Hindawi
Mobile Information Systems
Volume 2018, Article ID 4157156, 9 pages
https://doi.org/10.1155/2018/4157156

Research Article

TinyDroid: A Lightweight and Efficient Model for Android Malware Detection and Classification

Tieming Chen,[1] **Qingyu Mao**,[1] **Yimin Yang**,[1] **Mingqi Lv**,[1] **and Jianming Zhu**[2]

[1]*College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China*
[2]*College of Information Engineering, Central University of Finance and Economics, Beijing 100081, China*

Correspondence should be addressed to Tieming Chen; tmchen@zjut.edu.cn

Received 10 June 2018; Revised 19 August 2018; Accepted 27 August 2018; Published 17 October 2018

Academic Editor: Prosanta Gope

With the popularity of Android applications, Android malware has an exponential growth trend. In order to detect Android malware effectively, this paper proposes a novel lightweight static detection model, *TinyDroid*, using instruction simplification and machine learning technique. First, a symbol-based simplification method is proposed to abstract the opcode sequence decompiled from Android Dalvik Executable files. Then, N-gram is employed to extract features from the simplified opcode sequence, and a classifier is trained for the malware detection and classification tasks. To improve the efficiency and scalability of the proposed detection model, a compression procedure is also used to reduce features and select exemplars for the malware sample dataset. *TinyDroid* is compared against the state-of-the-art antivirus tools in real world using Drebin dataset. The experimental results show that *TinyDroid* can get a higher accuracy rate and lower false alarm rate with satisfied efficiency.

1. Introduction

With the fast development of mobile Internet, the popularity of mobile devices, and the rapid growth of mobile applications, smartphones have become the most popular tools for people to access the Internet. The statistics from Gartner show that more than 400 million smartphones were sold globally in the final quarter of 2015, with Android operating system accounting for 80.7% [1]. As of February 2016, the number of Android apps in Google Play is approaching 2 million. Meanwhile, internal application vulnerabilities and malware are the primary threat to the security of Android applications. Since application developers lack the necessary knowledge and skills about security code audit, it is easy to create vulnerable code in every development step in the software life cycle. Once the attackers exploit these vulnerabilities, they will damage the confidentiality, integrity, and availability of applications. The creator of an Android malware usually inserts a small malicious code to a popular application and spreads the malware through some third-party app stores without security management [2, 3]. According to a new report from the mobile department of Alibaba, 18% of the Android devices have ever been infected with viruses, and 95% of the popular mobile applications have been counterfeited and kinds of malware have evolved quickly [4]. The malicious activities of malware typically include system failure, information leakage, and data corruption, so malicious mobile code will damage the user's benefits.

The existing methods of malware detection include signature-based detection methods and behaviour-based detection methods. Signature-based detection methods detect malware by comparing the binary code of software with a database that stores all the signatures of the known malware. Although the signature-based detection methods have the advantages of simplicity and efficiency with high accuracy, they cannot detect an unknown malware and need to maintain a vast signature database.

On the other hand, behaviour-based detection methods detect malware by comparing the behaviour pattern of software with that of known malware. Although this method leads to a high rate of false alarm, it can effectively detect unknown malware and compensate for the disadvantages of signature-based detection methods. Two types

of analysis could be performed on behaviour-based detection, that is, static analysis [5–8] and dynamic analysis [9–14]. The dynamic analysis can analyze object-code obfuscation to some extent by monitoring the run-time situations of an application, which runs in a secure sandbox mode, but it requires a significant amount of computing resources and has low code coverage. Instead, static analysis can get features based on analysis on sequences of instructions obtained using reverse engineering. In a word, static analysis has the advantages of efficiency and high code coverage, and it is a relatively lightweight process as compared to the dynamic analysis.

In this paper, we firstly identify the problem of Android malware detection and summarize a model of threats faced by developers and researchers. Then, we propose a detection method named *TinyDroid* using instruction simplification and machine learning technique. The main contributions of this paper are twofold as follows:

(i) N-gram direct on the reduced symbolic Dalvik opcode sequences instead of the original full instructions

A simple symbol set is introduced to simplify the original Dalvik instructions sequence, where one series of instructions with similar function can be assigned as one symbol. N-gram technique is employed to handle the symbolic sequence.

(ii) Reduction on both features and number of samples for building a lightweight model

In order to achieve high efficiency of detection and classification, a further reduced scheme is proposed to largely cut down the number of N-gram items and training samples. Specifically, information gain is employed for attribute reduction and affinity propagation is used for sample selection.

2. Related Works

In this section, we review related work in two areas: usage of opcodes and N-gram to detect malware and lightweight approaches to detect Android malware.

2.1. Usage of Opcodes and N-Gram to Detect Malware. Some references study the effectiveness of opcodes for detecting or characterising malware. Rad et al. [15] use the histogram of opcodes to classify metamorphic virus family variants. This method takes into account the frequency distribution of opcodes but ignores the sequential patterns of opcodes.

