this section, we evaluate the effectiveness, detection performance of DroidPortrait. We first describe the dataset used in this evaluation. Second, we evaluate the efficiency of the behavioral characteristic selection, as discussed in Section 3.1.3. Third, we evaluate the behavioral portrait model build by selected behavioral characteristics. Finally, we use the DroidPortrait to detect Android malware from an unknown Android app and compare it with other well-known Android malware detection approaches.

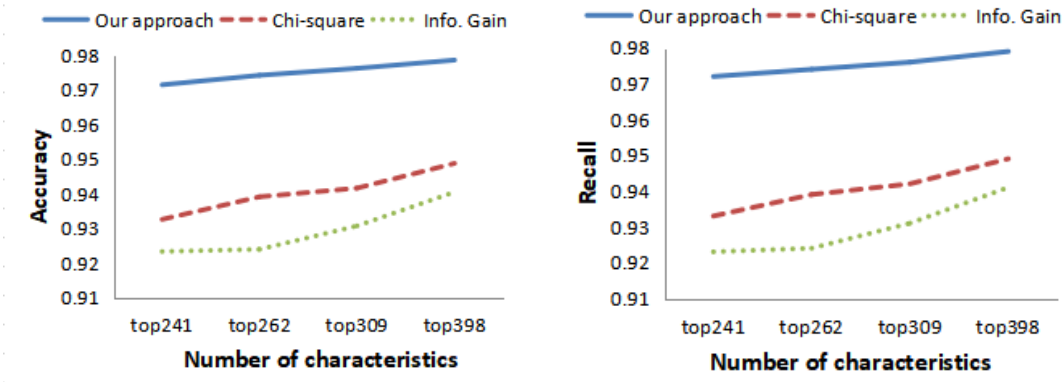
5.1. Dataset

The dataset used to evaluate our approach consists of three parts: the first part is composed of Android apps downloaded from the official Google Play market, which contains popular Android apps from each category. This part of the dataset is considered as benign apps because of the strict audit mechanism of the Google Play market, and we randomly choose 3986 apps from 90,000 of them. The second part of the dataset is composed of several well-known malware datasets, such as Drebin [7], Android Malware Genome Project and the Contagio Community. We collected 3986 Android malwares from these kinds of datasets in total. The third part of the dataset consists of Android apps downloaded from several unofficial markets, and we consider them an unknown type of Android app. We collect about 40,000 apps and randomly select 1515 among them.

5.2. Evaluating Behavioral Characteristic Selection Approach

5.2.1. Static Behavioral Characteristic Selection

To evaluate the performance of the behavioral characteristic selection approach, we first set the parameter α value as 0.5, 0.6, 0.7, 0.8 to select static behavioral characteristics from the extracted ones, and obtain four subsets of behavioral characteristics contain 241, 262, 309 and 398, respectively. Then, we also use two well-known feature selection approaches (Chi-Squared, Information Gain) to select four subsets with the same number of behavioral characteristics from the same extracted behavioral characteristics. Finally, we fed these selected behavioral characteristics into the optimized random forest algorithm to train the classification model to evaluate the performance of the proposed selection approach based on 10-fold cross-validation. The detailed results are shown in Figure 8.



(**a**) Accuracy comparison (**b**) Recall comparison

Figure 8. Performance comparisons.

Figure 8 shows that it significantly outperforms the other algorithms in terms of accuracy and recall. Because our approach selects behavioral characteristics based on their occurrence frequency, which can reflect intrinsic behavioral contain by Android malware. The other two selection algorithms select characteristics based on statistical results, which may select more common characteristics and cause lower accuracy and recall during model training. Moreover, with the number of selected characteristics increase, accuracy and recall of our approach keep increase. Because the selected characteristics could depict Android malware intrinsically and this kind result also reflects Android malware contain many different behaviors.

5.2.2. Dynamic Behavioral Characteristic Selection

In Section 3.1.3, we already discussed behavioral characteristics from the network and system call dimension selection approach. We fed these selected characteristics into the optimized random forest algorithm to evaluate the quality based on 10-fold cross-validation. Table 3 shows detailed results.

