Research on of Malicious Code Classification Based on Machine Learning

Min Ai[a], Zini Bie[b]

School of Information Safety and Engineering, Zhongnan University Economics and Law, Wuhan 430073, China.

[a]aimin7@163.com, [b]strongbzn@163.com

Keywords: Machine learning, malicious code, CNN, SVM, GRU-SVM

Abstract: With the rapid development of internet technology, malicious code analysis techniques are also developing, resulting in huge challenges for existing malicious code analysis technology. The existing malicious code analysis techniques are mainly divided into static analysis methods and dynamic analysis methods. However static analysis methods often cannot effectively solve the problem of malicious code obfuscation technology, which leads to weak availability of malicious code under static analysis. Compared with static analysis methods, dynamic analysis methods can effectively overcome the confusion of malicious code. However, there are some shortcomings in the dynamic analysis of malicious code: (1) The execution of malicious code is strictly restricted by the environment, and some virtual environments cannot even trigger the execution of code; (2) Each execution of malicious code can only obtain a single execution path; (3) It takes a long time to analyze massive malicious code, and the analysis efficiency needs to be improved. With the continuous development of machine learning, the method of malicious code analysis based on machine learning has received extensive attention. However, the existing machine learning-based malicious code classification method often requires manual design and participation in the feature extraction stage. This requires prior knowledge and cannot automatically learn the characteristics of malicious code, which affects the classification and clustering accuracy of malicious code to a certain extent. Therefore, in view of the shortcomings of the current malicious code analysis method based on machine learning, and the theory or method of machine learning, this paper focus on an in-depth study on the serialization representation of malicious code, static anti-obfuscation of malicious code, and classification methods of malicious code. First, every binary file of the malicious code is processed and converted into a two-dimensional array of n*k, which is a vectored representation of the malicious code. Then, the appropriate machine learning methods are trained to explore a suitable application model for malicious code classification.

1. Introduction

The continuous innovation of Internet technology has promoted the development of malicious code. Among the well-known malicious codes, Trojan horses, viruses, worms and other malicious codes are the most widely spread, causing immeasurable losses to individuals, enterprises and even the government. According to Tencent security 2017 Internet security report, in 2017 alone, the number of PC viruses intercepted nearly 3 billion times, the number of new viruses increased by 8.6% compared to 2016, the first decline in six years. The ransom-like virus is exploding. In 2017 the total number of extortion ransomosis samples reached 6.6 million. Moreover in China alone, there were two large-scale spreads. Nearly 5 millions computer users were attacked by the virus. Compared with the PC, the situation of the mobile terminal is much better but not optimistic. In 2017 alone, the Android virus was detected 1.24 billion times, the new prion was 15.45 million, and the number of infected users exceeded 188 million.

Looking at the impact of malicious code on society, for individuals, the invasion of malicious code can lead to personal privacy exposed to the open internet environment; for corporate, commercial secrets, especially outside the core technology, the leakage will greatly reduce the competitiveness of enterprises and thus have a negative impact on the development of enterprises.

For the country, in the context of the information age, the information security plays an increasingly important role. It has become an important part of the national development strategy. Therefore, it is very practical to study more effective malicious code detection technology.

2. Related work

In the 1980sas the first virus in the true sense, the Apple II [1,2]was born. Since then, the war of malicious code detection and anti-detection has been kicked off. Many scholars have begun to invest a lot of energy into the research of malicious code classification technology. In the study, the existing malicious code analysis techniques can be divided into static analysis and dynamic analysis. Static analysis technology can be applied to malicious code represented by different language levels, mainly divided into feature code based detection methods and behavior based analysis methods. Dynamic analysis technology mainly implements malicious code, and analysts can easily capture the behavior characteristics of malicious code (for example, file operations, network communication traces, etc.). These behavioral features will help malicious code analysts understand the properties of malicious code and then classify the malicious code.

Malicious code classification technology based on static analysis. In the 1990s, as an efficient traditional static analysis method, a code-based malicious code detection method was proposed [3]. This method is widely used to detect known malicious code, and its disadvantage is that it cannot effectively deal with malicious code variants and unknown malicious code. Since then, the researchers have improved this method, and proposed a new method to deal with unknown malicious code and malicious code variants [4]. It’s basic idea is to calculate the similarity between known malicious code and suspicious code, but its detection effect is still not ideal. Ashish et al. [5] proposed a simple, fast, and scalable method for identifying malware and cleaning software based on Windows PE file characteristics, using suspicious number of sections and frequency of function calls. Alazab et al. [6] proposed and evaluated a new method for detecting and classifying zero-day malware using multiple data mining techniques based on the frequency of Windows API calls, with high accuracy and efficiency. In 2011, Nataraj et al. [7] proposed an image-based malware visualization processing method that uses commonly image feature descriptors to describe malicious code behavior characteristics. However, such methods do not extract the characteristics of anti-aliasing malware. In 2015, Qian Yucun et al. [8] proposed a malicious code homology analysis and clustering method based on static analysis according to malicious code behavior characteristics. This method uses Jaccard coefficient to characterize malicious code control flow chart and the similarity of graph node instruction set. The description of the order information of the sequence of malicious code behaviors has been lost to some extent. Lee et al. [9] proposed to extract the malicious code API call sequence based on static analysis and construct a control flow chart based on the extracted sequence information, but the method is susceptible to packing or code confusion.