References [16–18] have investigated the effectiveness of N-gram opcodes extracted from the disassembled application and represented programs as N-gram density that made a robust indicator of malware. However, these methods have a large number of features and do not consider efficiency.

Wolfe et al. [19] extracted Java bytecode from the Android application and computed the N-gram frequencies of the bytecode. Then, they applied dimensionality reduction on the N-gram frequencies using principal components analysis (PCA). This approach, where the dimensionality of the feature set is very large and difficult to handle due to the unrestricted selection of features, goes to the other extreme when compared to the permission approach.

2.2. Lightweight Approaches to Detect Android Malware. Some static and lightweight mechanisms for detecting Android malware have been introduced. For example, in 2012, Zhao et al. [20] proposed a lightweight framework named RobotDroid that uses active learning on Android applications to induce an accurate detection model with minimal labeled samples.

In 2013, Santos et al. [21] proposed an efficient model, which is based on the frequency of the appearance of opcode sequences. Also, they provided empirical validation which shows that opcode sequence is feasible to detect unknown malware.

In 2014, Arp et al. [22] designed a lightweight method of Android malware detection. They apply linear-time static analysis and learning techniques for efficiency.

In 2017, DroidSieve [23] relies on several syntactic and resource-centric features, which are robust and computationally inexpensive to detect Android malware.

These methods have not considered the issue of scalability. This problem produces excessive storage requirements, increases time complexity, and impairs the general accuracy of the models.

3. Threat Model

Before introducing our detection method, it is necessary to summarize the threats we currently encounter in Android malware detection. Four threats arise from the Android malware detection:

Repacking and Reflection. A more recent trend we have observed is the increasing prevalence of Android malware leveraging packing technology. The application can use the secondary packaging to modify the source code or add a small amount of malicious code due to insufficient reinforcement. Java's reflection mechanism is expressed as a way to dynamically get information and call objects. Malware creators use reflection mechanisms as an important way to hide malicious behaviour in software. Malicious applications, in order to evade static analysis, can transmit malicious code by invoking sensitive methods at reflection at runtime.

Malware in MultiDEX. Android programs are compiled into Dalvik Executable (DEX) files. A typical Android app has a single DEX file, and some complex applications contain multiple DEX files. Therefore, some malware would include malicious code in multiple DEX files. If only a single DEX file is analyzed, it may not be able to distinguish whether it is malicious. This simple step can serve as an evasion technique against static analysis.

Run-Based Malware. Instant Run is a new feature in Android system. It allows developers to quickly deploy patches to a debug application .zip file into the application. Some malware authors hide the malware payload portion of their app in code fragments that are hidden in the zip file used by Instant Run. This approach to detection evasion can only be used on Android Lollipop and later SDK levels. Although it cannot be used on apps in Google Play, it was still possible to spread in the third application markets.

Native Code. To avoid static detectors at the bytecode level, malicious code relies on some special native API functions and kernel system functions to infect, spread, and hide. Moreover, sometimes, malware also embeds malicious payload within the native content.

Each threat can evade some methods of malicious detection. To cope with some of the above threats, *TinyDroid* is proposed to detect Android malware using instruction simplification and machine learning technique.

4. Design of TinyDroid

4.1. Overview. To solve the aforementioned problems, this paper focuses on lightweight machine learning-based detection of Android malware using Dalvik instructions simplification, exemplar selection, and optimization.

We develop a detection system *TinyDroid*, which has a high accuracy on detection and classification of Android malware. *TinyDroid* includes two procedures. The first one is to create the detection model and classification model. The second is to predict Android malware. The main steps are shown, respectively, in Figures 1 and 2.

Training set for the detection model is divided into two subsets: malware apps and benign apps. An APK file always contains several DEX files, which can be executed in Dalvik Virtual Machine. The APK file can be disassembled into smali codes by Apktool [24], where smali is an explanation for the Dalvik bytecode and can be simplified into symbolic instructions by our method. The detection model is supported by N-gram, exemplar selection method, and machine learning algorithm. Similarly, the classification model has the same steps as in Figure 1, except that it needs to divide the training set into different family subsets.

As shown in Figure 2, the apps to be predicted are firstly preprocessed and the reduced N-gram sequences are extracted. The malware can be detected based on the detection model. Furthermore, the information about malware family can be obtained based on the classification model.

The threats mentioned in Section 3 are almost the hardest part of all malicious detection methods. Our solution mitigates some threats as follows:

Repacking and Reflection. Our study has shown that reflection APIs are more frequently used by malware set than in the benign set, which makes them part of the feature vector for the classification.

Malware in MultiDEX. In the feature extraction process, we analyze all the DEX files to extract the opcodes and generate the N-gram sequences.

