Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 3, March 2019, pp. 772-781
DOI: 10.23940/ijpe.19.03.p6.772781

Cuckoo-based Malware Dynamic Analysis

Lele Wang[a], Binqiang Wang[a], Jiangang Liu[b], Qiguang Miao[c], and Jianhui Zhang[a,*]

[a]*National Digital Switching System Engineering and Technological Research Center, Zhengzhou, 450002, China*
[b]*Nanjing Information Technology Institute, Nanjing, 210000, China*
[c]*Department of Computer Science, Xidian University, Xi'an, 710071, China*

Abstract

Aiming at the problems of the huge number of malware currently in the big data environment, the insufficient ability of automatic malware analysis available, and the inefficiency of the classification of malicious attributes, in this paper, we propose a Cuckoo-based malware dynamic analysis system that can be extended, analyzed quickly, and has application value. The system proposes a semantic feature model based on deep learning, uses a deep recursive neural network model to describe the multi-layered aggregation relationship of program semantics, and builds a malware semantic aggregation model. The model can automatically complete the acquisition and analysis of behavioural features of unknown program samples and perform attribute discrimination on unknown program samples efficiently and accurately.

Keywords: cuckoo; dynamic analysis; deep learning

(Submitted on October 20, 2018; Revised on November 21, 2018; Accepted on December 23, 2018)

1. Introduction

The rapid development of Internet and mobile terminal technologies has caused Internet security to face an unprecedented and severe situation. Malicious programs are the primary threat to the Internet system. A malicious program, so-called malware, is a program with attack intention. Software that can cause damage to users, networks, computers, or other devices can be called malicious programs. Malware mainly includes viruses, worms, Trojans, etc. In 2017, the National Internet Emergency Center (CNCERT) obtained more than 2.53 million mobile Internet malicious programs through self-capture and exchanging with vendors, an increase of 23.4% over the same period of last year. The number of malware has maintained a high-speed growth trend for nearly eight years, as shown in Figure 1. [1] Attackers use techniques such as packer, anti-debugging, anti-tracking, and anti-virtualization to evade inspections, making it extremely difficult to extract malicious malware features from massive suspicious files, which are great challenges to the traditional malware analysis method.

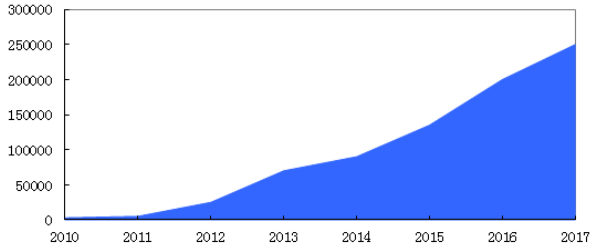

Figure 1. Trends in the number of mobile Internet malware captures in 2010-2017

Traditional methods of malicious program analysis cannot effectively guarantee the security of computer systems and the Internet; the main reasons are motivated by the following tips:

* Corresponding author.
E-mail address: ndsczjh@163.com

(1) The current number of malware is numerous, and the design of it is so exquisite that the traditional method of relying on manual analysis of massive data seems to be incapable.

(2) The rapid update of malware and the numerous attack strategies used by attackers make it easy to miss new attack features by matching the known features, which leads to the failure of malware analysis to meet the expected performance requirements.

With the development of virtual machine technology, the malware dynamic analysis system has entered a new stage. We build a safe and controllable operating environment for the operation of malware by using virtual machine technology. By monitoring its running process and capturing the information of the malware runtime, we judge the maliciousness according to the behavior of the program at runtime. The above method is considered the most effective method for malware analysis. At the same time, using the deep learning [2] method to automatically mine the deep features of malware will greatly improve the efficiency and accuracy of malware analysis.

2. Related Work

The traditional methods of malware analysis are mainly static analysis and dynamic analysis. Static analysis refers to statically scanning and analyzing the source code of a sample program without running the unknown sample program, including PE file header [3], plaintext strings in the program, and disassembly code information of the program. By these analyses, we determine the behavior and maliciousness of the program. The advantage of this method is that it can analyze multiple executable paths of the program, but it requires manual participation and high professional skills. Moser's research shows that static analysis cannot correctly classify such malicious code due to the use of new code obfuscation techniques [4]. Without running the sample program, static analysis can hardly obtain the interactive information between the sample runtime and the system, resulting in little information that can be extracted from automated analysis.

