Hindawi
Security and Communication Networks
Volume 2023, Article ID 2796988, 11 pages
https://doi.org/10.1155/2023/2796988

Research Article

Android Malware Detection Based on Program Genes

Qingfeng Li [1] **Guoqiang Chen** [2] **and Bo Li** [2]

[1]*Network Information Center, Northeast Forestry University, Harbin, China*
[2]*Information and Computer Engineering College, Northeast Forestry University, Harbin, China*

Correspondence should be addressed to Bo Li; 2020111884@nefu.edu.cn

Received 6 December 2022; Revised 3 February 2023; Accepted 10 March 2023; Published 15 April 2023

Academic Editor: Saed Alrabaee

Copyright © 2023 Qingfeng Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The security issues with mobile devices have received more attention as a result of the development of mobile Internet technology and the adoption of mobile intelligent terminal devices. It is becoming more crucial to quickly and effectively identify and remove harmful applications from systems in order to protect user data and personal devices. The Dalvik bytecode, permission applications, and system calls of Android apps are the main targets of the current Android malware analysis approaches. However, in recent years, an increasing amount of Android malware conceals harmful code in native code. The method for using program gene technology to identify malware on the Android platform is presented in this research. This method extracts executable library files from the Native layer binary executable files of Android programs and disassembles the library files to obtain program genes. Then, the programs' genes perform feature screening by the information gain method and next use Word2Vec to express the semantic abstraction of the screened features. Finally, the screened features were used in deep neural network models for training and detection. The experimental results demonstrate that compared with KNN, SVM, and other machine learning algorithms, the deep neural network model is more effective and the detection accuracy reaches up to 97.51%. Thus, it confirmed the feasibility of the Android malicious program detection method based on program genes in this paper.

1. Introduction

According to the 46th Statistical Report on the Development Status of the Internet in China [1] published by the Internet Information Office of the Central Committee of the Communist Party of China on September 29, 2020, although the domestic Internet user population's experience with Internet security issues over the previous six months decreased by 5.2 percent, mobile device security issues have become more prevalent in recent years. Domestic Internet security is still not in a promising state. Mobile security has grown to be an essential component of cyberspace security as 99.2% of the domestic Internet users now access the web using a mobile device.

It is indicated in the "Report on the State of Mobile Security in China in the First Half of 2020" [2], jointly published by 360 Internet Security Center and China Academy of Information and Communication Research, that, in the first half of 2020, 360 Security Brain intercepted a total of about 1.048 million new malicious program samples on mobile, an increase of 12.2% from the first half of 2019 (920000) and that the average daily interception of new mobile malware increased from 0 to 1 every day. About 0.6 million brand-new malicious software samples are captured on a daily average. The nation was infected by the new crown virus in 2020, especially during the Chinese New Year vacation. As a result, Internet use time rose dramatically, increasing the likelihood that users would encounter unlawful content on the network. The results in the research show that the sample volume of harmful program interception in the first half of 2020 increased, and malicious programs that nefariously drain users' bank accounts and steal users' personal information are mushrooming everywhere. As a result, the research about the mobile malicious program detection technology is essential for advancing the development of mobile terminal network security. Unlike Windows and Linux platforms, the instruction set used in the Android system changes greatly when each version is updated, and it becomes more and more difficult to dynamically extract the instruction flow executed by Android

programs. This creates a higher barrier for mobile malicious program detection. Therefore, it is more necessary to propose a method to detect malicious programs under the Android system by static analysis correlation. Our main contributions are as follows:

(1) We developed a program gene extraction method to extract binary executable library files of Native layer from Android application installation package. This method extracted program genes from the sample executable library files by static disassembly, used opcode to represent assembly instructions to streamline the program gene library, and obtained the basic block sequence files.

(2) We used a combination of information gain and Word2Vec model. First, we used information gain to count the number of occurrences of benign and malicious samples in the basic block sequence file and performed feature filtering by threshold. Next, we utilized the Word2Vec model to convert each basic block into a vector and store them.

(3) We constructed a deep neural network classifier for malicious programs, connected multiple hidden layers using full connectivity and added a dropout layer to prevent overfitting, and finally validated the model using a five-fold cross-validation method.