Malicious code classification technology based on dynamic analysis. M. Bailey et al. [10] proposed clustering of malicious code by analyzing malicious code behavior reports, the downside of which is the lack of external information to guide data analysis. Fredrikson [14], Dai J [15] and others proposed to obtain a malicious code behavior flow by dynamically tracking the API call sequence. The dynamic execution of the malicious code monitoring API call sequence can effectively eliminate the influence of the malicious code obfuscation technology, but is vulnerable to the execution environment, the execution path is single, and in the face of massive malicious code, its efficiency needs to be improved. Kolter et al. [11] classified malicious code based on malicious code behavior, and refined the classification results and studied the semantics of malicious code. Since then, the researchers have proposed a number of malicious code analysis and classification methods based on dynamic extraction API calls. Mohammad et al. [16] tested on the system call log of real malware, combining the powerful statistical pattern analysis capabilities of the Hidden Markov Model with the proven system call capacity as a distinguishing dynamic feature against malware confusion. Tian et al. [12] proposed a method of monitoring system API calls and counting the number of API calls and the frequency of specific API calls. Although this method can match the similarity of the same type of malicious code to some extent, it does not consider the problem of malicious code behavior sequences. Therefore malicious code writers can easily evade analysis by inserting or removing redundant API calls. Qiao Yanchen et al. [13] automated the determination of malicious code homology by obtaining the API calling habits of malicious code, which has a high clustering accuracy. For the processing of API call sequences, the method of malicious code detection based on the common function call sequence pattern effectively improves the detection rate of unknown malicious code.

However, with the continuous development of malicious code back analysis technology, the existing malicious code analysis technology faces enormous challenges: existing static analysis methods often cannot effectively solve the impact of malicious code obfuscation technology, resulting in the weak usability of malicious code under static analysis; compared with the static analysis method, although the dynamic analysis method can effectively overcome the problem of malicious code confusion, the dynamic analysis of malicious code also has the disadvantages of being vulnerable to the environment, obtaining only a single path for each execution, and low efficiency of analysis. With the continuous development of machine learning, malicious code analysis methods based on machine learning have received extensive attention, but traditional machine learning models often require manual design and participation in the feature extraction stage. This requires complete prior knowledge and cannot automatically learn the characteristics of malicious code, which to some extent affect the classification and clustering accuracy of malicious code. In view of this, this paper focuses on the shortcomings of existing malicious code analysis methods. Based on the theory and method of machine learning, firstly, the binary file of malicious code is preprocessed and converted into a two-dimensional array of n*k, i.e. vectored malicious code, then select the appropriate machine learning method to train it, and explore the application model suitable for malicious code classification.

3. Machine learning based malicious code classification technology

The machine learning-based malicious code classification is designed to extract effective features from malicious code, and conduct supervised learning and training based on malicious code features to achieve malicious code homology determination. This paper selects three representative algorithms in the field of machine learning: CNN (Convolutional Neural Network), SVM (Support Vector Machines), and GRU-SVM (Gated Recurrent Units-Support Vector Machines) are used for malicious code classification. By comparing the results of the three models under various evaluation indicators, a model more suitable for malicious code classification is explored.

The implementation flow of each model in this paper is shown in Figure 1: (1) First, the malicious code is preprocessed to obtain vectored malicious code; (2) The vectored malicious code is respectively brought into three classification models to train the classification model; (3) Based on different evaluation indicators, compare the three classification models and select a model more suitable for malicious code classification.

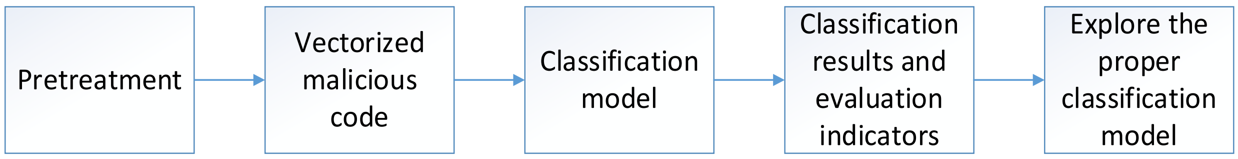
Pretreatment
Vectorized malicious code
Classification model
Classification results and evaluation indicators
Explore the proper classification model

Fig. 1 Model implementation flow chart

3.1. Pretreatment.

Since the static analysis method for malicious code generally analyzes the text of malicious code, dynamic analysis generally analyzes the characteristics of the calling sequence of malicious code, and the malicious code binary file includes the text features and the serialized features of the malicious code. Therefore, to some extent, the analysis of malicious code binary files can extract valid static features and dynamic features in malicious code. In view of this, this article first preprocesses the malicious code binary. The malicious code data set used in this paper is shown in Table 1, consisting of 25 different types of malicious code.

