Android Malware Application Detection Method Based on RGB Image Features in E-Commerce

Xingyu Li[1], Yongli Tang[1], Mary Subaja Christo[2], Zongqu Zhao[1*], Ying Li[1]

[1] Henan Polytechnic University, School of Computer Science and Technology, China,
[2] Department of Networking and Communications, School of Computing, SRM Institute of Science and Technology, India
spdogbear@163.com, yltang@hpu.edu.cn, marysubc@srmist.edu.in, zhaozong__qu@hpu.edu.cn, ying_li18@163.com

Abstract

In recent years, with the continuous development of Android applications, part of e-commerce business gradually promoted to the Android platform. Android malicious applications have become an important factor threatening E-Commerce security. The Android malware application detection methods based on machine learning play an important role in malware detection, and most of them construct self-defined structured features with static analysis technique. The accuracy and comprehensiveness of features are disturbed by shell and code obfuscation techniques. In the case of applications using shell and code obfuscation techniques, these techniques would lead to instability of detection result. In addition, the way of these methods, which mainly use disassembly technology to extract source data, would deteriorate the efficiency of detection. In view of such problems, we propose an android malware application detection method based on RGB image features. Our method uses RGB image visualization technology to directly transform binary files into unstructured RGB images. In the process of model training, we acquire advanced features autonomously by training the VGG16 model with RGB images. We perform a comprehensive analysis of our approach and other methods on the Android malware dataset. The results show good efficiency and the adaptability of our method.

Keywords: Android, Software security, E-Commerce, RGB image feature, VGG16

1 Introduction

Android is a free and open-source operating system based on Linux, widely used in E-Commerce. The Android operating system has three features: open-source code, strong hardware compatibility, and a large application market. Because of Android's openness, it becomes the main target of malicious applications. According to Android malware special report, in 2019, 360 Security Brain intercepted 1,809,000 Android malware samples, with an average daily increase of 5,000. The types of new Android malicious applications are cost consumption, privacy theft, remote control, rogue behavior, malicious deduction of fees, fraud and so on.

The Android malware applications is a constant threat to mobile E-Commerce. The security of Android applications has great significance for the sound development of the E-Commerce. In recent years, Android Malicious Application Detection Method (AMADM) is a hot issue in the field of software security. Considering the advantages of detecting unknown malicious applications, a variety of AMADMs based on machine learning have been proposed. These methods used machine learning algorithms to train models, build classifiers and predict unknown samples.

AMADMs based on machine learning have four parts: source data extraction, feature representation, model training and sample prediction. Source data extraction is to extract raw data from Android Application Package (APK) such as resource files, code files and configuration files. It would take a lot of time, when using disassembly techniques in this process. Feature representation is to construct self-defined structured features from raw data, which can reflect the information of applications. In this stage, the accuracy of structural features and the integrity of feature representation information depend on the prior knowledge used in dynamic analysis techniques or static analysis techniques. Many applications are protected with shell and obfuscation technologies. We couldn't extract a Java program completely, when the applications use shell technologies. The purpose of these technologies is to prevent external programs or software from analyzing the applications. Obfuscation technologies make code reading analysis more difficult in three ways: encode classes and methods with meaningless strings, add useless permissions and APIs to the application, and store the APIs call in ASCCLL codes. The first obfuscation method does not disturb the extraction and detection of API features. The second obfuscation disturbs the final detection effect. The third form of obfuscation prevents us from effectively extracting API features. When facing those applications which use shell and obfuscation techniques, it is difficult to get high-quality features. Model training is using representations to train some traditional machine learning models, such as support vector machines, decision tree, random forests and so on. All of these algorithms need structured input which has some rules and practical meaning. Recently, some researchers have been trying to train the model with deep learning algorithms. In comparison to traditional machine learning algorithms, deep learning algorithms can autonomously learn advanced features from image features. Furthermore, the input data of deep learning algorithms is more comprehensive than that of traditional machine learning algorithms. Sample prediction is to predict unknown applications with the model.

*Corresponding Author: Zongqu Zhao; E-mail: zhaozong__qu@hpu.edu.cn
DOI: 10.53106/160792642022112306017

We propose a new AMADM based on RGB image features. It has four parts: selection feature files, RGB visualization, model training and sample prediction. Selection feature files are to extract three types of binary files: the dex file, Android manifest files, and the certificates files in the META-INF folder. RGB visualization is to convert these files into a RGB image. Model training is to train VGG16 model with RGB images. Sample prediction is to predict unknown applications. The contributions of this paper are as follows:

(1) To improve the detection efficiency. When using binary data, we don’t need to disassemble the application. It improves detection efficiency.

(2) Convert binary code into RGB image. The method of RGB visualization directly converts binary files into a RGB image. This method dost not rely on prior knowledge. By training the VGG16 model with RGB images, we could obtain advanced features more intelligently.

(3) To improve the adaptability of detection system. The experiment results show that our method has better detection results against malicious applications than other methods. Our method has a high TPR for both benign and malicious applications.

2 Related Work

The type of feature in AMADMs mainly includes permissions, APIs, opcodes and graphs.