2. Related Work

Malicious program detection on the Android platform is the same as traditional malicious program detection techniques, which are classified into dynamic analysis and static analysis. The dynamic analysis method detects maliciousness by monitoring the behavior of the program during execution, such as function calls, resource access, and system calls. The static analysis method extracts static features from the disassembled code through specific disassembly tools to detect maliciousness, which does not require code execution, thus avoiding the consumption of time, space, and resources caused by application execution and achieving 100% code coverage. Mariconti et al. [3] proposed to build a malicious program detection system by constructing a call graph from Android program API calls and then extracted the call sequences using the Markov chain modeling. Lindorfer et al. [4] constructed a method to detect privilege elevation vulnerabilities in Android preinstalled software by statically intermediate representation of disassembled code to construct interprogram data flow graphs and control flow graphs, which are used for taint propagation analysis to detect possible vulnerabilities and information leakage points, and achieved a good detection rate. Yu and Tao [5] put forward a method for Android malware detection based on model library, where data of different populations were obtained by classifying permission information and applied to the model library, which led to a certain enhancement of the detection effect. By combining syntactic and semantic features in the detection method, Yanping et al. [6] utilized a computing adaptive feature weight with PSO. It used a support vector machine (SVM) method based on feature weights that are computed by information gain (IG) and particle swarm optimization (PSO) algorithms, overcome the defects of basic PSO, and improve the performance of SVM. Weiping et al. [7] made a method that transforms the dynamic API call sequence into a function call graph. The method fuses the transformed function call graph feature and the extracted permission request feature to perform a high-detection accuracy. Min et al. [8] proposed an Android malware detection model based on DT-SVM. The model extracts the original opcode and Dalvik opcode by reversing Android software. Meanwhile, the model effectively combines DT with SVM. Under the premise of maintaining a high-accuracy decision path, SVM is used to effectively reduce the overfitting problem in DT and thus improve the generalization ability. Although the majority of the aforementioned studies focus on structural similarity of malicious programs or similarity of system calls and permission usage of programs, they do not focus on machine code at the disassembly level from the perspective of homology, and most of the research works focus on permission requests, system calls, and Dalvik bytecode. The authors used both the genetic algorithm and machine learning method in [9, 10], which reduced feature dimensionality, proving that the combination of the genetic algorithm and the machine learning method has some advantages in Android malware detection. Xiao et al. [11] applied program technology to the task of malicious program detection on PC and also achieved better accuracy. The malicious code of Android platform is mainly binary code, including Dalvik virtual machine executables in DEX format and native binary executables in ELF format. Jin et al. [12] proposed enhancing classification accuracy by extracting program genes in the form of use-def chains from bytecode files in DEX format, but they did not take into consideration that, in recent years, the vast majority of Android malware prefers to write malicious logic into native layer binary executables to obtain features such as obfuscation, encryption, and other disguised malicious logic.

In this paper, the program gene technology is applied to the detection of Android malicious program. First, the static analysis method is adopted to extract program genes from the native binary executable of Android programs, and then, feature selecting is carried out through the method of information gain. Also, for feature semantic abstract expression, Word2Vec is employed, and finally, the deep learning model is used for training. These constitute the Android malware detection system.

3. Android Malware Detection Method Based on Program Genes

Using the static analysis methodology, this paper proposes a maliciousness detection method for Android apps from the viewpoint of program genes. The method is shown in Figure 1 and is based on program genes. This strategy adopts the native layer out of the installation package for an Android application, breaks down the library files to get the matching instruction sequence, and uses a program to identify all the fundamental building blocks in the instruction sequence to produce a basic block collection. As a consequence, the

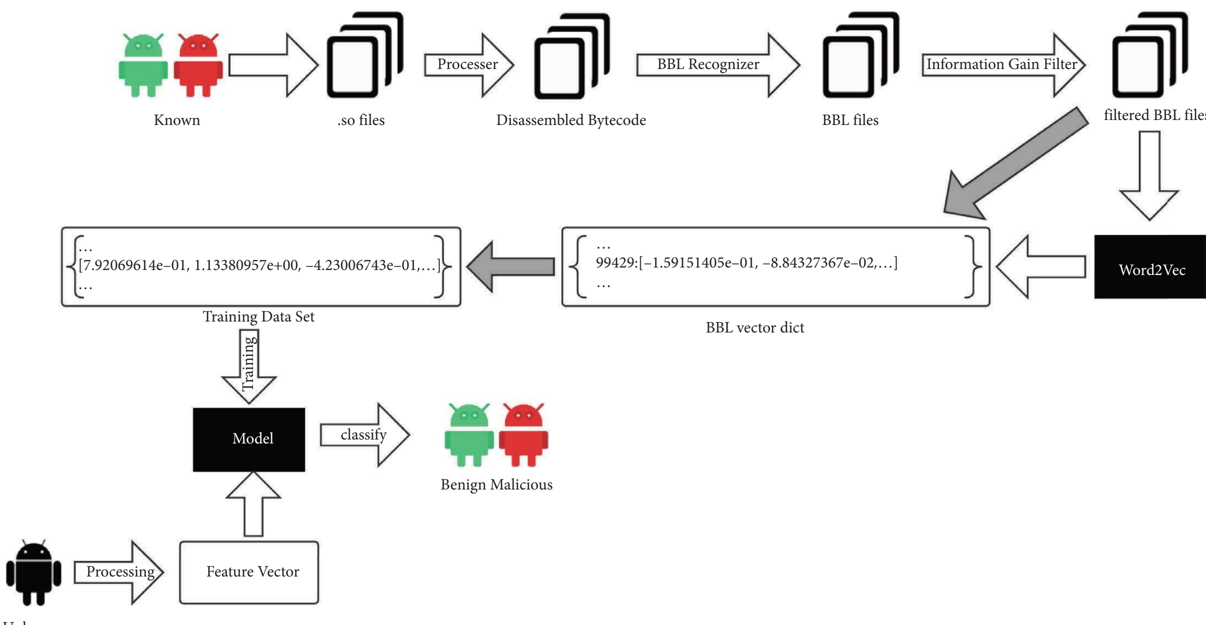


Figure 1: System architecture.

original binary file is transformed into a representation of a basic block index, and the basic block characteristics are then filtered based on the information gain. When the Word2Vec word embedding model receives the filtered basic block index files as the original corpus for training, a word vector dictionary comprising all the basic blocks is formed. The original Android installation samples, in accordance with this vocabulary, are converted into vector representations and sent to various classification models, such as deep neural networks for training.