Table 3. Results of the selected dynamic behavioral characteristic evaluation.

Behavioral Characteristics	Accuracy	Precision	Recall	F-Measure
Network	0.929	0.929	0.929	0.929
System call	0.907	0.907	0.907	0.907

In Table 3, we select four metrics to evaluate the quality of model training based on the selected dynamic behavioral characteristics. Results show that each metric can achieve more than 90%, which means the selected dynamic behavioral characteristics are suited for building a classification model to depict Android malware dynamic behavior, as these selected dynamic

behavioral characteristics can distinguish between Android benign apps and malware. For example, the number of bytes represents the number of bytes transmitted between app and server. Benign apps generally have a wide variety of functions, so their network activities are very diverse, which may include text messaging, web browsing, image viewing and others. Therefore, these network activities typically vary remarkably in terms of the number of packets. On the other hand, malware usually focuses on sending out private data, usually of standard size, regardless of smartphone use. As a result, we would assume that the number of packets per flow is consistent among different malicious apps.

5.3. Evaluation of Behavioral Portrait

5.3.1. Behavioral Portrait Visualization

In this evaluation, we visualize the behavioral portrait of the Android benign app and malware based on the selected behavioral characteristics and conduct a visual comparison between two kinds of app. The detailed comparison results are shown in Figure 9.

(**a**) (**b**)

Figure 9. Visualization comparison of behavioral portrait between Android benign app and malware. (**a**) Behavioral portrait of Android malware; (**b**) behavioral portrait of Android benign app.

From Figure 9, each kind of portrait contains several behavioral characteristics. First, we can see that majority of Android benign apps contain behavioral characteristics for app development, implement the functionality. However, most Android malware includes behavioral characteristics for accessing sensitive information and executing a command, load library and others. Second, we can nitidly understand specific functions provided by the Android benign app and malware. These findings could help Android security researchers to analyze Android malware behavior based on these functions.

5.3.2. Performance of Characteristic Behavioral Dimension

Next, we plan to evaluate the performance of behavioral characteristics from each dimension during ORF classification model training. The results are shown in Figure 10.

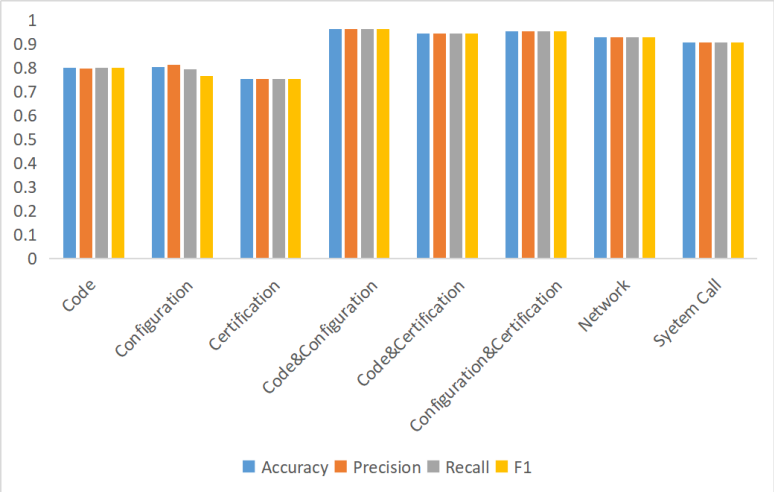


Figure 10. Performance of each dimension and dimension combination.

As shown in Figure 10, we evaluate the performance of five behavioral characteristic dimensions and two kinds of behavioral characteristic combinations. First, we found that each single static behavioral characteristic dimension (code, configuration, certification) cannot achieve a satisfying result, four metrics only achieve 0.8 in average about code and Configuration dimension, certification dimension only achieves 0.75 in four metrics. Because these behavioral characteristics only reflect a part of Android malware behavior. Second, the dimension of *Network* and *System Call* are two main dynamic behavioral characteristics which can reflect network and local behaviors of Android malware, so each of them can achieve a better result, four metrics can achieve 0.9 in average. Third, we combined two static behavioral characteristic dimensions and found that model training results are better than a single situation and close to using dynamic behavioral characteristics. Because static behavioral characteristic combination almost covers the static behavior of Android malware, which can depict Android malware more accurately.

