Research Article

An Android Malicious Application Detection Method with Decision Mechanism in the Operating Environment of Blockchain

Xingyu Li , Zongqu Zhao , Yongli Tang , Jing Zhang , Chengyi Wu , and Ying Li

Henan Polytechnic University, School of Computer Science and Technology, Jiaozuo, Henan 454000, China

Correspondence should be addressed to Yongli Tang; yltang@hpu.edu.cn

Received 6 November 2021; Revised 27 November 2021; Accepted 11 January 2022; Published 29 January 2022

Academic Editor: Yuling Chen

Copyright © 2022 Xingyu Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Recently, security policies and behaviour detection methods have been proposed to improve the security of blockchain by many researchers. However, these methods cannot discover the source of typical behaviours, such as the malicious applications in the blockchain environment. Android application is an important part of the blockchain operating environment, and machine learning-based Android malware application detection method is significant for blockchain user security. The way of constructing features in these methods determines the performance. The single-feature mechanism, training classifiers with one type of features, cannot detect the malicious applications effectively which exhibit the typical behaviours in various forms. The multifeatures fusion mechanism, constructing mixed features from multiple types of data sources, can cover more kinds of information. However, different types of data sources will interfere with each other in the mixed features constructed by this mechanism. That limits the performance of the model. In order to improve the detection performance of Android malicious applications in complex scenarios, we propose an Android malicious application detection method which includes parallel feature processing and decision mechanism. Our method uses RGB image visualization technology to construct three types of RGB image which are utilized to train different classifiers, respectively, and a decision mechanism is designed to fuse the outputs of subclassifiers through weight analysis. This approach simultaneously extracts different types of features, which preserve application information comprehensively. Different classifiers are trained by these features to guarantee independence of each feature and classifier. On this basis, a comprehensive analysis of many methods is performed on the Android malware dataset, and the results show that our method has better efficiency and adaptability than others.

1. Introduction

Blockchain is a new decentralized infrastructure and distributed computing paradigm emerging with the increasing popularity of digital cryptocurrencies such as Bitcoin. In recent years, many researchers have put forward a large number of researches on the security of blockchain from two aspects: security policy and behaviour detection. The security policies include a series of technologies such as privacy protection [1, 2] and data encryption [3]. Behaviour detections are mainly used to identify typical behaviours that existed in blockchain, such as mining [4, 5]. However, the malicious applications which result in these behaviours cannot be discovered by security policy and behaviour detection method. Android is a free and open-source operating system based on Linux, widely used in blockchain.

Due to Android's openness, it has become a main target of malicious applications. The types of new Android malicious applications are cost consumption, privacy theft, remote control, roguery, malicious deduction of fees, fraud, etc. Detecting malicious Android applications is greatly significant for improving the security of Android applications and protecting the blockchain users' secret keys and information security.

In order to effectively detect Android malicious applications, various methods have been proposed. These methods include the single-feature mechanisms [6–17] and the multifeatures fusion mechanisms [18–26]. The methods based on a single-feature mechanism usually train a classifier with one type of features which include APIs [8], permissions [6, 7], call graphs, images [10], or codes [9, 13–17]. As the structural complexity of these features increases, the

accuracy of methods is constantly improved. However, with the development of the malwares, the malicious behaviours would be presented in multiple places in the application simultaneously, such as signature files, configuration files, code files, DLL files, and so on. A classifier trained by a single-feature cannot detect various malwares effectively. Therefore, it is difficult to improve the performance of the method based on a single-feature mechanism when detecting various Android malicious applications. The multifeatures fusion mechanisms can solve the problems encountered by a single-feature mechanism. These mechanisms construct a group of hybrid features which include multiple types of data. Then they use hybrid features to train the classifiers. To some extent, these mechanisms could detect those malicious applications which present typical behaviours in multiple types of data sources. However, as multiple types of data sources are fused into a group of hybrid features, different types of data sources can interfere with each other during the classifier training process, which restricts the further improvement of the performance.

We propose an Android malicious applications detection method including a decision mechanism. The method introduces a concept of the parallel detection and fusing detection result. The parallel detection constructs a variety of images for different types of data sources to train multiple classifiers. The result decision layer is constructed to fuse the outputs of multiple classifiers. The advantages of our method are as follows:

- (i) Different types of images are constructed using parallel RGB image visualization techniques, and these images are employed to train multiple classifiers. These mutual independent classifiers can reduce the interference between different data sources.
- (ii) The result decision layer fuses the outputs of multiple classifiers with decision algorithms. It guarantees the independence and accuracy of subclassifiers and improves the performance of the primary classifier.