Dynamic analysis, so-called behavior analysis, refers to the running of a sample program in a controlled security environment, the monitoring and tracking of the execution and operation status of the program during the process runtime, the capture of the progress of the program, the use of system resources, and the flow of data flow [5]. On this basis, we can build a sandbox isolated from the host environment. In this sandbox, we run the client system and execute the program, and the monitoring program automatically monitors the running of the program and records its API call information. Bayer et al. [6] used a simulation technique to design a sandbox called TTAnalyze to run the sample and analyze the sample execution flow information; Willems et al. [7] realized the sandbox called CWSandbox with virtualization technology.

In the malware dynamic analysis system, the extraction of program behavior characteristics is one of the focuses of researchers. Ahmed [8] proposed to analyze the API call sequences when the program is running, and the API call sequences are used as the input of machine learning algorithm. Quellette et al. [9] obtained the characteristics of the sample program by analyzing the control flow graph when the program is running and used the deep probabilistic model to compare the similarity between the unknown samples and the representative sample features in known malicious programs. Wang [10] proposed a semantics-based malware behavior signature extraction and detection method to obtain malicious code behavior characteristics with strong anti-interference ability.

In recent years, researchers have proposed introducing deep learning techniques to malware analysis. Li [11] proposed a hybrid malicious code detection method based on auto encoder and Deep Belief Network (DBN). The auto encoder is used to reduce the dimension of data, and the deep belief network is applied to malicious code detection. Li et al. [12] proposed to apply the deep belief network in deep learning to extract the network features and classify the feature data by the softmax classifier to achieve the detection purpose.

Although these common methods of malware analysis have achieved certain results, there are still some inadequacies, and the following two main problems should be considered.

(1) The existing behavior feature extraction method for solving the malware dynamic analysis system is not effectively solved in terms of the feature representation of malware. The feature extraction method can neither maintain malicious semantics nor describe the aggregation and delivery structure of malicious semantics, which seriously restricts the performance of malware analysis.

(2) In the face of real-time massive malware, machine learning algorithms have good performance on small-scale

problems, but it is difficult to obtain models with sufficient performance in the current large-scale malware analysis problems. As the number of malware increases and its design becomes increasingly sophisticated, the time it takes for malware analysis is becoming longer and longer, resulting in a decline in the performance of malware analysis.

3. System Design

3.1. System Architecture

In view of the inadequacies in malware analysis, in this paper, our focus is on API information acquisition, malware behavior abstraction, and semantic aggregation. We automatically learn characteristics from a limited malware sample and construct a classification model with learning and evolution ability to realize the classification task of the unknown program sample. Designing the corresponding automated analysis system requires focusing on two aspects. The first aspect is how the system realizes automation to analyze the maliciousness of unknown programs and obtains feature information that reflects the nature of malware, thereby providing support for further analysis. The second aspect, in the face of the massive malware that appear in real time, is how the system can efficiently and accurately judge the attributes of unknown programs and complete the classification task for unknown sample programs.

In view of these two inadequacies, we conduct a more in-depth study on the analysis of malware from the aspects of API information acquisition, behavior abstraction, and semantic aggregation. In the big data environment, we design a Cuckoo-based malware dynamic analysis system.

Figure 2 is a schematic diagram of the overall framework of the system. The operation control module maintains a normal operation of the system and is responsible for monitoring the behavior of the malware samples during operation and various resources required for managing the system operation. The behavior abstraction module is responsible for abstracting the behavior of the sample runtime from the results of the dynamic analysis and completing the semantic feature construction based on deep learning. The semantic aggregation module describes the multi-layered aggregation relationship of malware semantics through the deep recurrent neural network model. The program analysis module is responsible for accurately estimating the degree and the type of program maliciousness based on the semantic aggregation model.