3.1. Procedural Genes and Extraction Methods

3.1.1. Definition of Program Genes. The computer program gene is the smallest unit in which the static components of a program are dynamically expressed. It is used to extract program behavior characteristics of a computer from the assembly instruction stream in which the program is running for semantic description and unique characterization of the behavior patterns of a computer program to reflect the program's true behavioral intent. Program genes have many characteristics with biological genes. In biology, genes are referred to as the fundamental genetic building blocks that regulate biological qualities, while a program's characteristics are mostly represented in its distinctive behavior. The assembly instructions included in a binary program are executed when it is run on a computer, and the type, order, and other properties of the executed assembly instructions define the specific features the program shows. Program genes, as defined previously, are the series of instruction streams that the CPU actually executes when a program is running dynamically. On Windows or Linux platforms, dynamic instruction streams can be extracted using tools such as the QEMU full virtualization sandbox and the Pin dynamic binary stubbing framework. However, between versions of Android, the instruction set has seen considerable changes. There is not a better simulated execution sandbox available right now for dynamically extracting the execution order of Android programs. In addition, this article aims to look at the viability of application genetic approaches for Android platform activities including the detection of harmful applications. As a result, this study adopts the following conventions while discussing the idea of application genes, which are obtained via static disassembly from example native layer executable binary library files.

3.1.2. Extraction of Program Genes. Program genes are bits of assembly instructions that represent certain actions and serve specific purposes. The assembly instruction can be divided into six levels from course to fine granularity: image file, program segment, function, instruction flow, basic block, and assembly instruction. In this study, the basic block is taken as the gene fragment of the program gene, and the basic block is employed as the minimum processing unit for the program gene extraction, subsequent information gain, and word vector embedding.

In this paper, we first extract the binary executable library file from the installation package of Android program samples, then, we disassemble the library file to extract all the codes of the code segment, and finally, we perform the basic block identification by identifying all the branch instructions, return instructions, exception instructions, and other instructions to divide the basic blocks accordingly. Even though the ARM instruction set is more streamlined than the x86 instruction set in terms of opcode kinds and numbers, there are still a huge number of basic blocks needed to construct these assembly instructions due to the thousands of possible opcode and operand combinations.

Due to this, we choose to employ opcodes to represent assembly instructions to simplify the program gene library while maintaining the behavioral information of the fundamental building blocks. However, it is not possible to completely overwrite binary code in the real world. We suggest the following potential mitigation plans in order to stimulate further study:

(1) normalize instructions for disassembly, mask immediate values, and address with specified symbols to greatly reduce the complexity of fundamental blocks

(2) The clustering algorithm is used to cluster the basic blocks drawn from multiple malicious samples and identify similar basic blocks as malicious code blocks, thus further filtering out useless genes that are of little help in identifying maliciousness

Algorithm 1 illustrates the procedure of extracting the program gene. The time complexity of this algorithm is O(n), and it processes each instruction mirror file sequentially, reading and cutting instructions to basic blocks in a linear fashion. In the single-threaded case, the algorithm occupies at most the storage space of all basic blocks in a file. In other words, the algorithm has an average time and space complexity and does not have excessive overhead. Figure 2 depicts the sequence of instructions used to disassemble the library file in a sample of a malicious program. This basic block identification procedure is then used to create the mapping dictionary, which contains all of the sample's basic blocks and index tags.

The following three components are created among the intermediate files during the process of program gene extraction.

(1) Instruction mirror file: In this work, the whole instruction sequence acquired by disassembly is referred to as the "library file md5 value int.txt" or "mirror file" for short. This file stores the function name, function address, assembly instruction address, and assembly instruction information about all instruction sequences.

(2) Basic block sequence file: The basic block sequence file in this work is known as the instruction image file using the basic block index and is entitled "library md5 value bbl.txt." This file is used to capture all instruction sequences as indexes and replace the recognized basic blocks with index numbers.

(3) Mapping dictionary: This document contains a mapping dictionary, which is a list of all the fundamental blocks and the index tags that relate to them. The fundamental blocks listed in the instruction image file are mapped to index numbers using this dictionary file.

After the previous extraction procedure, it is possible to obtain the instruction image files corresponding to the binary executable library files in all samples. All the files convert into basic block sequence files represented by basic block indexes using the mapping dictionary, which serves as the original gene pool for further processing and analysis work.

3.2. Feature Selection Based on Information Gain. Through the previous work of program gene extraction, we obtained a large number of instruction sequences represented by basic blocks, i.e., basic block sequence files. However, not all the basic blocks in these files are useful for identifying the maliciousness of programs, so, in this paper, we use information gain for feature screening of basic blocks [13], retaining the basic blocks with larger information gain values, and removing redundant information as much as possible to reduce the computational effort while retaining program behavior information useful for classification. In the following section, the information gain calculation formula is derived for this prediction model, and the sample features described in the following are the identified basic blocks.

The information gain is a common evaluation criterion in feature selection, which describes how much information a feature contributes to the prediction model, and the more information a feature contributes, the more important it is to the prediction model. The information gain of a feature in information theory is defined as the difference between the systematic entropy and the conditional entropy after fixing the feature. The systematic entropy is used to describe the uncertainty of the feature; for a prediction model, let the sample data set be D and let the classification space be $\{C_i, i = 1, 2, \ldots, n\}$; the entropy of the prediction model [14] is as shown in the following equation:

$$\text{Entropy}(C) = -\sum^{n} P(C_i)\log P(C_i), \tag{1}$$

where $P(C_j)$ is the probability that any sample in the sample data set has a category of.