2. Related Work

In this section, we will introduce Android malicious application detection methods under different mechanisms in detail.

2.1. The Methods Based on Single-Feature Mechanism. The methods based on single-feature mechanism usually train the classifier with one type feature. Almin and others [6] extracted the permission as feature from applications. Then, they adopted k-means algorithm to cluster these permissions. A detection method was developed by Li et al. [7]. They named it Significant Permission Identification (SIGPID) which identified an essential subset of permissions with three types of data analysis: permission ranking with negative rate, support based permission ranking, and permission mining with association rules. SIGPID trained SVM classifiers with the essential subset of permissions. Their method reduced the number of permissions that need to be analysed. Chen et al. [8] trained machine learning classifiers

using APIs feature extracted from the smali files. To improve the robustness of online malware detectors, they proposed a robust secure-learning paradigm. Zhang et al. [9] converted the opcode sequences into an image and finally performed further feature extraction using CNN. Nataraj et al. [10] mapped malicious applications to grayscale images firstly and then obtained features through the Gabor filter. Lin et al. [11] extracted the system call sequence of an app at run-time. Then they used subsequences to detect some special Android malware generated by piggybacking malicious payloads into benign applications. Munoz and others [12] selected predictive features from the metadata which was collected from Google play. Dixon and others [13] detected malware code behaviour by using the power consumption feature based on time and location. Karbab and others [14] proposed Mal-Dozer, an automatic Android malware detection and family attribution framework that relies on sequence classification using deep learning techniques. Canfora and others [15] characterized the frequencies of opcode N -grams and used the Random Forest classifier to test detection accuracy under different values of N . In order to get the best detection results, they carried out many tests under different parameters. The average detection rate reached 97%. Li et al. [16] extracted the number of opcodes in each sample into a binary matrix of the same size. Then they took advantage of binary matrices to train CNN. This detection system achieved an accuracy of 99%. Zhang et al. [17] calculated the n -gram value of the opcode. The value of the opcode was divided into SA-CNN slices to train CNN. The shape of every SA-CNN was (M, N) . The result showed that the experimental index was optimal when (M, N) was $(400, 10)$. Compared with Canfora and others [15], Li et al. [16] and Zhang et al. [17] avoided the tedious process of selecting parameters. Most of the parameters of CNN could be obtained through feature training process.

2.2. The Methods Based on Multifeatures Fusion. The methods based on multifeatures fusion are a concept of early fusion. These methods usually fuse multiple single features into a group of mixed features. The mixed features are used to train a traditional machine learning algorithm or a deep learning algorithm. Peiravian and others [18] used the call relationships between function packages and classes in the applications as a feature which could present APIs. They also got the permission application list from configuration files. This list was another feature which could present permissions. Then they fused these two classes of features into a feature set and used them to train traditional machine learning classifiers. The classifiers could detect malicious behaviours in Android applications. Afonso and others [19] constructed a group of hybrid features including API calls and system call traces. For classification, Random Forest classifier was utilized. Arp et al. [20] proposed a lightweight detection method, which was named Drebin. The Drebin classifier was trained by a group of mixed features which were made up of permissions and APIs. This method significantly enhanced the detection ability and efficiency. Zhang et al. [21] used binary values to represent opcodes,

permissions, and API usage frequency values in the application. The binary was converted into a RGB image as a feature. Deep learning algorithm could obtain rules hidden in the data by learning sample data. Han et al. [22] proposed a hybrid feature construction method named MalDAE that fused the dynamic and static API sequences. Feng et al. [23] fused manifest properties and API calls into a hybrid matrix; this matrix was the training feature of deep neural networks. Arshad and others [24] explained a 3-level hybrid malware detection model named SAMADroid. They extracted dynamic feature in level 1 and static feature in level 2 and trained machine learning classifier using dynamic and static feature in level 3. Suarez-Tangil and others [25] proposed the DroidSieve method from which several static features were extracted. These features included permissions, APIs, and application components. Holland et al. [26] and Quan et al. [27] adopted pattern match algorithm and the mixed feature to detect malwares.