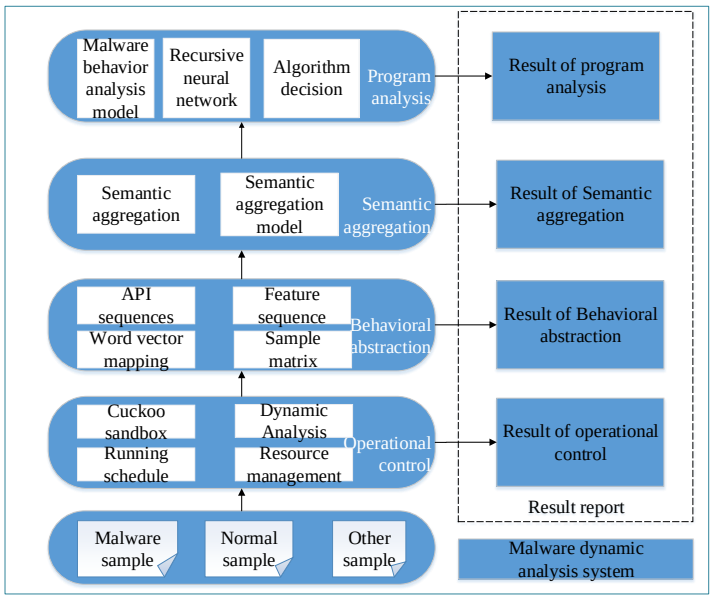

Figure 2. System architecture

3.2. The Operation Control Module

The operation control module is the basis of the malware dynamic analysis system. The main function of it is to let malware run in the controlled environment. It can automatically analyze the sample program and capture the API calls and parameters of the specific process. It can also capture the process loading module and related kernel data maintained by the operating system.

The monitoring of malware is implemented by calling the sandbox. Two virtualized technologies can be used to build the sandbox system. The main advantages and disadvantages are as follows (Table 1).

Table 1. Comparison of two virtualization technologies

	Virtual Machine	Simulator
Representative products	VMware, Virtual Box [13]	QEMU [14]
Characteristics	Simulate an independent and complete computer system that is completely isolated from the host system and has a complete hardware system	Simulate all hardware functions of the system
Differences	Most of the instructions are executed by the real hardware device, and only a small number of sensitive instructions need to be executed by the virtual machine monitor in a restricted environment	All instructions will be translated by the simulator's monitoring module

Considering the operational efficiency, we choose to use the virtual machine to build the sandbox system. Specifically, we use the Cuckoo platform to monitor the running of malware. The architecture of the Cuckoo platform is shown in Figure 3.

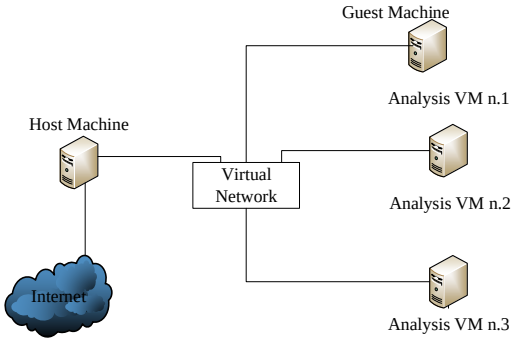

Figure 3. Cuckoo Sandbox architecture diagram

Cuckoo Sandbox [15] consists of a Cuckoo host machine and some guest machines. The guest machine communicates with the host machine through a virtual network. The core structure on the host machine is a hypervisor that manages these guest machines in a range of different environments. The host machine is responsible for scheduling, initiating analysis, monitoring malicious behavior, and generating analysis reports, and the guest machine has a virtualized, restricted environment for running, analyzing malware, and then transmitting the analysis results back to Cuckoo through the virtual network. The working process of Cuckoo Sandbox is shown in Figure 4.