Run-Based Malware. This is a difficult point of malicious detection and cannot be detected by static analysis. The method in this paper may be insufficient for such threats.

Native Code. Since our detection tool only works at Dalvik bytecode level, it is not able to detect any dangerous methods invoked. However, the use of invoking calls such as *invoke-direct* is also used as a feature by our model.

4.2. Feature Extraction and Simplification. Dalvik instructions contained 230 instructions such as method call instructions, data manipulation instructions, and return instructions [25]. For malicious detection, the full instruction set is too complicated. In addition, there are some instructions of the same semantics. Condensing these instructions would not affect the test results but also improve the efficiency of detection. For example, Hang et al. [26] classified all 218 Dalvik instructions into a simplified instruction set with only 13 types of instructions, which is called SDIL.

Thus, we have designed a more simplified symbolic instruction set that is suitable for more efficient machine learning. The detailed steps are as follows:

(1) We analyze many smali source code files and found that many instructions have multiple versions due to the different parameters. For example, consider two Dalvik move instructions (i.e., *move-object/from16 vAA, vBBBB; move-object/16 vAAAA, vBBBB*), the only difference between them is how many bits they use to represent registers indices (*vAA* and *vAAAA* require 8 and 16 bits). According to the above rules, finally, we identified 107 representative Dalvik instructions which have the core semantics and high frequency of occurrence.

(2) Then, we assign them into 10 types of symbols according to the semantics of each instruction. For example, *G* stands for all the jump instructions, it could be *goto*, *goto16*, or *goto/32*, because they have the almost the same semantics. Table 1 shows the full symbolic instruction set with different semantics we proposed.

(3) We use Apktool to decompile apk for the sake of getting smali source code. Then, we extract instructions according to the reduced instruction set. Finally, we replace ordered instructions with symbols.

N-gram [27] is widely used for natural language processing (NLP) and also used for the analysis of malware [16, 28]. When the N-gram sequences of Dalvik instructions are generated, they can be easily analyzed for the detection of Android malware. N-gram is a type of probabilistic model, and it assumes that the appearance of the nth word only correlates with the previous $n-1$ words. For example,

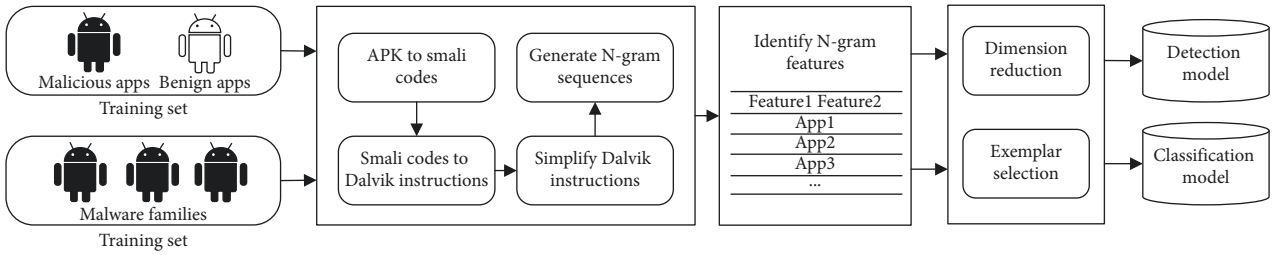

Figure 1: The structure of the proposed detection and classification model.

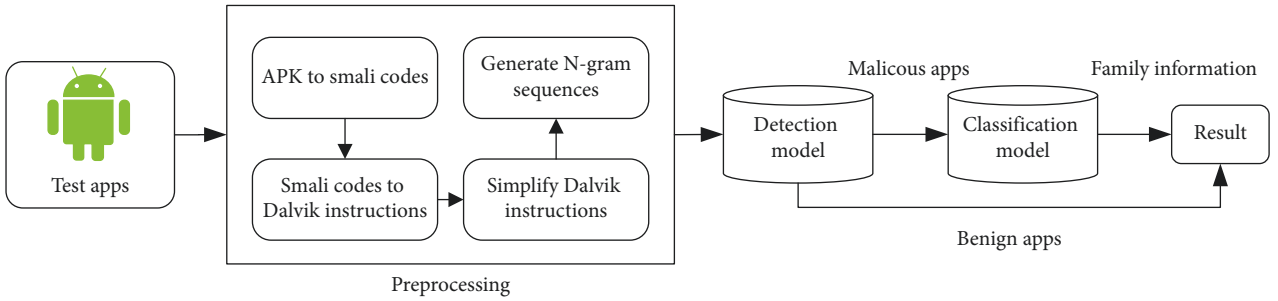

Figure 2: The process of predicting Android malware.

suppose that we have a sequence of symbolic instructions "MRGITPV," the 3-gram features are [{MRG}, {RGI}, {GIT}, {ITP}, and {TPV}]. Table 2 shows the occurrence rate of several 3-gram features in the selected APKs.