2.3. The Methods Based on Decision Mechanism. The methods utilizing decision mechanisms usually train multiple detection classifiers with single or multiple features. The results of the multiple classifiers are then fused into a final detection result using a decision algorithm or decision layer. Wu et al. [28] extracted multiple types of features from the application, which included permissions, subassemblies, the information of intent, and APIs. These features were used separately to train different classifiers using traditional machine learning algorithms. The detection results of these classifiers were combined in pairs to make a decision. The decision result was taken as the final test result. Tang et al. [29] extracted two types of features: opcode n-gram and the frequencies of duplicate code subblocks. They trained the XGBoost classifier with opcode n-gram and the Random Forest classifier with the frequencies of duplicate code subblocks. Then they added a decision algorithm after these two classifiers. Ananya et al. [30] used dynamic analysis to get application system calls, which was represented by the n-gram algorithm. The n-gram of the system calls was utilized as a feature to train a machine learning classifier and a DNN classifier separately. Finally, they adopted a decision algorithm to fuse the results of the two classifiers.

2.4. Summary and Analysis. With the development of malicious application technologies, the variability and uncertainty of the new type malicious applications greatly reduce the performance of the detection methods based on single-feature mechanism. The methods based on multifeatures fusion mechanism, which fuses multiple types of features into a set of tensors to train one type of classifier, solve the problem of single-feature mechanisms. They enriched the information in the features. The method based on multifeatures fusion mechanism could stably detect the polytropic malicious behaviours in applications. However, with the number of type features increasing, different types of features in the mixed tensor interfere with each other when training the classifier. It will increase the false positive rate for benign applications, reduce the recall rate for benign

applications, and limit further improvement of the detection capability.

Based on the above problems, we propose an Android malicious application detection method with a decision mechanism. Our method uses feature construction methods and detection algorithms which show good performance in single-feature mechanism and multifeatures fusion mechanism. Based on this, we enhance the performance. The features constructed by these methods can be divided into self-defined structured features and image features. Self-defined structure features include APIs, permissions, opcode, system calls, etc. It usually relies on the disassembly techniques when constructing these features. The accuracy and comprehensiveness of these features will be disturbed by shell and code obfuscation techniques which are often used to prevent the analysis of the application by external programs or software. Image features can be constructed with RGB image visualization technologies. These technologies convert binary files directly into RGB images. It avoids some of the detection problems of classifiers, which are caused by disassembly applications.

The classifiers of technologies can be classified into traditional machine learning classifiers and deep learning classifiers. Compared to traditional machine learning classifiers, the parameters and weights of deep learning classifiers could be obtained through a self-study process. The ability of the deep learning classifiers is more stable.

3. Our Approach

The Android malicious application detection model, shown in Figure 1, consists of three processes: parallel RGB image visualization, parallel detection classifier, and decision layer. The process of parallel RGB image visualization constructs three types of images which are utilized to train the detection classifiers separately. The process of parallel detection classifier consists of three separate classifiers which adopt the VGG16 algorithm. The purpose of the process decision layer is to fuse the outputs of the parallel detection classifier by using a decision algorithm.

3.1. Parallel RGB Image Visualization Technology. As shown in Figure 2, the parallel RGB image visualization technology will create three types of images: dex-image, manifest-image, and certificates-image. Dex file is the data source for dex-image, which contains all compiled Java code. Android manifest file is the data source for the manifest-image. It is usually stored in the root directory. The certificate files mainly contain MF, SF, and RSA files. They can be regarded as the containers of APK to record the digest information of all files in APK. These files are the data source of certificates-image.

These images are created by the image visualization technology. We get the dex files, the Android manifest files, and the signature files in the META-INF folder by unpacking the APK. Then we extract three binary strings, B_{dex} , $B_{manifest}$, $B_{certificates}$, from these files. B_i is a string

FIGURE 1: Android malicious application detection model diagram.

FIGURE 2: Parallel RGB image visualization technology.

containing 0 and 1. The RGB visualization technology is to convert B_i into a “red-green-blue” image. Each pixel is composed of three channels. The value range of each channel is $[0, 255]$. RGB visualization requires the following three basic steps.

- (i) Divide B_i into three equal segments of characteristic binary code of the same length b_{ir}, b_{ig}, b_{ib} . b_{ir} is the binary code snippet for the red channel. b_{ig} is the

binary code snippet for the green channel. b_{ib} is the binary code snippet for the blue channel.

