Chinese Journal of Electronics
Vol.32, No.5, Sept. 2023

EAODroid: Android Malware Detection Based on Enhanced API Order

HUANG Lu[1], XUE Jingfeng[1], WANG Yong[1], QU Dacheng[1], CHEN Junbao[1], ZHANG Nan[1], and ZHANG Li[2]

(1. *School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China*)

(2. *Department of Media Engineering, Communication University of Zhejiang, Hangzhou 310018, China*)

Abstract — The development of smart mobile devices brings convenience to people's lives, but also provides a breeding ground for Android malware. The sharp increasing malware poses a disastrous threat to personal privacy in the information age. Based on the fact that malware heavily resorts to system application programming interfaces (APIs) to perform its malicious actions, there has been a variety of API-based detection methods. Most of them do not consider the relationship between APIs. We contribute a new approach based on the enhanced API order for Android malware detection, named EAODroid, which learns the similarity of system APIs from a large number of API sequences and groups similar APIs into clusters. The extracted API clusters are further used to enhance the original API calls executed by an app to characterize behaviors and perform classification. We perform multi-dimensional experiments to evaluate EAODroid on three datasets with ground truth. We compare with many state-of-the-art works, showing that EAODroid achieves effective performance in Android malware detection.

Key words — Android malware, Malware detection, Deep learning, Application programming interface.

I. Introduction

Accompanied by the significant development of intelligence, human life has gradually been inseparable from mobile devices. Various applications, including social, travel, payment and game, make human life more convenient than ever. Unfortunately, as the mainstream mobile operating system, the Android platform is under more severe attacks. According to the latest report of 360 Security Center that about 2.065 million new mobile malware samples have been captured in the first quarter of 2021 alone, increasing by 426.5% from the first quarter of 2020. On average, about 23,000 new samples have been captured per day. The newly added samples are mainly tariff consumption, accounting for 91.5%, followed by privacy theft and rogue. Massive malware pose the major threat to mobile security and the considerable challenge to malware detection, resulting in economic losses for Android users.

Previous works have shown that it is an effective detection method to extract application features based on dynamic/static/hybrid methods and feed feature vectors into machine learning (ML) or deep learning (DL) model for classification. The static analysis does not need to run the application but extracts interesting features by checking the application's manifest file or executing code, which is relatively effective. But it is unable to analyze the application behavior during execution. The dynamic analysis can learn the runtime characteristic, but its low code coverage and high consumption bring many limitations to detection. The hybrid method combines dynamic and static analysis to extract various types of information from the application, leading to large-scale feature sets (even hundreds of thousands). Android platform provides an API framework for developers to interact with the system, applications, or hardware. It is an important detection method to use system API call information due to malware usually cannot bypass system APIs to perform malicious actions, which provides clues for detection. So, many studies rely on analyzing APIs or combining APIs with other features. However, most of these works only use shallow features such as API binary, API frequency, API *N*-gram, or malicious API patterns defined by experts, which treat API as independent of each oth-

Manuscript Received Dec. 26, 2021; Accepted Mar. 10, 2022. This work was supported by the National Natural Science Foundation of China (62172042), the National Key Research & Development Program of China (2020YFB1712104), and the Major Scientific and Technological Innovation Projects of Shandong Province (2020CXGC010116)).

 DOI:10.23919/cje.2021.00.451

er and ignore the potential association between them. Hence, the performance of classification will be affected after the specific API is changed or replaced. Based on previous observations [1] that different malware usually keep the same behavior but switch to different implementations of system API to avoid similarity detection. So, we propose to extract the relevance between APIs to enhance API-based features. Our insight is that the functional similarity of API will be reflected in the context of API sequence. Therefore, we treat each API as a word and convert them into embedding vectors using natural language processing (NLP) method and generate API clusters with similar functions or usage to enhance the original API call sequence.

Specifically, our method extracts the complete sequence of system API calls from Dalvik code and uses part of API sequences to train the API embedding model, i.e., API2Vec, which generates dense vectors of APIs. We expect related APIs should be gathered in the feature space. Then the API vectors are grouped into different clusters, which will be used to enhance the original sequence of API. Moreover, we utilize the order of API calls to abstract application behaviors, so we adopt the Convolutional Neural Network (CNN) on adjacency matrices generated from the API sequences. The proposed method effectively reduces the feature dimension and be sensitive to different implementations of similar behavior.

The contributions of this work are as follows.

1) We propose a new malware detection method by mining functional similarity between APIs to produce API clusters, and using adjacency matrices from enhanced sequences of API-cluster to abstract application behaviors.

2) We implement an API embedding model trained by sequences of API calls and obtain API clusters with functional similarity through clustering, which supports resilience to change of the feature size.

3) We construct a detection model named EAODroid and investigate its effectiveness through experiments on three datasets. Experimental results show that EAODroid outperforms the most advanced works based on static analysis.

The rest of the paper is organized as follows. First, Section II introduces related work. The new methodology will be described in Section III. Then, Section IV details the dataset, experimental procedures, and results. Lastly, the summary and future work will be concluded in Section V.

II. Related Work

Previous researchers have proposed many novel and efficient methods in the process of gaming with malware producers. In this section, we divide different methods into the following categories: based on dynamic and static analysis, based on deep learning, and based on the graph. It is worth mentioning that no matter which category, there is no lack of researchers who use APIs as feature.