The malicious code data set structure matrixing the binary code of the malicious code executable file is based on the malicious code executable file, and is also a key link to solve the problem of malicious code confusion. Specifically, matrixing a malicious code executable is to represent the entire malicious code executable as a n*k two-dimensional array. Therefore, the general process of preprocessing is shown in Figure 2: First, read the binary code of the executable file, convert each 8-bit binary code into a decimal number as an element in a two-dimensional array, and the range of elements in a two-dimensional array is 0-255. Then, each row of the matrix has k elements, that is, k 8-bit binary codes are converted into k real numbers, and the binary code is continuously converted until the complete malicious code is obtained, and finally the data representation of the whole malicious code is similar to a grayscale image. The data used in this paper is transformed into a 32*32-dimensional array by the above method.

Table.1. Malicious code Data Set

Family name	Adialer.C	Agent.FYI	Allaple.A	Allaple.L	Alueron.gen!J	Autorun.K
NO	122	116	2949	1591	198	106
Family name	C2LOP.P	C2LOP.gen!g	Dialplatform.B	Dontovo.A	Fakerean	Instantaccess
NO	146	200	177	162	381	431
Family name	Lolyda.AA1	Lolyda.AA2	Lolyda.AA3	Lolyda.AT	Malex.gen!J	Obfuscator.AD
NO	213	184	123	159	136	142
Family name	Rbot!gen	Skintrim.N	Swizzor.gen!E	Swizzor.gen!I	VB.AT	Wintrim.BX
NO	158	80	128	132	408	97
Family name	Yuner.A					
NO	800					

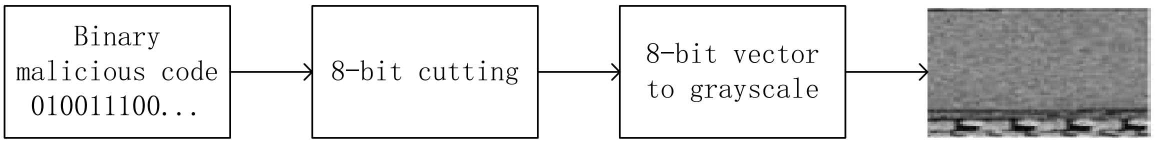

Fig. 2 Pretreatment process

3.2. CNN-based malicious code classification algorithm.

The malicious code executable file matrix is input, wherein the malicious code executable file includes confusing samples, and the CNN model automatically extracts features in the malicious code executable file matrix, thereby avoiding the influence of malicious code confusion on manual analysis. Static classification is achieved by supervised learning to classify malicious code. The CNN classification model is shown in Figure 3:

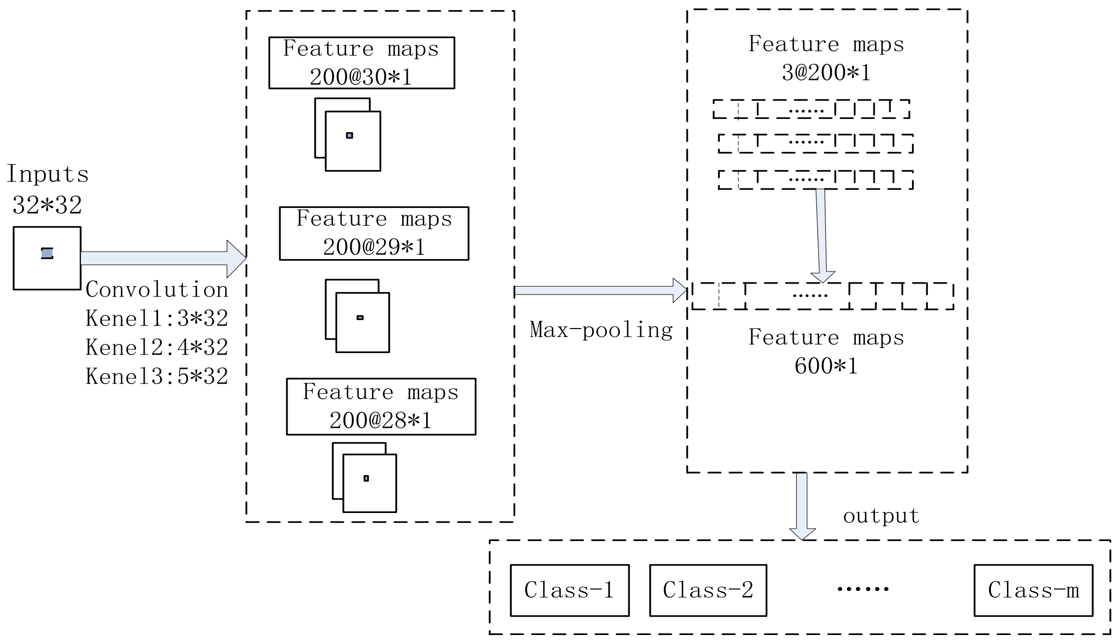

Fig. 3 CNN model

Convolution operations are performed on the original malicious code matrix using convolution kernels of three sizes in the convolutional layer. The width of the convolution kernel *filter_size* is set to 3, 4, and 5 respectively. The convolution kernels of each size have 200 convolution kernels with different initial weights. In the convolution process, set all convolution kernel lengths to k. Finally, a feature vector of (*n-filter_size*+1) dimension is obtained. The convolutional layer calculation formula is shown in equations (1) and (2):

$$h_t = \sum_{i=t}^{t+2} \sum_{j=1}^{k} (s_{ij} * w_{ij}) \tag{1}$$

$$\mathrm{V} = f(H + b) \tag{2}$$

Where k is the convolution kernel length and is also the number of malicious code matrix columns; n is the number of malicious code matrix rows; s_{ij} is the element in the malicious code matrix; w_{ij} is the convolution kernel parameter value; H is the vector obtained by the convolution operation, h_t represents the *t*th element of H, f represents a nonlinear activation function, b represents an offset vector, and V represents the acquired feature vector.