- (ii) Divide each code segment into subcode segments of 8-bit length. Each subcode segment represents the value of each channel, p_i , at the pixel point. So, $b_{ir} = \{p_{ir1}, p_{ir2}, \dots, p_{irj}\}$, $b_{ig} = \{p_{ig1}, p_{ig2}, \dots, p_{igj}\}$, $b_{ib} = \{p_{ib1}, p_{ib2}, \dots, p_{ibj}\}$. Suppose a binary code segment is 0101011011010111. This

process is 0101011011010111 → 01010110, 11010111 → 86, 215.

- (iii) Finally, transform b_{ir}, b_{ig}, b_{ib} into the pixel matrix of column J and row K . Transform b_{ir}, b_{ig}, b_{ib} into three matrices whose dimensions are $[J, K]$. Algorithm 1 is the way to get the column J and row K . Then pad the data of b_i into the matrix M_i whose dimension is $[J, K]$. Finally, use the Image.fromarray function to convert $[M_{ir}, M_{ig}, M_{ib}]$ into a RGB image.

3.2. Parallel Detection Classifier. In this process, we choose VGG16 neural network algorithm developed in 2014 to generate three classifiers: dex-classifier, manifest-classifier, and certificates-classifier. The VGG16 neural network algorithm has 16 parameter layers and 5 no-parameter layers: 13 convolutional layers, 3 fully connected layers, and 5 maximum pooling layers.

Convolution layer: The convolution layer consists of several convolution kernels. Formulas (1) and (2) are the calculation equations of the convolution layer.

$$N = \frac{(W - F + 2P)}{S + 1}, \quad (1)$$

$$c_j = f_1(\text{Conv}(\mathbf{M}, \mathbf{w}_j) + \mathbf{b}_j). \quad (2)$$

Max-pooling layer: The purpose of the Max-pooling layer is to extract the maximum value of the target region. The filter size is 2×2 . Stride is 2.

Fully connected layer: Before the fully connected layer, the output matrix of the last pooling needs to be stretched into a one-dimensional vector $\mathbf{z}$ by a flattening function. The output of the previous fully connected layer is the input of the next fully connected layer. Each node in the fully connection layer is connected to all nodes in the preceding layer. Formula (3) is the calculation formula of the fully connected layer.

$$\mathbf{y} = f_2(\mathbf{W}_f \cdot \mathbf{z}) + \mathbf{b}'_f. \quad (3)$$

3.3. Decision Layer. The outputs of three classifiers are 0 or 1, respectively. When the output of a classifier is 0, the classifier considers that such data sources cannot present typical behaviours. When the output of a classifier is 1, the classifier considers that such data sources can present typical behaviours. For example, if the typical behaviours are presented by the dex file, the detection value of dex-classifier is 1. In the process of parallel detection classifier, each application has three detection values which are the outputs of dex-classifier, manifest-classifier, and certificates-classifier. Then, we use the decision algorithm to fuse the three detection values. The output of decision algorithm is the predicted value for each application. When the result of any classifier is 1, the decision algorithm considers the application as a malicious application. For example, the outputs of the three classifiers are (x, y, z) . x is the result of dex-classifier. y is the result of manifest-classifier. z is the result of certificates-classifier. If $x == 1 | y == 1 | z == 1$, $o = 1$. The

application is malware. o is the output of the decision algorithm. If $x! = 1 \& y! = 1 \& z! = 1$, $o = 0$. The application is a benign application. Table 1 shows the results of all decisions algorithm.

4. Experiments

In order to evaluate detection techniques with decision mechanisms, we conduct experiments for stability evaluation in three datasets: AndMal2017, CICMalDroid2020, and DREBIN. Then, we verify the effectiveness of decision mechanism by comparing the detection results of dex-classifier, manifest-classifier, certificates-classifier, and decision mechanism. In the end, we compare the detection results under the three mechanisms: decision, single-feature, multifeatures fusion.

4.1. Environment and Datasets. The equipment used in our experiment is a machine with 32G RAM, 1T HDD, and Intel(R) Xeon(R) Silver 4214 CPU operating at 2.20 GHz. Table 2 shows the three datasets for our experiments.

