Packed malware variants detection using deep belief networks

Zhigang Zhang[1,2,*], *Chaowen* Chang[1], *Peisheng* Han[1] *and Hongtao* Zhang[1]

[1]Strategic Support Force Information Engineering University, Zhengzhou, 450000, China
[2]DaTang Centrol-China Electric Power Test Research Institute, Zhengzhou, 450006, China

Keywords: Malware, Deep belief network, Sensitive system call, Information gain.

Abstract. Malware is one of the most serious network security threats. To detect unknown variants of malware, many researches have proposed various methods of malware detection based on machine learning in recent years. However, modern malware is often protected by software packers, obfuscation, and other technologies, which bring challenges to malware analysis and detection. In this paper, we propose a system call based malware detection technology. By comparing malware and benign software in a sandbox environment, a sensitive system call context is extracted based on information gain, which reduces obfuscation caused by a normal system call. By using the deep belief network, we train a malware detection model with sensitive system call context to improve the detection accuracy.

1 Introduction

With the rapid growth of malicious software (malware) variants, traditional signature-based detection methods are failed to detect malware. In recent years, a series of malware variants detection methods have been proposed by using machine learning.

However, modern anti-detection techniques such as software packing, obfuscation, etc. can prevent detections by compressing, encrypting and obfuscating malicious programs which reduce the accuracy of detection. Although some unpacking techniques can recover the original programs, there are remain some limitations: Since a variety of public and private packers, the unpacking techniques cannot be always efficient.

To address such a problem, we prefer to adopt a dynamic analysis method to detect malicious behaviors in runtime. However, the malicious behavior of malware will be hidden and obfuscated by the normal behavior and the packing/unpacking in packed malware, which brings a challenge to detect packed malware.

To detect packed malware efficiently, we propose a packed malware variants detection method based on sensitive system calls and a deep belief network. We first gain the system call sequences of executables in a sandbox, then extract the sensitive system call by using information gain to reduce the obfuscation caused by normal behaviors and packers, and

* Corresponding author: 957473068@126.com

finally adopt the deep belief network to adaptively train a detection model to detect the abnormal malicious behaviors.

The main contributions are organized as follows: 1) In this paper we propose a packed malware detection method by using Deep Belief Networks (DBN). 2) We propose a sensitive system call extraction method to reduce the obfuscation caused by packing/unpacking behaviors and normal behaviors. 3) Theoretical analysis and experimental results show that the proposed method can detect packed malware, which achieves more than 92% of accuracy and takes less than 0.001 seconds of detection time..

The rest of this paper is organized as follows: Section 2 introduces the related works. Section 3 proposes our methods. Section 4 presents the experiments and Section 5 concludes this paper.

2 Related works

2.1 Static analysis-based detection methods

Static analysis-based detection methods usually extract operation code (op-code) by disassembly tools, and detects malware by analyzing the features of the code. McLaughlin et al. [1] embedded malware opcodes and trained a malware detection model with the N-Gram Convolutional Neural Network (CNN). Ming et al. [2] proposed a malware classification method that extracts API calls to construct API call subgraphs and classifies the family of malware based on the features of the API call subgraphs. Zhang et al. [3] proposed a feature-hybrid malware detection method to integrate op-codes and API calls by merging the hidden layers of the CNN and the back-propagation neural networks. Cesare et al. [4] proposed to extract the control flow graphs of executables and search the similarities between the unknown software and the malware by edit string distance. Zhang et al. [5] proposed an Android malware detection method which builds a graph of op-codes and extracts the globle topology features.

2.2 Dynamic analysis-based detection methods

Dynamic analysis-based detection methods usually analyze the malicious behaviors by a sandbox, virtual machine, etc. Huang et al. [6] proposed to analyze the behaviors of users and the behaviors of programs, and detect stealing behavior in Android system by searching similarities. Canzanese et al. [7] used N tuples of system calls to represent the system call sequence and used support vector machine to train the malware detection model. Yang et al [8] proposed to extract sensitive behaviors of software that affect system security and detect malware by comparing with sensitive behaviors in normal software and malware. Shabtai et al. [9] proposed to extract device states that affect system security and compare the differences between normal states and abnormal states in runtime to detect malware. Rieck et al. [10] proposed an automatic malware detection method by adopting system call and machine learning.