Permissions and APIs ware the security mechanisms which restricted the use of some restrictive functions in applications. [1-9] Peiravian et al. [1] 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. The list was another feature which could present the permissions. Then they fused the two classes of features into a set of features which was used to train the traditional machine learning classifiers. The classifiers could detect identify malicious behavior in Android applications. Arp et al. [6] proposed a lightweight detection method, which was named Drebin, based on permissions and APIs. This method significantly enhanced the detection ability and efficiency. In addition, they collected a data set of Android malicious applications, which also was named Drebin. This data set was applied to the studying of Android malicious applications. Suarez-Tangil et al. [7] proposed the DroidSieve method in which several static features were extracted. These features included permissions, APIs and application components. The method first detected whether the application had malicious behaviors. If so, they would classifiers it as belonging to a family of related malware. The detection rate of DroidSieve on 100,000 benign and malicious achieved 99.44%. Qiao et al. [8] proposed an AMADM which took permissions and APIs as features. Based on static analysis of the Android applications’ code files and resource files, they extracted permissions and API features in binary form. Then feature selection technology was used to reduce the dimension of the feature matrix and to improve efficiency. Finally, they trained different traditional machine learning classifiers to detect malicious Android applications. The result showed that the detection effect of permissions combined with API was better than that of permissions alone. Sun et al. [3] developed an AMADM method which was named SIGPID. It 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. By utilizing this method, the number of permissions which needed to be analyzed was reduced by 84%. Their detection method improved efficiency. Wu et al. [9] used static analysis mechanism to extract resting state information from applications, such as permissions, subassemblies, the information of Intent and APIs. This information could effectively detect different intentions in Android malicious applications.

The AMADMs based on permissions and APIs methods had a high detection rate for specific Android malwares. However, with the development of malicious applications, malware developers keep trying to bypass such detection methods by building structures similar to those of benign applications. There is a class of malicious applications in the production environment, which are based on benign applications and insert malicious code into benign applications. There are subtle differences, which couldn’t be reflect through permissions and APIs, between malicious applications and benign ones. In addition, in order to improve detection efficiency, the detection methods represented by Qiao et al. [8] need to select feature. It relies on the prior knowledge. The richness of researchers’ knowledge reserves directly affects the validity of features. Therefore, permissions and API characteristics greatly reduce the ability to detect malware in complex scenarios.

The object of feature extraction for the AMADMs based on opcodes and graphs [10-19] was the davlik code. So, these features could give a accurate expression than permissions and APIs. Opcode features [10, 16-19] employed the idea of natural language processing. In order to accurately and effectively detect the payment cracked type of Android malicious applications, Tang et al. [16] statically analyzed 300 Android payment cracked applications. The analysis results show that Android payment cracked applications mainly include logic cracked and content cracked. For these two cracked methods, they constructed opcode N-gram features based on function call control flow and Repeating code sub-block features. Then two groups of features were used to train XGBoost classifier and random forest classifier respectively. Finally, they used the decision algorithm to fuse the detection results of the two groups of classifiers. Experimental results showed that their detection method could effectively detect malicious behavior of payment cracked type in Android applications. Canfora et al. [17] characterized the frequencies of opcode *N-grams*, and tested the accuracy of detection at different values of *N*. The average detection rate was 97%. Experimental results showed that this method could effectively solve the detection problem of payment cracked applications. Chen et al. [18] proposed a clone detector which was named Nicad to identify consistent malwares. They extracted Java code from the binary code and found the classes of clones by Nicad method. Then to detect malware with these classes as signature. Their detection method could detect 95% of previously known malware. Zhang et al. [19] calculated the n-gram value of the opcode. Then the value of the opcode was divided into SA-CNN slices to train CNN. The shape of every SA-CNN is (M, N). The result showed that the experimental index is optimal when (M, N) is (400, 10). Graph features [11-15] adopted the call relationship of functions. Fan et al. [11] proposed a detection method named DAPASA. This method assumed two types of malicious application models: malicious behavior payloads of the piggybacked application and host malicious applications. Based on these two malicious application models, they generated a sensitive subgraph (SSG) to describe the potential malicious behavior in applications. Then they constructed five features from SSG to feed the machine learning algorithms. Hou et al. [12] used the HinDroid method to build API heterogeneous information network. The API heterogeneous information network represented four kinds of the relationship between APIs and applications: API call, API calls in the same code block, API calls with the same package name, and API calls with the same invoke type. They constructed characteristics based on these four relationships. Fan et al. [13] used the frequent subgraphs as the features which could present the malicious activity in the Android malware applications. On this basis, they developed an automated Android malicious applications classification system, which was named FalDroidAndroid.

Compared to permissions and APIs, the feature structure of opcodes and graphs is more complex. Opcodes and graphics feature extraction objects are assembly opcodes that could reflect the subtle differences between applications. Such features are based on the discipline of probability and statistics and do not depend on prior knowledge. In complex scenarios, they could detect mutated malware effectively. However, these studies did not consider the problem of disassembly applications using shell and obfuscation techniques. Shell technology and obfuscation technology increase the difficulty of obtaining core code in applications. It is difficult to disassemble accurate and complete Java code from applications that use shell and obfuscation techniques. The detection ability of the method based on opcode and graph will be limited and the detection accuracy couldn’t be further improved. In addition, disassembly technology also restricts the improvement of detection efficiency.

In recent years, some new types of AMADMs [10, 19-26] based on deep learning algorithms have been proposed by researchers. The application of deep learning algorithm provided a new direction to detect malwares. Li et al. [10] proposed an Android malware detection method based on deep learning algorithm. They first used disassembly technology to extract opcode sequences from applications and calculated the frequency of each opcode according to the official opcode sequences. Then the frequency vector of the opcode list was mapped to a binary matrix. Finally, the CNN neural network was trained by this binary matrix. The experiment result showed that the detection system achieved an accuracy of 99%. Zhang et al. [20] used binary values to represent opcodes, permissions, and API usage frequency values in the application. The binary was converted into a color image as a feature. The deep learning algorithm can obtain the rules hidden in the data, by learning the sample data. Compared to the AMADMs based on traditional machine learning algorithms, the detection models of these methods were more flexible. They showed good robustness in complex scenarios.

However, all of these methods still needed to disassembly applications. The disassembly problem of shell application was still unresolved.

3 Our Approach

In this section, we will detail three parts of our method: selection feature files, RGB visualization and model training.