For the feature B of the prediction model, the feature taking space is $\{B_i, j = 1, 2 \cdots m\}$, and the conditional entropy of the prediction model is shown in the following equation:

$$\text{Entropy}(C \mid B) = -\sum_{i=1}^{n}\sum_{j=1}^{m} P\left(C_i, B_j\right)\log P\left(C_i \mid B_j\right). \tag{2}$$

For any feature B_j, the conditional entropy of the prediction model is shown in the following equation:

$$\text{Entropy}\left(C \mid B_j\right) = -\sum^{n} P\left(C_i \mid B_j\right)\log P\left(C_i \mid B_j\right). \tag{3}$$

The following equation can be obtained by combining the conditional probability formula:

$$\text{Entropy}(C \mid B) = \sum^{m} P(B_i)\text{Entropy}\left(C \mid B_j\right). \tag{4}$$

For this prediction model, since there are only two values of occurrence and nonoccurrence in the feature space, that is, $m = 2$, assuming that the probability of occurrence of a feature in any sample from the sample set is $P(\mathrm{B}_j)$, and the

```
1 .func: 0x13ccL sub_13CC
2 .text: 0x13ccL LDR     R0, =(unk_E000 - 0x13D8)
3 .text: 0x13d0L ADD     R0, PC, R0; unk_E000
4 .text: 0x13d4L B       __cxa_finalize
5 .func: 0x13dcL char_to_nibble
6 .text: 0x13dcL STR     R11, [SP,#-4+var_s0]!
7 .text: 0x13e0L ADD     R11, SP, #0
8 .text: 0x13e4L SUB     SP, SP, #0xC
9 .text: 0x13e8L MOV     R3, R0
10 .text: 0x13ecL STRB    R3, [R11,#var_5]
11 .text: 0x13f0L LDRB    R3, [R11,#var_5]
12 .text: 0x13f4L SUB     R3, R3, #0x30
13 .text: 0x13f8L CMP     R3, #0x36 ; '6'; switch 55 cases
14 .text: 0x13fcL ADDLS   PC, PC, R3,LSL#2; switch jump
15 .text: 0x1400L B       loc_1560; jumptable 000013FC default case
16 .text: 0x1404L B       loc_14E0; jumptable 000013FC case 0
17 .text: 0x1408L B       loc_14E8; jumptable 000013FC case 1
18 .text: 0x140cL B       loc_14F0; jumptable 000013FC case 2
19 .text: 0x1410L B       loc_14F8; jumptable 000013FC case 3
20 ...
```

Figure 2: Example diagram of a disassembly instruction sequence.

Input: command image file;
Output: Gene bank and basic block sequence file;
1. GENE $\Longleftarrow \{\{\phi\}\}$;
2. foreach sample ∈ samples do
3. sample_gene ⟸ [];
4. for each function ∈ sample do
5. bbl ⟸ [];
6. for each ins ∈ sample do
7. if ins is Branch then
8. GENE ⟸ {hash (bbl): bbl};
9. sample_gene.append (hash (bbl));
10. else
11. bbl.append (ins);
12. end if
13. end for
14. clear (bbl);
15. end for
16. write 2 file (sample_gene);
17. end for

Algorithm 1: Gene bank builds and converts instruction image files into basic block sequence files.

probability of nonoccurrence is $P(B_j)$, the previous equation can be simplified in the following equation:

$$\text{Entropy}\,(C \mid B) = P\left(B_j\right)\text{Entropy}\left(C \,\middle|\, B_j\right) + P\left(\overline{B_j}\right)\text{Entropy}\left(C \,\middle|\, \overline{B_j}\right). \tag{5}$$

According to the definition of information gain, the feature B_j information gain in the following equation is obtained:

$$IG(X) = -\sum_{i=1}^{n} P(C_i)\log P(C_i) - P(B_j)\text{Entropy}\left(C \mid B_j\right) - P(\overline{B_j})\text{Entropy}\left(C \mid \overline{B_j}\right)$$
$$= -\sum_{i=1}^{n} P(C_i)\log P(C_i) + P(B_j)\sum_{i=1}^{n} P\left(C_i \mid B_j\right)\log P\left(C_i \mid B_j\right) + P(\overline{B_j})\sum_{i=1}^{n} P\left(C_i \mid \overline{B_j}\right)\log P\left(C_i \mid \overline{B_j}\right). \quad (6)$$

In order to calculate the information gain value of each feature, we have to count the number of times about each basic block appearing in the malicious and benign samples and then calculate the information gain value using equation (6), so as to filter the features according to the threshold value.

3.3. Feature Semantic Abstraction Representation. After the extraction of program genes and information gain, the next step is to perform the semantic abstract representation of the features to transform them into a vector form. However, the number of basic blocks after the previous information gain is still very large, and if one-hot encoding [15] is directly used as the abstract representation of the sample programs, it will make the data set into a large sparse matrix, which is unfavorable for computation, on the one hand, and lose semantic information such as contextual relationships of the basic blocks on the other hand. Therefore, this paper adopts the Word2Vec word vector generation model [16], which is a shallow neural network model consisting of an "input layer-hidden layer-output layer" and a simple neural network. The Word2Vec model can map each word to a vector space of arbitrary dimension to represent the relationship between words. Word2Vec relies on skip-grams or CBOW to build neural word embeddings, and the trained word vectors retain good contextual semantic relationship information.

In this paper, we use the Word2Vec module in the python open-source third-party toolkit Gensim [17] for word embedding training of basic blocks, traversing all basic blocks in all basic block sequence files and feeding them into the Word2Vec model as the original corpus for training to obtain a 500-dimensional word vector dictionary, which includes the embedding of all basic block indexes in the corpus vectors. We can use this dictionary to convert all samples into vectors for storage and provide data for subsequent model training and testing. The setting of the word vector dimension is a key parameter in the whole prediction system, and the length of the word vector determines the amount of information it can carry, which will be further explored later to find its optimal value.