3 Methodology

3.1 Overview of our method

The overview of our malware detection method is shown in Figure 1. To capture the behavior of executables, we first run executables in a sandbox Cukoo and get the logs of each

executables, The log contains a time series of system calls which represents the interactions between executables and the operating system. Then we use a Bi-gram model to build system call bi-grams to represent the local semantic among system calls. Base on information theory, we use information gain to extract the sensitive system call bi-grams. Finally, we adopt Deep Belief Networks (DBN) to train a malware detection model with these sensitive system calls.

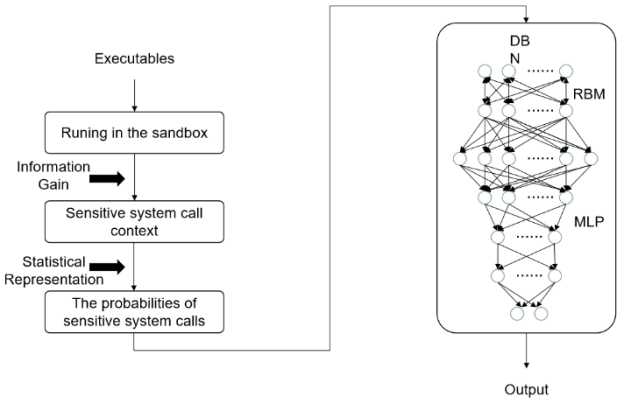

Fig. 1. The overview of our approach.

3.2 Sensitive system call extraction

We collect the log of runtime system call through a sandbox, named cuckoo, and the log of system calls includes system call, time stamp, input and output data, etc. Since we find that the probability distribution of system call context in malware is quite different from that in benign, we propose a sensitive system call extraction method based on infromation gain.

In order to extract the sensitive system call context, we analyze the probability distribution of the system call bi-grams in malware and benign, and calculate the information gain for each system call bi-grams, according to Eq. (1), where x_{ij} is the system call bi-gram, y is the class of an executable. Let $p(x_{ij})$ be the probability of the system call bi-grams x_{ij}, according to Eq. (2), where $p(y)$ is the probability of the malware samples, $p(x_{ij} \mid y)$ is the conditional probability of x_{ij} when the executable is malware. When $Gain$ approaches 1, it means that the system call bi-grams is sensitive.

$$p(x_{ij}) = \frac{F_{ij}}{\sum F_{ij}} \tag{1}$$

$$\mathrm{G}ain = p(x_{ij} \mid y) \cdot \log \frac{p(x_{ij} \mid y)}{p(x_{ij}) \cdot p(y)} \tag{2}$$

To reduce the obfuscated system calls caused by normal behaviors and packing/unpacking behaviors. We maintain the sensitive system call context and remove the rest system calls. After that, we represent the sensitive system call context by a vector of probabilities of the sensitive system calls. Let $Vec = \{p(x_1), p(x_2), ..., p(x_n)\}$ be the

vector of the statistical representation of the sensitive system calls, where $p(x_i)$ is the probability of the sensitive system call in the sensitive system call context.

3.3 Training and detection

Once we have represented the sensitive system calls, we use Deep Belief Networks (DBN) to detect packed malware.

3.3.1 Architechture

The DBN consists of a Restricted Boltzmann Machine (RBM) and a multi-layer perceptron (MLP). The RBM is used to embed the original statistical representation of sensitive system calls to optimize the inputs of MLP, that improves the convergence accuracy and speed.

The RBM has two layers: the visible layer and the hidden layer. The visible layer inputs the statistical representation of the sensitive system calls and the hidden layer outputs the embedding vector of statistical representation of the sensitive system calls. The neurons between the visible layer and the hidden layer are fully connected. In the process of two-way information transmission, the two layers of neurons share the connection weights. In this paper, the RBM by N times of Gibbs sampling. The neurons in each layer are randomly activated. The activated neurons are calculated by the Sigmoid function, which is shown in Eq. (3). The output of the hidden layer of the RBM will be sent to the next MLP.

The MLP used in this paper has three layers: an input layer, a hidden layers and an output layer. The input layer fully connected to the next hidden layer and the hidden layer fully connect to the next output layer according to Sigmoid function as shown in Eq. (3). The connection weights between the two adjacent layers are initialized by random values. The output layer outputs the probability of malware and the probability of benign.

$$Sigmoid = \frac{1}{1+e^{-\sum w \cdot x + b}} \tag{3}$$

3.3.2 Training process

Since the output of the RBM is the input of the MLP and the training processes of the RBM and the MLP are independent, we train the RBM and the MLP respectively.