3.1 Selection Feature Files

The process of selection feature files is to extract three types of binary files from the APK files: dex files, Android manifest files and certificates files in the META-INF folder. All of these files will be utilized in the process of RGB visualization.

The APK files is a type of application package which is used in Android option systems. The package mainly includes five types of files which have been compiled: the dex file, resources files, assets files, certificates files and Android manifest files. The dex file is a binary file which contains all of the compiled Java code. The androidmanifest.xll file is a manifest file of applications. It is usually stored in the root directory. The file mainly declares three types of information which is necessary for application: the name of application package, application components and permissions. The certificates files, which mainly contain MF, SF and RSA files, can be regarded as the containers of APK. They record the digest information of all files in the APK. The resources files and assets files are used to store the application’s scene resource files such as images, audio, and interface code.

The AMADMs based on machine learning usually extracted androidmanifest.xml file and dex file to construct features. In the process of source data extraction, they needed to decompile these files. The accuracy of decompile results would affect the quality of feature.

It was difficult to get complete davlik code from shell applications by decompiling. Besides, decompiling shell applications would take a lot of time. So, the features quality of opcodes and graphs would be greatly reduced, when detecting shell applications.

Unlike other methods, we don’t disassemble the dex file, the androidmanifest.xml file and the certificates files in the META-INF folder. Our method is divided into two parts: (1) Unpack the APKs. In this part, we just extract all the files from the APKs and don’t decompile these files. (2) Select feature files. In this part, we screen out the dex file, androidmainfest.xml file and the certificates files in the META-INF folder from all the files which have been extracted from the APKs.

3.2 RGB Visualization

Figure 1 is the detailed process of RGB visualization. The process mainly includes binarization and RGB visualization parts.

We get dex files, androidmanifest.xml files, and the certificate files in the META-INF folder by unpacking the APK. Binarization is to generate B by splicing the binary of these files. B is a string containing 0 and 1. Algorithm 1 is the way to splice the binary data of these files. F_d, F_a, F_c is dex files, androidmanifest.xml files, and the certificate files in the META-INF.

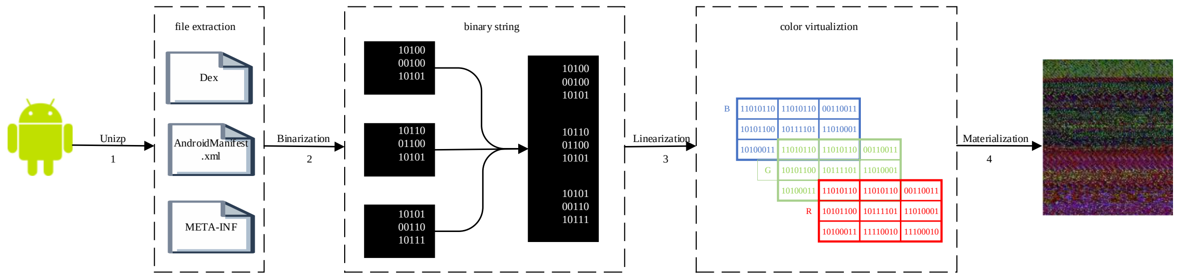

Figure 1. The process of Android RGB visualization

Algorithm 1. Reading the data of files

```
Input: $F_d$, $F_a$, $F_c$
Output: $B$
$Array_d$, $Array_a$, $Array_c$
Read.binary($F_d$, $F_a$, $F_c$).to_array()
For $arrayi$ in ($Array_d$, $Array_a$, $Array_c$):
    For $i$ in length($array_i$):
        $B$.append($arrat_i[i]$)
```

The RGB visualization method is to convert B into a "red-green-blue" three-channel 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.

We divide B into three equal segments of characteristic binary code of the same length b_r, b_g, b_b by algorithm 2. b_r is the binary code snippet for the red channel. b_g is the binary code snippet for the green channel. b_b is the binary code snippet for the blue channel.

Algorithm 2. Divide B into b_r, b_g, b_b

```
Input: $B$
Output: $b_r$, $b_g$, $b_b$
While (length($B$)%3! = 0): $B$.append(0)
$b_r = B$[0: 1/3 length($B$)]
$b_g = B$[1/3 length($B$): 2/3 length($B$)]
$b_b = B$[2/3 length($B$): length($B$)]
```

Divide each code segment into sub-code segments of 8-bit length. Each sub-code segment represents the value of each channel, P_i, at the pixel point. So, $b_r = \{P_{r1}, P_{r2}, \ldots, P_{ri}\}$, $b_g = \{P_{g1}, P_{g2}, P_{gi}\}$, $b_b = \{P_{b1}, P_{b2}, P_{bi}\}$. The range of P_i is [0, 255]. Suppose a binary code segment is 0101000101110100. This process is 0101000101110100$\rightarrow$ 01010001, 01110100$\rightarrow$ 81, 116.

Transform b_r, b_g, b_b into three matrices whose dimensions are [J, K]. Algorithm 3 is the way to get the column J and row K. Then padding the data of b_i into the matrix M_i whose dimension is [J, K]. Finally, use the Image.fromarray function to convert [M_r, M_g, M_b] into a RGB image.

3.3 Model Training

3.3.1 Preprocessing

Before training the model, we need to do some preparatory work which is named preprocessing. Preprocessing mainly includes three parts: image standardization, setting labels and data set segmentation. (1) Image standardization is to transform the images in a data set into images with the same dimension. We use the transforms.Compose function to transform the image into a standard $224\times224\times3$ image. (2) Setting labels is to set the image of the malware to value 1 and the image of the benign to value 2. (3) Data set segmentation is to take 90% of the data set as the training set and 10% of the data set as the test set.

