An Efficient Combined Deep Neural Network based Malware Detection Framework in 5G Environment*

Ning Lu[a,b], Dan Li[b], Wenbo Shi[b,**], Pandi Vijayakumar[c], Francesco Piccialli[d] and Victor Chang[e,**]

[a]School of Computer Science and Technology, Xidian University, Xi'an, China
[b]School of Computer Science and Engineering, Northeastern University, Shenyang, China
[c]Department of Computer Science and Engineering, University College of Engineering Tindivanam, Tindivanam, India
[d]Department of Mathematics and Applications "R.Caccioppoli", University of Naples Federico II, Campania, Italy
[e]School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough, United Kingdom

ARTICLE INFO

Keywords:
5G network
Internet of Things (IoT) networks
android-based applications
malware detection
combined deep neural network

ABSTRACT

While Android smart phones are widely used in 5G networks, third-party application platforms are facing a rapid increase in the screening of applications for market launch. However, on the one hand, due to the receipt of excessive applications for listing, the review requires a lot of time and computing resources. On the other hand, due to the multi-selectivity of Android application features, it is difficult to determine the best feature combination as a criterion for distinguishing benign and malicious software. To address these challenges, this paper proposes an efficient malware detection framework based on deep neural network called DLAMD that can face large-scale samples. An efficient detection framework is designed, which combines the pre-detection phase of rapid detection and the deep detection phase of deep detection. The Android application package (APK) is analyzed in detail, and the permissions and opcodes feature that can distinguish benign from malicious are quickly extracted from the APK. In addition, the random forest with good effect is selected for importance selection and the convolutional neural network (CNN) which automatically extracted the hidden pattern inside features is selected for feature selection, so as to select the feature subset that can distinguish the attributes most. In the experiment, real data from AMD datasets and third-party application download platform are used to verify the high efficiency of the proposed method. The results show that the F1-score index of this method can reach 95.69%.

1. Introduction

The evolution of 5G networks has facilitated the development of the IoT industry, which makes it possible to interconnect everything. Along with convenience to people, cyber-attacks are also increasing rapidly as the IoT is widely used in providing e-commerce, giving online access to healthcare, communication, and billing systems (including malware attacks on the device). Especially Android-based smartphones, as one of the most popular IoT platforms at present, the operating system that has occupied 85% of the global smartphone market [1] has received closer attention from hackers. Taking the Chinese market as an example, 360 Security Brain alone intercepted about 1.809 million new malicious software samples of mobile terminals in 2019, and intercepted about 950 million malicious attacks against mobile phone users nationwide [2]. Due to the openness of the Android system, Android users can download software from the official store, and can also obtain richer applications from third-party sources, which provides attackers with more convenient conditions to implement malicious behavior. With the emergence of malware, users and their devices face serious damages and threats, including information leakage, system damage, and mobile bots [3]. For this reason, it is necessary to ensure that the applications provided by the third-party application market to users are safe and reliable, and the application market needs to strengthen the security audit of many applications before they are put on the market. Therefore, it is urgent to propose an efficient method for detecting Android malware.

Android malware detection aims to classify applications of unknown nature into benign and malicious, which is essentially a classification problem. On this basis, its main goal is to build a feasible classification model that can represent the relationship between APK features and labels (benign and malicious), and then find the optimal solution of the model. According to the different ways of acquiring features, these methods are divided into source package-based features, runtime-based features and hybrid-based features methods [4, 5, 6]. The source package-based features analysis method usually obtains feature information by opening APK and analyzing the files obtained [7, 8, 9, 10]. Runtime-based features analysis utilizes runtime or simulated runtime characteristics [11, 12]. As a result, source package-based analysis has advantages over runtime-based analysis in terms of time and computational consumption. In the face of massive samples, it is necessary to reduce the detection time as much as possible in order to improve efficiency. Therefore, the detection method based on source package analysis is more appropriate. At the same time, in order to ensure

*This work supported by the National Natural Science Foundation of China (Nos.62072093, 62072092, 61601107and U1708262); the China Postdoctoral Science Foundation (No.2019M653568); the Fundamental Research Funds for the Central Universities (No.N2023020); the Natural Science Foundation of Hebei Province of China (No.F2020501013). Dan Li is the co-first author of this work.

**Corresponding author.
E-mail address: Victor Chang (ic.victor.chang@gmail.com), Wenbo Shi (shiwb@neuq.edu.cn)
ORCID(s):

the validity of the classification results, the selected features must be representative. In the source package-based detection method, a Dalvik virtual machine instruction, which can analyze the operation behavior of applications from the bottom. It is designed by Google and is called opcode. The execution of each opcode instruction results in the corresponding operation in the internal register. So, this is a very representative sample feature. Due to the increase in the number of opcodes in applications, the complexity of selecting key opcode features increases and the timing sequence of opcodes is lost. Further, it will be difficult to ensure the speed and accuracy of detection by using opcodes as features to detect a good deal of application samples. Therefore, detection based on opcode faces two challenges in terms of ensuring accuracy and improving detection speed.

Challenge 1: the high cost of detection. A wide range of samples to be tested will inevitably lead to an increase in the cost of calculation and time, which puts forward higher requirements for detection methods. The process of the existing runtime-based features extraction method is more complicated than that of the source-package-based features extraction method because it needs to monitor the runtime state. This way of working causes the method to take longer and more computing resources [13, 14, 15, 16]. At the same time, this method has the disadvantage of low code coverage, making it difficult to ensure that all execution routes of the application are covered [17]. The detection methods that hybrid-based features also has the defect of runtime-based features extraction. Because it means extracting both runtime and source-package features, this doubles the difficulty of the extraction process, making time and computational resources more expensive [18, 19]. However, some deep learning analysis methods based on source package features reduce the efficiency due to the design of overly complex neural networks [20].These methods are especially not suitable for the situation with huge samples. Therefore, how to efficiently solve a huge number of software detection problems is the first technical challenge.

Challenge 2: the problem of selecting features. The selection of feature subsets will affect the results of the detection method, and different feature inputs will lead to differences in results. Android applications' access to system resources and user permissions is regulated by the permission system, and different permissions correspond to different behaviors, so important permissions need to be filtered out. Removing irrelevant features and redundant features without causing loss of important information can make the classification results more accurate. The highest permissions required for clean and malicious applications are listed, but only considering the frequency count of a single permission cannot provide better malware detection performance [21]. However, using the Term Frequency-inverse Document Frequency weighting method to calculate the permission value of each permission is not comprehensive [22], and sometimes important permissions may not appear many times. The same problem occurs in the selection of opcode features[23].Only counting the distribution of opcodes will not only lead to the situation that the frequency information cannot be fully expressed, but also the problem of missing the order information of the opcodes [24]. To make matters worse, as the value of n in n-gram increases, the huge overhead caused by the explosive growth of the parameter space makes this method unable to handle longer sequences [25]. Therefore, how to choose the feature that best reflects the difference is the second technical challenge.