3.4. Malicious Program Classifier Based on Deep Learning. Through the semantic abstract representation of the previous features, a word vector dictionary of all basic blocks is obtained, which is used to transform each basic block in the basic block sequence file into a vector representation, so that each sample is a two-dimensional matrix containing multiple vectors, and then, each sample is compressed into a one-dimensional vector representation. In this paper, we use the method of accumulating and averaging all the basic block word vectors of each sample, which can ensure that the original text feature information is not lost to the maximum extent. Through the dimensionality reduction process, the whole sample space is a two-dimensional vector set, and we use a deep neural network [18] based on deep learning [19] to construct a malicious program classifier.

In the process of model construction, in order to fully evaluate the performance of the model to reduce the overfitting problem, we lead the k-fold cross-validation method [20] to apply the test set data on the trained model for testing. Moreover, the relevant classification result parameters are defined as shown in Table 1, and the accuracy, precision, recall, and $F1$ score are obtained as the performance indexes of the model.

The accuracy rate identifies the proportion of malicious samples that are predicted to be malicious; the precision rate is the proportion of malicious samples that are truly malicious among all predicted malicious samples; the recall rate identifies the proportion of malicious samples that are correctly determined among all malicious samples; the $F1$ score is the weighted summed average of the accuracy and recall rates.

In this paper, we use the architecture of a deep neural network as shown in Figure 3. This neural network uses a fully connected way to connect multiple hidden layers and adds a dropout layer in the middle of each layer to prevent overfitting, randomly discarding the updates of certain parameters during the gradient backward update and using ReLU as the activation function in the neurons in order to mitigate the effect of the gradient disappearance problem.

4. Experiment and Analysis

4.1. Experimental Settings. The original samples used in this paper are Android application installer files. The dataset contains 8000 malicious sample records and 2000 benign sample records, where the malicious samples are obtained by random sampling from VirusShare_Android_APK_2018.zip provided by https://virusshare.com/ [18]. Moreover, benign sample records are downloaded from the Android App Store. We keep the json file of each apk sample after processing, which contains program name, package name, version Vector dictionaries, and so on. To reduce the sensitivity of model test results to data division, all experimental results in this paper are the results of 5-fold cross validation. Table 2 shows the attribute distribution of the dataset.

4.2. Experimental Process. In this paper, the previous experimental sample set is processed as follows:

Table 1: Definition of classification result parameters.

Source of samples	Testing results	
	Malicious sample	Benign sample
Malicious sample	TP	FN
Benign sample	FP	TN

(1) extract the native binary executable library file for each sample.
(2) disassemble the library files, extract the instruction sequences for all code segments, and obtain the corresponding instruction image files.
(3) refer to the ARM instruction set design to achieve the basic block recognizer, traverse all instruction image files, and generate the basic block index dictionary.
(4) convert all instruction image files into basic block sequence files according to the basic block index dictionary.
(5) count the number and frequency of occurrence of all basic blocks in the basic block sequence file and calculate the information gain of each basic block and then select the basic block features to generate the basic block sequence file.
(6) The filtered basic block sequence files are fed into the Word2Vec model as a corpus for training, and the word-embedding vector dictionaries for all basic blocks are obtained. We take the basic block sequence files as corpus into the Word2Vec model to train and then get word embedding vector dictionaries for all basic blocks
(7) Based on the word embedding vector dictionaries of the previous basic blocks, the samples are transformed into vector representations. We embed the basic block in the vector dictionaries and then transform the samples into the vectors.
(8) input vectors into the deep neural network to train and test.

4.3. Analysis of Experimental Results. In this paper, the following experiments are designed to verify the feasibility of the malicious program detection model designed in this paper and to determine the values of the key parameters in this system. First, we determine the information gain threshold of selecting basic blocks in the information gain method. Second, we determine the vector length of the word vector dictionary trained by the Word2Vec model. Because too short vector length may lead to the loss of semantic information, while too long vectors may lead to information redundancy and reduce the recognition efficiency and accuracy of the final model. Finally, we compare with other traditional machine learning models and identify the model that suits the system architecture designed in this paper.

We set the vector length of the word vector dictionary output by Word2Vec at 500 and use the deep neural network

