Stealthy Malware Detection Based on Deep Neural Network

Shoupu Lu¹,²*, Qingbao Li¹, Xinbing Zhu¹

¹State Key laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, Henan Province, 450001, China
²Henan University of Economics and Law, Zhengzhou, Henan Province, 450000, China

58458878@qq.com (Shoupu Lu), qingbaoli@139.com (Qingbao Li), xinbingzhu@139.com (Xinbing Zhu)

Abstract: Network attacks using advanced local hiding technology have not only increased, but also become a serious threat. However, attacks using these technologies cannot be detected through traffic detection, and some attacks imitate benign traffic to avoid detection. To solve these problems, a malware process detection method based on process behavior in possibly infected terminals is proposed. In this method, a deep neural network is introduced to classify malware processes. Firstly, the recurrent neural network is trained to extract the characteristics of process behavior. Secondly, training convolutional neural network is used to classify feature images generated by trained RNN features. The experiments results show that this method can effectively extract the features of malicious processes, and the AUC of ROC curve is 0.97 in the best case.

1. Introduction

The Internet has now become an important part of our work and life. However, cyber-attacks by stealthy malware, especially malware that has hidden in systems for long to obtain valid information, pose a serious threat to the Internet. In recent years, malware has increased year by year. In 2018, CNCERT/CC detected more than 100 million cases of malware, the cyber-attacks made by which were more advanced and complex [1]. In these advanced attacks, unknown malware that has not been confirmed by security vendors is often used to circumvent malware detection systems, and the malware generally adopts some hidden technology that hides their existence, such as process hiding and file hiding technology. In addition, advanced malware has emerged and by dynamically modifying their codes, it evades the case where signature features are matched. Therefore, it is difficult to protect the terminal from all attacks, and the demand for post-infection countermeasures is also increasing.

Using traffic data to detect malware infection is one of the post-infection countermeasures. However, discovering the traffic of recent malware is difficult because it mimics benign traffic. Besides, the attacker’s purpose has changed to profit from stealing intellectual property and privacy, making the attack more unnoticeable and covert. In order to achieve long-term information stealing, malware tends to reduce the communication frequency with C&C communication to avoid traffic detection. Therefore, detecting malware infections based on traffic is not easy.

In summary, this paper refers to the study by Charan [2] and proposes a new malware process detection method, using process behaviors to detect if the terminal is infected. In order to make the characteristics of malware processes widely applicable, two types of deep neural networks (DNN) are
utilized to adapt to different characteristics of each operational process. One is the recursive neural network (RNN) for feature extraction, and the other is the convolutional neural network (CNN) for feature classification. The training phase based on a language model with long-term and short-term memory (LSTM) to construct a feature extractor firstly records the process behavior. Features are then extracted from the trained RNN and a feature image is generated. Finally, the CNN is trained using feature images with malware or benign annotations. The verification phase determines the malware by calculating DNN probability.

2. Related methods

2.1. Malware detection method

There are two types of countermeasures against malware. One is to detect malware files before they are executed to prevent terminal infections, and the other is to detect infected terminals to minimize the scope of infection.

In the countermeasure against infection, a mode-based detection method is usually employed. It can be divided into a detection method based on signature, on behavioral heuristic search, and on the feature state space model.

2.1.1. Detection method based on signature

Malware detection based on code signature is a technology for comparing the malware signature (hashes or code fragments) to a signature database associated with known attacks. Although signatures cannot be used directly to discover new loopholes, they can be indirectly matched for signatures due to overlapping components between malware [3][4]. In addition, even if the codes deform, there are still some n-gram byte sequences that are specific to the common type. Machine learning for malware classification through n-gram and sequence analysis has been extensively studied and deployed as part of anti-malware systems [5][6].

2.1.2. Detection method based on behavioral heuristic analysis

At the host level, signatures are not the only heuristic algorithm used for intrusion detection. The system call sequence [7-9] for intrusion and anomaly detection is a particularly popular alternative to rootkit analysis because hooked IAT or SSDT entries normally produce duplicate system call patterns.

2.1.3. Detection method based on feature/state space model

Machine learning models can be divided into feature space models and state space models. The feature space model is designed to take signature/behavior features as spatial dimensions and parameterize the high-dimensional feature spaces of each category.

The state space model is intended to infer the probability of a sequence. The most common type of state space model is the Markov Model. Although the Markov assumption is not always valid, it makes sequential inference easier and it is usually reasonable. The Hidden Markov Model (HMM) is the most widely used Markov model and is particularly useful in code analysis, including identifying variant viruses [10, 11].

Among studies on post-infection countermeasures, many focus on traffic data. Otsuki detects malware infections using the occurrence frequency of ASCII codes in the traffic payload and the length of HTTP requests [12]. However, behavior-based detection method has a higher detection error rate. What’s more, using only traffic-based data is not enough because malware will mimic benign traffic, making malware traffic less detectable.