1. Dynamic and static analysis based malware detection

The static analysis method refers to extracting features by analyzing the code, configuration or resources. It has high efficiency and fast speed, but it is easily affected by code confusion, packing, dynamic loading, etc. Scalas *et al.* [2] used the frequency of occurrence of system APIs to construct feature vectors to distinguish ransomware, malware and goodware. MaMaDroid [3] extracts the conversion probability between APIs from the abstracted API sequences through Markov chain. Their works abstract API to family/package/class to reduce feature-dimension and impact of API changes, but cannot represent the functional similarity between APIs well. Other features, such as permissions [4]–[6], opcode [7]–[10] and source code [11] are also widely used. Many works combine API with other features for analysis. Li *et al.* [12] mines the frequent pattern of Permission & API, assigns weights to items according to the number of items, and constructs a weighted naive Bayesian classifier based on the weight. RepassDroid [13] believes that sensitive API triggering without user participation is dangerous. All API trigger points are traced from the code and used as additional attributes of APIs to distinguish them. Meanwhile, PIKADroid [14] constructs the set of all (entrypoint, target-API) pairs and calculate the malicious scores of different pairs. When there are too many malicious pairs in one Android package (APK), there is reason to think it is suspicious. All of the works have selected APIs with custom rules, but the binary features are shallow and easy to be affected by code obfuscation.

Different from static analysis, the dynamic analysis uses runtime behaviors, including system calls, network and access behaviors, etc. SWORD [15] computes typical paths of system-call traces to construct the feature space. Wang *et al.* [16] builds a platform to collect HTTP traffic flow generated by the application and analyzes the traffic data as text using NLP to identify malicious traffic. Maybe we can consider the analysis of covert VoIP (voice over Internet protocol) traffic [17] in future. Many researchers will customize dynamic features, which requires a sufficient understanding of malware. DroidCat [18] determines 70 indicators from method calls and inter-component communication intents, which can distinguish malware from goodware. DroidSpan [19] defines 52 dynamic features based on

sensitive access distribution from the dynamic call trace and further studies the sustainability of features and models. The combination of dynamic and static features will enable researchers to analyze malicious behavior further. CoDroid [20] is a hybrid method that uses the mixed sequence of dynamic system call and static opcode. DroidPortrait [21] builds static and dynamic behavior databases from five dimension: configuration, code, certification, network and system call, and realizes the correlation between different behaviors through ML. Even though our method currently only uses static features, our theory can also be applied to the system call sequence extracted by dynamic analysis.

2. Deep learning based malware detection

In order to combat the evolving malware and adversarial samples for ML model [22], researchers use the deep learning algorithm to improve detection performance. With help of the deep learning algorithm, features can be learned automatically without manual selection, which greatly improves efficiency. Multimodal [23] uses five feature vectors and inputs each vector to different initial deep neural networks (DNNs) separately. The five initial networks are not connected but are merged into the final DNN. It turns out that training different networks with different features and merging is better than training a single model with merged features. DeepRefiner [24] adopts a two-layer network structure. The second layer network can further detect the APK samples that cannot be determined in the first layer network, which can reduce the false positive rate effectively, but does not solve the problem of conflict between two layers. TC-Droid [25] applies TextCNN to mine the difference between analysis reports of samples, which replace manual feature engineering effectively. Andro-Simnet [26] builds a weighted network graph of similarity samples and cluster samples into families with an unsupervised model. To fight against the evolution of malware, DroidEvolver [27] updates the feature sets and aging models if there is any drift. Specifically, it trains five online learning models to build a model pool. When an unknown sample is identified as drift, the pseudo label will be generated according to the voting mechanism and used to update the aging models. In addition, DroidFusion [28] studies the influence of the combination of different initial machine learning models on the performance. Works based on deep learning use a large number of features, or directly raw data, which will lead to high computational complexity. Although our method applies the deep learning algorithm CNN, our feature dimensions are adjustable.

3. Graph based malware detection

Methods based on raw features such as permission, API, system call, etc., are easy to be escaped, and the high-dimensional sparse feature vectors will also bring unnecessary resource consumption. On the contrary, the structural features generated based on graph (e.g., function call graph, FCG) are more robust. Considering that the implementation of malicious behaviors cannot bypass API calls, there have been many works using API-related graphs. FalDroid [29] performs graph matching and clustering according to the structural similarity between sensitive API nodes in graph. The frequent graphs in malicious families will be identified as features and assigned weighted scores as evidence for family classification. GefDroid [30] also uses structural role of API nodes, but the similarity calculation of API is based on the API node embedding through struc2vec. In addition, they construct a malware link network and applied and community detection algorithms to cluster malware into families. AndrEnsemble [31] computes fuzzy hash value of function nodes in call graphs and aggregates all hash graphs of one malicious family to extract the frequent API set. GDroid [32] maps APIs and apps into a heterogeneous graph and uses graph convolutional network (GCN) model to obtain the node embedding to represent samples. These works are based on single API nodes, which is effective when using the same API implementations between malware. When switching to the different API implementations, the diametrically opposite results will be computed. Using functions including API calls as graph node is more flexible. Cai *et al.* [33] and Feng *et al.* [34] both convert the function nodes in FCG into vector form. The former is based on word embedding, and the latter one extracts internal attributes of function as features. The final vector representations of applications are learned through GCN or graph neural network (GNN) model. GSDroid [35] combines dynamic analysis and graph signal to extract low dimensional feature vectors from system call graph. Graph-based malware detection is one of our interests, but larger applications will generate larger-scale graphs to be difficult to analyze. The key is how to construct a meaningful graph and extract meaningful information.