5.3.3. Performance of Behavioral Portrait

Next, we plan to evaluate the performance of the proposed behavioral portrait. To achieve this, we fed the portrait into several classification algorithms to train the classification model, such as optimized random forest (ORF), SVM, random forest (RF), C4.5. We also compare the results of these models and detailed results are shown in Figure 11.

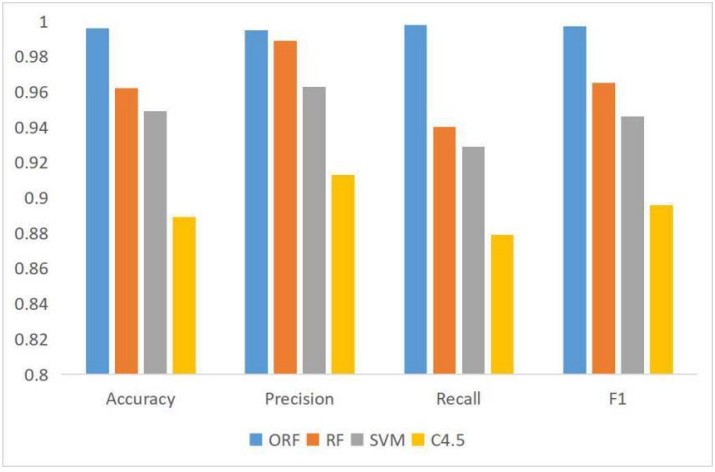


Figure 11. Performance comparison of classification model training.

From Figure 11, we compare ORF algorithm with three other well-known classification algorithms by four metrics (accuracy, precision, recall and F1-measure). The results show that using the proposed behavioral portrait and classification algorithm to build a classification model could achieve satisfying performance. We also found that ORF algorithm can achieve better performance than the other three algorithms during model building. Compare to *RF* algorithm, and the ORF algorithm has a more significant probability of selecting better performance decision trees than traditional *RF* algorithm during the iteration process, which would lead to higher performance. Compare to *C4.5* algorithm, ORF algorithm consists of several decision trees, each decision tree equals *C4.5* algorithm. Therefore, ORF can achieve better performance than *C4.5*. Compare to the *SVM* algorithm, ORF algorithm focuses on the importance of behavioral characteristics during model training, SVM training model based on statistical theory. Therefore, ORF can achieve better performance than *SVM*.

5.4. Evaluating the Performance of Unknown Android App Detection

In this section, we evaluate the performance of DroidPortrait when detecting Android malware from an unknown Android app dataset and compare it with three previous research works, namely FEST [33], Drebin [7] and DroidAPIMiner [5]. To conduct this experiment, we utilize the unknown type of Android apps downloaded from well-known unofficial markets that contain 1515 Android apps. Figure 12 shows the detection results of our approach and the comparison approaches.

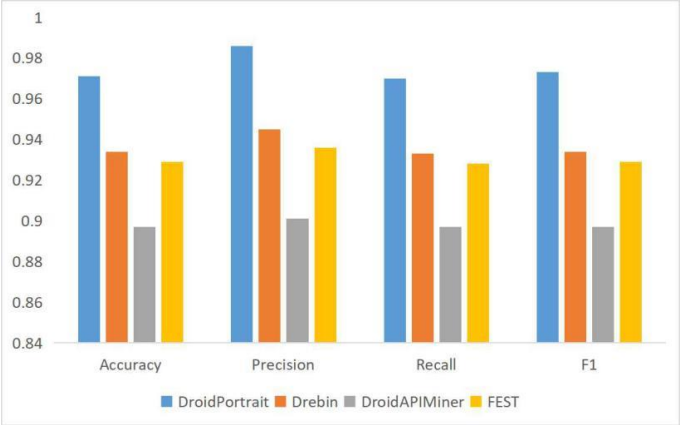


Figure 12. Performance comparison during detecting Android malware from unknown Android app dataset.