Since the number of N-gram features is excessive, so we use information gain (IG) for feature reduction as it has been proven to be effective [29]. The information gain of a given feature X with respect to the class attribute Y is the reduction in uncertainty about the value of Y, after observing values of X. It is denoted as $IG(Y|X)$. The uncertainty about the value of Y is measured by its entropy defined as

$$H(Y) = -\sum_{i} P(y_i)\log_2(P(y_i)), \quad (1)$$

where $P(y_i)$ is the prior probabilities for all values of Y. The uncertainty about the value of Y after observing values of X is given by the conditional entropy of Y given X defined as

$$H(Y|X) = -\sum_{j} P(x_j)\sum_{i} P(y_i|x_j)\log_2(P(y_i|x_j)), \quad (2)$$

where $P(y_i|x_j)$ is the posterior probabilities of Y given the values of X. The information gain is thus defined as $IG(Y|X) = H(Y) - H(Y|X)$. According to this measure, a feature X is regarded more correlated to class Y than feature Z, if $IG(Y|X) > IG(Y|Z)$. By calculating information gain, we can rank the correlations of each feature to the class and select key features based on this ranking.

4.3. Exemplar Selection and Optimization. Recent researches show that the clustering algorithm affinity propagation [30] can be used to generate good representative data for intrusion detection training [31]. The intrusion detection training is very similar to the malware detection training, so we choose this clustering algorithm in our paper. Affinity propagation is employed as a reduction method to select a smaller set composed of representative exemplars from the original large training data. When the size of the training dataset becomes smaller, the detection model efficiency would be improved after the sample optimization.

Affinity propagation (AP) clusters instances by passing messages between data points iteratively. Unlike clustering algorithms such as k-means, AP does not require the number of clusters to be determined or estimated before running the algorithm, and the only parameter that should be set is the *preference*. The number of clusters generated by AP is decided by its *preference*, so we choose the preferences between the minimum and the maximum of the similarities to generate the expected number of clusters that are separately distributed.

The time complexity of AP is $O(N^2T)$, where N is the number of samples and T is the number of iterations until convergence. Further, the memory complexity is $O(N^2)$ if a dense similarity matrix is used, but reducible if a sparse similarity matrix is used.

5. Evaluation

5.1. Experimental Setup. Classification algorithm is a typical method of machine learning. The most common evaluation metrics include true positive (TP), false positive (FP), true negative (TN), and false negative (FN) [32]. These four metrics can make up a confusion matrix as shown in Table 3.

Depending on these basic metrics, a series of common evaluation metrics can also be generated as follows:

Table 1: Symbolic instruction set definition.

Symbols	Semantics	Quantity	Representative opcodes prefixes
C	Comparison	5	cmpl-float|cmpg-float|cmpl-double|cmpg-double|cmp-long
D	Definition	11	const|const/4|const/16|const-wide|const/high|const-string
M	Manipulation	13	move|move-wide|move-object|move-result|move-exception
R	Return	4	return|return-void|return-wide|return-object
L	Monitor	2	monitor-enter|monitor-exit
G	Jump	3	goto|goto/16|goto/32
I	Judgment	12	if-eq|if-ne|if-lt|if-ge|if-gt|if-le|if-eqz|if-nez|if-ltz|if-gez|if-gtz|if-lez
T	Reading	21	aget|iget|sget|aget-wide|aget-object|aget-boolea|aget-byte|aget-char
P	Writing	21	aput|iput|sput|aput-wide|aput-object|aput-boolea|aput-byte|aput-char
V	Method call	15	invoke-virtual|invoke-super|invoke-direct|invoke-static

Table 2: Illustration for the frequency statistics of 3-gram features.

	CCC	CCD	CCM	CCR	...	VVV
DroidKungFu.apk	0.032016	0.010601	0.019507	0.036822	...	0.071454
BaseBridge.apk	0.059207	0.011801	0.018162	0.032164	...	0.047486
Plankton.apk	0.030508	0.006391	0.017376	0.038064	...	0.053952
...	...	...	...	...	...	...

Table 3: Confusion matrix.

Prediction	Malicious	Benign
Malicious	TP	FN
Benign	FP	TN

$$\begin{aligned}
\text{TP rate} = \text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\
\text{FP rate} &= \frac{\text{FP}}{\text{FP} + \text{TN}}, \\
\text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\
\text{accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \\
\text{F} - \text{measure} &= \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}.
\end{aligned} \tag{3}$$

True-positive rate (TPR) is sometimes called detection rate or recall rate. False-positive rate (FPR) is sometimes called false alarm rate. Receiver operating characteristic (ROC) is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the TPR against FPR at various threshold settings. The area under the curve (AUC) is the area under the ROC curve. The closer that this value is to 1, the better the performance is.