III. Methodology

In this section, we describe our approach in detail. Fig.1 illustrates the architecture of EAODroid. There are the following four main stages: 1) Extracting the complete system API call sequences from applications. 2) Learning API embedding through the API2Vec model, and then using the K-means algorithm to group APIs in clusters. 3) Enhancing the original API sequences according to the API clusters and constructing the adjacency matrix. 4) Applying the CNN model on the adjacency matrix to extract order information and classify unknown applications.

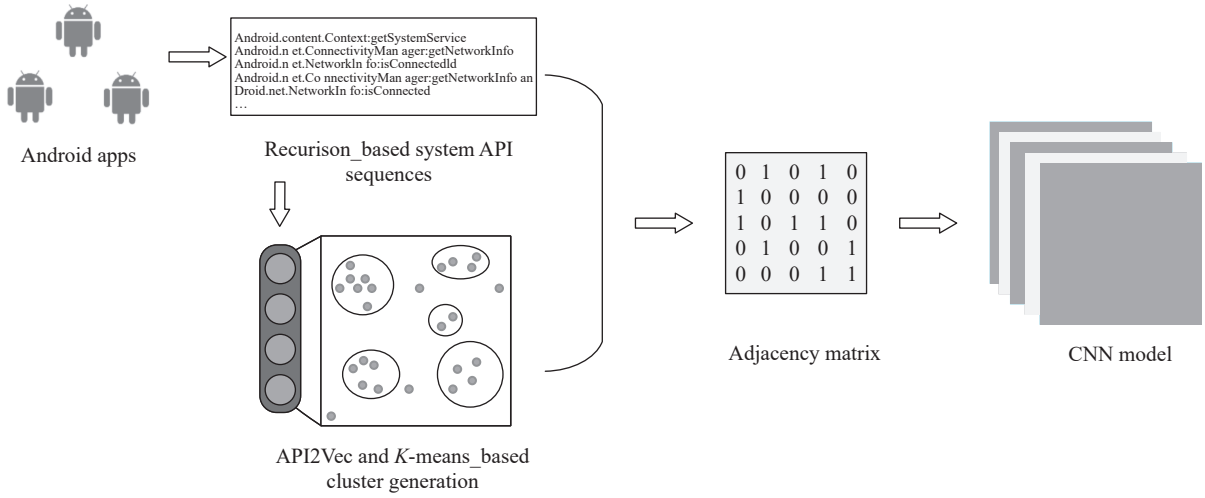

Fig. 1. Illustrates the overall workflow of EAODroid.

1. Recursion-based system API call sequence extraction

The first step of EAODroid is to extract system API call sequences from APK files. An Android app is normally written in Java and compiled to Dalvik code stored in classes.dex file and the compiled code and other resources are packaged into an APK file. We first use a decompile tool (ApkTool[*1]) to get the Dalvik code from APK files. Each class and its methods are defined in the corresponding smali file. Then, we identify the invocation statements ("invoke-∗") to extract callees from the Dalvik code in turn. Generally, the callee is not necessarily a system API, it may be a user-defined function. Compared with system API, user-defined functions are easier to be modified and avoided by attackers using code obfuscation technology, which not only increases computational complexity but also decreases model performance. So, we only focus on system API. We implement a recursion-based system API extraction to eliminate user-defined functions while retaining the complete API call sequence. In short, when the current callee is a user-defined function, all callee in this function will be queried down until the final sequence does not contain another user-defined one. Fig.2 shows an example, the function "download()" invokes another user-defined function "createFileF()", so all system callees in function "createFileF()" are used to replace itself when generate the sequence of "download()".

2. API cluster generation

To extract and utilize the functional similarity of APIs, we implement an API embedding model, API2Vec, trained by the sequences of system API, which converts APIs into dense vectors. Then the clustering algorithm *K*-means is used to group similar APIs into clusters. Fig.3 illustrates this process.

1) API embedding. The API2Vec model is inspired by the task of word embedding in NLP. It treats single system APIs as words, and API sequences in one smali file will be regarded as one row in the training corpus for API2Vec. The inconsistent length of API sequences extracted from smali files will not affect the embedding process, because the API2Vec extracts API pairs in the sliding window as model input in API sequences. For example, given an API sequence "java.io.IOException. printStackTrace java.net.HttpURLConnection.getInputStream java.io. InputStream.Read java.io.FileOutputStream.Close android.Util.Log.e", the intermediate API "java.io. InputStream.Read" is treated as the output and its contexts as the input. The training object is similar to the CBOW (continuous bag of words) model, which learns dense vectors to represent each API. If two different APIs have similar contexts, they will be closer in the embedding space, and EAODroid can use the similarity to capture more advanced semantic information.

2) API clustering. We apply the *K*-means to group API embedding in clusters to obtain API clusters with functional similarity. Furthermore, Android malware usually invokes sensitive API calls to perform malicious behaviors. To narrow the scope of analysis, we use a set of sensitive APIs summarized by Susi [36] as the key API for malware detection tasks, including 18044 sources and 8278 sinks. Note that the number of clusters K is optional, and the appropriate number of clusters can be considered by combining consumption and detection performance.