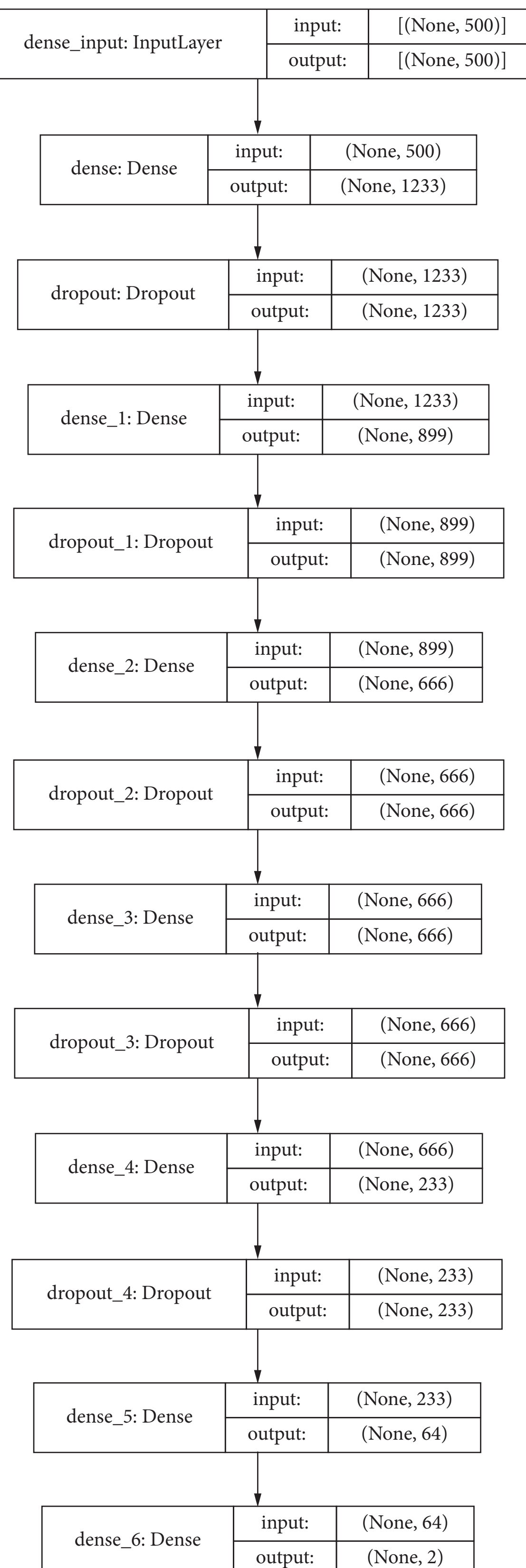


Figure 3: Deep neural network structure.

Table 2: Distribution of attributes of datasets used in the experiment.

Classes	Training	Tests
Benign	1600	400
Malware	6400	1600

Table 3: Neural network model scores with different information gain thresholds.

IG	Accuracy (%)	*f*1 score (%)	Precision (%)	Recall (%)
0.00	94.49	97.10	96.65	97.58
1.00	95.29	98.25	98.45	98.07
2.00	97.08	98.94	98.70	98.69
4.00	95.91	97.89	98.13	97.69
6.00	96.20	98.11	97.86	98.37
8.00	95.67	97.73	97.83	97.68
10.00	95.68	98.03	97.69	98.40
12.00	95.59	98.01	98.51	97.53

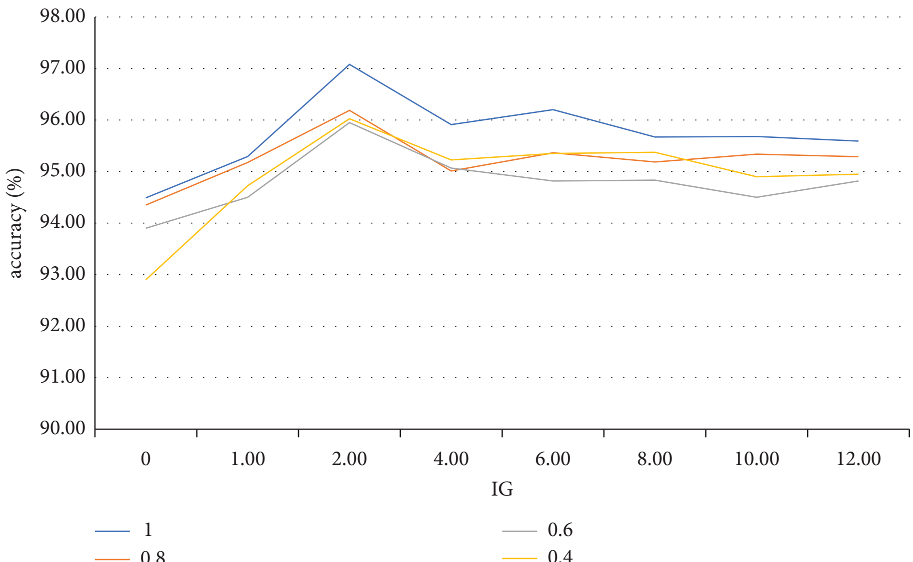


Figure 4: Split data set information gain optimal threshold search.

Table 4: Neural network model scores with different word vector lengths.

Embedding lens	Accuracy (%)	*f*1 score (%)	Precision (%)	Recall (%)
100	96.95	98.08	97.71	98.48
200	97.21	98.29	98.59	97.51
300	97.51	98.85	98.74	98.98
400	97.09	98.17	97.95	98.40
500	97.08	98.94	98.70	98.69
600	96.66	98.54	98.60	98.50
700	96.64	98.51	98.23	98.82
800	96.41	98.43	98.09	97.33

to train which is obtained in Table 3. In the experiment of this paper, setting the information gain threshold as 2.0 is the most appropriate. In this experiment, it is important to ensure practicality that the model parameter settings are not affected by the dataset. To determine whether the IG threshold is affected by the size of the dataset, we slice the dataset and take a subset of the original dataset for the same experiment using the nonrepeat sampling method, and the results are shown in Figure 4. The subsets with different data size can get the optimal value around the information gain threshold of 2.0, so it can be seen that the information gain threshold is not affected by the dataset size.