4.2. Evaluation Parameters. In order to evaluate the effectiveness of our proposed method, we adopt some evaluation parameters, including precision, accuracy, TPR, f1-score, receiver operating characteristic (ROC) curve, and Area Under Curve (AUC). These parameters help us to evaluate the effectiveness of our method. TP is the number of applications correctly classified as malicious. FP is the number of benign applications incorrectly classified as malicious. TN is the number of benign applications correctly classified as benign. FN is the number of malicious applications incorrectly classified as benign.

Precision is defined as

$$\text{Precision} = \frac{\text{TN}}{\text{FN} + \text{TN}} \cdot \frac{\text{TP}}{\text{FP} + \text{TP}}. \quad (4)$$

Accuracy can be calculated by

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}}. \quad (5)$$

TPR, also known as recall rate, is defined as

$$\text{TPR (Recall)} = \frac{\text{TN}}{\text{TN} + \text{FP}} \cdot \frac{\text{TP}}{\text{TP} + \text{FN}}. \quad (6)$$

Naturally, F1-score is defined as

$$\text{F1 - score} = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}. \quad (7)$$

The ROC curve is called sensitivity curve. All points on the curve reflect the same sensitivity, which is the result of the response to the same signal stimulus under several different criteria. AUC is the area enclosed by ROC curve and coordinate axes.

4.3. The Detection Effect of Three Datasets. Table 3 presents the results of the decision mechanisms utilized in AndMal2017, CICMalDroid2020, and DREBIN datasets. As

```

(i) Input  $b_{ij}$ ; Output:  $J \& K$ 
 $l = \text{length}(b_{ij})$ ;  $J = \text{sqrt}(l)$  If  $l \% J == 0$ :  $K = l // J$ 
Else: low = pow( $J, 2$ ) high = pow( $J + 1, 2$ )
If
 $(l - \text{low}) \geq (\text{high} - l)$ :  $J = J + 1$   $J = J - 1$ 
While  $J = J - 1$ :
If  $(l \% J)$ : break  $J - = 1$   $K = l // J$ 

```

ALGORITHM 1: Getting the column J and row K

TABLE 1: Decision algorithm.

Dex-classifier	Manifest-classifier	Certificates-classifier	Decision result
0	0	0	0
0	1	0	1
0	0	1	1
0	1	1	1
1	0	0	1
1	1	0	1
1	0	1	1
1	1	1	1

TABLE 2: Datasets used in the experiment.

	AndMal2017	CICMalDroid2020	DREBIN
Benign application	1000	2000	2000
Malicious applications	500	500	500

TABLE 3: The detection effect of three datasets%.

Datasets		AndMal2017	CICMalDroid2020	DREBIN
TPR	Benign	96	94	96
	Malicious	89	87	84
F1-score	Benign	95	95	92
	Malicious	88	83	91
Precision	Benign	96	97	98
	Malicious	87	85	90
Accuracy		94	92	93

shown in Table 3, the TPR, f-score, and precision of malicious applications are all between 85% and 90%. The TPR, f-score, and precision of benign applications are all between 95% and 98%. The accuracy of the datasets is 94%, 92%, and 93%, respectively.

Figure 3 is the ROC curve for the three datasets, which shows the AUC curve and ROC values for the three datasets. The AUC values are 0.92, 0.88, and 0.87.

As shown in the experimental data in Table 3, the evaluation parameters of the three datasets fluctuate within a small range. There are no extreme differences in the experimental data due to the variation of the datasets. This indicates that our method has good detection stability. In different complex scenarios, the Android malicious application detection method with a decision mechanism can maintain excellent performance.

4.4. The Detection Results of Primary Classifier and Subclassifiers. The primary classifier is the model shown in Figure 1. The subclassifiers are the three classifiers before the decision layer: dex-classifier, manifest-classifier, and certificates-classifier.

The detection results of primary classifier and subclassifiers on dataset AndMal2017 are shown in Table 4. The malicious' TPR of the subclassifiers is between 40% and 50%. The malicious' F1-score of subclassifiers is between 55% and 65%. The malicious' precision of subclassifiers is between 80% and 90%. All the benign applications' evaluation parameters of subclassifiers are above 85%. All the malicious' evaluation parameters of primary classifier are above 85%. All the benign applications' evaluation parameters of primary classifier are above 95%. The accuracy rates of the primary classifier and subclassifiers are 94%, 88%, 82%, and

FIGURE 3: The ROC curve of three datasets.

TABLE 4: The evaluation parameters of primary classifier and subclassifiers%.