3. Feature matrix construction

Due to the different writing habits of malware de-

[*1]https://ibotpeaches.github.io/Apktool/

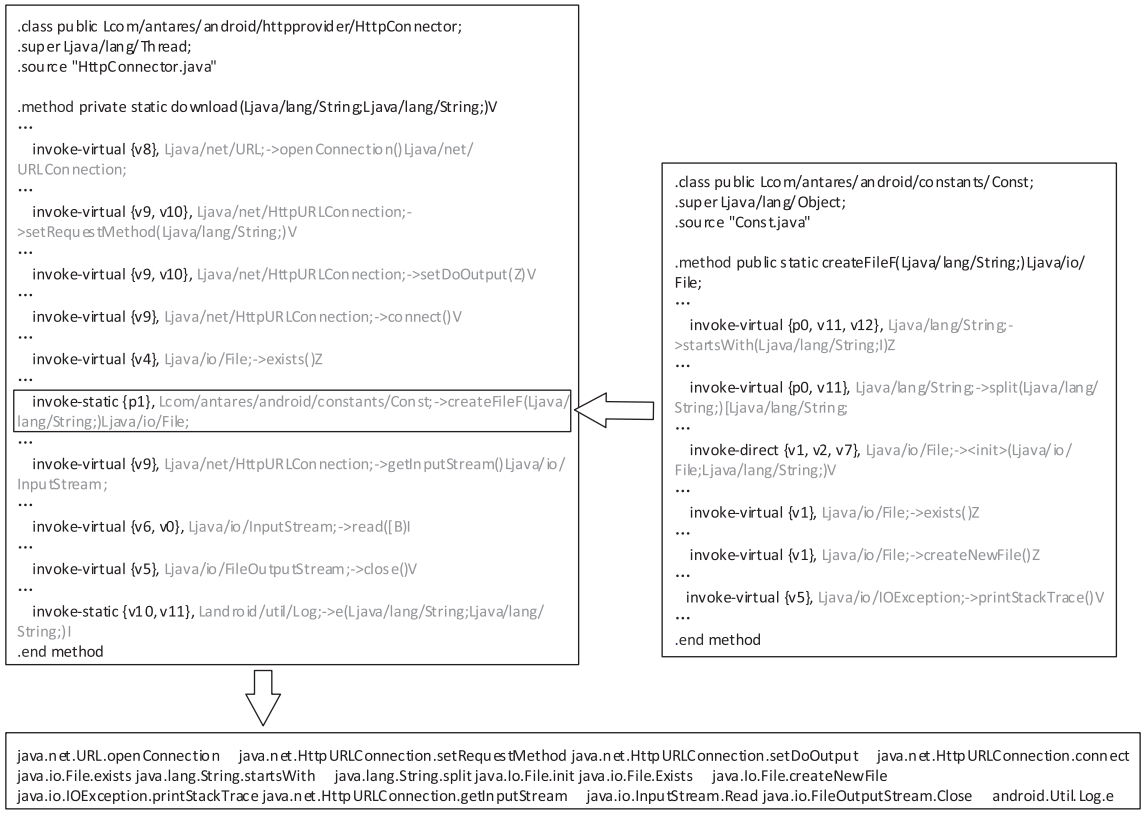

Fig. 2. Instance of the recursion-based system API sequences.

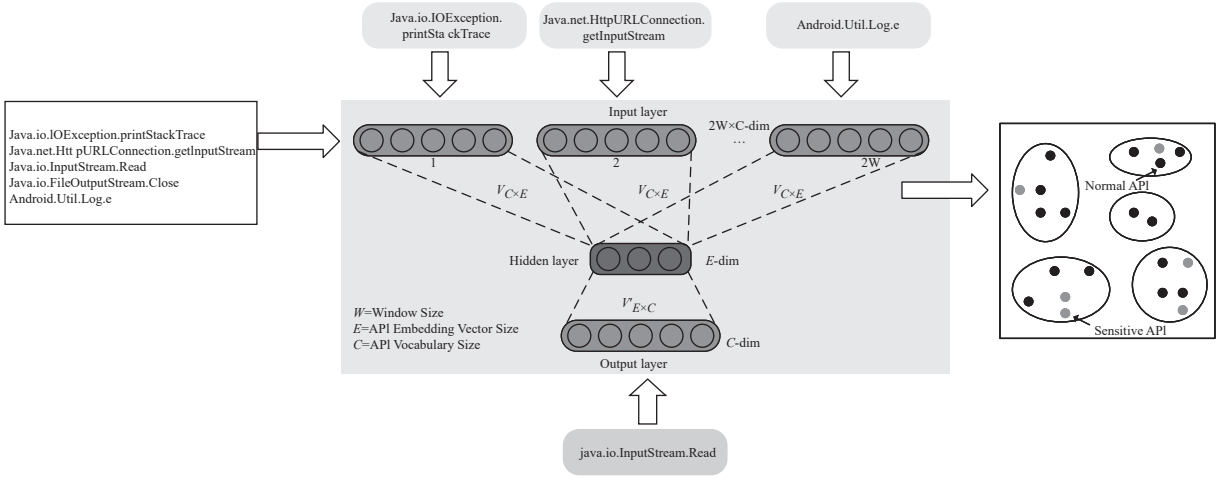

Fig. 3. API embedding and clustering.

velopers, or the update iteration of homologous malware, completely consistent API sequences often do not appear. Luckily, when performing similar malicious behaviors, the order of key APIs usually does not change much. Most of the time, the change is to add or delete some API calls. So, in order to explore the correlation between these incomplete sequences, we construct API adjacency matrix from API sequences with API clusters to extract order information. Another benefit is that compared with methods that use graphs directly, using the adjacency matrix can effectively reduce the high complexity.