Aiming at the above challenges, a phased detection framework based on the combination of pre-detection method and deep learning method is proposed. It is designed to reduce the time and computing cost of detection in large-scale applications as much as possible. In the pre-detection phase, the important permission features filtered by the random forest are combined with the back propagation (BP) network classifier for rapid screening. Pre-detection quickly divides the samples to be detected into malicious and suspicious, which can greatly save time and improve detection efficiency. It is ensured that only suspicious samples need to enter the second phase of deep detection, and the overall detection efficiency is improved by reducing the number of samples entering the deep detection phase. In the deep detection phase, after the enhancement of the CNN and the extraction of key local information, the opcode sequence features are classified by using the long short-term memory network (LSTM) which is good at processing the temporal information. The purpose of this is to ensure that CNN automatically extracts important opcodes features and obtains relative position information of the sequences.The problem of manually selecting the opcodes is solved, the relative position information lost due to the truncated sequence is retained, and more accurate features are selected for LSTM.

1.1. The goal and contributions of the paper

The goal of this paper is to provide a more efficient malware detection framework for third-party markets of 5G-supported Android applications, which can reduce the adverse effects of malware introduced into 5G networks. The contributions are summarized as follows.

- For Android malware detection, an efficient detection framework based on hybrid deep neural networks is proposed, which can quickly and effectively identify malware and benign software.
- A pre-detection method of BP network based on permission features and a joint deep detection method of CNN and LSTM based on opcodes features are designed. The pre-detection adopts the permission features extracted rapidly and selects important features in combination with the random forest. The results of pre-detection can reduce the number of samples to be tested in the deep detection phase and improve the detection efficiency of large samples. In the deep detection phase, a feature selection method that automatically extracts the hidden mode of the opcodes is designed. It employs CNN to extract local key information to obtain the key opcodes and the timing sequence

characteristics of the relative position information between the key opcodes, which improves the accuracy of detection.

- Experimental results of real data sets demonstrate the high efficiency of proposed method. Experiments were carried out on the pre-detection phase and the deep detection phase respectively, and the accuracy reached more than 93%. Especially in the experiment of the whole detection framework, the accuracy is improved by about 2%-3% compared with a single stage. In comparison experiments with traditional machine learning algorithms, the accuracy is improved by about 10%-20%.

The organization of the rest of entire paper is shown below. Section 2 is a description of android software detection problems. The Section 3 expounds the DLAMD method, including the framework and special details and Section 4 conducts experiments to estimate the composite indicator of DLAMD. Section 5 and Section 6 are related work and conclusions respectively.

2. Problem Statement

2.1. Problem Definition

Dalvik is a virtual machine designed by Google for the Android platform. There are Dalvik instructions in the smali file obtained by decompilation. Opcode is derived from Dalvik instructions in smali files, such as "iput-object", "invoke-direct" and "return-void". These opcodes all represent different operations and contain sequence information in the order of appearance. An APK can be represented as a sequence of opcodes composed of opcodes. At this time, the problem of malware detection is transformed to the judgment of benign and malicious applications based on opcodes features by analyzing the different manifestations of corresponding instruction sequences on their respective labels.

2.2. Problem Decomposition

There are hundreds of opcodes, and some of them will appear repeatedly in different positions. Since only extracting the frequency of the opcodes cannot fully express all the information, and truncating the opcode sequence will lose the order information. Therefore, it is a relatively good choice to consider using opcode sequences to represent APK behavior. However, excessive length of opcodes sequence will lead to the introduction of redundant information and reduce the detection efficiency. By now, the problem of malware detection based on opcodes features can be decomposed into two subproblems:

1) The inefficiency caused by length dependence based on opcodes features. First, the permissions that can be obtained more quickly are obtained, then the key features are sorted by random forest, and finally the BP network is used for rapid pre-detection. The detailed process of pre-detection is described in Section 3.2.

2) Key feature selection based on opcodes features. After a quick pre-detection phase, the nature of the part of the sample judged to be malware is determined. The other part of the software that is classified as suspicious will enter the deep detection phase to obtain its opcodes features and utilize CNN to extract key opcode features and relative position information to provide more accurate judgments.

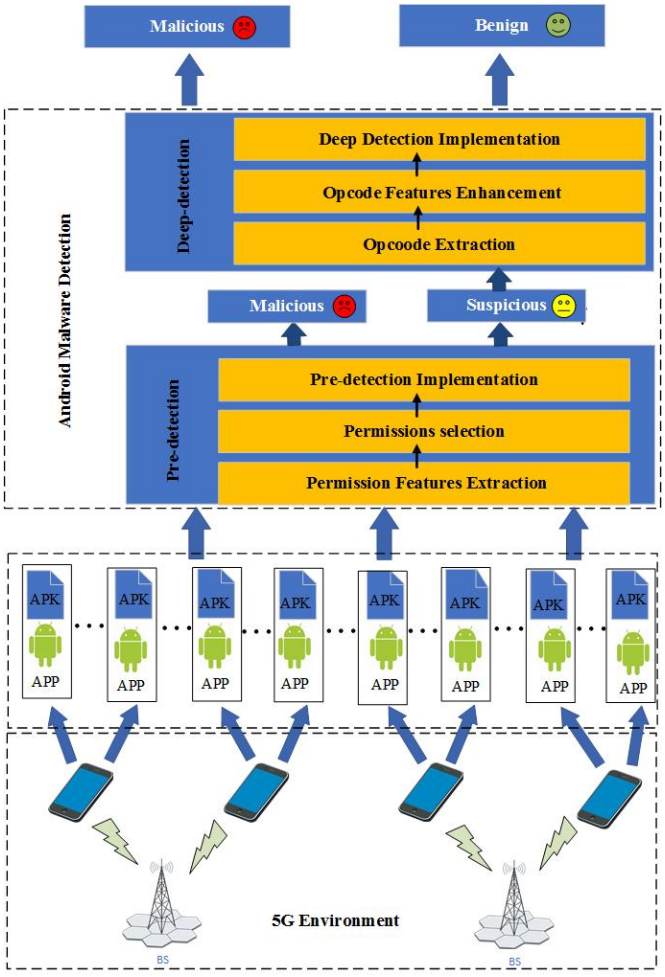

Figure 1: The framework of the malware detection method

3. Methodology

In this section, the structure of the DLAMD is first shown to have a global view. Then, we concretely represent how to use the permission feature for pre-detection, as well as the rules for selecting permission feature. For explaining the principle of the deep detection network clearly, the structure and parameter settings of the combined deep learning network are depicted.

3.1. Framework

The architecture of DLAMD is shown in Figure. 1, and DLAMD is separated into two parts: the pre-detection and

the deep detection. The APK to be detected under 5G network is the input of the framework. Two typical features of APK are used in the DLAMD framework: permissions and opcodes. The input of the pre-detection phase in DLAMD is the features vector containing permissions information. Then, according to the result of the pre-detection, applications are divided into suspicious and malicious. Finally, the opcodes feature vector of the suspicious softwares will be the input of the deep detection in the classification model.

The first phase is the pre-detection, which is used for preliminary rapid detection of various Android applications and is intended to exclude some malware with obvious characteristics. It greatly reduces the time and computing cost of deep detection and improves the detection efficiency of the whole framework. Pre-detection requires preliminary screening of a wide range of applications, so its characteristics such as fast detection speed and easy feature extraction are necessary. Pre-detection is designed for two considerations. On the one hand, the permission mechanism, as the core of the Android system security mechanism, can remind users of the application permissions required by applications and help users make decisions to avoid data leakage or malicious attacks. At the same time, permissions required by the application must be declared in the Androidmanifest.xml file, which is easy to obtain. On the other hand, the random forest algorithm and BP neural network are both lightweight detection methods, which can meet the requirements of fast detection speed required for pre-detection.