From Figure 12, we found that our approach can achieve the best performance among the four approaches. For example, our approach achieves 0.971 in overall accuracy, and the other three works achieve 0.929, 0.934 and 0.897, respectively. DroidPortrait can achieve better precision, recall and f-measure score than *FEST*, Drebin and *DroidAPIMiner*, which are 0.986, 0.97 and 0.973, respectively. The reason contains three parts: first, these approaches extract a part of behaviors from Android malware, Drebin focuses on static behavior, and DroidAPIMiner focuses on API behavior. Our approach extracts more comprehensive behaviors than three of them to build an accurate, informative behavioral portrait of Android malware. Second, we employ a feature selection approach after feature extraction and select more intrinsic behavioral characteristics. Third, we optimize the random forest algorithm to build a classification model with better generalization ability.

6. Conclusions

In this study, we design and develop an Android malware behavioral portrait construction approach that can depict Android malware behaviors accurately and comprehensively and then detect them with excellent performance. To achieve this goal, this study mainly finishes these works as followings:

1. We extract two categories of behavioral characteristics from the Android app, namely, static behavioral characteristics and dynamic behavioral characteristics. These kinds of characteristic can cover Android malware behavior as much as possible. Meanwhile, we proposed two kinds of characteristic selection approaches for both kinds of behavioral characteristics to find intrinsic behavioral characteristics for Android malware detection;
2. We divide these behavioral characteristics into five dimensions to build a behavioral tag. Then, we employ these behavioral tags are utilized to build a portrait of Android malware. After portrait construction, we employ two kinds of machine learning algorithms to train the Android malware behavioral portrait;
3. We conduct three kinds of experiments to evaluate the proposed approach. First, we evaluate static and dynamic behavioral characteristic selection and using the selected behavioral characteristics to train classification models can achieve more than 90% accuracy and recall. Second, we evaluate the portrait of Android malware, which includes portrait visualize, the performance of behavioral dimension and portrait. We also compare with three well-known classification algorithms. Third, we evaluate the performance of our approach in detecting unknown Android malware and compare it with three well-known detection approaches.

Author Contributions: Conceptualization, X.S.; methodology, X.S. and X.L.; validation, X.S., W.L. (Wenjia Li) and K.-C.L.; formal analysis, X.L. and W.L. (Wei Liang), writing—original draft preparation, X.S. and W.L. (Wenjia Li); writing—review and editing W.L. (Wenjia Li) and K.L.; visualization, X.S. and W.L. (Wei Liang); supervision, L.X., X.L.; funding acquisition L.X. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported in part by the Science and Technology Project of Hunan Province of China (No. 2017SK1040), supported by the Research Foundation of Education Bureau of Hunan Province, China (No. 18B548), supported by the Science and Technology Projects of Hunan Province (No. 2018JJ2108), supported by the Open Research Fund of Hunan Provincial Key Laboratory of Network Investigational Technology (No. 2017WLZC006).

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

References

1. 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]
2. Aafer, Y.; Du, W.L.; Yin, H. Droidapiminer: Mining api-level features for robust malware detection in Android. In Proceedings of the International ICST Conference on Security and Privacy in Communication Networks, Orlando, FL, USA, 23–25 October 2013; pp. 86–103.
3. Talha, K.A.; Alper, D.I.; Aydin, C. Apk auditor: Permission-based android malware detection system. *Digit. Investig.* **2015**, *13*, 1–14. [CrossRef]
4. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. In Proceedings of the Annual Symposium on Network and Distributed System Security, San Diego, CA, USA, 23–26 February 2014.
5. Xie, N.; Wang, X.; Wang, W.; Liu, J. Fingerprinting Android malware families. *Front. Comput. Sci.* **2018**, *13*, 637–646. [CrossRef]
6. Liang, W.; Li, K.C.; Long, J.; Kui, X.; Zomaya, A. An Industrial Network Intrusion Detection Algorithm based on Multi-Characteristic Data Clustering Optimization Model. *IEEE Trans. Ind. Inform.* **2019**, *16*, 2063–2071. [CrossRef]
7. Deepa, K.; Radhamani, G.; Vinod, P.; Shojafar, M.; Kumar, N.; Conti, M. Identification of Android malware using refined system calls. *Concurr. Comput. Pract. Exp.* **2019**, 31. [CrossRef]
8. Yerima, S.Y.; Alzaylaee, M.K.; Sezer, S. Machine Learning Based Dynamic Analysis of Android apps with Improved Code Coverage. *EURASIP J. Inf. Secur.* **2019**, *1*, 1–24. [CrossRef]
9. Li, X.; Niu, J.; Ma, J.; Wang, W.; Liu, C. Cryptanalysis and improvement of a biometrics-based remote user authentication scheme using smart card. *J. Netw. Comput. Appl.* **2011**, *34*, 73–79. [CrossRef]