We used a dataset of real Android applications and malware to conduct the experiments. The dataset includes all Android malware from Drebin [33]. Additionally, all the benign applications are randomly collected from Google Play [34]. Previous work [35] has pointed out that only 2 in 52208 applications were malware from Google Play, so the applications can be considered as benign applications.

We utilize Waikato Environment for Knowledge Analysis (Weka) to train data and create the model. Weka is an open-source software, and it integrates a lot of machine learning algorithms, so we choose it as our experimental tool.

5.2. Malware Detection Performance. We randomly choose 300 malware and benign samples, respectively, as experimental data. 2-gram sequences, 3-gram sequences, and 4-gram sequences are generated, respectively. These sequences are respectively fed into Random Forest, Support Vector Machine (SVM), *k*-Nearest Neighbor (kNN), and Naive Bayes to build the classifier. The time complexity is shown in Table 4. 10-fold cross-validation is selected to evaluate our method. The results are shown in Tables 5–7.

AUC, TPR, FPR, precision, recall, and F-Measure are selected as performance measurement indicators. In particular, AUC is an important indicator because it can comprehensively reflect TPR and FPR. As shown in Table 5, Random Forest has the best performance on AUC and FPR, and its TPR is a little less than kNN. As shown in Table 6, every indicator of Random Forest is the best. As shown in Table 7, Random Forest performs best on the two indicators of F-Measure and AUC. Through the above analysis, we can see Random Forest is the optimal algorithm among 2-gram sequences, 3-gram sequences, and 4-gram sequences.

Next, we will find out the optimal N-gram sequences based on Random Forest. By analyzing Tables 5–7, we can easily find that Random Forest's AUC from good to bad: 4-gram, 3-gram, and 2-gram; Random Forest's FPR from good to bad: 3-gram, 4-gram, and 2-gram. The performance of 2-gram sequences is the worst, so it is no longer to be considered. 4-gram sequences' AUC is 2%, which is higher than 3-gram sequences', but 4-gram sequences' TPR is 30% higher than 3-gram sequences'. Through the above analysis, 3-gram sequences based on Random Forest is the optimal method.

Table 4: Classifiers' time complexity.

Classifier	Time complexity	Parameter
Random Forest	$O(\text{ndlogn})$	n is the number of samples and d is the number of features
SVM	$O(\max(n,d), \min((n,d)^2))$	n is the number of points and d is the number of features
kNN	$O(\text{ndk})$	n is the number of samples and d is the number of features
Naive Bayes	$O(\text{nd})$	n is the number of samples and d is the number of features

Table 5: The detection results based on 2-gram sequences.

Classifier	TPR	FPR	Precision	Recall	F-Measure	AUC
Random Forest	0.915	0.106	0.876	0.915	0.895	0.97
SVM	0.852	0.173	0.801	0.852	0.826	0.84
kNN	0.93	0.136	0.848	0.93	0.887	0.897
Naive Bayes	0.87	0.221	0.763	0.87	0.813	0.888

Table 6: The detection results based on 3-gram sequences.

Classifier	TPR	FPR	Precision	Recall	F-Measure	AUC
Random Forest	0.956	0.079	0.908	0.956	0.931	0.982
SVM	0.922	0.13	0.853	0.922	0.886	0.896
kNN	0.952	0.148	0.84	0.952	0.892	0.903
Naive Bayes	0.867	0.17	0.807	0.867	0.836	0.903

Table 7: The detection results based on 4-gram sequences.

Classifier	TPR	FPR	Precision	Recall	F-Measure	AUC
Random Forest	0.956	0.103	0.884	0.956	0.918	0.984
SVM	0.944	0.106	0.879	0.944	0.911	0.919
kNN	0.904	0.088	0.894	0.904	0.899	0.909
Naive Bayes	0.9	0.103	0.877	0.9	0.888	0.917

Different size of sample dataset would affect the evaluation results. Three different sizes of sample dataset are randomly generated: 600 samples, 1200 samples, and 2400 samples. The 3-gram sequences based on Random Forest method is used for the experiment. The final model is tested by 10-fold cross-validation. As shown in Table 8 and Figure 3, the results show that performance becomes better when the number of samples increases.

5.3. Comparison of Training Time. In order to further improve malware detection performance, especially for a large volume of training data, affinity propagation is used to generate the excellent representative data for malware detection training. We randomly choose 2400 samples as training data. Then, the training data are processed by AP algorithm; 834 exemplars and 516 exemplars are generated, respectively. The representative exemplars are trained to create a detection model and the model is tested by 10-fold cross-validation. As shown in Table 9, TPR and FPR show little change, but the time cost has declined dramatically. The time performance is improved after exemplar selection.