Define a $K \times K$ matrix, where K is the number of API clusters. Given that $\boldsymbol{S} = \{\boldsymbol{a_1}, \boldsymbol{a_2}, ..., \boldsymbol{a_n}\}$ is an API sequence, where a_i just belong to only one cluster C_{a_i}, The feature matrix is obtained by

$$\text{Matrix}[m][l] = \begin{cases} 1, & \exists\,(\boldsymbol{a_i}, \boldsymbol{a_j}, i < j \leq n) \text{ in } \boldsymbol{S}, \\ & \boldsymbol{a_i} \in C_m, \boldsymbol{a_j} \in C_l \\ 0, & \text{elsewhere} \end{cases} \tag{1}$$

4. CNN based classification

The CNN model has translation and scale invari-

ance, which can capture the feature block in the adjacency [37]. So, We use the CNN model to perform classification in the proposed method. The architecture of the CNN model used in this paper is shown in Table 1. It includes an input layer, four convolutional layers, a maximum pooling layer, a fully connected layer, and a softmax layer. The input of the CNN model is the adjacency matrix generated in the previous stage, and the output is a two-dimensional vector representing the probability of being benign and malicious. To prevent the gradient from exploding, we perform batch normalization on the feature map of each convolutional layer.

Table 1. CNN Structure

Model	Structure
CNN	Input Layer(64, 64, 1)
	Conv2D(3, 3 × 3)
	BatchNormalization()
	Conv2D(32, 3 × 3)
	BatchNormalization()
	Conv2D(64, 3 × 3)
	BatchNormalization()
	Conv2D(128, 3 × 3)
	BatchNormalization()
	GlobalMaxPool2D()
	Linear(2)
	Softmax()

IV. Evaluation

In this section, we first introduce the dataset and experimental settings, and then we conduct extensive experiments to evaluate the effectiveness of the proposed EAODroid.

1. Dataset

We obtain malware samples from the two malware datasets: 1) Drebin [38]; 2) AMD [39]. Both two datasets have been widely used in previous researches. The benign samples are collected from the two sources: 1) Xiaomi application market[*2], a popular third-party Android app market in China, 2) PlayDrone [40]. We randomly select a part of samples to compose our experimental datasets. Samples that failed to be analyzed by Apktool have been deleted. Besides, since the baseline method compared in our paper uses FlowDroid [41] to extract call graphs, we also delete samples that cannot be analyzed by FlowDroid. Finally, we compose three different experimental datasets as shown in Table 2.

2. Experimental environment

The experiments of EAODroid are conducted on a computer with Intel i7-6700 CPU(3.4 GHz) and 32G of RAM. And the proposed EAODroid is implemented using Python with several packages: Scikit-learn, TensorFlow, and Matplotlib.

Table 2. Summary statistics of the experimental dataset

Dataset	Benign	Drebin	AMD	Total
D1	5643	5532	0	11175
D2	5643	0	5619	11262
D3	5643	2800	2800	11243

3. Parameter settings

The number of API call sentences are used to train the API2Vec model accounted for 25% of each dataset. The embedding and window size of API2Vec are set in such a way to take a balance between efficiency and performance. We selected different parameter sets according to the experience of previous researchers. The API embedded dimension is selected in {64,128,256}, and the window size is determined in {5,10}. Through experiments, it is found that the larger embedding dimension and window size do not significantly improve the final detection performance. In order to improve efficiency, the output embedding representation for each API is set to 64-dimension and the window size is 5. The influence of different cluster numbers K on detection performance will be detailed in the rest of the paper. According to the experiment, the final K value of the K-means algorithm is set to 64 and the feature matrix used by EAODroid is set to (64,64) in experiments of detection performance. The batch size is 256 and the learning rate is 0.001. The parameters of the CNN model are shown in the next subsection.

4. Parameter settings

We conduct tenfold cross-validation to measure the performance of EAODroid and each dataset is split into 90% for training and 10% for testing. We evaluate the performance of EAODroid using four indicators: F1-measure, Accuracy, Recall, and Precision. Like most researches, true positive (TP) indicates that malicious applications are correctly classified as malicious, false positive (FP) indicates that benign applications are incorrectly classified as malicious, while correctly classified benign applications are labeled as true negative (TN) and incorrectly classified benign applications are labeled as false negative (FN). Depending on these basic metrics, the four indicators can be generated as follows:

The ratio of the number of samples correctly classified to the number of all samples is defined as Acc.

$$\text{Acc} = \frac{TN + TP}{TN + TP + FN + FP}$$

The ratio of the number of correctly classified mali-

[*2] http://app.mi.com/

cious samples to the number of all samples identified as malicious is defined as Precision:

$$\text{Precision} = \frac{TP}{TP + FP}$$

The ratio of the number of correctly classified malicious samples to the number of all malicious samples is defined as Recall:

$$\text{Recall} = \frac{TP}{TP + FN}$$

The definition of F1-measure is as follows:

$$\text{F1-measure} = 2 \times \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$

5. Performance evaluation

1) Detection performance. In this experiment, we evaluated the general performance of EAODroid on the malware detection task under dataset D1. Simultaneously we compare EAODroid with traditional machine learning such as K-nearest neighbor (KNN), Random forest (RF), support vector machine (SVM), naive Bayes (NB), logistic regression (LR). Table 3 shows the comprehensive results of different algorithms. The proposed EAODroid exceeds common machine learning methods in most of the performance metrics and achieves the highest F1-measure reaching 99.5%.

Table 3. Detection performance of EAODroid and common ML algorithm

Method	Acc	Precision	Recall	F1-measure
EAODroid	0.995	0.994	0.996	0.995
RF	0.992	0.986	0.998	0.992
SVM	0.991	0.989	0.993	0.991
KNN	0.986	0.977	0.996	0.986
NB	0.913	0.859	0.985	0.918
LR	0.993	0.991	0.995	0.993

2) Different feature dimensions. In this experiment, we discuss the performance results with different feature dimensions under dataset D1. The feature dimension is according to the K number of API clusters, which should be built based on the same API2Vec model to avoid other variables. As shown in Fig.4, we adjust K to (8, 16, 32, 64,128) and experimental results show that our model can still achieve satisfactory results despite the use of the lower-dimensional feature. When the dimension rises to 128, the performance drops in a small range, which shows that too many features introduce useless information and even interfere with model learning.