10. Li, X.; Ma, J.; Wang, W.; Xiong, Y.; Zhang, J. A novel smart card and dynamic ID based remote user authentication scheme for multi-server environment. *Math. Comput. Model.* **2013**, *58*, 85–95. [CrossRef]
11. Li, X.; Niu, J.; Khan, M.K.; Liao, J. An enhanced smart card based remote user password authentication scheme. *J. Netw. Comput. Appl.* **2013**, *36*, 1365–1371. [CrossRef]
12. Enck, W.; Gilbert, P.; Chun, B.; Cox, L.; Jung, J.; McDaniel, P.; Sheth, A. TaintDroid: An information-flow tracking system for real-time privacy monitoring on smartphones. *ACM Trans. Comput. Syst.* **2014**, *32*, 1–29. [CrossRef]
13. Su, X.; Liu, X.; Lin, J.; He, S.; Fu, Z.; Li, W. De-cloaking Malicious Activities in Smartphones Using HTTP Flow Mining. *KSII Trans. Internet Inf. Syst.* **2017**, *11*, 3230–3253.
14. Wu, F.; Lu, J.; Cao, W. Multi Feature Detection of Malicious Programs Based on Android Platform. *J. Chin. Comput. Syst.* **2018**, *39*, 151–155.
15. Nath, H. Static Malware Analysis Using Machine Learning Methods. In *Recent Trends in Computer Networks and Distributed Systems Security*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 440–450.
16. Gorla, A.; Tavecchia, I.; Gross, F.; Zeller, A. Checking app behavior against app descriptions. In Proceedings of the 36th International Conference on Software Engineering (ICSE), Hyderabad, India, 31 May–7 June 2014; pp. 1025–2035.
17. Yu, F.; Saswat, A.; Isil, D.; Alex, A. Apposcopy: Semantics-based detection of android malware through static analysis. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), Hong Kong, China, 16–21 November 2014; pp. 576–587.
18. Zhu, H.; You, Z.; Zhu, Z.; Shi, W.; Chen, X.; Cheng, L. DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing* **2017**, *272*, 638–646. [CrossRef]
19. Xie, L.; Li, S. Android malware detection model based on Bagging-SVM. *J. Comput. Appl.* **2018**, *38*, 818–823.
20. Park, J.; Chun, H.; Jing, S. API and permission-based classification system for Android malware analysis. In Proceedings of the 2018 International Conference on Information Networking (ICOIN), Chiang Mai, Thailand, 10–12 January 2018; Volume 1, pp. 930–935.
21. Junaid, M.; Liu, D.G.; Kung, D. Dexteroid: Detecting malicious behaviors in Android apps using reverse-engineered life cycle models. *Comput. Secur.* **2016**, *59*, 92–117. [CrossRef]
22. Wu, D.; Mao, C.; Wei, T.; Lee, H.; Wu, K. DroidMat: Android Malware Detection through Manifest and API Calls Tracing. In Proceedings of the 2012 Seventh Asia Joint Conference on Information Security, Tokyo, Japan, 9–10 August 2012; pp. 62–69.
23. Yan, L.K.; Yin, H. Droidscope: Seamlessly reconstructing the os and Dalvik semantic views for dynamic android malware analysis. In Proceedings of the 21st USENIX Conference on Security Symposium, Bellevue, WA, USA, 8–10 August 2012; p. 29.
24. Burguera, I.; Zurutuza, U.; Tehrani, S.N. Crowdroid: Behavior-based malware detection system for Android. In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, Chicago, IL, USA, 17 October 2011; pp. 15–26.
25. 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. [CrossRef]
26. Mehtab, A.; Shahid, W.B.; Yaqoob, T.; Abbas, H.; Afzal, H.; Saqib, M.N. AdDroid: Rule-Based Machine Learning Framework for Android Malware Analysis. *Mob. Netw. Appl.* **2019**, *24*, 1–13. [CrossRef]
27. Hou, S.; Saas, A.; Chen, L.; Ye, Y. Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs. In Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW), Omaha, NE, USA, 13–16 October 2016; pp. 104–111.
28. Ali, F.; Badrul, A.N.; Rosli, S. Evaluation of Network Traffic Analysis Using Fuzzy C-Means Clustering Algorithm in Mobile Malware Detection. *Adv. Sci. Lett.* **2018**, *24*, 929–932.
29. Mohammed, K.; Suleiman, Y.; Sakir, S. DL-Droid: Deep learning based android malware detection using real devices. *Comput. Secur.* **2020**, *89*, 101663.
30. Su, X.; Chuah, M.; Tan, G. Smartphone Dual Defense Protection Framework: Detecting malicious applications in Android Markets. In Proceedings of the 2012 8th International Conference on Mobile Ad-hoc and Sensor Networks, Chengdu, China, 14–16 December 2012; pp. 153–160.
31. Yu, L.; Pan, Z.L.; Liu, J.J.; Shen, Y. Android Malware Detection Technology Based on Improved Bayesian Classification. In Proceedings of the 2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control, Shenyang, China, 21–23 September 2013; pp. 1338–1341.