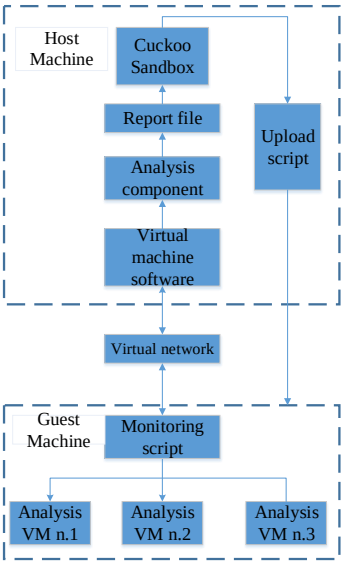

Figure 4. The working process of Cuckoo Sandbox

First, start the Cuckoo Sandbox main program and guest machines. The Cuckoo Sandbox injects the monitoring script file into the client, and the sample is executed in the client. The monitoring script records several information during the execution of the sample. After the execution is completed, the monitoring script sends the recorded result back to the Cuckoo where the virtual machine software is located through the virtual network, and the analysis component in the host analyzes and generates a report file. Finally, the virtual machine software uses the snapshot to restore the virtual machine in the guest machine to its original state.

The result of the operation monitoring is a function call sequence (API List), which includes the API sequences and parameter values called when the malware runs. The result will be an input file of the behavior abstraction module.

3.3. The Behavior Abstraction Module

The deep learning algorithm is used to solve the malware analysis problem. The behavior abstraction module is a key module of the malware analysis system. The so-called deep learning uses a multi-layer neural network to achieve the function of machine learning. Therefore, the use of neural networks for behavior abstraction of sample programs requires the conversion of the program to be tested into an acceptable form of input for the neural network, which is mainly done in two steps.

The first step is to solve the behavior abstraction problem. After the malware sample is analyzed by Cuckoo Sandbox, the API sequence called by it is obtained. Although the call sequence contains a lot of information related to malware behavior, the level of abstraction of the API sequence is too low, there is a large amount of redundant information, and the data format is also inconsistent. The API call sequence needs to be abstracted into an easily processed data form, which can be used by the algorithm.

There are four commonly used behavior abstract methods. The comparison of the principles, advantages, and disadvantages is shown in Table 2.

Table 2. Comparison of two virtualization technologies

Methods	Principles	Advantages and Disadvantages
n-gram [16-18]	The API sequence or instruction sequence is converted into many grams, and combined with commonly used feature selection methods to obtain more grams	The model learning results can be directly mapped to high-dimensional feature space vectors and applied to machine learning algorithms. Incomplete description of the program behavior results in unsatisfactory detection results
Control flow graph and data flow graph	Describe the program's execution path and memory access trajectory with a formalized language	Taking the lowest instruction stream of the system as the analysis data, what the behavior abstraction and behavior description obtain are still the lower-level behavior
API data dependency graph [19-22]	The input parameter of one API may be the return value or output parameter of another API, and multiple API functions are combined in a certain data dependency to achieve a complete logical and interpretable program behavior	The time complexity and space complexity of the graph algorithm are usually high, making it difficult to process massive samples
The minimal security-relevant behavior abstraction [23]	Centering on the sensitive resources of the system, the parameter dependence analysis of the API sequence is carried out	Abstract all successful and complete operations on the sensitive recourses of system

In this paper, the minimal security-relevant behavior abstraction is used to extract the features of the program runtime, and the function information in the API list is abstracted into behavior information that can represent the meaning of the function operation. The input of the API List file is the first step of behavior abstraction, and the output is the feature sequence.

The second problem to be solved is the need to convert the sequence of features obtained from behavior abstraction into input forms acceptable to the neural network. The method adopted in this paper is to map the feature sequence obtained from behavior abstraction into a fixed length word embedding and synthesize the word embedding into a matrix according to the calling sequence of the feature sequence. The matrix is the input matrix of the neural network. This is the basis for the use of deep learning algorithms.

This method of mapping the feature sequence of the sample program into a word embedding not only preserves the semantics of the behavioral elements in the vector space, but also reflects the similarity between the various behavioral elements.

There are two main ways to represent the word embedding. One is one-hot representation, which is to represent each word as a very long vector. This representation makes any two word vectors independent. We do not know whether the two words are related if we only know the two words.

The other way is distributed representation, which was first proposed by Hinton in 1986 [24]. The basic idea of it is to map each word in the language to a fixed-length short vector by training. Then, put these vectors together to form a word embedding space. Word embeddings are not directly related to semantics, but by mapping the feature sequences of sample programs into word embedding, the processing of feature sequences can be transformed into operations on vectors in vector space, and the similarity of vectors is used to represent the similarity between feature sequences.