The second phase is the deep detection. Applications that are judged to be suspicious in the pre-detection will serve as inputs to this phase.This can not only reduce the pressure of deep detection but also improve the accuracy of detection. This phase mainly utilizes the CNN and LSTM to detect the applications by using opcodes feature. It avoids manual selection of feature, and can also take care of the time sequence of the opcodes and improve the detection efficiency.

3.2. Pre-detection phase

The pre-detection is specially designed to avoid time overhead and calculation cost in consideration of actual requirements. Here, the pre-detection is described in detail in three subsections: rapid extraction of features, selection of permission features, and implementation of preliminary classification.

3.2.1. Extraction of Permission Features

Here is a detailed description of the permissions features extraction process. To boost the detection speed of the pre-detection phase,instead of the currently widely used open source tools such as Androguard [26], we chose the more direct parsing method, which only gets permission characteristics from the Androidmanifest.xml file. This can effectively speed up the extraction.

Since the AndroidManifest.xml file is usually stored as an encrypted binary file, it needs to be further parsed to extract the permissions features. This file consists of four parts: Header (including the magic number and file size of the file); String Chunk (the string resource pool); ResourceId Chunk (the system resource id information); XmlContent Chunk (the specific information in the manifest file). All the permissions information we need is stored in String Chunk, so we only need to parse this part to get the required information, and it can also reduce the time taken for extraction. First, we need to locate this part, then read the binary information byte by byte, finally convert them into string information that we can understand, and extract the permission information.

For all the samples of the experiment, we extracted a total of about 330 types of permissions features as feature dictionaries for detection model training and classification. For each sample, permissions feature extracted corresponds to the selected feature dictionary word vectorization, that is, the position in the feature dictionary is marked as 1 and the other positions are marked as 0. The vectorization of the bag of words is used to indicate the extracted permission feature.

3.2.2. Selection of Permission Features

In order to better detect malicious applications, improve the generalization ability of detection of unknown applications, and reduce overfitting, it is necessary to obtain the importance ranking of features through the process of selecting, deleting, and merging all permission features.Here's how to use the random forest to select permissions obtained in the previous section.

A permission feature selection method based on random forest and Gini coefficient is established. Consider the selection strategy of permissions features from two perspectives, one is whether the permission function is different. If the selected features are not different in the samples of the two labels, then this permission is not helpful for analysis and detection applications. Another point is whether the selection permissions are related to the classification of the application. More relevant functions are considered more important for application detection. Therefore, it is necessary to choose an appropriate, correct and robust feature selection method. Because random forest has high accuracy and robustness, and is easy to use, the pre-detection phase adopts random forest-based methods, and uses the Gini coefficient to measure the important coefficient of each permission feature, and then determines the advantages and disadvantages of the feature according to the value of the coefficient.

Random forest based on subsets belong to the training data set and train multiple decision trees. In the decision tree building step, permissions are used as a segmentation feature. To ensure that the selected permission feature is the best segmentation feature in all feature subsets, it is necessary to ensure that the permission feature has sufficient importance compared to other features, that is, the degree of importance is higher than other features. Using random forest to select the permission features of an application is to make a decision based on the degree of contribution of each permission feature on each decision tree, then the average value is taken, and finally the permission features are selected based on the contribution value. In other words, we need to calculate the contribution of each permission feature,

and then compare them one by one according to the threshold set in advance. If it is greater than the threshold, it is the selected permission feature.

Meanwhile, the method which aims at calculating the feature contribution is particularly critical. Gini coefficient is a widely used measurement method similar to information entropy and information gain, which can be used to represent the impureness of data. The Gini coefficient is a number between 0 and 1. When it is closer to 1, it means that the degree of inequality is higher, and vice versa. The best segmentation permission feature is usually selected according to the degree of impureness of the child nodes after segmentation. The lower the degree of impureness, the more inclined the class distribution is. Generally, the smallest Gini coefficient is used as the basis for division. The Gini function of the probability distribution is defined as:

$$Q(r) = \sum_{s=1}^{S} r_s(1 - r_s) = 1 - \sum_{s=1}^{S} r_s^2. \tag{1}$$

Among them, S indicates S categories, and r_s and the probability that a sample is assigned to the s categories. Then we assume that the data set V is classified into sub-dataset V_1 and V_2, based on the condition that whether feature U is equal to a certain value u. In this case, the Gini of V is:

$$Q(V, U) = \frac{|V_1|}{|V|} Q(V_1) + \frac{|V_2|}{|V|} Q(V_2), \tag{2}$$

$Q(V, U)$ is the uncertainty of V divided by feature $U = u$. The larger the Gini coefficient, the greater the uncertainty of the sample set.The Gini coefficient and the important value score of each permission feature are calculated respectively, and the most important permission feature is selected for detecting malicious application functional programs.

Therefore, on the premise of obtaining the importance of features, the steps of feature selection for permissions are as follows. The first is to calculate the importance of Android features and sort them in descending order. The second is to set the proportion of features to be deleted, and then obtain a new permission feature set according to the calculated importance of permission features.The third is to repeat the above operation until a threshold number of permission features are left. So far, the feature selection work is completed. Figure. 2 shows the top 20 most important Android application permission features, and we can find that the features of permissions such as READ_PRIVILEGED_PHONE_STATE and WRITE_SYNC_SETTINGS are more important. By constructing the random forest using permissions features and calculating the Gini coefficient, it is possible to further normalize and obtain the important value scores of each permission. Among the extracted 330 permission features, the most important 45 are selected as the key features for detection.

3.2.3. Implementation of Pre-detection

The optimized design of BP network structure is also carried out to ensure that the classification index can be improved as much as possible while the rapid detection is ensured.The classification method based on permission feature and BP neural network is introduced in detail.

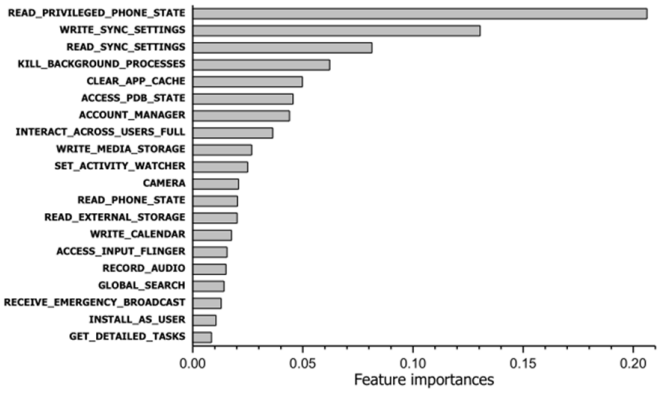

Figure 2: Top 20 permissions distribution

The neural network is a simplified artificial intelligence algorithm modeled after biological neurons. BP neural network is a classic network structure in artificial neural networks. The BP neural network model has good non-linear mapping capabilities, which can solve more complex detection problems, and has a strong self-learning ability for feature information. But the BP neural network is not perfect and has certain limitations. When designing a structural model based on BP neural network, the following optimizations were performed in this paper: adjusting model parameters and network structure, optimizing the learning strategy and loss function of the network model, accelerating the model's convergence speed, and enhancing the classification accuracy of the model. The BP network structure usually includes the input layer, hidden layer, and output layer. Forward propagation calculation and back propagation training are two components of neural network structure. Based on the above rules, the network model designed in this section is an input layer, an output layer and four hidden layers. All neuron nodes use RELU as the activation function. To prevent model overfitting, dropout is introduced into the network and the random inactivation probability is set at 0.2.