After convolution kernel convolution with length k and width *filter_size*, the real matrix $n*k$ of the original malicious code matrix is transformed into 200 feature vectors of length (*n-filter_size*+1). The model is connected to the pooling layer after the convolutional layer. Using the 1-max pooling layer, for the 600 different feature vectors obtained by the convolutional layer, the maximum value is selected in each feature vector to represent the entire vector to form a new 600-dimensional feature vector. The pooling layer calculation process is as shown in the formula (3), where $\mathrm{v}_i^j (i = 1,2,...,200; j = 1,2,...,n - \mathrm{filter\_size} + 1)$ represents the j-th element value in the i-th (*n-filter_size*+1)-dimensional feature vector obtained by the convolution operation, and $\mathrm{x}_i (i = 1,2,...,200)$ represents the real value obtained by the feature vector mapping after the pooling operation:

$$x_{\mathrm{i}} = \max_{1 \le j \le n - \mathrm{filter\_size} + 1} v_i^j \tag{3}$$

The model is connected to the fully connected layer after the pooling layer. The number of implicit neural nodes in the fully connected layer is m, that is, the number of categories of malicious code family classification. The Softmax classifier is used as the output of the entire CNN classification model to calculate the category probability.

3.3. SVM-based malicious code classification algorithm.

Support Vector Machine (SVM) was proposed in the 1990s. It is a machine learning method based on VC dimension theory and structural risk minimization. When the samples involved in training are limited, it seeks the best compromise between the learning ability and complexity of the model, so that the model has the best generalization ability. In the case of fewer training samples, the SVM seeks to find the optimal hyperplane to separate the different categories. At the same time, when solving the nonlinear problem, it converts the nonlinear problem into a linear problem by introducing a kernel function and then processes it.

The SVM algorithm was originally designed for the binary classification problem. When dealing with multiple types of problems, it is necessary to construct a suitable multi-class classifier. There are two main methods for constructing SVM multi-class classifiers: one-to-many method and one-to-one method. This paper mainly uses a one-to-many method to classify malicious code. In the training, the samples of a certain category are classified into one class, and the other remaining samples are classified into another class, so that the samples of k categories construct k trained SVM models. When classifying, the unknown sample is classified as the one with the largest classification function value. The objective function of the SVM is shown in equation (4):

$$\begin{gathered}\min_{\mathrm{w,b},\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i \\ s.t.\ y_i(w^T x_i + b) \ge 1 - \xi_i, \xi_i \ge 0, i = 1, ..., N\end{gathered} \tag{4}$$

Where w is the weight coefficient vector and b is a constant. C is a penalty coefficient, which controls the degree of punishment for misclassified samples, and plays a role in balancing model complexity and loss error. ξ_i is the relaxation factor, which is used to adjust the number of misclassified samples that exist in the hyperplane to allow for classification.

3.4. GRU-SVM-based malicious code classification algorithm.

GRU is a variant of LSTM, which in turn is a variant of RNN. RNN is often used for text analysis, which can model sequence data, but does not analyze long-dependent text very well. LSTM can solve this defect of RNN well. As a variant of LSTM, GRU simplifies nearly one-third of the parameters on the basis of LSTM, synthesizing the LSTM's forgotten and input gates into a single update gate, as well as mixing cell states and hidden states. Although the GRU model is simpler than the standard LSTM model, the effect is comparable to LSTM. The structure of the GRU is shown in Figure 4 below. z_t and r_t in Figure 4 represent the update gate and the reset gate, respectively. The update gate is used to control the degree to which the status information of the previous moment is brought into the current state. The larger the value of the update gate is, the more the status information is brought in at the previous moment. The reset gate controls how much information is written to the current candidate set $\tilde{h}_t$ in the previous state, the smaller the reset gate, the less the information of the previous state is written. Equation (5) to (8) shows the calculation of z_t, r_t, $\tilde{h}_t$ and h_t, respectively.

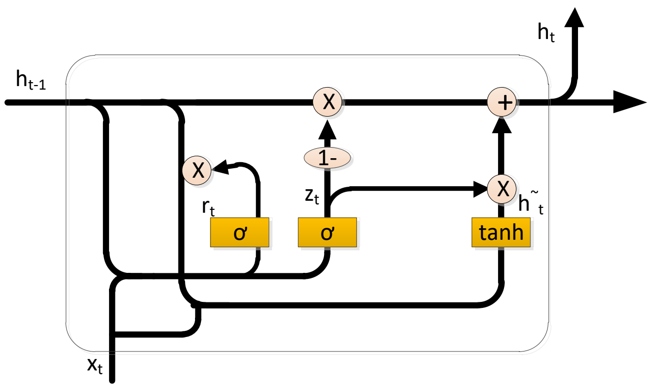

Fig. 4 GRU structure diagram

$$z_t = \sigma(W_z * [h_{t-1}, x_t]) \tag{5}$$

$$r_t = \sigma(W_r * [h_{t-1}, x_t]) \tag{6}$$

$$\tilde{h}_t = \tanh(W * [r_t * h_{t-1}, x_t]) \tag{7}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{8}$$

h_{t-1} represents the output of the previous neuron, x_t represents the input of the current neuron, and W_z represents the weight of the gate. In the GRU-SVM model, the data h_t obtained after GRU processing is no longer using the traditional softmax classifier, but is classified by the SVM to obtain the classification result.