Classifiers		Primary	Dex-classifier	Manifest-classifier	Certificates-classifier
TPR	Benign	96	99	93	98
	Malicious	89	48	40	42
F1-score	Benign	95	93	89	92
	Malicious	88	62	58	56
Precision	Benign	96	88	86	87
	Malicious	87	89	80	85
Accuracy		94	88	82	86

86%, respectively. The AUC values, shown in Figure 4, are 0.92, 0.73, 0.67, and 0.70.

According to the experimental data in Table 4, subclassifiers and the primary classifier have similar detection performance for benign applications. However, the evaluation parameters of the primary classifier for malicious applications are twice that of the subclassifier. According to the needs of application developers, the malicious behaviour of malwares is mainly distributed in dex files, configuration files, and dynamic link library files. Each subclassifier of the primary classifier can detect malicious behaviour in only one class of files. The following is an example of using the dex-classifier. When the typical behaviours are presented by the Java code, the dex-classifier can accurately detect malicious. Based on multiple subclassifiers, the primary classifier containing decision algorithms makes up for the inability of subclassifiers to detect the multiple type sources of the typical behaviours. The primary classifier first uses subclassifiers to the sources of the typical behaviours in various files. In this process, the parallel detection of subclassifiers does not interfere with each other. The outputs of the subclassifiers are then fused with the decision algorithm. The primary classifier identifies applications with maliciousness detected by any subclassifier as malware. The primary classifier increases the weight of detecting malicious

applications. As a result, the detection performance of malicious applications has been significantly improved.

4.5. The Effectiveness under Different Mechanisms. We use different detection mechanisms to conduct experiments on the dataset AndMal2017. Table 5 shows the results. The TPR, f1-score, and precision of the single-feature mechanism against malicious applications are 63%, 69%, and 70%. The TPR, f1-score, precision of the multifeatures fusion mechanism against malicious applications are 83%, 76%, and 71%. The TPR, f1-score, and precision of the decision mechanism against malicious applications are 89%, 88%, and 87%. The TPR, f1-score, and precision of the single-feature mechanism against benign applications are 95%, 93%, and 90%. The TPR, f1-score, and precision of the multifeatures fusion mechanism against benign applications are 91%, 93%, and 95%. The TPR, f1-score, and precision of the decision mechanism against benign applications are 96%, 95%, and 96%. The accuracy of decision mechanism, single-feature mechanism, and multifeatures fusion mechanism is 94%, 85%, and 89%, respectively. The AUC values, as shown in Figure 5, are 0.92, 0.78, and 0.87.

As shown in the experimental data in Table 5, the detection ability of the multifeatures fusion mechanism is better than that of the single-feature mechanism. In essence,

FIGURE 4: The ROC curve of primary classifier and subclassifiers.

TABLE 5: The evaluation parameters of different mechanisms%.

Detection mechanisms		Decision	Single-feature [7]	Multifeatures fusion [21]
TPR	Benign	96	95	91
	Malicious	89	63	83
F1-score	Benign	95	93	93
	Malicious	88	69	76
Precision	Benign	96	90	95
	Malicious	87	70	71
Accuracy		94	85	89

FIGURE 5: The ROC curve of different mechanisms.

the single-feature mechanism could only detect malicious behaviour represented by one type of feature. Take for example the sensitive permission features used by Li et al. [7].

Their method can only detect malicious behaviour for sensitive permissions. Their methods cannot effectively detect those applications which implement malicious behaviours through Java code, API calls, and dynamic link libraries. The multifeatures fusion mechanism is an improvement on the single-feature mechanism, which fuses many different types into one tensor feature. Tensor features include many types of features, such as code features, APIs, and permissions. Similar to the multifeatures fusion mechanism, our method converts different types of files into images. Compared with the single-feature mechanism, the multiple images used in our method and the mixed features used in the multifeatures fusion mechanism contain more abundant and comprehensive information. Therefore, the detection capability of our method and multifeatures fusion mechanism for malicious applications is much higher than that of the single-feature mechanism.

However, the recall rate of multifeatures fusion mechanism for benign applications is weaker than that of single-feature mechanism and our proposed method. The multifeatures fusion mechanism fuses multiple types of features into a set of tensors with fixed dimensions for training one type of classifier. Different types of features fused in a set of tensors will interfere with each other. It reduces the performance of the classifier for benign applications and limits

the improvement of the overall performance of the classifier. Our method separates different data sources and trains different subclassifiers by adopting parallel detection method. This approach avoids the problem of interference between data sources. Finally, the decision algorithm is used to fuse the detection results of multiple subclassifiers and improve the performance of the main classifier.