When using word embedding, we must mention the statistical language model. So far, the acquisition of word vectors has been based on the training of language model. We use mathematical methods to describe language models: given a string $S=\{w_1 w_2 \cdots w_t\}$, judge the probability $P(S)$ that it belongs to natural language. Commonly used language models are all approximate solutions to $P(w_t \mid w_1, w_2, \cdots, w_{t-1})$, which can be obtained from the simple inference:

$$P(w_1, w_2, \cdots, w_t) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \cdots \times P(w_t \mid w_1, w_2, \cdots, w_{t-1})$$

Mnih and Hinton first extended the deep learning theory to natural language processing, proposed a "log-bilinear" model [25], and later proposed a "hierarchical log-bilinear" model of hierarchical thought in 2008. The model improves the efficiency on the basis of ensuring the training effect.

In this paper, the Hierarchical Log-bilinear Language Model (HLBL) [26] is used to map the word embedding, the binary tree with the feature sequence as the leaf node is established, and the linear structure is replaced by the tree structure. The HLBL model is trained by using the feature sequences of all training samples as a corpus, and each feature sequence is mapped into a fixed-length word embedding. According to the order of API calls, the word embedding is combined into a matrix, which is used as an input matrix for solving malware analysis using neural networks.

The goal of the solution is to find the corresponding word embedding representation of every word v_i in the word embedding matrix and then associate it with the current word (v_3 in Figure 5) through the link matrix, which is represented by the energy function.

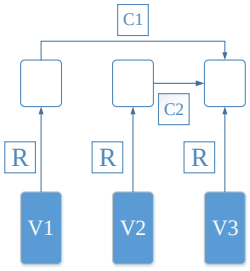

Figure 5. Log bi-linear model schematic

The log bi-linear model can be written in the form of a neural network:

$$h = \sum_{i=1}^{t-1} H_i C(w_i) \tag{1}$$

$$y_j = C(w_j)^T h \tag{2}$$

Combine Equations (1) and (2) into $y_j = \sum_{i=1}^{n-1} C(w_j)^T H_i C(w_i)$. $C(w)$ is the word embedding corresponding to the word w. Models like $x^T My$ are called bilinear models. In Equation (1), h represents a hidden layer directly with semantic information, and the dimension of the hidden layer is m, which is consistent with the dimension of the word embedding. The matrix $H_i = m \times m$ represents the contribution of the i^{th} word to the t^{th} word after a transformation of H_i. In Equation (2), y_j is equal to the inner product of $C(w_j)$ and the predicted word vector h, which can reflect the similarity between the two, and it directly indicates the predicted log probability that the next word is w_j.

Using the above two-step behavior abstraction method, not only can the feature of malware suitable for the deep learning algorithm be extracted from the API call sequence, but also the extracted program features can maintain the semantics of the malware in the vector space. The method can also provide an ideal data foundation for subsequent behavior analysis.

3.4. The Semantic Aggregation Module

Traditional malware analysis methods, after the end of behavior abstraction, usually use the machine learning algorithm model to complete the detection, clustering. and classification tasks, regardless of the characteristics of the malware analysis problem itself. In fact, the semantics of malware (in Figure 6) are aggregated layer by layer from low to high, usually including the following levels:

(1) Handle semantic aggregation of the bottom level semantic elements: usually, some APIs associated with the same handle constitute an aggregation relationship, completing a single atomic operation for a particular system resource.

(2) The resource manipulation aggregation of the middle level semantic elements: the aggregation relation made up of a specific system resource, which is composed of the operating process within the lifetime of the malware, represents the complete operation intention of a system resource.

(3) Business logic aggregation of the high-level semantic elements: the design of malware must be based on one or more specific business logics, that is, attack intents. These business logics span multiple system resources and are aggregated in specific timing relationships.