Algorithm 3. Getting the column J and row K

```
Input: $b_i$
Output: $J$ & $K$
$l = length(b_i)$
$J$ = sqrt($l$)
If $l$%$J$==0: $K = l//J$
Else:
    $low$ = pow($J$, 2)
    $high$ = pow($J$ + 1, 2)
    If ($l - low$) > = ($high - l$): $J = J + 1$
    $J = J - 1$
    While ($J$ > 0):
        If ($l$%$J$): break
        $J$- = 1
    $K = l//J$
```

3.3.2 VGG16 Model

We choose VGG16 neural network algorithm as the model training algorithm. VGG16 network is a deep network model developed in 2014. It has 16 training parameter networks. As shown in Figure 2, VGG16 consists of 13 convolutional layers, 5 maximum pooling layers and 3 full connection layers. The parameters of convolutional layers and full connection layers are obtained by training. In addition, we use transfer learning in our work. The parameters of convolutional layers in our model are pre-trained. We only train the parameters of full connection layers. There are three advantages to adopting transfer learning: (1) The model has excellent initial performance.(2) In the training process, the model is improved at fast rate.(3) After training, the model has excellent convergence results.

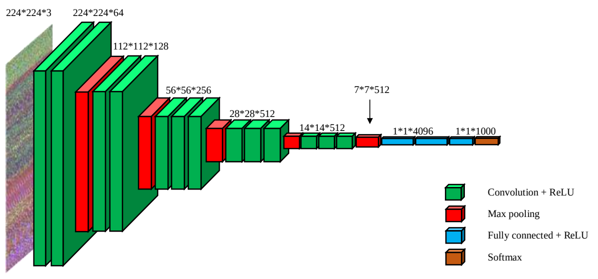

Figure 2. VGG16 model

Convolution layer: The convolution layer consists of several convolution kernels. Formula (1) is the calculation formula of the convolution layer. N is the size of output matrix $\mathbf{M}$. W is the input size. F is the convolution kernel size. P is the filling value size. S is the step size. Formula (2) is the operation formula of the convolution kernel. $\mathbf{W_j} \in R^{3\times3}$, $\mathbf{W_j}$ is the parameter matrix of the convolution kernel. is the convolution kernel set with dimension $R^{3\times3}$. $\mathbf{M}$ is the input matrix. $\mathbf{b_j}$ is the bias parameter matrix for each convolution kernel. f_1 is the ReLU activation function.

$$N = (W - F + 2P)/\, S + 1. \tag{1}$$

$$c_j = f_1(Conv(\boldsymbol{M}, \mathbf{w_j}) + \mathbf{b_j}). \tag{2}$$

Max-pooling layer: As shown in Figure 3, 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.

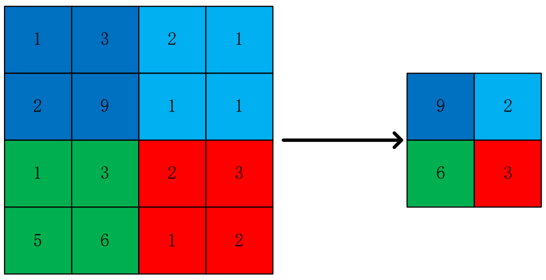

Figure 3. Max-pooling

Full connection layer: Before the full connection layer, the last pooled output matrix needs to be stretched into a one-dimensional vector **z** by using the flatten function. Then the output of the previous full connection layer is the input of the next fully connection layer. As shown in Figure 4, each node in the full connection layer is connected to all nodes in the preceding layer. Formula (3) is the calculation formula of the fully connection layer. $\mathbf{W_f}$ is the weight matrix of the full connection layer. **b'** is the offset term matrix. **y** is the output matrix of the full connection layer.

$$\mathbf{y} = f_2(\mathbf{W_f z}) + \mathbf{b}'. \tag{3}$$

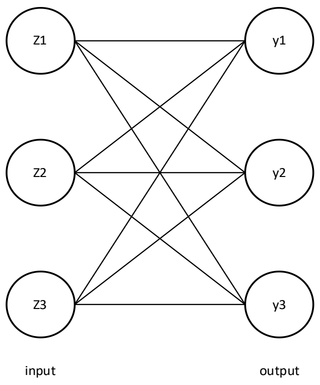

Figure 4. Full connection layers

4 Experiments

In order to verify the effectiveness of our method in Android malicious application detection, we mainly answer the following three questions (Qs) through experiments:

(1) Q1: Whether our method has good detection ability?
(2) Q2: Whether our method has better adaptability compared with other methods?
(3) Q3: Whether our method is more efficient?

4.1 Environment and Datasets

The equipment used in our experiment is a machine with 192G RAM, 2T HDD and Intel(R) Xeon(R) Silver 4214 CPU operating at 2.20 GHz.

Table 1 shows the three data sets which were utilized in our experiment.

Table 1. Data sets used in the experiment

	AndMal	CICMalDroid	DREBIN
Benign application	1500	3000	3000
Malicious applications	500	1000	1000

4.2 Evaluation Parameters

The evaluation parameters used in the experiment include precision, accuracy, TPR, f1-score, receiver operating characteristic (ROC) curve and Area Under Curve (AUC). TP is the number of malicious applications which are correctly classified as malicious applications. FP is the number of benign applications which are misclassified as malicious applications. TN is the number of benign applications which are correctly classified as benign applications. FN is the number of malicious applications which are misclassified as benign applications.

Precision is the percentage of the benign/malicious applications, which are correctly predicted, of all identified benign/malicious applications.

$$\text{Precision} = \frac{TP}{FP+TP} / \frac{TN}{FN+TN}\ . \tag{4}$$

Accuracy is the percentage of the applications, which are correctly predicted, of all applications.