Then, we compare the proposed method with several baselines. Drebin [23] is a lightweight method of Android

Table 8: The results of different size of samples.

Number of samples	TPR	FPR	Precision	AUC
600 samples	0.956	0.086	0.908	0.982
1200 samples	0.980	0.079	0.917	0.992
2400 samples	0.988	0.077	0.921	0.994

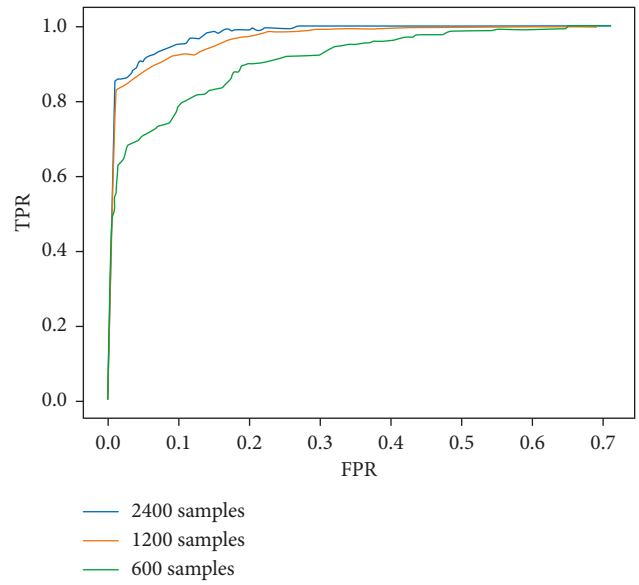

Figure 3: ROC curves with different size of samples.

Table 9: The time and performance effectiveness with different size of exemplars selected.

Training data	Training time	TPR	FPR
2400 samples	2.9 s	0.988	0.077
834 exemplars (after AP)	0.86 s	0.964	0.083
516 exemplars (after AP)	0.56 s	0.959	0.091

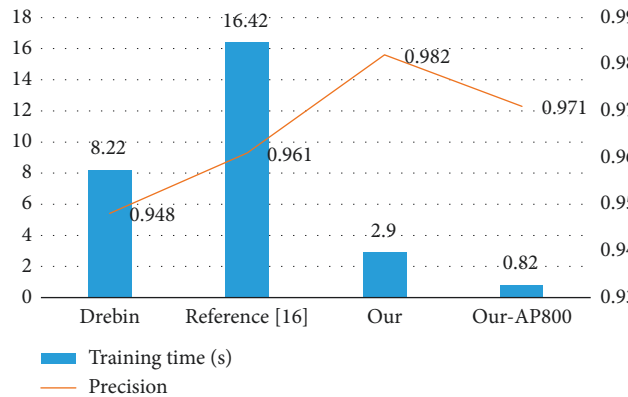

Figure 4: Comparison of training time.

Table 10: Detection rates of TinyDroid and antivirus scanners.

Num.	TinyDroid	Avira	Dr.Web	AVG	Kaspersky	ESET	GDATA	Rising	MSE	Avast
2000 samples	95.3%	100.0%	97.0%	94.0%	91.5%	91.0%	66.0%	49.5%	15.5%	11.0%
4000 samples	98.6%	99.1%	92.7%	92.3%	78.7%	90.0%	42.9%	45.0%	7.5%	10.2%

Table 11: False-positive rates of TinyDroid and antivirus scanners.

Num.	TinyDroid	Avira	Dr.Web	AVG	Kaspersky	ESET	GDATA	Rising	MSE	Avast
2000 samples	5.0%	0.5%	0.5%	0%	0%	0%	0.5%	0.5%	0%	0%
4000 samples	1.4%	1.6%	0.9%	0%	0%	0%	0%	1.2%	0%	0%

malware detection. They apply linear-time static analysis and learning techniques for efficiency. Canfora et al. [16] proposed a method using frequencies of N-grams of opcodes to detect Android malware effectively. We extract the corresponding number of features as much as possible for experiment and use the same dataset (randomly choose 2400 samples) to train three models, respectively. The results are shown in Figure 4.

Here, we divide our method into two parts. In the first part, we only use opcode instruction symbolization and N-gram techniques to reduce feature dimensions (Our in Figure 4). In the second part, on the basis of the first part, we use the AP clustering algorithm for sample selection (Our-AP800 in Figure 4).

We can see that the time spent to perform model training in our method (first part) is about 2.9 s, which is better than the other two baselines. Subsequently, we performed AP clustering algorithm processing on the samples and selected 800 center samples to retrain. We can see that the accuracy of the model has decreased a little, but the time cost has declined dramatically. And our method still has higher accuracy than other methods. The experimental results clearly show that *TinyDroid* is better than the state-of-the-art methods.