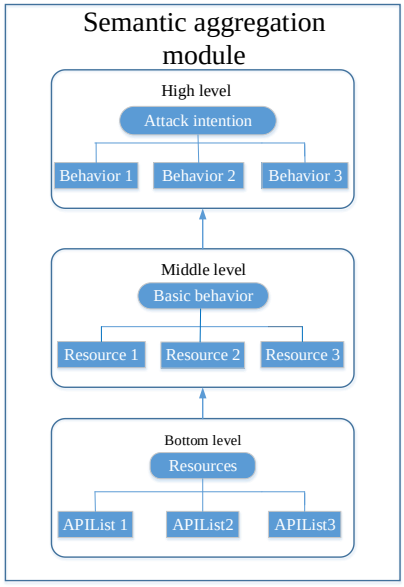

Figure 6. Three levels of semantic aggregation

This multi-level semantic aggregation and transfer relationship has a tree-like recursive structure, similar to the combination of words in a neural network language model. Recursive Neural Network (RNN) [27] can be used to construct the malware semantics aggregate model. In the process of semantic aggregation, it is not combined in the order of words but

is based on the degree of approximation of semantics and the first combination of semantic similarities. In this way, the semantic aggregation is gradually completed from the beginning of the word until the semantics of the whole sentence is obtained, and a bottom-up tree structure is formed in the process of combination.

In the recursive neural network, besides obtaining the combination meaning of word combination, it is necessary to ensure that there are not too many parameters nor too much calculation. Therefore, in addition to the meaning of the vector descriptor, each node uses a matrix to describe how the word affects the meaning of its adjacent words. The concept of tensor is introduced to represent the product of the vector and the matrix. The tensor-based combination function is used to replace the original linear function, and all nodes share the same tensor parameters. Then, it can reduce the overall number of parameters and the amount of calculation.

In this paper, we use the Recursive Neural Tensor Network (RNTN) [28] to construct a malware semantic aggregation model. The structure of the single-layer recursive tensor neural network is shown in Figure 7. Each dashed box represents one of the d slices ($V^{[1:d]}$), and the synthesis process of word embedding (a,b,c) is performed by the combination function based on the tensor product, as shown in Figure 8.

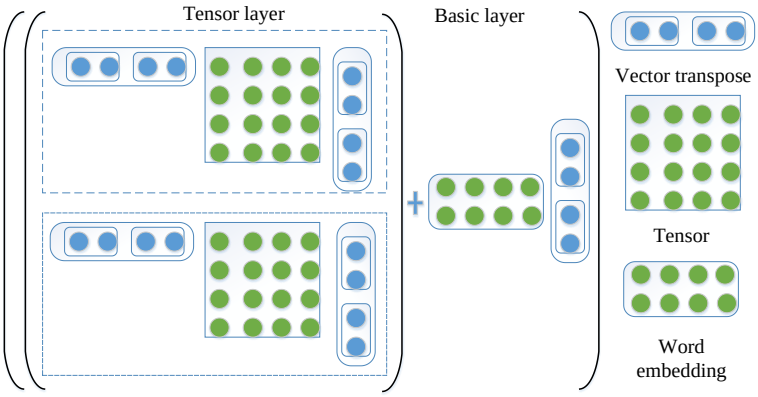

Figure 7. Structure of the Recursive Neural Tensor Network

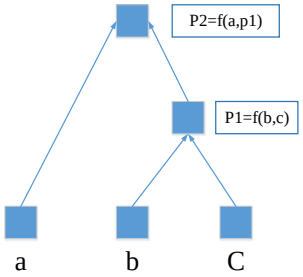

Figure 8. Schematic diagram of word embedding synthesis process

Calculate P_1 :

$$p_1 = f\left\{\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}\right\}$$

Calculate P_2 :

$$p_2 = f\left\{\begin{bmatrix} a \\ p_1 \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} a \\ p_1 \end{bmatrix} + W \begin{bmatrix} a \\ p_1 \end{bmatrix}\right\}$$

Where $V^{[1:d]} \in R^{2d \times 2d \times d}$ is a tensor that defines multiple bilinear forms, W is a weight, and f is an activation function.

3.5. The Program Analysis Module

The classification of malware is based on the estimation of maliciousness in the semantics of the program and the analysis of malicious categories. In this paper, the hierarchical logarithmic bilinear model is used to map the word vectors and extract the malware features that maintain the semantic relationship. Based on this feature representation, the semantic aggregation model of malware is established by the recursive tensor neural network model, and the degree of maliciousness and the transfer relationship of categories in the multi-layer semantic aggregation process are studied in detail. The feature extraction, neural network structure, and neural network output correspond to the similarity, aggregation structure, and semantic category of malware semantics, respectively. Therefore, this model can be used to accurately estimate the degree and the type of maliciousness of the program, thus completing the semantic-based malware classification detection task. Figure 9 is the analysis process of the malware dynamic analysis system.