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

TPR is the percentage of benign/malicious applications, which are correctly predicted, of all benign/malicious applications.

$$\text{TPR} = \frac{TP}{TP+FN} / \frac{TN}{TN+FP} \quad . \tag{6}$$

F1-score is defined as

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

The ROC curve is also 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. The horizontal axis of the ROC curve is false positive rate (FPR), which is the probability of incorrectly predicting a positive example. The vertical axis of the ROC curve is true positive rate (TPR).

$$\text{FPR} = \frac{\text{FP}}{\text{FP+TN}} \quad . \tag{8}$$

AUC is the area surrounded by roc curve and coordinate axes. The value range of AUC is [0, 1]. When AUC is [0, 0.5), the constructed detection model can’t detect unknown samples. When AUC is 0.5, the detection model is a random prediction model. When AUC is (0.5, 1], the detection model can effectively detect the unknown samples. The higher the value of AUC, the stronger the model detection capability.

4.3 Answering Q1: The Detection Effect of Our Method Used in Different Datasets

Before training the model, we extract the RGB image features applied to AndMal, CICMalDroid and DREBIN. As shown in Figure 5, there are significant differences between malicious and benign applications in three data sets. Moreover, the feature images of benign applications between different data sets have high similarity.

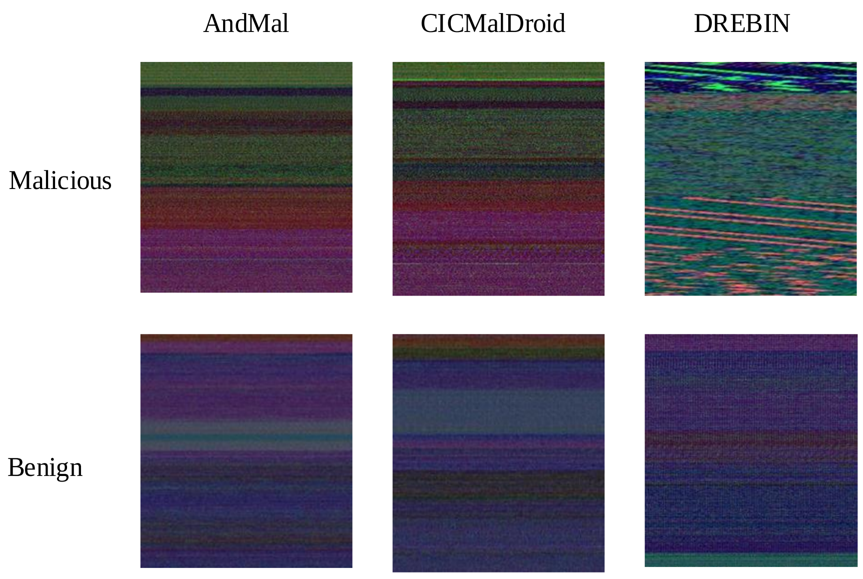

Figure 5. RGB images features of three datasets

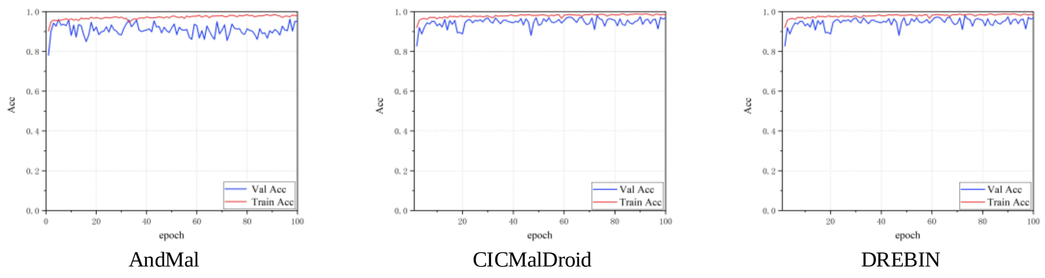

Figure 6. The experiment result of three datasets

The VGG16 neural network model in our method is trained by using three data sets respectively. As shown in Figure 6. After 100 rounds of training, the AndMal, CICMalDroid and DREBIN training sets are nearly 99 percent accurate. The test set is around 95% accurate. Because we use transfer learning, the accuracy of the training-test set is about 80% at the beginning of model training.

Table 2 lists the detection results of our method utilized in AndMal, CICMalDroid and DREBIN data sets. The TPR values of the benign application in the three data sets are 98%, 99% and 98%. The TPR for malicious applications are 91%, 81% and 84%. The f1-score values of benign applications are 97%, 95% and 95%. The f1-Scores of malicious applications are 93%, 87%, and 90%. The precision values of benign applications are 96%, 91% and 91%. The precision values of malicious applications are 95%, 98%, and 97%. Accuracy values of RGB image features in the three data sets are 96%, 93% and 94%.

Table 2. The detection effect of three datasets %

Datasets		AndMal	CICMalDroid	DREBIN
TPR	Benign	98	99	98
	Malicious	91	81	84
f1-score	Benign	97	95	95
	Malicious	93	87	90
precision	Benign	96	91	91
	Malicious	95	98	97
accuracy		96	93	94

Figure 7 shows the ROC curve and AUC values for the three data sets. The AUC values are 0.94, 0.89 and 0.91. These values indicate that the VGG16 detection model trained by our method is a strong detection model.