4. Experiment and analysis

4.1. Experiment environment.

The experimental data set was taken from the Malimg data set created by Nataraj et al. The data set consists of 32*32 matrices, a total of 25 categories. Experimental environment: win8 operating system, AMD cpu, 8g memory; PyCharm + python 3.6, using tensorflow 1.12 framework to build the model, using tensorboard to visualize the results.

4.2. Analysis of results.

In the process of training and verifying of malicious code classification model, this paper uses classification accuracy rate *accuracy*, cross entropy loss function value *loss*, *precision*, *recall*, *F1-score*, *confusion matrix*, change rate index *ROC curve* and fuzzy evaluation index *AUC* as evaluation criteria for the model.

4.3. Classification accuracy rate accuracy.

The classification accuracy rate is used to judge the correct rate of the model classification of malicious code, as shown in formula (9), Where *N* is the total number of malicious samples and N_t is the number of malicious code samples correctly classified by the classification model:

$$accuracy = \frac{N_t}{N} \tag{9}$$

4.4. Cross entropy loss function value loss.

The cross entropy loss function is used to describe the error function between the actual output calculated by the model and the standard output. The smaller the function value, the closer the actual output value is to the standard output value. The cross entropy loss function is shown in equation (10):

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k}\{y^{(i)} = j\}\log\frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k}\theta_l^T x^{(i)}}\right] \tag{10}$$

Note l\{true value expression\}=1, l\{false value expression\}=0. m is the number of samples; $x^{(i)}$ represents the input vector of the ith sample; $x^{(i)} = (1, x_1^{(i)}, x_2^{(i)}, \dots, x_p^{(i)})$ is the p+1-dimensional vector; $y^{(i)}$ represents the value of the corresponding category of the ith sample, and k is the total number of sample categories.

4.5. Precision.

The *accuracy rate* indicates the ratio of the number of correct predictions to the total number of samples in the prediction results. For this article, the accuracy rate estimates the proportion of all results predicted to be the correct type of malicious code, as shown in equation (11):

$$TP/(TP+FP) \tag{11}$$

Where *TP* (True Positive) represents the true value is true and the predicted value is true; *FN* (False Negative) represents the true value is false, the predicted value is true; *FP* (False Positive) represents the true value is true, and the predicted value is false; *TN* (True Negative) represents the true value is false, and the predicted value is false.

4.6. Recall.

The *recall rate* indicates how many types of samples have been predicted correctly. For this article, in a malicious code type, the recall rate represents the ratio of the number of samples that are predicted to be correct to the samples that are assigned to this class. As shown in formula (12):

$$TP/(TP+FN) \tag{12}$$

4.7. F1-score.

Both the *accuracy rate* and the *recall rate* are often contradictory, and the *F1 score* is the harmonic average of the two, combining the results of both. As shown in formula (13):

$$F1 = 2TP/(2TP+FN+FP) \tag{13}$$

4.8. Confusion matrix.

The *confusion matrix* is calculated by comparing the position and classification of each measured pixel with the corresponding position and classification in the classified image. The *confusion matrix* can be used to characterize the accuracy of a classifier.

4.9. ROC curve.

ROC focuses on two indicators: *true positive rate* (*TPR = TP / [TP + FN]*) and *false positive rate* (*FPR = FP / [FP + TN]*). Intuitively, *TPR* stands for the probability that the positive case can be divided correctly, and *FPR* stands for the probability that the negative case is divided into positive cases. In *ROC* space, the abscissa of each point is *FPR* and the ordinate is *TPR*, which depicts the trader's trade-off between *TP* (true positive) and *FP* (wrong positive).

4.10. AUC.

The value of *AUC* is the size of the area under the *ROC curve*. Typically, the *AUC* value is between 0.5 and 1.0, with a larger *AUC* representing better performance.

Table.2. Different models' comparative results under various evaluation indicators

Models / Indicators	CNN	SVM	GRU-SVM
accuracy	0.7421	0.9151	0.95
loss	0.7221	0.3743	0.12
AUC	0.97	0.96	0.96