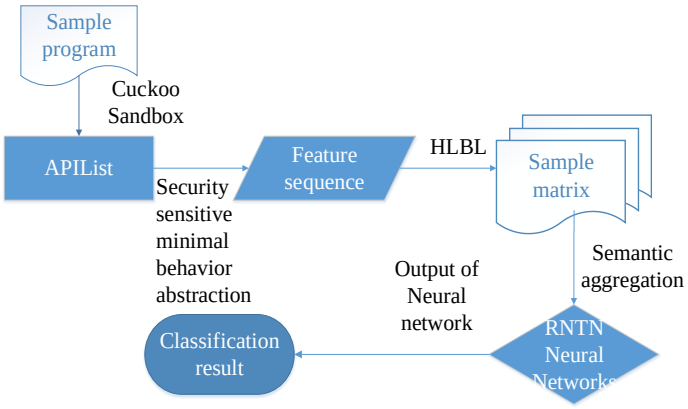

Figure 9. Analysis process of malware dynamic analysis system

4. Conclusions

In this paper, in the analysis of traditional malware, there are problems of insufficient feature extraction of automatic extraction programs and low efficiency of malicious attribute determination. Through the in-depth study of massive malware samples, a dynamic analysis system based on Cuckoo malware is proposed to solve the problem of semantic loss in the extraction of traditional malware. This thesis uses a recursive tensor neural network to establish a multi-layer semantic aggregation model of malware and describes the multi-layer semantic aggregation and transmission relationship of malware, which provides a basis for dynamic analysis of malware. After testing, the system worked well and achieved the desired design results. In the next step, we will focus on exploring more efficient deep learning model training methods to further improve the training speed and ability to respond to malware.

References

1. J. R. Xue, J. W. Fang, and P. Zhang, "A Survey of Scene Understanding by Event Reasoning in Autonomous Driving," *International Journal of Automation and Computing*, Vol. 15, No. 3, pp. 1-18, 2018
2. G. Hinton, S. Osindero, M. Welling, and Y. W. Teh, "Unsupervised Discovery of Nonlinear Structure using Contrastive Backpropagation," *Cognitive Science*, Vol. 30, No. 4, pp. 725-731, 2006
3. J. R. Bai, J. F. Wang, and Z. Q. Zhao, "Malware Detection Approach based on Structural Feature of PE File," *Computer Science*, Vol. 40, No. 1, pp. 122-126, 2013
4. A. Moser, C. Kruegel, and E. Kirda, "Limits of Static Analysis for Malware Detection," in *Proceedings of Twenty-third Annual Computer Security Applications Conference*, pp. 421-430, 2007
5. M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A Survey on Automated Dynamic Malware-Analysis Techniques and Tools," *ACM Computing Surveys*, Vol. 44, No. 2, pp. 1-42, 2012
6. U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic Analysis of Malicious Code," *Journal in Computer Virology*, Vol. 2, No. 1, pp. 67-77, 2006
7. C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis using CWsandbox," *IEEE Security & Privacy*, Vol. 5, No. 2, pp. 32-39, 2007
8. F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using Spatio-Temporal Information in API Calls with Machine Learning Algorithms for Malware Detection," in *Proceedings of ACM Workshop on Security and Artificial Intelligence*, pp. 55-62, 2009
9. J. Ouellette, A. Pfeffer, and A. Lakhotia, "Countering Malware Evolution using Cloud-Based Learning," in *Proceedings of International Conference on Malicious and Unwanted Software*, pp. 85-94, 2013