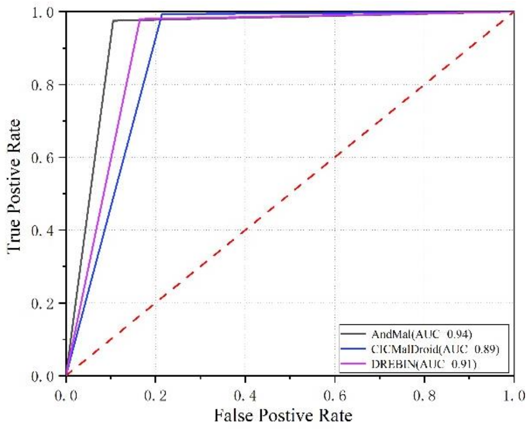

Figure 7. The ROC curve of three datasets

4.4 Answering Q2: Compare the Detection Ability Our Method with Other Methods

Table 3 lists the detection results of RGB image features and other three methods on the AndMal2017 dataset. The TPR, f1-score and precision of malicious applications are higher than the other three methods. The TPR values of malicious applications in our method and the other three methods are 91%, 87%, 63% and 73%. The f1-score values of malicious applications in our method and the other three methods are 93%, 83%, 69% and 72%. The precision values of malicious applications in our method and the other three methods are 95%, 79%, 77% and 71%. The accuracy values of our method and other three methods are 96%, 92%, 88% and 89%.

Table 3. The detection effect of different features %

Methods		RGB image	n-gram [19]	API [2]	Binary [10]
TPR	Benign	98	94(-4)	95(-3)	92(-6)
	Malicious	91	87(-4)	63(-28)	73(-18)
f1-score	Benign	97	95(-2)	93(-4)	93(-4)
	Malicious	93	83(-10)	69(-24)	72(-21)
precision	Benign	96	97(+1)	91(-5)	93(-3)
	Malicious	95	79(-16)	77(-18)	71(-24)
accuracy		96	92(-4)	88(-8)	89(-7)

Effective detection of malicious applications is of great significance to protect mobile e-commerce. In order to visualize the advancement of our approach, in Table 3, we indicate the absolute difference of the detection data between their method and ours in parentheses. It is worth noting that the detection data of our method for malicious applications is significantly higher than the other three methods. For example, for the detection data of malicious applications, our method has 28% higher TPR than API [2], 21% higher f1-score than Binary [10] and 16% higher precision than n-gram [19].

Figure 8 shows the ROC curve and AUC values on the AndMal data set. The AUC values of our method and other three methods are 0.94, 0.91, 0.79 and 0.83. The above schemes can effectively detect unknown malicious applications. However, the detection ability of the model trained by our method is better than the other three methods.

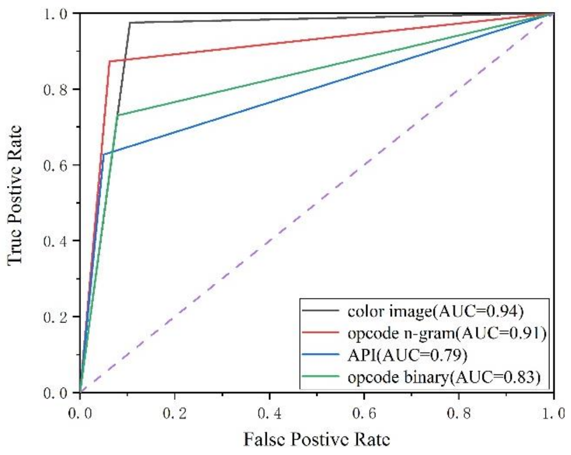

Figure 8. The ROC curve of different features

4.5 Answering Q3: Compare the Time and Space Used by Our Method and Other Methods

We compared the time and space, which is spent on our method and the other three methods, with 50 applications as examples.

Figure 9 is the line box diagram of time distribution. The time which is spent on our method was about 5s. The extraction time of the other three methods is about 15s. The three methods took three times as long as ours.

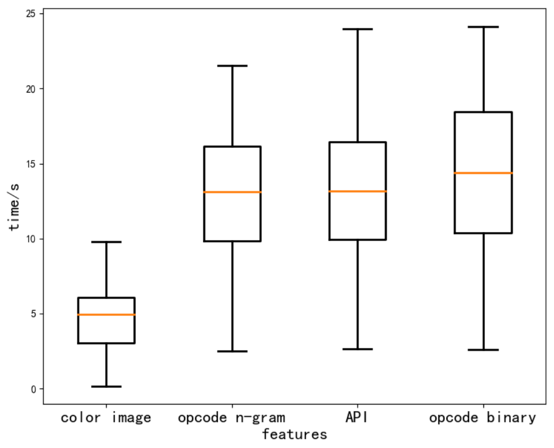

Figure 9. Running time distribution diagram

Figure 10 shows the line-box diagram of the distribution of the file size generated. The size of files generated by our method is much smaller than the methods based on opcode N-gram, API and Opcode binary.

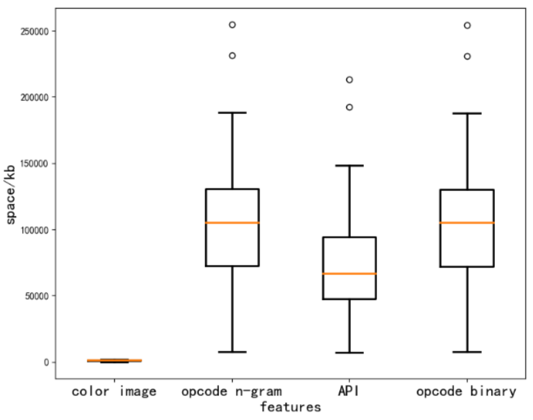

Figure 10. Excessive files size distribution diagram

4.6 Discussion

Table 4 lists the performance comparison between our method and the other three methods. The detection accuracy of our method and the method used in reference 19 is much higher than that of reference 2 and 10. The proposed method is superior to the above three methods in the time and space required for detection. To sum up, the Android malware detection method based on RGB image features has good detection performance and efficiency.