Table.3. Deep Learning Using CNN for Malware Classification Report

	precision	*recall*	*F1-score*
Adialer.C	0.95	0.99	0.97
Agent.FYI	0.99	1.00	1.00
Allaple.A	0.75	0.71	0.73
Allaple.L	0.60	0.85	0.71
Alueron.gen!J	0.92	0.91	0.92
Autorun.K	1.00	1.00	1.00
C2LOP.P	0.30	0.41	0.35
C2LOP.gen!g	0.35	0.29	0.32
Dialplatform.B	1.00	0.99	0.99
Dontovo.A	1.00	1.00	1.00
Fakerean	0.99	0.85	0.92
Instantaccess	0.99	0.99	0.99
Lolyda.AA1	0.97	0.91	0.94
Lolyda.AA2	1.00	0.97	0.99
Lolyda.AA3	1.00	0.64	0.78
Lolyda.AT	0.61	0.50	0.55
Malex.gen!J	0.00	0.00	0.00
Obfuscator.AD	1.00	1.00	1.00
Rbot!gen	0.76	0.76	0.76
Skintrim.N	0.92	0.82	0.87
Swizzor.gen!E	0.41	0.17	0.24
Swizzor.gen!I	0.54	0.15	0.24
VB.AT	0.92	0.99	0.95
Wintrim.BX	0.00	0.00	0.00
Yuner.A	1.00	1.00	1.00
Avg/total	0.77	0.78	0.77

Table.4. Deep Learning Using SVM for Malware Classification Classification Report

	precision	*recall*	*F1-score*
Adialer.C	0.95	1.00	0.97
Agent.FYI	0.97	0.97	0.97
Allaple.A	0.90	0.87	0.89
Allaple.L	0.84	0.87	0.85
Alueron.gen!J	0.98	0.96	0.97
Autorun.K	0.98	1.00	0.99
C2LOP.P	0.75	0.70	0.73
C2LOP.gen!g	0.80	0.72	0.76

Dialplatform.B	0.97	1.00	0.99
Dontovo.A	1.00	1.00	1.00
Fakerean	0.98	0.98	0.98
Instantaccess	0.97	1.00	0.98
Lolyda.AA1	0.97	0.97	0.97
Lolyda.AA2	0.99	0.99	0.99
Lolyda.AA3	0.95	0.98	0.96
Lolyda.AT	0.90	0.92	0.91
Malex.gen!J	0.77	0.82	0.80
Obfuscator.AD	0.91	1.00	0.95
Rbot!gen	0.96	0.76	0.96
Skintrim.N	0.89	0.96	0.92
Swizzor.gen!E	0.72	0.73	0.72
Swizzor.gen!I	0.78	0.71	0.75
VB.AT	0.96	0.96	0.96
Wintrim.BX	0.95	0.92	0.93
Yuner.A	1.00	1.00	1.00
Avg/total	0.91	0.91	0.91

Table 2 shows the results of the malicious code classification method based on the three models under different indicators such as classification *accuracy rate,* loss function value *loss* and fuzzy evaluation index *AUC*. Table 3, Table 4 and Table 5 respectively show the effectiveness of the malicious code classification method based on different models under the *precision, recall* and *F1-score* indicators: the *precision* of CNN is about 0.77, and the *recall* is about 0.78,the *F1-score* is about 0.77; the SVM has a *precision* of about 0.91, the *recall* is about 0.91, and the *F1-score* is about 0.91; the GRU-SVM model has an *precision* of about 0.95, a *recall* of about 0.92, and a *F1-score* of about 0.93.In summary, after analyzing Table 2, Table 3, Table 4 and Table 5, we can know that among the three different classification algorithms the GRU-SVM model performs best in malicious code classification that not only its *loss* value is low, the *precision* and the *recall* and *F1-score* also perform best in the three classification algorithms.

Table.5. Deep Learning Using GRU - SVM for Malware Classification Report

	precision	*recall*	*F1-score*
Adialer.C	0.25	0.99	0.40
Agent.FYI	0.99	0.98	0.99
Allaple.A	0.93	0.94	0.93
Allaple.L	0.94	0.87	0.90
Alueron.gen!J	0.99	0.95	0.97
Autorun.K	0.99	0.82	0.90
C2LOP.P	0.96	0.82	0.88
C2LOP.gen!g	0.97	0.82	0.89
Dialplatform.B	0.99	0.97	0.98
Dontovo.A	1.00	0.98	0.99
Fakerean	0.99	0.95	0.97
Instantaccess	1.00	0.98	0.99
Lolyda.AA1	0.98	0.94	0.96
Lolyda.AA2	0.99	0.94	0.97
Lolyda.AA3	1.00	0.97	0.99
Lolyda.AT	0.99	0.94	0.96
Malex.gen!J	0.98	0.72	0.83

Obfuscator.AD	1.00	0.98	0.99
Rbot!gen	0.99	0.93	0.96
Skintrim.N	1.00	0.93	0.96
Swizzor.gen!E	0.96	0.80	0.87
Swizzor.gen!I	0.97	0.79	0.87
VB.AT	0.99	0.97	0.98
Wintrim.BX	0.99	0.83	0.90
Yuner.A	0.98	0.99	0.98
Avg/total	0.95	0.92	0.93