5. Conclusion

We propose an Android malicious application detection method which includes a decision mechanism to enhance the security of the blockchain operating environment. It improves the performance of Android malicious applications in complex scenarios. The parallel RGB image visualization technology in our method constructs three types of RGB images from dex files, manifest files, and certificates files. These images are then used to train three types of classifiers, respectively. Such technology reduces the interference between different data sources. Furthermore, we adopt a decision mechanism which adds a decision layer to the subclassifiers. The decision layer improves the performance by fusing the results of the three subclassifiers. Although our approach will be costly when loading multiple subclassifiers simultaneously, the efficiency of our scheme would be improved, if the subclassifiers can be deployed on different cloud detection servers.

Data Availability

The data used to support the findings of this study are included within this article.

Conflicts of Interest

The authors declare there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Research Foundation of Support Plan of Scientific and Technological Innovation Team in Universities of Henan Province 20IRTSTHN013, Shaanxi Key Laboratory of Information Communication Network and Security, Xi'an University of Posts & Telecommunications, Xi'an, Shaanxi 710121, China, ICNS202006, the Fundamental Research Funds for the Universities of Henan Province, NSFRF210312, Youth Talent Support Program of Henan Association for Science and Technology, 2021HYTP008, and Project supported by the PhD Foundation of Henan Polytechnic University, B2021-41.

References

- [1] Y. Chen, J. Sun, Y. Yang, T. Li, X. Niu, and H. Zhou, "PSSPR: a source location privacy protection scheme based on sector phantom routing in WSNs," *International Journal of Intelligent Systems*, vol. 37, no. 2, pp. 1204–1221, 2022.
- [2] C. Song, Y. Zhang, X. Gu, L. Wang, and Z. Liu, "A trajectory substitution privacy protection scheme in location-based services," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 13, pp. 4771–4787, 2019.
- [3] Y. Chen, S. Dong, T. Li, Y. Wang, and H. Zhou, "Dynamic multi-key FHE in asymmetric key setting from LWE," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 5239–5249, 2021.
- [4] T. Li, Z. Wang, G. Yang, Y. Cui, Y. Chen, and X. Yu, "Semi-selfish mining based on hidden Markov decision process," *International Journal of Intelligent Systems*, vol. 36, no. 7, pp. 3596–3612, 2021.
- [5] T. Li, Z. Wang, Y. Chen, C. Li, Y. Jia, and Y. Yang, "Is semi-selfish mining available without being detected?" *International Journal of Intelligent Systems*, 2021.
- [6] S. B. Almin and M. Chatterjee, "A novel approach to detect android malware," *Procedia Computer Science*, vol. 45, pp. 407–417, 2015.
- [7] Li Jin, 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. 14, pp. 3216–3225, 2018.
- [8] L. Chen, S. Hou, Y. Ye, and L. Chen, "An adversarial machine learning model against android malware evasion attacks," in *Proceedings of the Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, pp. 43–55, Springer, Beijing, China, July 2017.
- [9] J. Zhang, Q. Zheng, H. Yin, Ou Lu, and Y. Hu, "IRMD: malware variant detection using opcode image recognition," in *Proceedings of the 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 1175–1180, IEEE, Wuhan, China, December 2016.
- [10] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, pp. 1–7, Pittsburgh, PA, USA, July 2011.
- [11] Y.-D. Lin, Y.-C. Lai, C.-H. Chen, and H.-C. Tsai, "Identifying android malicious repackaged applications by thread-grained system call sequences," *Computers & Security*, vol. 39, pp. 340–350, 2013.
- [12] M. Alfonso, I. Martin, A. Guzman, and J. A. Hernandez, "Android malware detection from Google Play meta-data: selection of important features," in *Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS)*, pp. 701–702, IEEE, Florence, Italy, September 2015.
- [13] D. Bryan and S. Mishra, "Power based malicious code detection techniques for smartphones," in *Proceedings of the 2013 12th IEEE international conference on trust, security and privacy in computing and communications*, pp. 142–149, IEEE, Melbourne, VIC, Australia, July 2013.
- [14] K. ElMouatez Billah, D. Mourad, D. Abdelouahid, and M. Djedjiga, "MalDozer: automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.
- [15] C. Gerardo, D. Lorenzo Andrea, M. Eric, M. Francesco, and V. Corrado Aaron, "Effectiveness of opcode ngrams for detection of multi family android malware," in *Proceedings of the 2015 10th International Conference on Availability, Reliability and Security*, pp. 333–340, IEEE, Toulouse, France, August 2015.
- [16] D. Li, L. Zhao, Q. Cheng, N. Liu, and W. Shi, "Opcode sequence analysis of Android malware by a convolutional