The above contents have described the basic network structure of pre-detection. Based on the above network structure, the output value of the network model can be calculated forward. It is also necessary to use the final output of forwarding propagation to calculate the partial derivative of the error, to obtain the loss function, to evaluate and back propagate to tune the network parameters. To enhance training model's convergence speed and detection results of the algorithm model, this paper chooses Adam optimization algorithm and cross-entropy loss function.Adam adaptive estimation algorithm not only guarantees effective on sparse gradients but also has good performance on non-steady-state and online problems. This algorithm is helpful for the efficient calculation of the entire algorithm model, while requiring less memory, and is suitable for non-steady-state targets and high noise or sparse matrix problems. With cross-

entropy as the loss function, the similarity between the distribution of real labels and the distribution of model prediction labels can be predicted to optimize the network structure 's parameters. Besides, the adopted loss function avoids the gradient dispersion problem to a certain extent and accelerates the convergence rate of the model.

3.3. Deep Detection Phase

The deep detection method based on CNN and LSTM using opcode is described in detail. As shown in Figure. 3, we divide the detection process of this phase into three modules: opcode extraction, feature enhancement and selection, and deep detection.

3.3.1. Opcode Extraction

Here is about the process of extracting the opcode sequence features of the input for deep detection, and the generation process of the opcode sequences conversion into the data format that can be input into the detection network.

Extracting the opcode sequences from the APK file mainly includes the following three important steps [27]. Firstly, the smali file is obtained from the decompiled APK file. Then, we extract only the Dalvik opcode from the file and discard the operands. Finally, an opcode sequence file is obtained from the Android opcode constant list. The decompiler Apktool [28] used in this section is an open source tool that is often used for reverse engineering. It can decompile Android programs and rebuild them after modifications. After decompiling the samples using Apktool, the files with the suffix smali is extracted, which is the core of the application samples at runtime. This file contains all the method functions required for the execution of the samples, and we only need to extract and record the opcodes representing the methods. Finally, we map these opcode files to the list of android Dalvik instruction set to get the numeric opcode sequence file. Its concrete representation is the hexadecimal representation of the 0-255 sequence of numbers, representing the corresponding opcode.

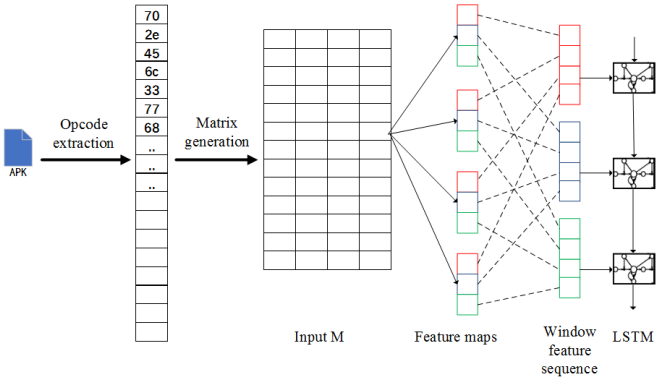

Figure 3: Deep Detection Structure

After the previous step, we obtain the opcode sequences. Because of data format problems, these sequences cannot be directly used as input data for CNN, and further transformations are required to make them into the matching data format. All opcodes are converted into one-hot vectors, that is, vectors with index bits of 1 and the remaining bits being 0. In this way, the opcode sequence can be transformed into an opcode matrix which can be input into the CNN. However, a new question appeared at this time, that is, the opcode matrix is too sparse and the dimension is too high, which will cause the efficiency of the feature extraction of the convolutional layer to become very low. Therefore, it is necessary to centralize the feature information to improve calculation efficiency. We multiply the opcode matrix by a randomly initialized embedding matrix, which concentrates the feature information, thereby obtaining the final form of the input convolutional neural network M.

3.3.2. Feature Enhancement and Selection

Given the continuity of the feature code sequence, this paper chooses to use a CNN model that is used for classification. The reason is that this model can amplify the correlation between opcode features, which can select and obtain key features accurately.It is described how CNN enhances and selects the features of opcode sequences while ensuring that sequence information is obtained.

The model designed here adopts some design advantages of Text-CNN [29] model to maintain the order information of opcode sequences. In order to match the features of the opcode sequences, multiple filters with a width equal to the width of the feature matrix input of the embedding layer are used in the convolutional network. The convolution operation is performed to extract new features under the condition that the width of each filter is kept constant. Multiple filter convolution operations yield multiple feature maps. The formula is defined as follows:

$$c_i = Relu(Conv(M, w_i) + b_i), \tag{3}$$

where, the activation function is $RELU$. The convolution operation on the input matrix M is represented by $Conv$. The weight of j-th filter is $w_j \in \mathbb{R}(h \times g)$, and the bias is b_j. $n - h + 1$ convolution operations is implemented in the j-th filter in the entire input matrix, and the corresponding feature map in obtained. Summary of the different feature maps extracted for all filters is as follows.

$$C = [c_1|c_2|...|c_p]^T. \tag{4}$$

The convolutional network layer mainly obtains the output matrix C by performing a convolution calculation on the opcode sequence represented in the input matrix M. After convolution, the result matrix needs to be input to the pooling layer for further feature extraction. Generally speaking, max-pooling and k-max pooling [30] is applied to the feature mapping after convolution to select important one or k important features. However, the input of LSTM is a specified sequence input, and pooling will destroy the structure of the sequence due to the discontinuity of the selected features. Therefore, considering that LSTM is connected behind CNN, pooling is not applied after convolution operation.At this point, the process of extracting the feature information of the convolutional neural network module is all completed.

3.3.3. Deep Detection

The function of deep detection is to use LSTM network to detect and analyze the feature vectors of the output of CNN.LSTM network is a kind of time-delay neural network, which is more suitable for time series operation behavior.The unit structure of LSTM is more complex, and it has a better training effect on sequence data.

The core operation is mainly calculated and controlled by three "gates", and the cell status is selected to be updated by using the information at the previous moment and the current input feature information. The specific LSTM model is shown in Figure. 4. $X = x_1, x_2, \ldots, x_n$ is the input sequence of the LSTM model, namely the intermediate feature vector of the output of the convolutional network, and $H = h_1, h_2, \ldots, h_n$ be the LSTM model calculates vector sequence of hidden layer nodes, then calculate $Y = y_1, y_2, \ldots, y_n$, the corresponding output vector sequence. The control and calculation of transmission state is implemented using a set of " gates " in each time step of the LSTM, which remember feature information that needs to be memorized for a long time, and forget unimportant feature information. The LSTM model mainly determines and calculates the transition between c_t (the time step cell state) and h_t (the current hidden state) through $h(t-1)$ (the hidden state) of the preceding time step and the input vector x_t of the current time step, and three calculation control " gates ", namely " output " o_t, " forget " f_t, " input " i_t. The calculation conversion process of LSTM is specifically defined as follows.