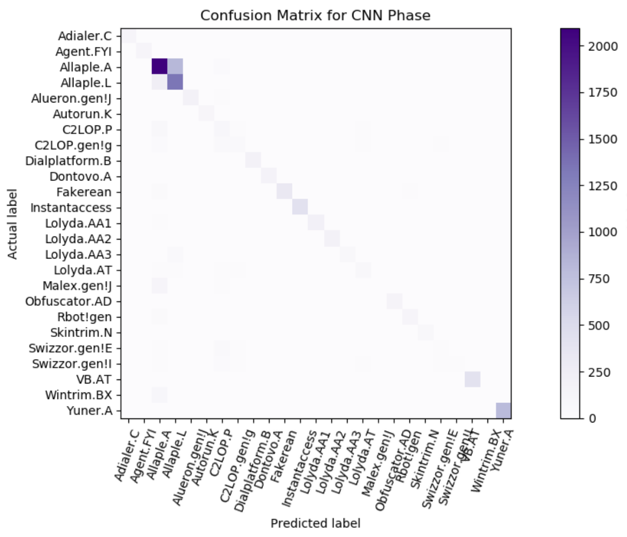

Fig. 5 CNN performance in the confusion matrix

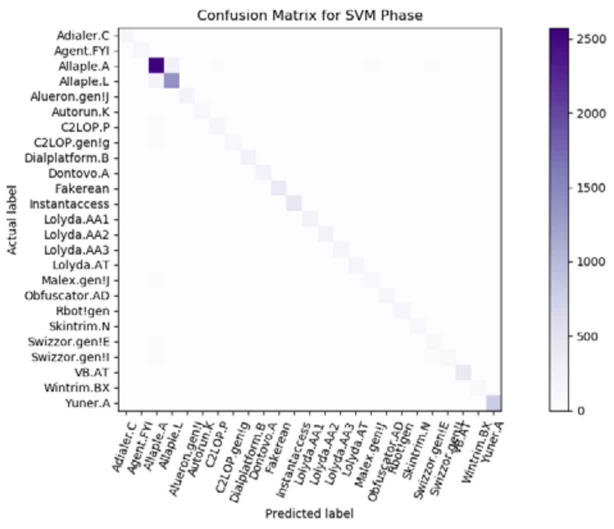

Fig. 6 SVM performance in the confusion Matrix

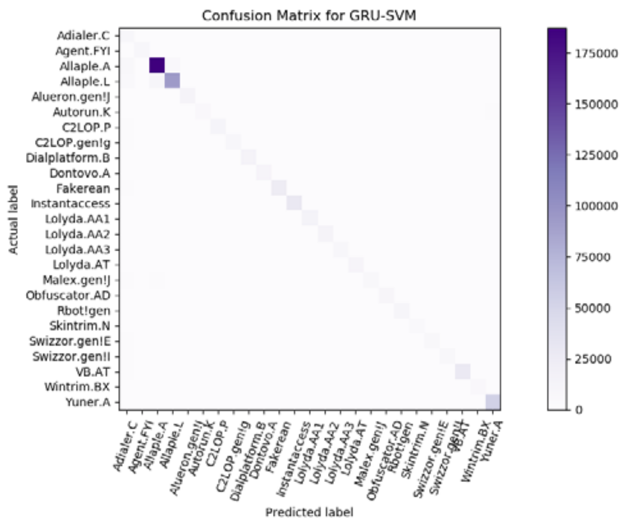

Fig. 7 GRU-SVM performance in the confusion matrix

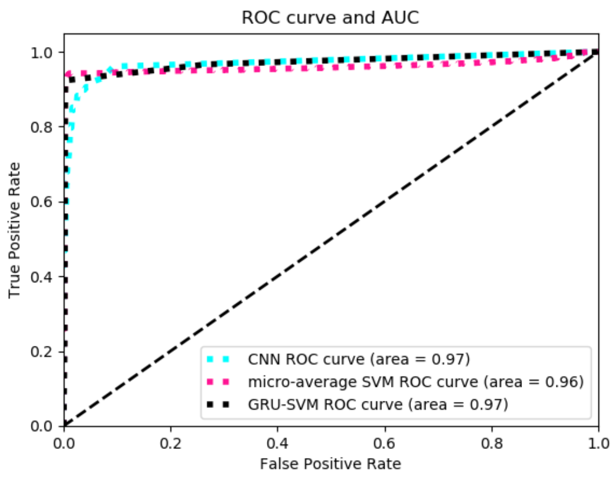

Fig. 8 ROC curve and AUC of CNN and SVM and GRU-SVM

Figure 5, Figure 6, and Figure 7 show the performance of each classification algorithm in the confusion matrix. As can be seen from Table 3 and Figure 5, CNN does not ideally classify these malicious codes such as Allaple. A, Allaple. L, Lolyda. AA3, Lolyda. AT, C2LOP.PC2LOP. gen!g, Swizzor. gen!E, Swizzor.gen!I, etc. After analyzing the big categories they belong to, we know that Allaple.A and Allaple.L belong to one category and Lolyda.AA3 and Lolyda.AT belong to the same category, similarly, C2LOP.P and C2LOP.gen!g, Swizzor.gen!E and Swizzor.gen!I belong to the same class. It can be seen that CNN is not ideal for classifying malicious codes that belong to a same large class. Table 4 and Figure 6 shows that compared with CNN, SVM greatly improves the classification effect of malicious code belonging to the same category. For example, for the classification effect of Allaple. A, AllapleL, CNN can only reach between 0.30 and 0.40, while SVM can reach between 0.70 and 0.80. It can be seen that SVM has better classification effect than CNN. It can be seen from Table 5 and Figure 7 that compared with SVM, GRU-SVM has better classification effect. In addition to inheriting SVM's advantage of better effect on classifying malicious code belonging to a large class, GRU-SVM's classification accuracy rate *accuracy* and other evaluation indicators are higher than SVM. It can be seen that GRU-SVM can better extract the characteristics of vectored malicious code binary files.