We train the RBM by N times of Gibbs sampling. The neurons in each layer are randomly activated according to the Sigmoid function. The two adjacent layers of neurons transfer information in two directions and share connection weights. The expectation function is according to Eq. (4), and the weights are updated by Eq. (5), where, v represents the vector in the visible layer, h represents the vector in the hidden layer and w represents the shared connection weight, $E(v,h)_{visible}$ represents the expectation of the visible layer, and $E(v,h)_{hidden}$ represents the expectation of the hidden layer.

$$E(v,h) = -\sum v \cdot b - \sum h \cdot c - \sum v \cdot h \cdot w \tag{4}$$

$$\Delta w = w + \alpha \cdot (E(v,h)_{visible} - E(v,h)_{hidden}) \tag{5}$$

We train the MLP by the gradient descent method. In this paper, we use the minimum square error as the loss function of MLP, according to Eq. (6), Eq. (7) and Eq. (8), , where x is the input, y is the label, $h(x)$ is the calculated value of neural network, w is the connection weight between two adjacent layers of neurons, and α is the step length of each iteration. Through back propagation, we update the weights between two adjacent layers based on the chain rule, according to Eq. (9).

$$Loss = -\frac{1}{2n} \cdot \sum (y - h(x))^2 \tag{6}$$

$$h(x) = \frac{1}{1 + e^{-u(x)}} \tag{7}$$

$$u(x) = \sum wx + b \tag{8}$$

$$\begin{aligned} \Delta w &= \alpha \cdot \frac{\partial_{Loss}}{\partial_{h(x)}} \cdot \frac{\partial_{h(x)}}{\partial_{u(x)}} \cdot \frac{\partial_{u(x)}}{\partial_{w}} \\ &= \alpha \cdot \frac{\partial_{-\frac{1}{2n} \cdot \sum (y - h(x))^2}}{\partial_{h(x)}} \cdot \frac{\partial_{\frac{1}{1+e^{-u(x)}}}}{\partial_{u(x)}} \cdot \frac{\partial_{\sum wx+b}}{\partial_{w}} \\ &= \alpha \cdot (-\frac{1}{n}) \cdot \sum (y - h(x)) \cdot h(x)(1 - h(x)) \cdot \sum x \end{aligned} \tag{9}$$

3.3.3 Detection process

In this paper, the detection model is formed by setting the fixed weights between two adjacent layers of neurons, the fixed number of network layers, the fixed number of neurons in each layer and other parameters after training the RBM and MLP. When detecting an unknown packed executable, we first extract the sensitive system call context and use a statistical vector to represent the probabilities of sensitive system calls, and then send the statistical vector as an input to the detection model. The detection model outputs the probabilities of malware and benign through forward passing. If the probability of malware is large enough and bigger than the probability of benign, then the packed executable is malware.

4 Experiments

In this paper, different groups of experiments are designed, and different groups of sample data are used for analysis. 10-fold cross validation method is used for verification. The experimental results are the average of 10 groups of experimental results.

4.1 Experimental setup

All of the methods for comparison are implemented in the same environment and the same configurations, such as CPU, Memory, Hard disk, Operating system (OS), Java virtual machine, as shown in Table 1.

Table 1. The experimental setup.

Project	Parameter
CPU	2*Xeon4116, 12C/85W/2.1GHz
Memory	256G DDR4 2666MHz
Hard disk	2*480G SSD
OS	Ubuntu 16.04
JVM	JRE 1.8

4.2 Data sets

The malware data sets used in this paper are collected from vxheaven website [15] and the benign data sets are collected from our personal computers. Some of the malware samples and some of the benign samples in the data sets are packed by several packers, such as ASPack [11], UPX [12], VMProtect [13], ZProtect [14], which will be used for packed malware detection. A part of the malware samples and a part of benign samples are used for training the others are used for detection, as shown in Table 2.

Table 2. The data sets.

Data items	Number	Total	Purpose
Virus(unpacked)	540	671	train
Virus(packed)	131		test
Worm(unpacked)	503	605	train
Worm(packed)	102		test
Trojan(unpacked)	1376	1824	train
Trojan(packed)	448		test
Benign(unpacked)	2298	2895	train
Benign(packed)	597		test

4.3 Pre-processing

For each malware sample and benign sample in the data sets, we capture the log of runtime system calls as the behaviors of executables by using the sandbox Cuckoo.