5.4. Comparison with Real World AV Scanners. In this part, two different sizes of sample dataset are chosen to evaluate the detection rate of *TinyDroid*. For this experiment, we randomly split the dataset into a training set (60%) and a testing set (40%). To compete with common antivirus products in practice, we send each sample to the VirusBook service [36] and get the output from 9 common antivirus scanners (Avira, Dr.Web, AVG, Kaspersky, ESET, GDATA, Rising, MSE, and Avast). Finally, we obtain the detection rates and false-positive rates by the statistics of output.

The results of the experiments are shown in Table 10 and Table 11. While most scanners detect over 90% of the malware, the detection rates for some scanners are below 50%. Obviously, these antivirus scanners may not be specialized in detecting mobile applications. On the 2000 samples' dataset, *TinyDroid* provides the third best performance with detection of 95.3% and outperforms 7 out of the 9 scanners. On the 4000 samples' dataset, *TinyDroid* provides the second best performance with detection of 98.6% and outperforms 8 out of the 9 scanners. This shows that *TinyDroid* achieves better detection rates when the number of samples increases. Also, these samples have been public for a longer period, so almost all antivirus scanners provide proper signatures for this dataset. The machine learning method has much more advantage than the traditional technique when the samples are unknown malware.

Table 12: The results on the classification of malware family using TinyDroid.

Malware family	TPR	FPR	Precision	Recall	*F*-Measure	AUC
Plankton	0.949	0.003	0.974	0.949	0.961	0.999
DroidKungF	0.833	0.003	0.972	0.833	0.897	0.996
GinMaster	1	0.037	0.755	1	0.86	0.997
FakeDoc	0.976	0	1	0.976	0.988	1
FakeInstalle	0.865	0.006	0.941	0.865	0.901	0.978
Opfake	1	0.019	0.882	1	0.938	0.997
BaseBridge	0.816	0	1	0.816	0.899	0.995
$K_{\min}$	1	0	1	1	1	1
Iconosys	1	0	1	1	1	1

The false-positive rates of the antivirus scanners range from 0% to 1% on our dataset of benign applications. Although the false-positive rate of *TinyDroid* is higher than the antivirus scanners, Table 11 shows that the false-positive rate of *TinyDroid* has seen a decline when the number of samples increases.

5.5. Malware Classification Performance. In addition to the detection performance, we need to evaluate the malware classification performance. In this part, we select 9 top malware families as the experimental data. The experimental data contains a total of 900 samples and is separated into a training set (60%) and a testing set (40%). The results of the experiments are shown in Table 12. All the values of AUC are higher than 0.97 and the values of FPR are lower than 0.04. *TinyDroid* shows a good performance on malware classification.

6. Conclusion

This paper proposes a novel and lightweight static detection system *TinyDroid*. We firstly use the reverse engineering to get the Dalvik instructions from DEX files and simplify the instructions into a small symbol set. Then, the detection model integrating N-gram, exemplar selection method, and machine learning algorithm is set up based on those above reduced symbolic sequences. We compare *TinyDroid* against

the antivirus scanners in real world. The experimental results show that *TinyDroid* outperforms the state-of-the-art tools on detection and classification.

There are still some deficiencies and improvements to be found in this study: The samples used in this paper mainly originate from the academic sample set of the organization or platform and lack metamorphic malware samples. Metamorphic malware could escape the proposed method. It is clearly vulnerable to obfuscation and packing.

In the next step, according to the limitation, dynamic analysis methods can be studied and combined to extract more useful features. Meanwhile, other optimization methods can also be further studied to keep the lightweight and improve the detection efficiency.

Data Availability

Previously reported (Android Malware Applications) data were used to support this study and are available at (https://doi.org/10.14722/ndss.2014.23247). These prior studies (and datasets) are cited at relevant places within the text as references [16, 24]. And the benign applications are randomly collected from Google Play and are available at (https://play.google.com/store). All samples included in this study are available upon request by contact with the corresponding author.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This paper was partially supported by the Key National Natural Science Foundation of China with Grant nos. 61772026, U1509214, and 61602412.

References

[1] "Gartner Report," http://www.gartner.com/newsroom/id/3215217.

[2] A. Shabtai, Y. Fledel, U. Kanonov et al., "Google android: a state-of-the-art review of security mechanisms," 2009, https://arxiv.org/abs/0912.5101.

[3] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'11)*, pp. 3–14, ACM, Chicago, IL, USA, 2011.

[4] Alibaba Security, http://jaq.alibaba.com/.

[5] H. Kang, J. W. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, Article ID 479174, 2015.

[6] J. W. Jang and H. K. Kim, "Function-oriented mobile malware analysis as first aid," *Mobile Information Systems*, vol. 2016, Article ID 6707524, 11 pages, 2016.

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

[8] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine learning based Android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 3216–3225, 2018.