Table 4. Performance comparison

Indicator	Accuracy	Time	Space
RGB image	high	low	small
n-gram [19]	high	high	big
API [2]	moderate	high	big
Binary [10]	moderate	high	big

Discuss Q1: As shown in Figure 5, the RGB images of benign applications are well-organized granular images. The structures of these images have a clear sense of layered. However, the structures of malicious' RGB images are chaotic. The malicious behaviors of malicious applications mainly include cost consumption, privacy theft, remote control, permission abuse and so on. In order to avoid detection, the malicious behavior mentioned above usually does not appear in the application alone. Malicious application developers would inject malicious behavior code and configuration information into the code and configuration information of benign behavior. It does not interfere with the normal operation of the applications. But this operation disrupts the overall structure of the code and configuration information. The RGB visualization observes application configuration information, code information, and certificate information from a macroscopic perspective. On the image got by the RGB visualization, the malicious applications present a phenomenon of chaotic structure. Because there is no malicious behavior in the benign applications, the overall structure of configuration information, code information, and certificate information is not disturbed. The RGB images of benign applications is an ordered layered structure image. We use RGB images of three data sets to train the VGG16 algorithm. According to the changes in detection accuracy of training set and test set shown in Figure 6, our method has high detection accuracy in the three data sets. So, our method has good detection ability.

Discuss Q2: In Table 3, the detection result of our method for malicious applications is better than the other three detection methods. There are two main factors influencing these methods based on opcode n-gram and opcode binary: obfuscation technology and shell technology. Obfuscation techniques make code analysis more difficult. The methods based on opcode n-gram and opcode binary need to locate the key code information in the code. The accurate information of code processed by obfuscation techniques is difficult to obtain through static analysis. It increases the difficulty of obtaining accurate features. Shell technology which could increase the difficulty of obtaining critical code is a code protection technology. So, it's difficult to obtain detailed and rich opcode information. Although it's hard to use disassembly technique to extract accurate opcode information from the applications which are protected by the obfuscation and shell technology, all of the applications' information is still stored intactly in the dex file as binary data. The RGB visualization technology is to convert all binary data of the object files into images. This process eliminates disassembly techniques. Shell technology and obfuscation technology do not interfere with extracting the information of application. Our approach ensures maximum information integrity. In addition, the API features are simple in structure. Malicious applications can circumvent the detection systems by using the same APIs of benign applications. Compared with API features, the advanced features obtained by training VGG16 with RGB images are more complex. Malicious applications can't bypass detection by such methods.

Discuss Q3: As shown in Figure 9, the methods based on opcode n-gram, opcode binary and APIs take three times as long as our method. These three methods need disassembly application when constructing feature. Disassembly applications, especially which using shell technology, require a lot of time. Our approach uses RGB visualization techniques to transform application-critical information into images. Finally, training VGG16 is used to generate neural network detection model. This process does not utilize disassembly technology, saving the disassembly application time. The results of Figure 10 show that the space used by the other three methods is much larger than ours. The reason is that the other three methods inevitably generate many excessive files when constructing features. Instead, our method uses binary files directly to construct features and does not produce excessive files.

5 Conclusion

In order to protect the security of E-Commerce applications on Android platforms, we propose an Android malware application detection method based on RGB image features. In this paper, we designed three groups of experiments to evaluate our approach from the perspectives of stability, performance and efficiency. The results show good results in feature processing efficiency and the adaptability of our method. The RGB visualization technology in our method converts binary files into RGB pictures. Such technology not only improves the speed of data processing, but also avoids information loss caused by disassembly. Compared with the self-defined structural features of traditional machine learning, the form of images has visual form which almost retains all the information about the applications. When detecting special applications which utilize shell and obfuscation techniques, the deep learning technology which was adopted by our method in the process of model training has better detection performance. Although deep learning takes a long time to train when processing a large number of samples, the mobile E-Commerce would improve the detection efficiency of the model by using server-side training and client-side detection.

Funding

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.

References

[1] N. Peiravian, X. Zhu, Machine learning for Android malware detection using permission and API calls, *2013 IEEE 25th international conference on tools with artificial intelligence*, Herndon, VA, USA, 2013, pp. 300-305.

[2] P. P. K. Chan, W. K. Song, Static detection of Android malware by using permissions and API calls, *2014 International Conference on Machine Learning and Cybernetics*, Lanzhou, China, 2014, pp. 82-87.

[3] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, H. Ye, Significant permission identification for machine-learning-based android malware detection, *IEEE Transactions on Industrial Informatics*, Vol. 14, No. 7, pp. 3216-3225, July, 2018.

[4] K. W. Y. Au, Y. F. Zhou, Z. Huang, D. Lie, Pscout: analyzing the android permission specification, *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, Raleigh, North Carolina, America, 2012, pp. 217-228.

[5] Y. Aafer, W. Du, H Yin., Droidapiminer: mining api-level features for robust malware detection in android, *International Conference on Security and Privacy in Communication Systems*, Sydney, Australia, 2013, pp. 86-103.

[6] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, DREBIN: effective and explainable detection of Android malware in your pocket, *Network & Distributed System Security Symposium*, San Diego, California, America, 2014, pp. 23-26.

[7] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, L Cavallaro, Droidsieve: Fast and accurate classification of obfuscated android malware, *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, Scottsdale, Arizona, USA, 2017, pp. 309-320.

[8] M. Qiao, A. H. Sung, Q. Liu, Merging permission and api features for android malware detection, *2016 5th IIAI International Congress on Advanced Applied Informatics*, Kumamoto, Japan, 2016, pp. 566-571.

[9] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, K. P. Wu, Droidmat: Android malware detection through manifest and API calls tracing, *2012 Seventh Asia Joint Conference on Information Security*, Tokyo, Japan, 2012, pp. 62-69.

[10] D. Li, L. Zhao, Q. Cheng, N. Lu, W Shi, Opcode sequence analysis of Android malware by a convolutional neural network, *Concurrency and Computation Practice and Experience*, Vol. 32, No. 18, Article No. e5308, September, 2020.