As can be seen from Figure 8, the *AUC* performance of the SVM is not as good as that of CNN and GRU-SVM, so we can know that the SVM classification performance is not ideal compared to the other two classifiers. The three *ROC curves* show that the performance of the three classifiers is similar, but the performance of CNN is relatively less than ideal. Combining the performance of the three classifiers in ROC and AUC, it can be seen that GRU-SVM performs best.

5. Summary

This paper implements three models CNN, SVM and GRU-SVM to classify malicious code. Taking the binary code matrix of the malicious code executable file as input, the CNN model can automatically extract the malicious code features and semantics, effectively solve the malicious code confusion problem in static analysis, and implement the family classification of malicious code according to the automatically extracted feature vector, obtain new feature vectors through convolutional layer and pooling layer, and train the model through self-feedback and adjustment of the neural network to improve classification. The GRU model is generally used for long text processing. It can retain important features through various Gates to ensure that it will not be lost when long-term propagation. Therefore, GRU is similar to CNN and can effectively solve the confusion problem in the static analysis of malicious code. At the same time, the GRU has the feature of processing long texts, and the GRU can solve the time serialization feature of the malicious code binary files that the CNN cannot handle. Therefore, in the feature extraction stage of the malicious code binary file, the GRU can better extract the binary code’s important features. It can be seen from the experimental evaluation indicators that SVM can analyze the important features of malicious code more than CNN, and thus obtain better classification results. GRU-SVM combines the characteristics of GRU and SVM, and the malicious code binary file is processed by the GRU, and the extracted important features are used as input of the SVM, and the data is classified by the SVM. From the above experimental results, it can be seen that compared with CNN and SVM models, GRU-SVM performs best in terms of *precise, accuracy, recall* , or in terms of *confusion matrix, F1-score, AUC,* etc. Therefore, among the three classification models CNN, SVM, and GRU-SVM, GRU-SVM is more suitable for classification of malicious code.

Although this paper explores a malicious code classification model based on machine learning, but the classification accuracy of this classification model is about 96%, compared with other machine learning methods, there is still room for improvement. And the input to the model is just the result of matrixing the malicious code binary, without extracting the features of other malicious code as input, so there is still a lack of malicious code preprocessing, and later work will focus on the research of malicious code feature extraction. In addition, this paper only studies the malicious code classification based on CNN model, SVM model and GRU-SVM model, and has not tried other machine learning. The method is therefore not comprehensive enough in terms of model comparison.

References

[1] Ed S, Lenny Z., Decisive battle malicious code, Electronic Industry Press, Bei Jing, 2005,pp.1-10.

[2] Pengbo Zhu:Research on Malicious Code Detection Technology Based on Machine Learning Algorithm (2018).

[3] Lo R W, Levitt K N, Olsson RA, MCF: a malicious code filter, Computers & Security.14 (1995)541-566.

[4] Sung A H, Xu J, Chavez P, et al,Static analyzer of vicious executables (SAVE),Computer Security Applications Conference,2004,pp.326-334.

[5] Ashish Saini, Ekta Gandotra, Divya Bansal Sanjeev Sofat, Classification of PE Files using Static Analysis, ResearchGate. DOI: 10.1145/2659651.2659679.

[6] Alazab M, Venkatraman S, Watters P, et al,Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures,In Proc. Australasian Data Mining Conference .(2011).

[7] L Nataraj,S Karthikeyan, G Jacob,Malware Images: Visualization and Automatic Classification. DOI: 10.1145/2016904.2016908.

[8] Yucun Qian,Guojun Peng,Ying Wang,et al,Malicious code homology analysis and family clustering,Computer Engineering and Applications,2015,51 (18),pp.76-81.

[9] Lee J,Jeong K,Lee H,Detecting metamorphic malwares using code graphs, ACM Symposium on Applied Computing.DBLP,2010,pp.1970-1977.

[10] Bailey M, Oberheide J, Andersen J, et al.Automated classification and analysis of internet malware,International Conference on RecentAdvances in Intrusion Detection. Springer-Verlag,2007pp.178-197.

[11] Fredrikson M,Jha S,Christodorescu M,et al,Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors,IEEE Symposium on Security and Privacy. IEEE Computer Society,2010pp. 45-60.

[12] Dai J,Guha R,Lee J,Efficient Virus Detection Using Dynamic Instruction Sequences,Journal of Computers,2009,4 (5).

[13] Kolter J ZMaloof MA,Learning to Detect and Classify Malicious Executables in the Wild,Journal of Machine Learning Research,20067(4),pp.2721-2744.

[14] Imran M,Afzal M T,Qadir M A, Using hidden markov model for dynamic malware analysis: First impressions,International Conference on Fuzzy Systems & Knowledge Discovery. IEEE (2016).

[15] Tian R,Islam R,Batten L,et al,Differentiating malware from cleanware using behavioural analysis,International Conference on Malicious and Unwanted Software. IEEE,2010pp.23-30.

[16] Yanchen Qiao,Xiaochun Yun,YongZheng Zhang,et al,Automatic code homology determination method based on calling habit,Electronic Journal,2016,44 (10),pp.2410-2414.