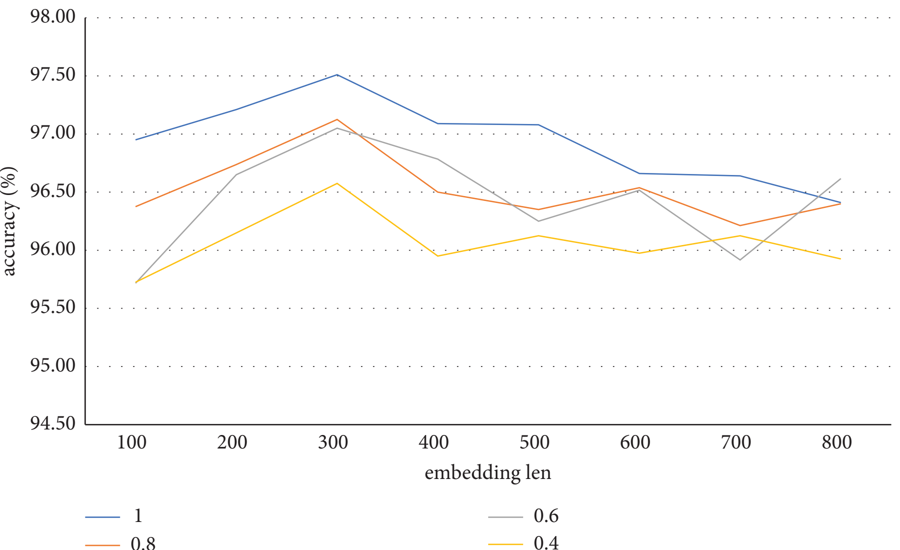
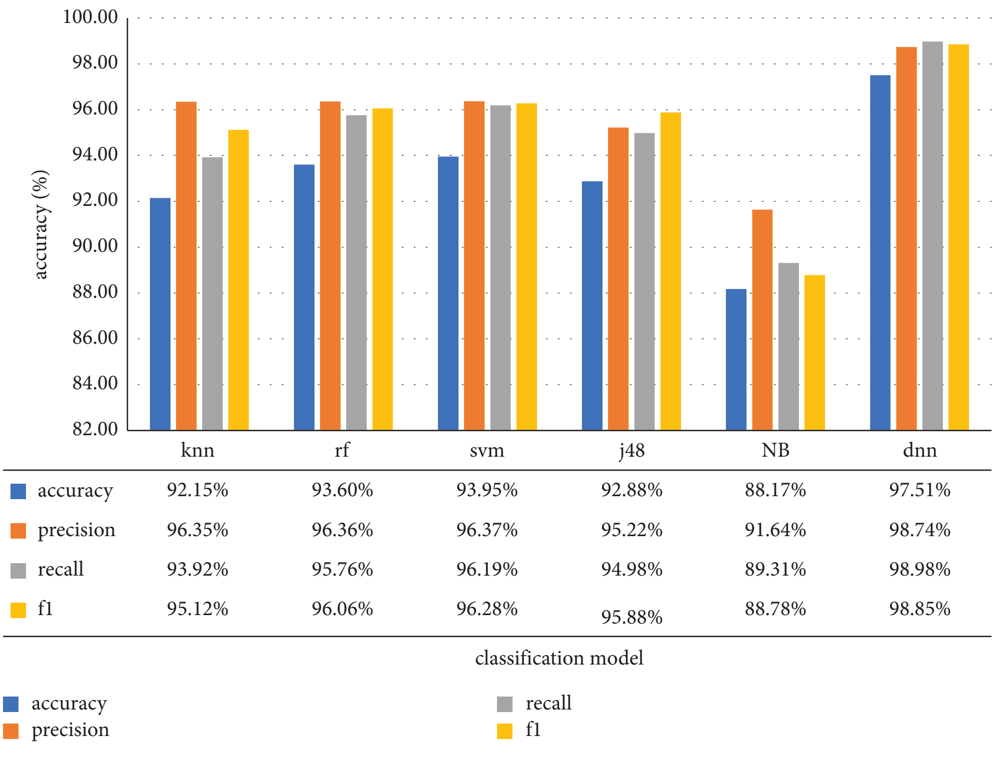


Figure 5: Search for optimal value of word vector length for segmented dataset.



	knn	rf	svm	j48	NB	dnn
accuracy	92.15%	93.60%	93.95%	92.88%	88.17%	97.51%
precision	96.35%	96.36%	96.37%	95.22%	91.64%	98.74%
recall	93.92%	95.76%	96.19%	94.98%	89.31%	98.98%
f1	95.12%	96.06%	96.28%	95.88%	88.78%	98.85%

Figure 6: Different classification model scores.

After determining the IG threshold, the next step is to find the optimal value for the length of the embedding vector obtained from the Word2Vec word vector model. In this group of experiments, the neural network model based on the information gain threshold of 2.0 is used as the classifier and trained to get the comparison experiments as shown in Table 4. When the word vector length is 300, we can get higher accuracy and achieve better performance in terms of precision, recall, and $F1$ score. In the four sets of comparison experiments as shown in Figure 5, it can be seen that the model achieves the optimal value around 300. Therefore, it is observed that the length of the embedding vector set by the Word2Vec model training is less affected by the size of the dataset.

In order to prove the effectiveness of this model, this paper conducted comparison experiments with four classification models, namely, K-nearest neighbor, random forest, support vector machine, J48 decision tree, Naive Bayes, and deep neural network. The results of the training and testing are shown in Figure 6. In this paper, the deep

neural network model has the best performance among the four models, and its overall performance is better than the other classification models, with the highest accuracy rate reaching 97.51%, precision rate, recall rate, and *f*1 score also have better performance and fitting degree.

Through the analysis of the previous experiments, the Android malware detection method proposed in this paper shows a good discrimination ability and proves that the Android malware detection based on program genes is feasible.