- neural network,” *Concurrency and Computation: Practice and Experience*, vol. 32, Article ID e5308, 2020.
- [17] B. Zhang, W. Xiao, X. Xiao, A. K. Sangaiah, W. Zhang, and J. Zhang, “Ransomware classification using patch-based CNN and self-attention network on embedded N-grams of opcodes,” *Future Generation Computer Systems*, vol. 110, pp. 708–720, 2020.
 - [18] P. Naser and X. Zhu, “Machine learning for android malware detection using permission and api calls,” in *Proceedings of the 2013 IEEE 25th international conference on tools with artificial intelligence*, pp. 300–305, IEEE, Herndon, VA, USA, May 2013.
 - [19] A. Monte, M. Favero, G. Abed, J. Barroso, and P. Lício, “Identifying Android malware using dynamically obtained features,” *Journal of Computer Virology and Hacking Techniques*, vol. 11, pp. 9–17, 2015.
 - [20] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, “Drebin: effective and explainable detection of android malware in your pocket,” *Ndss*, vol. 14, pp. 23–26, 2014.
 - [21] H. Zhang, J. Qin, B. Zhang, H. Yan, J. Guo, and F. Gao, “A multi-class detection system for android malicious apps based on color image features,” in *Proceedings of the International Conference on Security and Privacy in New Computing Environments*, pp. 186–206, Springer, Tianjin, China, April 2020.
 - [22] W. Han, J. Xue, Y. Wang, L. Huang, Z. Kong, and L. Mao, “MalDAE: detecting and explaining malware based on correlation and fusion of static and dynamic characteristics,” *Computers & Security*, vol. 83, pp. 208–233, 2019.
 - [23] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, “A performance-sensitive malware detection system using deep learning on mobile devices,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1563–1578, 2020.
 - [24] A. Saba, A. Shah Munam, A. Wahid, A. Mehmood, H. Song, and H. Yu, “Samadroid: a novel 3-level hybrid malware detection model for android operating system,” *IEEE Access*, vol. 6, pp. 4321–4339, 2018.
 - [25] S.-T. Guillermo, D. Santanu Kumar, A. Mansour, K. Johannes, G. Giorgio, and C. Lorenzo, “Droidsieve: fast and accurate classification of obfuscated android malware,” in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pp. 309–320, Texas, America, March 2017.
 - [26] B. Holland, D. Tom, K. Suresh, J. Mathews, and R. Nikhil, “Security toolbox for detecting novel and sophisticated android malware,” vol. 2, pp. 733–736, in *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, pp. 733–736, IEEE, Florence, Italy, May 2015.
 - [27] D. Quan, L. Zhai, Y. Fan, and P. Wang, “Detection of android malicious apps based on the sensitive behaviors,” in *Proceedings of the 2014 IEEE 13th international conference on trust, security and privacy in computing and communications*, pp. 877–883, IEEE, Beijing, China, September 2014.
 - [28] D.-J. Wu, C.-H. Mao, T.-E. Wei, and H.-M. Lee, K.-P. W. Droidmat, Android malware detection through manifest and api calls tracing,” in *Proceedings of the 2012 Seventh Asia Joint Conference on Information Security*, pp. 62–69, IEEE, Tokyo, Japan, August 2012.
 - [29] A. Ananya, A. Aswathy, T. R. Amal, P. G. Swathy, P. Vinod, and S. Mohammad, “SysDroid: a dynamic ML-based android malware analyzer using system call traces,” *Cluster Computing*, vol. 23, pp. 1–20, 2020.
 - [30] Y.-l. Tang, X.-y. Li, Z.-q. Zhao, and Y.-F. Li, “Detection method for android payment cracked application,” *Journal of Beijing University of Posts and Telecommunications*, vol. 44, no. 4, p. 95, 2021.