10. W. Rui, D. G. Feng, Y. Yi, and S. U. Pu-Rui, "Semantics-based Malware Behavior Signature Extraction and Detection Method," *Journal of Software*, Vol. 23, No. 2, pp. 378-393, 2012
11. Y. Li, R. Ma, and R. Jiao, "A Hybrid Malicious Code Detection Method based on Deep Learning," *International Journal of Software Engineering and its Applications*, Vol. 9, No. 5, pp. 205-216, 2015
12. L. I. Chun-Lin, Y. J. Huang, H. Wang, and C. X. Niu, "Detection of Network Intrusion based on Deep Learning," in *Proceedings of Information Security & Communications Privacy*
13. J. Watson, "VirtualBox: Bits and Bytes Masquerading as Machines," *Linux Journal*, Vol. 2008, No. 166, 2008
14. F. Bellard, "QEMU, A Fast and Portable Dynamic Translator," in *Proceedings of USENIX Annual Technical Conference*, FREENIX Track, pp. 41-44, 2005
15. C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser, "The Cuckoo Sandbox," (http://cuckoosandbox.org, 2012)
16. J. Choi, H. Kim, C. Choi, and P. Kim, "Efficient Malicious Code Detection using N-gram Analysis and SVM," in *Proceedings of the 14th International Conference on Network-based Information Systems*, Vol. 16, pp. 618-621, 2011
17. E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, et al., "An Investigation of Byte N-gram Features for Malware Classification," *Journal of Computer Virology and Hacking Techniques*, Vol. 14, No. 1, pp. 1-20, 2018
18. R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, et al., "Unknown Malcode Detection using OPCODE Representation," in *Proceedings of the First European Conference on Intelligence and Security Informatics*, Vol. 5376, pp. 204-215, 2008
19. A. A. E. Elhadi, M. A. Maarof, and A. H. Osman, "Malware Detection based on Hybrid Signature Behaviour Application Programming interface Call Graph," *American Journal of Applied Sciences*, Vol. 9, No. 3, pp. 283-288, 2012
20. P. Faruki, V. Laxmi, M. S. Gaur, and P. Vinod, "Mining Control Flow Graph as API Call-Grams to Detect Portable Executable Malware," in *Proceedings of the Fifth International Conference on Security of Information and Networks*, pp. 130-137, ACM, October, 2012
21. B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based Malware Detection using Dynamic Analysis," *Journal in Computer Virology*, Vol. 7, No. 4, pp. 247-258, 2011
22. S. Alam, I. Traore, and I. Sogukpinar, "Annotated Control Flow Graph for Metamorphic Malware Detection.," *The Computer Journal*, Vol. 58, No. 10, pp. 2608-2621, 2015
23. Y. Cao, Q. Miao, J. Liu, and L. Gao, "Abstracting Minimal Security-Relevant Behaviors for Malware Analysis," *Journal of Computer Virology and Hacking Techniques*, Vol. 9, No. 4, pp. 193-204, 2013
24. G. E. Hinton, "Learning Distributed Representations of Concepts," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Vol. 1, pp. 12, 1986
25. A. Mnih and G. Hinton, "Three New Graphical Models for Statistical Language Modeling," in *Proceedings of International Conference on Machine Learning*, pp. 641-648, ACM, 2007
26. A. Mnih and G. E. Hinton, "A Scalable Hierarchical Distributed Language Model," *Advances in Neural Information Processing Systems*, pp. 1081-1088, 2009
27. R. Socher, C. D. Manning, and A. Y. Ng, "Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks," in *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, Vol. 2010, pp. 1-9, 2010
28. R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, et al., "Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631-1642, 2013

Lele Wang is a Ph.D. student at the National Digital Switching System Engineering and Technology Research Center whose main research direction is information security.

Binqiang Wang is a professor and doctoral tutor at the National Digital Switching System Engineering and Technology Research Center. His research interests include network security and broadband information networks.

Jiangang Liu is a researcher at the Nanjing Information Technology Institute whose main research direction is information security.

Qiguang Miao is a professor and Ph.D. supervisor in the School of Computer Science at Xidian University as well as a director of the China Computer Federation (CCF), chairman of CCF YOCSEF, member of the CCF Artificial Intelligence and Pattern Recognition Committee, standing committee member of the CCF Computer Vision Committee, and IEEE senior member. His main research directions include intelligent image processing, machine learning, and high performance computing.

Jianhui Zhang is an associate research fellow at the National Digital Switching System Engineering and Technology Research Center whose main research direction is broadband information networks.