5. Conclusions

In response to the sharp increase on malicious programs in Android, we suggest a method for detecting malicious programs that uses program gene technology on the Android platform. The effectiveness of this paper's method is confirmed through multiple sets of experimental analysis on a dataset of 10,000 malicious and benign sample programs, which can provide a corresponding method for the Android malicious program analysis and malicious program detection. In the absence of binary program source code, a complete malicious program detection method is designed and implemented. It started with program gene extraction from Native layer disassembly code. Then, the programs' genes perform feature screening by the information gain method and next use Word2Vec to express the semantic abstraction of the screened features. Finally, the screened features were used in deep neural network models for training and detection. However, in the face of malware using various techniques to evade detection, the use of natural language processes and machine learning techniques alone cannot identify the latest malicious programs, which is the current limitation of this paper. Therefore, in the follow-up work, more features that can be used to determine the maliciousness of programs can be added and dynamic features such as permission requests and system API calls can also be integrated to help implement a better approach. We will keep researching the applications of this technology that have gained a lot of value in recent years, such as the identification of malicious software and code traceability analyses.

Data Availability

The data used to support the findings of this study are included within the article. The data presented in this study are available upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by New Infrastructure and University Informatization Research Project (Grant no. XJJ202205017) and The Ministry of Education's Cooperative Education Project (Grant no. 220604719062313).

References

[1] Internet Network Information Center, *Statistical Report on the Development of Internet in China*, China Internet Network Information Center, Beijing, China, 2020.

[2] Security Technology Co, *China Mobile Security Status Report for the First Half of 2020*, 360 Mobile Phone Guard & 360 Enterprise Security & China Academy Of Information And Communication Research, Beijing, China, 2020.

[3] E. Mariconti, L. Onwuzurike, and P. Andriotis, "Mamadroid: detecting android malware by building Markov chains of behavioral models," 2016, https://arxiv.org/pdf/1612.04433.pdf.

[4] M. Lindorfer, M. Neugschwandtner, C. Platzer, and Marvin, "Efficient and comprehensive mobile app classification through static and dynamic analysis," in *Proceedings of the 2015 IEEE 39th annual computer software and applications conference*, pp. 422–433, Taichung, Taiwan, July 2015.

[5] D. Yu and L. Tao, "Android malware detection method based on model base," *Computer Applications and Software*, vol. 39, no. 1, pp. 328–333+338, 2022.

[6] Y. Xu, C. Wu, and K. Zheng, "Computing adaptive feature weights with pso to improve android malware detection," *Security and Communication Networks*, vol. 2017, Article ID 3284080, 14 pages, 2017.

[7] W. Wang, C. Ren, H. Song, S. Zhang, and P. Liu, "Fgl_droid: an efficient android malware detection method based on hybrid analysis," *Security and Communication Networks*, vol. 2022, Article ID 8398591, 11 pages, 2022.

[8] M. Yang, X. Chen, Y. Luo, and H. Zhang, "An android malware detection model based on dt-svm," *Security And Communication Networks*, vol. 2020, Article ID 8841233, 11 pages, 2020.

[9] A. Fatima, R. Maurya, M. K. Dutta, R. Burget, and J. Masek, "Android malware detection using genetic algorithm based optimized feature selection and machine learning," in *Proceedings of the 2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pp. 220–223, Budapest, Hungary, June 2019.

[10] J. Lee, H. Jang, S. Ha, and Y. Yoon, "Android malware detection using machine learning with feature selection based on the genetic algorithm," *Mathematics*, vol. 9, no. 21, p. 2813, 2021.

[11] D. Xiao, B. Liu, B. Cui, X. Wang, and S. Zhang, "Malware prediction technology based on program gene," *Journal of Network and Information Security*, vol. 4, no. 8, pp. 21–30, 2018.

[12] J. Han, S. Zheng, B. Zhao, and W. Sun, "Android malware detection and classification based on software gene," *Application Research of Computers*, vol. 36, no. 6, pp. 1813–1818, 2019.

[13] R. B. Pereira, A. Plastino, B. Zadrozny, and L. H. Merschmann, "Information gain feature selection for multi-label classification," *Journal of Information and Data Management*, vol. 6, pp. 48–58, 2015.

[14] O. Darwish, A. Al-Fuqaha, G. Ben Brahim, and M. A. Javed, "Using MapReduce and hierarchical entropy analysis to speed-up the detection of covert timing channels," in *Proceedings of the 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 1102–1107, Valencia, Spain, September 2017.

[15] L. Jie, C. Jiahao, Z. Xueqin, Z. H. Yue, and L. I. Jiajun, "One-hot encoding and convolutional neural network based

anomaly detection," *Journal of Tsinghua University*, vol. 59, no. 7, pp. 523–529, 2019 Jun 21.
[16] D. Jatnika, M. A. Bijaksana, and A. A. Suryani, "Word2vec model analysis for semantic similarities in English words," *Procedia Computer Science*, vol. 157, pp. 160–167, 2019.
[17] M. M. Haider, M. A. Hossin, H. R. Mahi, and H. Arif, "Automatic text summarization using Gensim Word2Vec and K-means clustering algorithm," in *Proceedings of the 2020 IEEE Region 10 Symposium (TENSYMP)*, pp. 283–286, Dhaka, Bangladesh, November 2020.
[18] Y. Wang and B. Wang, "Android malicious application detection based on deep learning," *Computer engineering and design*, vol. 41, no. 10, pp. 2752–2757, 2020.
[19] O. Darwish, A. Al-Fuqaha, G. Ben Brahim, I. Jenhani, and A. Vasilakos, "Using hierarchical statistical analysis and deep neural networks to detect covert timing channels," *Applied Soft Computing*, vol. 82, Article ID 105546, 2019.
[20] T. T. Wong and P. Y. Yeh, "Reliable accuracy estimates from k-fold cross validation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1586–1594, 1 Aug. 2020.