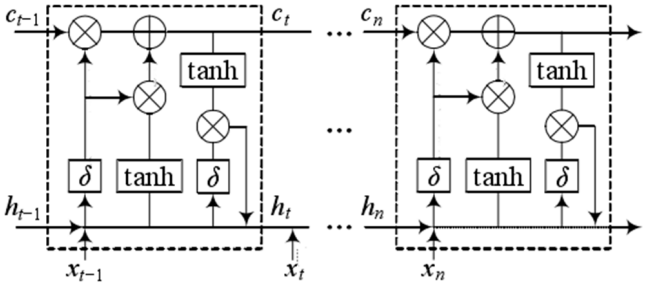

Figure 4: Long Short-Term Memory Network

First of all, in the LSTM model, it is necessary to determine which opcode feature information will be discarded out of the cell state c_t at time t by the forget gate f_t. The formula is defined as follows:

$$f_t = \sigma(W_f \cdot [h(t-1), x_f] + b_f), \tag{5}$$

where σ represents the *sigmoid* function, which can map a real number to the range (0,1). At this time, the *sigmoid* method will read the hidden state h_t during $t-1$ time and the input sequence x_t , and output a value to each opcode characteristic information in the cell state $c(t-1)$ at time $t-1$ in order to decide whether the information is retained.

Secondly, how much new opcode feature information needs to be input into the cell state at time t is determined by the LSTM model. In first part, the decision to decide which information is updated lies in the input gate i_t. Here is the formula definition:

$$i_t = \sigma(W_i \cdot [h(t-1), x_t] + b_i), \tag{6}$$

where the effect of the sigmoid function is similar to that of the forget gate. In the second part, l_t decides which opcode characteristic information will be the candidate update content. The formula is defined as follows:

$$l_t = \tanh(W_l \cdot [h(t-1), x_t] + b_l), \tag{7}$$

where *tanh* is a hyperbolic tangent function, which can map real numbers to activation functions in the interval between (-1,1). By combining these two steps, the cell status can be updated.

Again, the LSTM model will update the cell state at time t based on the forget gate f_t and the input gate i_t. The formula is defined as follows:

$$c_t = f_t \odot c(t-1) + i_t \odot b_o, \tag{8}$$

where $\odot$ represents the multiplication of the corresponding elements of the operation matrix and requires proof of the same type. Finally, it is up to the LSTM model to determine which part of the cell state at time t can be delivered from the o_t output. The formula is defined as follows:

$$o_t = \sigma(W_o \cdot [h(t-1), x_o] + b_o), \tag{9}$$

where the effect of the sigmoid function is also similar to that in the above formula. At the same time, the hidden state h_t at time t need to be updated to facilitate participation in the calculation in the next cell. The formula is defined as follows:

$$h_t = o_t \odot \tanh(c_t). \tag{10}$$

Through the continuous cycle of forgetting, input and output of the cells, the LSTM model can fully analyze the time sequence of the opcode sequences feature. The input of the fully connected layer is the output of LSTM model, and then this layer outputs the results. Finally, the classification output layer chooses to use the softmax model to judge the malicious and non-malicious behavior of the application. The function realized by this layer is to perform mathematical calculations on the weights of the features of the foregoing layer, to gain an accurate mixture of the elements, and output the result of the corresponding target.

4. EXPERIMENTAL STUDIES

This section verifies the optimality of the DLAMD through experiments. First, the data set is used to validate the performance of the pre-detection and deep detection. More importantly, the comparison experiment between the deep detection algorithm and the overall framework will also be conducted, which can verify the necessity of stratification and pre-detection. Other machine learning models are then used to compare with the DLAMD presented in this paper.

4.1. Experimental Setup

The data set used in the experiment, the evaluation index of android software detection results, and the description of the experimental environment are shown below.

Data set description: The dataset in this paper contains positive and negative samples, namely malicious applications and benign applications. Among them, the malicious data set mainly comes from VirusTotal [31] and the public data set Drebin. Virustotal is a website that provides free suspicious file analysis services with a regularly updated virus definition library. The benign software used in this paper is collected from third-party application platforms in China, such as Tencent App Store and 360 App Market. Therefore, all applications have been effectively tested. 2000 malicious APKs and 2000 benign APKs were extracted for experimental analysis. In order to further verify the performance of DLAMD in large-scale data sets, a data set containing 7800 applications including 3900 benign and 3900 malware is introduced. We divide the data set as follows, 90% of which are the training and validation sets, and the remaining 10% as the test set.

Evaluation criteria: The following five parameters are used as evaluation indicators in this article to quantitatively evaluate the effectiveness of the detection model network. The calculation formula is shown below:

$$Accuracy = \frac{RC + RA}{RC + RA + FC + FA}, \tag{11}$$

$$Recall = \frac{RC}{RC + RA}, \tag{12}$$

$$Precision = \frac{RC}{RC + FC}, \tag{13}$$

$$F1 - Score = \frac{2RC}{2RC + FC + FA}, \tag{14}$$

$$AUC = \frac{\sum_{ins_i \in positiveclass} rank_{ins_i} - \frac{M \times (1+M)}{2}}{M \times N}, \tag{15}$$

where RC and RA respectively represent the number of real correct and real negative, while FC and FA are the number of fake correct and fake negative respectively.

Experiments environment: The experimental environment is built based on the Python language, and the experimental platform mainly uses open source packages or tools such as pandas, sklearn, matplobtlib, and numpy. The tools for extracting features of opcodes of Android applications are mainly baksmali and apktool. The framework for deep learning is built with TensorFlow, cuda and cudnn. The training and testing stages of the model are all completed on the GPU server, and the main GPUs used are Tesla K80 GPU and GTX 1080 T1.

4.2. Result analysis

The experimental results of the data set will be analyzed in detail here. To reflect the accuracy of the overall framework, the pre-detection phase and the deep-detection phase are tested separately. Finally, the detection performance of the separate deep detection and the overall detection framework is compared.

4.2.1. Pre-detection

The results and performance of the pre-detection are experimentally verified. From the perspective of model training and classification results, experimental demonstrations are carried out.

The data sets are trained the network detection model, and the evaluation indices of training and verification are recorded respectively. First enter the data set network detection model for training, and record the loss and accuracy of training and verification. As shown in Figure. 5, with the increase of experimental steps, the loss of the model gradually decreases, and the training result of the detection model is judged. We use the loss function to compare the gap between the predicted value and the actual value of the detection network to help optimize the detection network. At the same time, it can quantitatively show the effect of pre-dection in distinguishing the properties of the software. The value of this loss function is expected to be extremely minimal, which means that the classification effect of the model meets the expectation. Meanwhile,it can quantitatively display the detection effect of the detection model. The smaller the loss function value is, the better the detection effect of the obtained detection model will be.