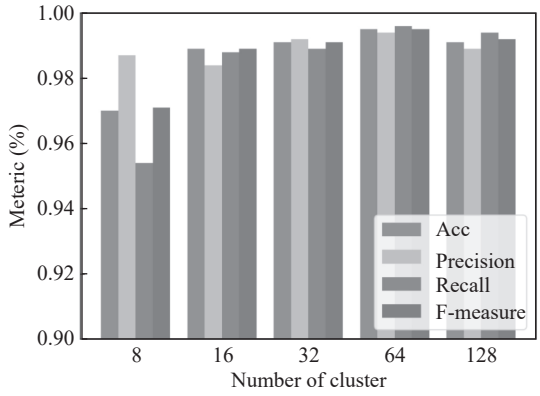

Fig. 4. Performance comparison of different feature dimension.

3) Comparison with other detection methods using Drebin dataset. To compare with other state-of-art works, we investigate previously proposed detection approaches. All works use different types of features to build their models. Unfortunately, most of them are not publicly available and difficult to reproduce with the same parameters. So we select works that utilize the malware samples from Drebin and compare the performance under the Drebin dataset. As shown in Table 4, compared with detection approaches based on other features, EAODroid is superior to others in terms of accuracy and F1-measure.

In addition, we consider MaMaDroid as the baseline and compare detection results under the same dataset D1. MaMaDroid abstracts API calls into Family or Package and computes the conversion frequency between abstracted API calls through Markov chains. It trains an RF classifier and achieves high performance. To reproduce MaMaDroid, we obtain the API family list (number is 11) and the latest API package list (number is 357, MaMaDroid is 340) according to the

Table 4. Performance comparison with previous works under Drebin (paper survey)

System	Dataset (Malware, Benign)	Features	Classification algorithm	Precision/F1-measure
EAODroid	(5534,5643)	API	CNN	**0.994/0.995**
ProDroid [42]	(5000,500)	API	PHMM	0.930/0.939
CoDroid [20]	(2978,2707)	Opc + Sys	CNN-BiLSTM	0.954/0.986
TinyDroid [8]	(2400,2400)	Opc	RF	0.921/NA
FAMD [43]	(5560,5666)	Per + Opc	Catboost	0.980/0.973
Frenklach *et al.* [44]	(5500,5500)	Source Code	RF	NA/0.976

Note: API: API call; Opc: Opcode; Per: Permission.

description in their paper. As shown in Table 5, we can see that EAODroid has improved almost every indicator. The results clearly illustrate that our method has better performance on malware detection. MaMaDroid has very different feature dimensions in two different modes, ranging from one hundred to hundreds of thousands. The proposed method is more flexible to support any dimension of the feature under the premise of ensuring accuracy.

Table 5. Performance comparison with MaMaDroid

Method	Feature dimension	Acc	Precision	Recall	F1-measure
EAODroid	(11,11)	0.986	0.979	0.994	0.986
EAODroid	(64,64)	**0.995**	**0.994**	**0.996**	**0.995**
MaMaDroid(family)	11 × 11	0.946	0.953	0.933	0.943
MaMaDroid(package)	357 × 357	0.977	0.968	0.964	0.966

4) Generalization ability. In this experiment, we use three different datasets in Table 1 to verify the generalization ability of EAODroid. We use a ten-fold cross-validation method to train and test these three datasets. As shown in Table 6, our model achieves high classification results in three different datasets, but the classification with Drebin is better than AMD. Samples of AMD dataset have a larger span (2010–2016) and greater diversity. Malware based on different levels of API framework will have large differences in the use of sensitive APIs, which will affect model performance. How to further eliminate the false positives caused by API updates is also one of our next tasks.

Table 6. Performance under different datasets

Dataset	Acc	Precision	Recall	F1-measure
D1	0.995	0.994	0.996	0.995
D2	0.972	0.975	0.967	0.970
D3	0.966	0.963	0.973	0.968

5) Runtime performance. In this section, we evaluate the time consumption of EAODroid to demonstrate its effectiveness. The first phase of EAODroid involves extracting API sequences from APKs and the time consumption of this phase depends on the size of analyzing APKs. For example, 113 KB malware requires 0.68 s, while 21.3 MB malware requires 53.52 s to complete. Next, we discuss the time spent in the API cluster generation, which includes two parts: API embedding and API clustering. In the embedding phase, API2Vec takes about 108 s to complete the embedding process by taking the sequences extracted from about 2800 samples as the corpus. The API clustering phase completes in about 2 s when K is set to 64. Due to the low feature dimension, the phase of feature matrix generation is fast and takes less than 0.2 s per sample. Another crucial phase is detection model training. Our training set D1 contains more than 10000 samples and the CNN model takes approximately 177.2 s with 50 epochs to exceed the performance shown Table 3. In summary, our experiments show that the time consumption of EAODroid is acceptable in real word.

V. Conclusion and Discussion

In this paper, we propose a new Android malware detection method based on enhanced API order, named EAODroid. It collects callees from invoke statements to form system API sequences and constructs adjacency matrix, replacing use API call graph. EAODroid needs to disassemble Android applications to get enough Dalvik code. However, some applications apply techniques such as shelling and dynamic loading to avoid analysis or protect copyright, which makes it unable to extract features. But, if we can get the true execution entry of these applications and find hidden executable code, it will be still effective.