[11] M. Fan, J. Liu, W. Wang, Z. Z. Tian, T. Liu, Dapasa: detecting android piggybacked apps through sensitive subgraph analysis, *IEEE Transactions on Information Forensics and Security*, Vol. 12, No. 8, pp. 1772-1785, August, 2017.

[12] S. Hou, Y. Ye, Y. Song, M. Abdulhayoglu, Hindroid: An intelligent android malware detection system based on structured heterogeneous information network, *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Halifax, NS, Canada, 2017, pp. 1507-1515.

[13] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zhang, T. Liu, Android malware familial classification and representative sample selection via frequent subgraph analysis, *IEEE Transactions on Information Forensics and Security*, Vol. 13, No. 8, pp. 1890-1905, August, 2018.

[14] K. Chen, P. Liu, Y. Zhang, Achieving accuracy and scalability simultaneously in detecting application clones on android markets, *Proceedings of the 36th International Conference on Software Engineering*, Hyderabad, India, 2014, pp. 175-186.

[15] W. Zhou, Y. Zhou, M. Grace, X. Jiang, S. Zou, Fast, scalable detection of "Piggybacked" mobile applications, *Proceedings of the third ACM conference on Data and application security and privacy*, San Antonio, Texas, USA, 2013, pp. 185-196.

[16] Y. L. Tang, X. Y. Li, Z. Q. Zhao, Y. F. Li, Detection method for Android payment cracked application, *Journal of Beijing University of Posts and Telecommunications*, Vol. 44, No. 4, pp. 95-101, August, 2021.

[17] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, C. A. Visaggio, Effectiveness of opcode ngrams for detection of multi family android malware, *2015 10th International Conference on Availability, Reliability and Security*, Toulouse, France, 2015, pp. 333-340.

[18] J. Chen, M. H. Alalfi, T. R. Dean, Y. Zou, Detecting android malware using clone detection, *Journal of Computer Science and Technology*, Vol. 30. No. 5, pp. 942-956, September, 2015.

[19] B. Zhang, W. T. Xiao, X. Xiao, A. K. Sangaiah, W. Z. Zhang, J. 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, September, 2020.

[20] H. Zhang, J. Qin, B. Zhang, H. Yan, J. Guo, F. Gao, A multi-class detection system for Android malicious apps based on color image features, *International Conference on Security and Privacy in New Computing Environments*, Lyngby, Denmark, 2020, pp.186-206.

[21] K. Yu, L. Tan, S. Mumtaz, S. Al-Rubaye, A. Al-Dulaimi, A. K. Bashir, F. A. Khan, Securing Critical Infrastructures: Deep-Learning-based Threat Detection in IIoT, *IEEE Communications Magazine*, Vol. 59, No. 10, pp. 76-82, October, 2021

[22] K. Yu, L. Tan, C. Yang, K. K. R. Choo, A. K. Bashir, J. J. P. C. Rodrigues, T. Sato, A Blockchain-based Shamir's Threshold Cryptography Scheme for Data Protection in Industrial Internet of Things Settings, *IEEE Internet of Things Journal*, Vol. 9, No. 11, pp. 8154-8167, June, 2022.

[23] F. Ding, G. Zhu, Y. Li, X. Zhang, P. K. Atrey, S. Lyu, Anti-Forensics for Face Swapping Videos via Adversarial Training, *IEEE Transactions on Multimedia*, Vol. 24, pp. 3429-3441, 2022.

[24] F. Ding, G. Zhu, M. Alazab, X. Li, K. Yu, Deep-Learning-Empowered Digital Forensics for Edge Consumer Electronics in 5G HetNets, *IEEE Consumer Electronics Magazine*, Vol. 11, No. 2, pp. 42-50, March, 2022.

[25] F. Ding, K. Yu, Z. Gu, X. Li, Y. Shi, Perceptual Enhancement for Autonomous Vehicles: Restoring Visually Degraded Images for Context Prediction via Adversarial Training, *IEEE Transactions on Intelligent Transportation Systems*, Vol. 23, No. 7, pp. 9430-9441, July, 2022.

[26] L. Tan, K. Yu, N. Shi, C. Yang, W. Wei, H. Lu, Towards Secure and Privacy-Preserving Data Sharing for COVID-19 Medical Records: A Blockchain-Empowered Approach, *IEEE Transactions on Network Science and Engineering*, Vol. 9, No. 1, pp. 271-281, January-February, 2022.

Biographies

Xingyu Li received the B.E. degree from Railway Police College, in 2019, and he is currently pursuing the M.S. degree in Henan Polytechnic University. His research interests include Android security and artificial intelligence.

Yongli Tang is Professor in the College of Computer Science and Technology at Henan Polytechnic University. He is a member of the Education and Science Work-ing Committee of Cryptography Society, the Henan comprehensive evaluation expert. His research areas include information security and cryptography algorithms.

Mary Subaja Christo is obtained B.E. in Computer Science and Engineering from St. Xavier's Catholic College of Engineering. Also holds an M.Tech in Information Technology from Sathyabama University, Chennai, and Ph.D. in the specialization of network security in Sathyabama University, Chennai. She is working as an Associate Professor in the Department of Computer Science and Engineering (School of Computing) in SRMIST (SRM UNIVERSITY), Kattankulathur, Chennai, India. Her research interests include but not limited to; computer networks, wireless communications, WSNs, IoT, Machine Learning, Block Chain Technology.

Zongqu Zhao is the teacher of the College of Computer Science and Technology at Henan Polytechnic University. He received the Ph.D. degree from Sichuan University, in 2015. His research interests include cryptography, network security and malicious code analysis.

Ying Li received the B.E. degree from Henan Polytechnic University, in 2020, and she is currently pursuing the M.S. degree in Henan Polytechnic University. Her research interests include modern cryptography and network security.