32. Wang, C.Z.; Liang, G.; Yang, J.; Chen, W. The Method of Malware Detection based on Information Gain Characteristics Optimization Select. *Netw. Comput. Secur.* **2013**, *4*, 13–17.
33. Zhao, K.; Zhang, D.F.; Su, X.; Li, W.J. Fest: A Feature Extraction and Selection Tool for Android Malware Detection. In Proceedings of the 20th IEEE Symposium on Computers and Communication, Larnaca, Cyprus, 6–9 July 2015; pp. 714–720.
34. Yang, H.; Zhang, Y.; Hu, Y.; Liu, Q. Android malware detection method based on permission sequential pattern mining algorithm. *J. Commun.* **2013**, *34*, 106–115.
35. Heermann, P.D.; Khazenie, N. Classification of multispectral remote sensing data using a back-propagation neural network. *IEEE Trans. Geosci. Remote Sens.* **1992**, *30*, 81–88. [CrossRef]
36. Srivastava, N.; Hinton, G.; Krizhevsky, A. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
37. Liang, W.; Huang, W.; Long, J.; Zhang, K.; Li, K.; Zhang, D. Deep Reinforcement Learning for Resource Protection and Real-time Detection in IoT Environment. *IEEE Internet Things J.* **2020**. [CrossRef]
38. Lipowski, A.; Lipowska, D. Roulette-wheel selection via stochastic acceptance. *Phys. A Stat. Mech. Appl.* **2012**, *391*, 2193–2196. [CrossRef]
39. Liang, W.; Fan, Y.; Li, K.; Zhang, D.; Gaudiot, J. Secure Data Storage and Recovery in Industrial Blockchain Network Environments. *IEEE Trans. Ind. Inform.* **2020**. [CrossRef]
40. Liang, W.; Tang, M.; Long, J.; Peng, X.; Xu, J.; Li, K.C. A Secure FaBric Blockchain based Data Transmission Technique for Industrial IoT. *IEEE Trans. Ind. Inform* **2019**, *15*, 3582–3592. [CrossRef]