In the future, we will further perform dynamic analysis to extend the existing method. At the same time, with continuous malware evolution, the aging of detection models has become an urgent issue. Recent effort been contributed to updating models and evolving feature set. However, the issue has not been fully addressed. We can start from the malware itself and study how malware of the same family evolves and what unchanged characteristics will be maintained. We believe it will be of great help for us to define new immutable characteristics.

References

[1] X. H. Zhang, Y. Zhang, M. Zhong, *et al.*, "Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, pp.757–770, 2020.

[2] M. Scalas, D. Maiorca, F. Mercaldo, *et al.*, "On the effectiveness of system API-related information for Android ransomware detection," *Computers & Security*, vol.86, pp.168–182, 2019.

[3] L. Onwuzurike, E. Mariconti, P. Andriotis, *et al.*, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models (Extended Version)," *ACM Transactions on Privacy and Security*, vol.22, no.2, article no.14, 2019.

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

[5] X. Jiang, B. L. Mao, J. Guan, *et al.*, "Android malware detection using fine-grained features," *Scientific Programming*, vol.2020, article no.5190138, 2020.

[6] J. Li, L. C. Sun, Q. B. Yan, *et al.*, "Significant permission identification for machine-learning-based Android malware

detection," *IEEE Transactions on Industrial Informatics*, vol.14, no.7, pp.3216–3225, 2018.

[7] A. Pektaş and T. Acarman, "Learning to detect Android malware via opcode sequences," *Neurocomputing*, vol.396, pp.599–608, 2020.

[8] T. M. Chen, Q. Y. Mao, Y. M. Yang, *et al.*, "TinyDroid: A lightweight and efficient model for Android malware detection and classification," *Mobile Information Systems*, vol.2018, article no.4157156, 2018.

[9] N. McLaughlin, J. M. del Rincon, B. Kang, *et al.*, "Deep Android malware detection," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, Scottsdale, AZ, USA, pp.301–308, 2017.

[10] W. N. Niu, R. Cao, X. S. Zhang, *et al.*, "OpCode-level function call graph based Android malware classification using deep learning," *Sensors*, vol.20, no.13, article no.3645, 2020.

[11] R. Mateless, D. Rejabek, O. Margalit, *et al.*, "Decompiled APK based malicious code classification," *Future Generation Computer Systems*, vol.110, pp.135–147, 2020.

[12] J. W. Li, B. Z. Wu, and W. P. Wen, "Android malware detection method based on frequent pattern and weighted naive Bayes," in *Proceedings of the 15th International Annual Conference*, Beijing, China, pp.36–51, 2018.

[13] N. N. Xie, F. P. Zeng, X. X. Qin, *et al.*, "RepassDroid: Automatic detection of Android malware based on essential permissions and semantic features of sensitive APIs," in *Proceedings of 2018 International Symposium on Theoretical Aspects of Software Engineering*, Guangzhou, China, pp.52–59, 2018.

[14] J. Allen, M. Landen, S. Chaba, *et al.*, "Improving accuracy of Android malware detection with lightweight contextual awareness," in *Proceedings of the 34th Annual Computer Security Applications Conference*, San Juan, PR, USA, pp.210–221, 2018.

[15] S. Bhandari, R. Panihar, S. Naval, *et al.*, "SWORD: semantic aWare andrOid malwaRe detector," *Journal of information Security and Applications*, vol.42, pp.46–56, 2018.

[16] S. S. Wang, Q. B. Yan, Z. X. Chen, *et al.*, "Detecting Android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics and Security*, vol.13, no.5, pp.1096–1109, 2018.

[17] C. Liang, X. M. Wang, X. S. Zhang, *et al.*, "A payload-dependent packet rearranging covert channel for mobile VoIP traffic," *Information Sciences*, vol.465, pp.162–173, 2018.

[18] H. P. Cai, N. Meng, B. Ryder, *et al.*, "DroidCat: effective Android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol.14, no.6, pp.1455–1470, 2019.

[19] H. P. Cai, "Assessing and improving malware detection sustainability through app evolution studies," *ACM Transactions on Software Engineering and Methodology*, vol.29, no.2, article no.8, 2020.

[20] N. Zhang, J. F. Xue, Y. X. Ma, *et al.*, "Hybrid sequence-based Android malware detection using natural language processing," *International Journal of Intelligent Systems*, vol.36, no.10, pp.5770–5784, 2021.

[21] X. Su, L. J. Xiao, W. J. Li, *et al.*, "DroidPortrait: Android malware portrait construction based on multidimensional behavior analysis," *Applied Sciences*, vol.10, no.11, article no.3978, 2020.

[22] X. M. Wang, J. Li, X. H. Kuang, *et al.*, "The security of machine learning in an adversarial setting: a survey," *Journal of Parallel and Distributed Computing*, vol.130, pp.12–23, 2019.

[23] T. Kim, B. Kang, M. Rho, *et al.*, "A multimodal deep learning method for Android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol.14, no.3, pp.773–788, 2019.

[24] K. Xu, Y. J. Li, R. H. Deng, *et al.*, "DeepRefiner: Multi-layer Android malware detection system applying deep neural networks," in *Proceedings of 2018 IEEE European Symposium on Security and Privacy*, London, UK, pp.473–487, 2018.

[25] N. Zhang, Y. A. Tan, C. Yang, *et al.*, "Deep learning feature exploration for Android malware detection," *Applied Soft Computing*, vol.102, article no.107069, 2021.