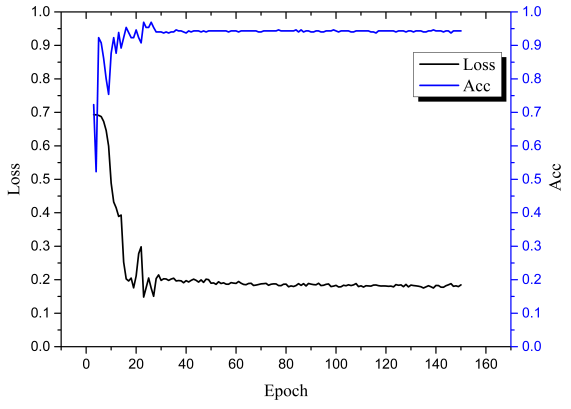

Figure 5: Loss and accuracy of pre-detection's training

After the detection model is trained, it needs to be tested and verified on the test set to more objectively and accurately measure the detection effect of the detection model. The experimental effect of the pre-detection is verified by comparison with related algorithms, such as support vector machine (SVM), logistic regression (LR), random forest (RF), decision tree (DT), multinomialNB (MNB) and multilayerperceptron (MLP).The results of the pre-detection are shown on two data sets. Pre-D400 and Pre-D7800 represent the test results of the pre-detection on 400 and 7800 data samples after training and verification, respectively. Through a series of experimental tests on the test set, multiple test results such as accuracy, recall, accuracy , F1-score and AUC are obtained. As shown in Figure 6, on the 400 sample data set, the accuracy of the model test results obtained by the pre-detection is about 93%. At the same time, the precision and recall indica-

Table 1
Pre-detection's training and detection time results

Number of sample	Training time	Number of sample	Detection time
APK files(3600)	39.06s	APK file(1)	0.0275s

Table 2
Two types of features extraction time

Number of sample	Feature type	Extraction time
APK files(7788)	permission	172.8625s
APK file(7795)	opcode	5h07min18s + 7h12min19s

tors have reached more than 91%. In the 7800 large samples dataset, although the index has declined slightly because of the dramatic increase in the number of samples, it still maintains 84.12% on the comprehensive measurement index of F1-score, which is 5.57% higher than the 78.55% of MNB.

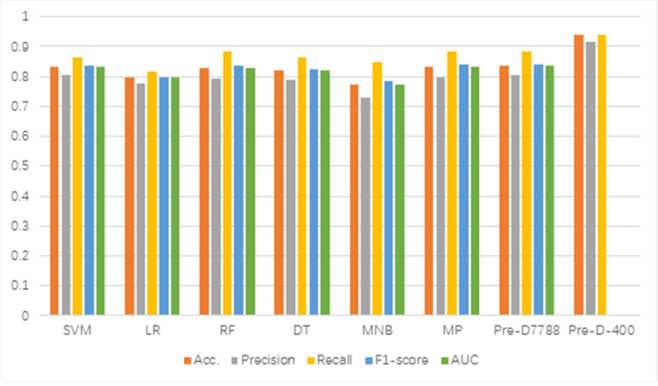

Figure 6: comparison results of pre-detection's and other methods

It can be seen from the above results that the pre-detection model proposed in this paper has good detection results on both test data set and training data set. In view of the fact that the main application scenario proposed in the pre-detection phase is to solve the preliminary detection of many unknown applications in the application market.Therefore, the detection speed of the detection method is more critical, that is, the training time of the model and the detection time of the unknown Android application are also particularly critical. Based on the above considerations, this section also designs corresponding experiments and records the time consumed by the experiment. Table 1 shows specific information.

The training time in Table 1 is the time used for training 3600 samples after the permission features have been extracted, including feature selection time and model training time. The detection time of the pre-detection is the detection time of an APK file, including the sum of the extraction features time, feature selection, and classification time. In order to further compare the time saved by the pre-detection for the overall detection framework, this paper compares the extraction time of permission features and opcode features.Because the opcode feature extraction took too long, the batch of samples was processed twice.It can be seen from Table 2 that the extraction efficiency of permission features is much higher than the extraction speed of opcode features, which can be said to be an order of magnitude time surpass.Here, the advantages of pre-detection can be clearly demonstrated. Pre-detection can quickly screen out a part of malicious samples for deep detection, reducing the number of samples that need to extract opcode features and perform deep detection, thereby improving detection efficiency.

Therefore, the pre-detection which is proposed in this paper has absolute advantages in ensuring the detection effect and detection speed.

4.2.2. Deep Detection

It is necessary to verify the effectiveness of the deep detection phase. To verify the detection effect of the proposed detection model based on CNN and LSTM.

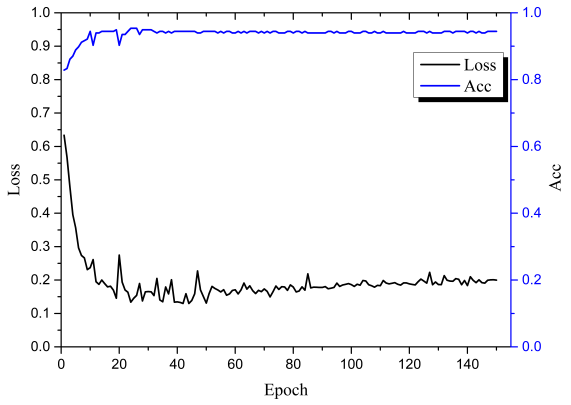

Figure 7: Loss and accuracy of Deep detection's training

In the deep detection, we propose a detection model based on CNN and LSTM. During the experiment, the CNN and the LSTM network are simultaneously trained as an overall model.It is necessary to adjust the hyperparameters such as convolution filters and the number of network layers to make the model in a relatively optimal state, thereby improving various evaluation indicators. As Figure 7 shows , in the process of increasing the epoch value of the abscissa to 20, the loss and accuracy respectively maintained gradually decreasing and increasing trend. After reaching 30 epoch, the curve was basically stable. At this time, the model gradually converges and stabilizes, indicating that the training effect is as expected.

In Figure. 7, the accuracy of the training model and the accuracy of the validation set improves gradually as the value of the horizontal axis slowly increases, and the detection accuracy reaches more than 91% after about 20 epochs. While the time of training iterations is increasing, the training loss and verification loss of the model gradually decrease, and finally stabilize to about 0.25, which proves that the model

Table 3
Experimental results of whole detection framework and separate deep detection

Model	Accuracy	Precision	Recall	F1-Score	AUC
Detection Framework	0.9583	0.9524	0.9615	0.9569	0.9584
Deep Detection	0.9375	0.9457	0.9385	0.9421	0.9374

gradually converges. Figure. 7 shows that the detection model has good performance in the training phase, and can gradually reduces fluctuations, and eventually tends to converge.

Like the pre-detection, the deep detection needs to be tested and verified on the test set after training, in order to measure the detection effect of the model more objectively and accurately.Through a series of experimental tests, the accuracy, precision, recall and other test results were obtained.At the same time, the DLAMD is compared with a separate deep detection model to further verify the advantages of the detection framework.In order to follow the principle of experimental fairness, the same data set is used in the comparison experiment, and the proportion of training set, verification set and test set is consistent.

As shown in TABLE 3, the proposed DLAMD and the separate deep detection have achieved good detection results, of which the DLAMD has obtained an accuracy of 95.83% and an precision of 95.24%. Compared with using only the deep detection model, the DLAMD has significantly enhanced in various evaluation indicators. The F1-score has increased to 95.69%, which is 1.48% higher than before. The AUC increased to 95.84%, which is 2.1% higher than before. Other evaluation indicators have been improved to varying degrees. It can be seen that the detection framework which is proposed in this paper can enhance the performance of Android application software detection.

4.3. Performance Comparison of DLAMD and Machine Learning Methods

The experimental verification of the performance indicators of the entire detection model is crucial.And the comparative experiment between DLAMD and machine learning methods is described in detail in the following content.