[9] C. Wang, Z. Li, X. Mo, H. Yang, and Y. Zhao, "An Android malware dynamic detection method based on service call co-occurrence matrices," *Annals of Telecommunications*, vol. 72, no. 9-10, pp. 607–615, 2017.

[10] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of android malware and Android analysis techniques," *ACM Computing Surveys*, vol. 49, no. 4, pp. 1–49, 2017.

[11] K. O. Elish, X. Shu, D. D. Yao, B. G. Ryder, and X. Jiang, "Profiling user-trigger dependence for Android malware detection," *Computers & Security*, vol. 49, pp. 255–273, 2015.

[12] Y. Liu, K. Guo, X. D. Huang, Z. Zhou, and Y. Zhang, "Detecting android malwares with high-efficient hybrid analyzing methods," *Mobile Information Systems*, vol. 2018, Article ID 1649703, 12 pages, 2018.

[13] X. Wang, D. F. Zhang, X. Su, and W. J. Li, "Mlifdect: Android malware detection based on parallel machine learning and information fusion," *Security and Communication Networks*, vol. 2017, article 6451260, 14 pages, 2017.

[14] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital Investigation*, vol. 13, pp. 22–37, 2015.

[15] B. B. Rad, M. Masrom, and S. Ibrahim, "Opcodes histogram for classifying metamorphic portable executables malware," in *Proceedings of International Conference on e-Learning and e-Technologies in Education (ICEEE)*, pp. 209–213, IEEE, Lodz, Poland, September 2012.

[16] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Effectiveness of opcode ngrams for detection of multi family Android malware," in *Proceedings of 10th International Conference on Availability, Reliability and Security (ARES)*, pp. 333–340, IEEE, Toulouse, France, August 2015.

[17] B. Kang, S. Y. Yerima, K. McLaughlin, and S. Sezer, "N-opcode analysis for Android malware classification and categorization," in *Proceedings of International Conference On Cyber Security And Protection Of Digital Services (Cyber Security)*, pp. 1–7, IEEE, London, UK, June 2016.

[18] P. O'Kane, S. Sezer, and K. McLaughlin, "N-gram density based malware detection," in *Proceedings of World Symposium on Computer Applications & Research (WSCAR)*, pp. 1–6, IEEE, Sousse, Tunisia, January 2014.

[19] B. Wolfe, K. Elish, and D. Yao, "High precision screening for Android malware with dimensionality reduction," in *Proceedings of 13th International Conference on Machine Learning and Applications (ICMLA)*, pp. 21–28, IEEE, Detroit, MI, USA, December 2014.

[20] M. Zhao, T. Zhang, F. B. Ge, and Z. J. Yuan, "RobotDroid: a lightweight malware detection framework on smartphones," *Journal of Networks*, vol. 7, no. 4, pp. 715–722, 2012.

[21] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences*, vol. 231, pp. 64–82, 2013.

[22] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: effective and explainable detection of android malware in your pocket," in *Proceedings of 21st Annual Network and Distributed System Security Symposium (NDSS'14)*, pp. 1–15, San Diego, CA, USA, February 2014.

[23] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Lorenzo Cavallaro, "DroidSieve: fast and

accurate classification of obfuscated android malware," in *Proceedings of Seventh ACM on Conference on Data and Application Security and Privacy*, pp. 309–320, ACM, Scottsdale, AZ, USA, March 2017.
[24] Apktool, https://github.com/iBotPeaches/Apktool/.
[25] Dalvik opcodes, http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html.
[26] H. Dong, N.-Q. He, G. Hu, Q. Li, and M. Zhang, "Malware detection method of android application based on simplification instructions," *Journal of China Universities of Posts and Telecommunications*, vol. 21, pp. 94–100, 2014.
[27] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J.D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.
[28] E. Raff, R. Zak, R. Cox et al., "An investigation of byte n-gram features for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 1, pp. 1–20, 2018.
[29] W. Wang, S. Gombault, and T. Guyet, "Towards fast detecting intrusions: using key attributes of network traffic," in *Proceedings of Third International Conference on Internet Monitoring and Protection (ICIMP'08)*, pp. 86–91, IEEE, Bucharest, Romania, June-July 2008.
[30] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.
[31] T. M. Chen, X. Zhang, S. C. Jin, and O. Kim, "Efficient classification using parallel and scalable compressed model and its application on intrusion detection," *Expert Systems with Applications*, vol. 41, no. 13, pp. 5972–5983, 2014.
[32] J. W. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*, Elsevier, New York, NY, USA, 2011.
[33] The Drebin Dataset, http://user.informatik.uni-goettingen.de/~darp/drebin/.
[34] Google Play, https://play.google.com/.
[35] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proceedings of 10th International Conference on Mobile Systems, Applications, and Services*, pp. 281–294, ACM, Ambleside, UK, June 2012.
[36] VirusBook, https://www.virusbook.cn/.