[26] H. M. Kim, H. M. Song, J. W. Seo, *et al.*, "Andro-Simnet: Android malware family classification using social network analysis," in *Proceedings of the 2018 16th Annual Conference on Privacy, Security and Trust*, Belfast, Ireland, pp.1–8, 2018.

[27] K. Xu, Y. J. Li, R. Deng, *et al.*, "DroidEvolver: Self-evolving Android malware detection system," in *Proceedings of 2019 IEEE European Symposium on Security and Privacy*, Stockholm, Sweden, pp.47–62, 2019.

[28] S. Y. Yerima and S. Sezer, "DroidFusion: A novel multilevel classifier fusion approach for Android malware detection," *IEEE Transactions on Cybernetics*, vol.49, no.2, pp.453–466, 2019.

[29] M. Fan, J. Liu, X. P. Luo, *et al.*, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Transactions on Information Forensics and Security*, vol.13, pp.1890–1905, 2018.

[30] M. Fan, X. P. Luo, J. Liu, *et al.*, "Graph embedding based familial analysis of Android malware using unsupervised learning," in *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering*, Montreal, QC, Canada, pp.771–782, 2019.

[31] O. Mirzaei, G. Suarez-Tangil, J. M. de Fuentes, *et al.*, "AndrEnsemble: Leveraging API ensembles to characterize Android malware families," in *Proceedings of 2019 ACM Asia Conference on Computer and Communications Security*, Auckland, New Zealand, pp.307–314, 2019.

[32] H. Gao, S. Y. Cheng, and W. M. Zhang, "GDroid: Android malware detection and classification with graph convolutional network," *Computers & Security*, vol.106, article no.102264, 2021.

[33] M. H. Cai, Y. Jiang, C. Y. Gao, *et al.*, "Learning features from enhanced function call graphs for Android malware detection," *Neurocomputing*, vol.423, pp.301–307, 2021.

[34] P. B. Feng, J. F. Ma, T. Li, *et al.*, "Android malware detection via graph representation learning," *Mobile Information Systems*, vol.2021, article no.5538841, 2021.

[35] R. Surendran, T. Thomas, and S. Emmanuel, "GSDroid: Graph signal based compact feature representation for Android malware detection," *Expert Systems with Applications*, vol.159, article no.113581, 2020.

[36] S. Rasthofer, S. Arzt, and E. Bodden, "A machine-learning approach for classifying and categorizing Android sources and sinks," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, pp.1–15, 2014.

[37] Z. P. Yu, R. Cao, Q. Y. Tang, *et al.*, "Order matters: Semantic-aware neural networks for binary code similarity detection," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, New York, NY, USA, pp.1145–1152, 2020.

[38] D. Arp, M. Spreitzenbarth, M. Hübner, *et al.*, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, pp.23–26, 2014.

[39] F. g. Wei, Y. p. Li, S. Roy, *et al.*, "Deep ground truth analysis of current Android malware," in *Proceedings of the 14th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Bonn, Germany, pp.252–276, 2017.

[40] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of Google play," in *Proceedings of 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, Austin, TX, USA, pp.221–233, 2014.

[41] S. Arzt, S. Rasthofer, C. Fritz, *et al.*, "FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM SIGPLAN Notices*, vol.49, no.6, pp.259–269, 2014.

[42] S. K. Sasidharan and C. Thomas, "Prodroid—an Android malware detection framework based on profile hidden Markov model," *Pervasive and Mobile Computing*, vol.72, article no.101336, 2021.

[43] H. P. Bai, N. N. Xie, X. Q. Di, *et al.*, "FAMD: A fast multifeature Android malware detection framework, design, and implementation," *IEEE Access*, vol.8, pp.194729–194740, 2020.

[44] T. Frenklach, D. Cohen, A. Shabtai, *et al.*, "Android malware detection via an app similarity graph," *Computers & Security*, vol.109, article no.102386, 2021.

HUANG Lu was born in 1997. She received the B.E. degree in software engineering from the Central South University. She is now a Ph.D. candidate of Beijing Institute of Technology. Her research interests include Android malware detection and software security. (Email: hhuangluu@163.com)

XUE Jingfeng was born in 1975. He is a Professor and Ph.D. Supervisor in Beijing Institute of Technology. His main research interests focus on network security, data security, and software security. (Email: xuejf@bit.edu.cn)

WANG Yong was born in 1975. She received the Ph.D. degree in computer science from Beijing Institute of Technology. She is an Associate Professor of Beijing Institute of Technology. Her research interests include cyber security and machine learning, and software security. (Email: wangyong@bit.edu.cn)

QU Dacheng was born in 1974. He is a Professor in Beijing Institute of Technology. His main research interests focus on social network, recommender systems, and bioinformatics. (Email: qudc@bit.edu.cn)

CHEN Junbao was born in 1999. He received the B.E. degree in software engineering from Beijing Institute of Technology. He is currently pursuing the master's degree with School of Computer Science and Technology, Beijing Institute of Technology. His research interests include federated learning and AI security. (Email: chen.junbao@outlook.com)

ZHANG Nan was born in 1991. He received the B.E. degree in computer application technology from Anyang Normal University. He is a Ph.D. candidate of Beijing Institute of Technology. His research interests include machine learning, malware detection, and information security. (Email: nanzhang611@bit.edu.cn)

ZHANG Li (corresponding author) received the Ph.D. degree in computer application technology from Beijing Institute of Technology, Beijing, China. She is currently an Associate Professor of the Communication University of Zhejiang. Her current research interests include digital forensics, machine learning, and information security. (Email: nythhsg@sina.com)