In order to further verify the detection performance of the overall detection framework proposed in this paper, a set of comparative experiments are designed to illustrate.Some machine learning algorithms with better classification effects are selected for simulation.The same data set is compared and analyzed to ensure the fairness of the detection framework applied to Android applications.The specific experimental results are shown in Table 4. In order to more intuitively evaluate the framework proposed in this paper, Decision tree, logistic regression, naive Bayes, random forest and multilayer perceptron are selected to carry out the comparison experiment. All comparative experiments employ the same dataset, with 2000 malware and 2000 benign softwares. The training and testing datasets have the same division ratio. The experiment results are reported in five evaluating indexes mentioned above. The results of comparative experiments confirm that the accuracy of the proposed DLAMD exceeds other detection algorithms by about 5%-10%, and the same increase is also seen in precision and recall. Stable comprehensive evaluation indexes of F1-Score and AUC can be increased by 10% -20%. The detection framework proposed in this paper is more optimized and stable than the better-performing random forest.This proves that the detection framework in this paper is more robust and improves the detection effect. The detection framework which is proposed in this paper is more optimized and stable than random forest detection.

Specifically, methods such as logistic regression and naive Bayes, as shown in Table 4, have large fluctuations in recall and precision, and low detection accuracy.This is due to the difficulty of opcodes feature selection and sequence information, which makes the detection algorithm unstable, and it is difficult to understand the internal laws of malicious applications.The DLAMD in this paper utilizes an efficient deep learning detection framework, adopts the pre-detection to filter out part of applications, and then employs the CNN to initially extract the features of opcode sequences, and finally analyzes the sequence relationship between the opcode sequences through LSTM.The detection framework is more comprehensive and fully analyzes the local opcode information. Therefore, the experimental results are relatively better, and the detection efficiency is improved while ensuring performance indicators as much as possible.

5. Related Work

In the 5G era, the terminal security of the Internet of Everything is particularly important, especially for the application security of android phones with huge number of users, which requires more extensive research into new technologies [32, 33, 34, 35].In general, runtime feature and hybrid feature detection methods can also achieve good detection results. Because they both need to obtain the data and behavior of the application at runtime, they need to be simulated in a sandbox or run in a real device, resulting in high computational cost and time consuming.Obviously, these two analysis methods are not suitable for the large-scale detection of the third-party application market under the 5G environment concerned, which requires an efficient and accurate detection method. Here, the focus of this paper is the detection method based on the features of the source package.

In terms of run-time feature analysis methods and hybrid feature methods, Zhang W *et al.* [36] combined runtime features and permissions of system categories, analyzed the trigger points of malicious code and established a birthmark library. They used DAMBA to design a hybrid detection method to reduce the range of possible malware families to obtain better accuracy. The scheme in [37] incorporated network traffic and system permissions, and proposed a hybrid detection model called NTPDroid. They used the FP-Growth algorithm to develop the proposed model and

Table 4
Comparison of different detection methods

Methods	Benign/Malware	Acc.	Pre.	Recall	F1-Score	AUC
Logistic Regression	2000/2000	0.7502	0.9811	0.5203	0.6795	0.7489
Random Forest	2000/2000	0.8500	0.8889	0.8001	0.8422	0.8501
Decision Tree	2000/2000	0.8000	0.8750	0.7000	0.7778	0.8001
Naive Bayes	2000/2000	0.7101	0.6428	0.9000	0.7502	0.7000
Multilayer Perceptron	2000/2000	0.8102	0.9903	0.6012	0.7482	0.8000
DLAMD	2000/2000	0.9583	0.9524	0.9615	0.9569	0.9584

to produce frequent patterns making up of traffic characteristics and permissions. A hybrid feature detection method combining static extracted permission features and API features extracted under running program state was proposed [18], achieving 93% accuracy. Feng P *et al.* [13] proposed EnDroid, which was used to classify malicious software by the integration algorithm by extracting the behavior characteristics of the monitoring program at run time, achieving good performance. Cai H *et al.* [38] proposed DroidCat, a dynamic application classification technology. It used a variety of dynamic characteristics based on method calls and inter-component communication intentions without involving permissions, application resources, or system calls. The use of runtime features of the above methods will cause significant time and resource consumption for samples' preprocessing. However, the hybrid feature analysis method needs to acquire both runtime and source package-based features, which will consume more time and computing resources. Therefore, considering the large-scale data facing, this paper will not consider using these methods.

In the source package-based feature analysis method, Alotaibi A proposed a network called MalResLSTM [39]. To collect complex functions and underground structures from malware, this method imposes time restriction on the architecture of deep learning. Since this method only extracted eight independent and different static features, the generated vector has no semantic information and no deep semantic information of feature mining. The framework proposed by Kim T G *et al.* [40] considered various features of Android. The eigenvector selection method is optimized by using the methods based on existence and similarity, and the classification is carried out by using the multi-mode machine learning. The excessively complex network design results in the increase of computing resources and the consumption of time. An Android malware classification model based on sensitive opcode sequence analysis of code-specific semantic information was proposed [41]. They used opcodes, sensitive APIs, STRs, and actions to construct sensitive semantic feature-sensitive opcode sequences, and proposed to analyze specific semantic information of the code, and generated semantic correlation vectors for Android malware family classification based on this feature. Because the model only extracts the opcodes contained in methods that contain sensitive elements, the model may not be able to generate sensitive opcode sequences for further analysis, which may affect the final detection results. Arora A *et al.* [42]built a malware diagram and compare it to a normal sample diagram by extracting permission pairs from the application's manifest file. Due to the existence of dangerous permission pairs, many normal social and communication applications of Google Play Store have been identified as malicious applications, so the proposed model has a high FPR. Kang B J *et al.* [25] proposed a method based on n-gram opcode features, and used machine learning to identify and classify malware. It used data segmentation technology to perform feature selection and can be extended to 10-gram opcodes. As the value of n increases, the amount of calculations processed will increase exponentially, and sequence information may be lost.

Unlike the above methods, the method proposed in this paper designs a two-stage detection, which provides a rapid screening method for large numbers of samples. Moreover, CNN is used to realize automatic mining of internal semantic information, which not only extracts the key information of input, but also ensures the relative position information. LSTM are well suited for dealing with issues that are highly correlated with time series and provide a strong guarantee for subsequent classification.

6. Conclusion

Aiming at large-scale of android applications with unknown attributes to be detected, an efficient detection method based on combined deep neural network under 5G network is proposed. It divides the detection into two stages. Firstly, the pre-detection features are obtained by combining the quickly acquired permission features with the random forest feature selection, and the BP network is combined to carry out rapid classification. Then, a CNN-based key opcodes time series feature selection method is adopted to reduce the impact of reduced efficiency caused by ultra-long sequences and automatically obtain key time sequence information. LSTM learns and classifies the automatically filtered time sequence features, and obtains the final classification result of suspicious samples. Therefore, it has certain usability for malware detection in large-scale third-party application market in the 5G era.

References

[1] Taheri, R., Ghahramani, M., Javidan, R., Shojafar, M., Pooranian, Z., Conti, M., 2020. Similarity-based android malware detection using hamming distance of static binary features. Future Generation Computer Systems 105, 230–247.

[2] Lab, .B., . Android malware special report 2019. https://blogs.360.cn/post/review_android_malware_of_2019.html. Accessed February 28, 2020.

[3] Ashawa, M.A., Morris, S., 2019. Analysis of android malware detection techniques: a systematic review .