From our data sets, we capture 139 kinds of Windows system calls. Through the analysis of the probability of each system call in malware and benign software, we find that the system calls with a significant distribution in malware mainly include Regopenkeyexw, RegCloseKey, Regopenkeyexa, Regqueryvalueexa, Regqueryvalueexw, Findfirstfileexw, Ntdelayexecution, etc.

4.4 Accuracy analysis

In this paper, we compare with several machine learning methods, such as DBN, DBSCAN clustering method and support vector machine (SVM) method. The experimental results are shown in Table 3. From the results, we find that the DBN method achieves 92.6% of accuracy while DBSCAN method only achieves 78.3% of accuracy and SVM method achieves 86.3% of accuracy. The experimental results show that our proposed method can detect packed malware from different malware families.

Table 3. The results of accuracy comparison.

Method	ACC	PRE	REC	F1
DBN	92.6 %	96.3 %	89.6 %	92.8 %
DBSCAN	78.3 %	85.1 %	72.4 %	78.2 %
SVM	86.3 %	92.2 %	81.2 %	86.4 %

4.5 Time cost analysis

The training and detection time cost experimental results of several machine learning methods are shown in Table 4. The results show that the DBN method takes less than 0.001 seconds of detection time and 277.4 seconds of training time, while the DBSCAN method takes less than 3.1 seconds of detection time and does not need any training time, and the SVM method takes less than 0.04 seconds of detection time and 46.0 seconds of training time.

Table 4. The results of training and detection time cost comparison.

Method	**Training time (second)**	**Detection time (second)**
DBN	277.4	< 0.001
DBSCAN	N/A	3.1
SVM	46.0	0.04

5 Conclusion

In this paper we propose a packed malware detection method which first capture the runtime system call sequences of executables in a sandbox Cukoo, then extract the sensitive system call by using information gain to reduce the obfuscation caused by normal behaviors and packing/unpacking behaviors, and finally adopt the deep belief network to adaptively train a detection model to detect packed malware. Theoretical analysis and experimental results show that the proposed method can detect packed malware, which achieves more than 92% of accuracy and takes less than 0.001 seconds of detection time.

References

1. N. McLaughlin, J. M. del Rincon, B. Kang, S. Yerima, Deep Android Malware Detection. ACM Conference on Data and Application Security and Privacy, 2017.
2. Ming Fan, Jun Liu, Xiapu Luo, Kai Chen, Zhenzhou Tian, Qinghua Zheng, Ting Liu. Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis. IEEE Transaction on Information Forensics and Security, 2018, 13(8): 1890-1905.
3. Jixin Zhang, Zheng Qin, Hui Yin, Lu Ou, Kehuan Zhang, A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding, Computers & Security, 2019, 84: 376-392.
4. Silvio Cesare, Yang Xiang, Wanlei Zhou. Control Flow-Based Malware Variant Detection. IEEE Transactions on Dependable and Secure Computing, 2014, 11(4): 307-317.
5. Jixin Zhang, Zheng Qin, Kehuan Zhang, Hui Yin, Jingfu Zou, Dalvik Opcode Graph based Android Malware Variants Detection, IEEE Access, 2018, 2018, 6: 51964 - 51974.

6. Jianjun Huang, Xiangyu Zhang, Lin Tan. AsDroid detecting stealthy behaviors in Android applications by user interface and program behavior contradiction. ACM/IEEE International Conference on Software Engineering, 2014: 1036-1046.
7. Canzanese R. et al. System call-based detection of malicious processes. In proc. of IEEE international conference on software quality, Reliability and Security, 2015: 119–24.
8. Wei Yang, Xusheng Xiao, Benjamin Andow. AppContext diffffferentiating malicious and benign mobile app behaviors using context. ACM/IEEE International Conference on Software Engineering, 2015: 303-313.
9. Asaf Shabtai, Yuval Elovici, Uri Kanonov, Yael Weiss, Chanan Glezer. Andromaly: a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems, 2012, 38(1): 161-190.
10. Konrad R et al Automatic analysis of malware behavior using machine learning. J Comput Secur 2013, 19:639–668.
11. ASPack, http://www.aspack.com, 2019.
12. UPX, https://upx.github.io, 2019.
13. VMProtect, https://vmpsoft.com/products/vmprotect/, 2019.
14. ZProtect, https://tuts4you.com/download.php?view.3017, 2019.
15. VXHeaven,https://hypestat.com/info/vxheaven.org, 2019.