[4] Zachariah, R., Akash, K., Yousef, M.S., Chacko, A.M., 2017. Android malware detection a survey, in: 2017 IEEE international conference on circuits and systems (ICCS), IEEE. pp. 238–244.

[5] Patel, Z.D., 2018. Malware detection in android operating system, in: 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), IEEE. pp. 366–370.

[6] Bakour, K., Ünver, H.M., Ghanem, R., 2018. The android malware static analysis: techniques, limitations, and open challenges, in: 2018 3rd International Conference on Computer Science and Engineering (UBMK), IEEE. pp. 586–593.

[7] Wang, W., Gao, Z., Zhao, M., Li, Y., Liu, J., Zhang, X., 2018. Droidensemble: Detecting android malicious applications with ensemble of string and structural static features. IEEE Access 6, 31798–31807.

[8] Guo, Y., Yang, L., Gao, X., Wu, K., 2016. The static detection analysis technology of android source codes, in: 2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC), IEEE. pp. 288–292.

[9] Şahın, D.Ö., Kural, O.E., Akleylek, S., Kiliç, E., 2018. New results on permission based static analysis for android malware, in: 2018 6th International Symposium on Digital Forensic and Security (ISDFS), IEEE. pp. 1–4.

[10] Taheri, R., Javidan, R., Shojafar, M., Pooranian, Z., Miri, A., Conti, M., 2020. On defending against label flipping attacks on malware detection systems. Neural Computing and Applications , 1–20.

[11] Shankar, V.G., Somani, G., Gaur, M.S., Laxmi, V., Conti, M., 2017. Androtaint: An efficient android malware detection framework using dynamic taint analysis, in: 2017 ISEA Asia security and privacy (ISEASP), IEEE. pp. 1–13.

[12] Aktaş, K., Sen, S., 2018. Android malware detection based on runtime behaviour, in: 2018 26th Signal Processing and Communications Applications Conference (SIU), IEEE. pp. 1–4.

[13] Feng, P., Ma, J., Sun, C., Xu, X., Ma, Y., 2018. A novel dynamic android malware detection system with ensemble learning. IEEE Access 6, 30996–31011.

[14] Kaushik, P., Yadav, P.K., 2018. A novel approach for detecting malware in android applications using deep learning, in: 2018 Eleventh International Conference on Contemporary Computing (IC3), IEEE. pp. 1–4.

[15] Malik, J., Kaushal, R., 2016. Credroid: Android malware detection by network traffic analysis, in: Proceedings of the 1st acm workshop on privacy-aware mobile computing, pp. 28–36.

[16] Jin, H., Li, Y., Yang, Y., 2020. Minerbs: Detecting android malware based on runtime behavior sequence, in: 2020 12th International Conference on Communication Software and Networks (ICCSN), IEEE. pp. 216–222.

[17] Yan, L.K., Yin, H., 2012. Droidscope: Seamlessly reconstructing the {OS} and dalvik semantic views for dynamic android malware analysis, in: Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12), pp. 569–584.

[18] Kuo, W.C., Liu, T.P., Wang, C.C., 2019. Study on android hybrid malware detection based on machine learning, in: 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), IEEE. pp. 31–35.

[19] Surendran, R., Thomas, T., Emmanuel, S., 2020. A tan based hybrid model for android malware detection. Journal of Information Security and Applications 54, 102483.

[20] Zhang, B., Xiao, W., Xiao, X., Sangaiah, A.K., Zhang, W., Zhang, J., 2020. Ransomware classification using patch-based cnn and self-attention network on embedded n-grams of opcodes. Future Generation Computer Systems 110, 708–720.

[21] Zhou, Y., Jiang, X., 2012. Dissecting android malware: Characterization and evolution, in: 2012 IEEE symposium on security and privacy, IEEE. pp. 95–109.

[22] Yuan, H., Tang, Y., Sun, W., Liu, L., 2020. A detection method for android application security based on tf-idf and machine learning. Plos one 15, e0238694.

[23] Gaviria de la Puerta, J., Sanz, B., 2017. Using dalvik opcodes for malware detection on android. Logic Journal of the IGPL 25, 938–948.

[24] Canfora, G., Mercaldo, F., Visaggio, C.A., 2015. Mobile malware detection using op-code frequency histograms, in: 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE), IEEE. pp. 27–38.

[25] Kang, B., Yerima, S.Y., Sezer, S., McLaughlin, K., 2016. N-gram opcode analysis for android malware detection. arXiv preprint arXiv:1612.01445 .

[26] , . Androguard. https://github.com/androguard/androguard.

[27] Li, D., Zhao, L., Cheng, Q., Lu, N., Shi, W., 2020. Opcode sequence analysis of android malware by a convolutional neural network. Concurrency and Computation: Practice and Experience 32, e5308.

[28] , . Apktool. https://code.google.com/android/apk-tool.

[29] Kim, Y., 2014. Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 .

[30] Zhao, L., Li, D., Zheng, G., Shi, W., 2018. Deep neural network based on android mobile malware detection system using opcode sequences, in: 2018 IEEE 18th International Conference on Communication Technology (ICCT), IEEE. pp. 1141–1147.

[31] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C., 2014. Drebin: Effective and explainable detection of android malware in your pocket., in: Ndss, pp. 23–26.

[32] Xiao, X., 2019. An image-inspired and cnn-based android malware detection approach, in: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE. pp. 1259–1261.

[33] Naway, A., Li, Y., 2018. A review on the use of deep learning in android malware detection. arXiv preprint arXiv:1812.10360 .

[34] Odusami, M., Abayomi-Alli, O., Misra, S., Shobayo, O., Damasevicius, R., Maskeliunas, R., 2018. Android malware detection: A survey, in: International Conference on Applied Informatics, Springer. pp. 255–266.

[35] Sabhadiya, S., Barad, J., Gheewala, J., 2019. Android malware detection using deep learning, in: 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), IEEE. pp. 1254–1260.

[36] Zhang, W., Wang, H., He, H., Liu, P., 2020. Damba: detecting android malware by orgb analysis. IEEE Transactions on Reliability 69, 55–69.

[37] Arora, A., Peddoju, S.K., 2018. Ntpdroid: a hybrid android malware detector using network traffic and system permissions, in: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), IEEE. pp. 808–813.

[38] Cai, H., Meng, N., Ryder, B., Yao, D., 2018. Droidcat: Effective android malware detection and categorization via app-level profiling. IEEE Transactions on Information Forensics and Security 14, 1455–1470.

[39] Alotaibi, A., 2019. Identifying malicious software using deep residual long-short term memory. IEEE Access 7, 163128–163137.

[40] Kim, T., Kang, B., Rho, M., Sezer, S., Im, E.G., 2018. A multimodal deep learning method for android malware detection using various features. IEEE Transactions on Information Forensics and Security 14, 773–788.

[41] Jiang, J., Li, S., Yu, M., Li, G., Liu, C., Chen, K., Liu, H., Huang, W., 2019. Android malware family classification based on sensitive opcode sequence, in: 2019 IEEE Symposium on Computers and Com-

munications (ISCC), IEEE. pp. 1–7.
[42] Arora, A., Peddoju, S.K., Conti, M., 2019. Permpair: Android malware detection using permission pairs. IEEE Transactions on Information Forensics and Security 15, 1968–1982.