2.2. Deep neural network

A neural network (NN) is a mathematical model that simulates the brain network. DNN is a NN with many hidden layers, which enables automatic learning of concrete and abstract features, so it has been widely studied and applied in various fields such as image recognition and language processing.
Dropton proposed by Hinton can improve the performance of DNNs \cite{13} as it lowers the interdependence between neurons by reducing the output of some neurons to avoid overfitting. Krizhevsky \cite{14} dramatically reduces the error rate of object recognition data sets by using CNNs.

RNN has achieved good results in various areas of sequential data, including language processing and speech recognition. Mikolov proposes a Recurrent Neural Network Language Model (RNNLM), a language model using simple RNNs \cite{15}. The purpose of language modeling is to predict the next word based on previous input. Pascanu puts forward a malware detection method using RNN \cite{16}, and constructs an API call language model by training RNN to generate fixed-length feature vectors, and then classifies them by logistic regression.

3. Detection method based on process behavior

This section presents a malware process detection method to discover potentially infected terminals. The method will apply DNN in two phases. Phase one extracts process activities through RNN and summarizes them into feature vectors. The feature vectors are then transformed into images, which are classified based on CNN so as to classify malware.

3.1. Overview

The basic principle and working process of this method are shown in Figure 1. The behavior of a process consists of various activities, such as file management, which involve multiple operations. When the behavior of a process is recorded as an API call sequence, a single API call represents an operation of the process, and multiple API calls represent an activity, and the API call of the entire record represents the behavior of the entire process. This hierarchy is similar to the writing structure. An article consists of different sentences, which involve multiple words. Therefore, the characteristics of the process behavior can be extracted through the language model, and trained as RNN. The trained RNN is then used to verify the feature vector extracted.

![Figure 1 Basic working principle of detection method based on process behavior](image)

The feature vector in the behavior log of the process is extracted and converted into an image containing various local features representing the process activity. Through training local features, CNN is applied to classify these images. In this way, classifiers can be created by CNN training image features of malware and benign processes.

According to Fig. 1, the training is divided into four parts. First, monitor the process behavior and generate a log file. Second, use the log file to train the RNN to build a behavioral language model. Third, the trained RNN extracts features from the log file and converts them into images. Finally, the feature image is trained based on CNN, so that the trained CNN marks the feature image as malware or benign software.

Next, the trained DNN is used to evaluate the verification process. First, the trained RNN generates a feature image from the process behavior log file. These images are then sorted by the trained CNN to determine whether it is malware.
3.2. Process behavior record
Within a pre-defined interval, this paper records the behavior of all executed processes at a predetermined length and time. Process Monitor is used to record the related operational behavior of system calls including ReadFile, RegSetValue, and Thread Start. The behavior items recorded by Process Monitor are shown in Table 1. Result shows the result codes of the specific process to perform an operation, such as SUCCESS, ACCESS DENIED, FILE NOT FOUND. Detail shows some information about the parameters in an operation. In this paper, the operational information used for the record is considered to be a process behavior rather than an API call sequence.

The behavior recording process adopts the following method: First, perform 5 minutes of log recording, and then continue recording for 5 minutes at intervals of 10 times. Second, create log files for each process with different PIDs. The log for each record contains seven items shown in Table 1. Last, use the Event and Result input contexts to classify the malware and make the reminder the process identifier. From this section, process operations are defined as a collection of Event and Result.

| Table 1 Recorded information |   |
|-----------------------------|---|
| Process name                | Name of a process that performs an operation |
| Time                        | Time to perform an operation |
| PID                         | Process ID to perform an operation |
| Event                       | The name of an event action |
| Path                        | The current path in which an operation is performed |
| Result                      | The result of the operation after the execution |
| Detail                      | Details about performing an action |

3.3. RNN training
This section builds a behavioral language model based on process operations and uses an RNN with LSTM units as the model. The RNN consists of an input layer $x$, a normal hidden layer $h^1$, two LSTM layers $h^2$ and $h^3$, and an output layer $y$. In the training phase, Dropout is used for acyclic connections. The RNN process is shown in Figure 2. The input vector $x$ is the converted 1-hot vector representing a separate operation $O_{Pt}$. The conversion process proceeds as follows.

1) Create a dictionary with ID and process operations associated with each other;
2) Convert the operation to a 1-hot vector, where the position associated with the operation ID is filled with 1 and the others are filled with 0.

RNN uses log files for repeated training. First, select a log file and convert the operations $\{O_{P1}, O_{P2}, ..., O_{PL}\}$ to 1-hot vectors $\{x^1, x^2, ..., x^L\}$. Each 1-hot vector $x_i$ is sequentially input into the RNN and outputs a predicted $y_i$. The loss function is then calculated by comparing $y_i$ and $x_{i+1}$. After the input process operation $T$, the weights are updated by back propagation.
Because it is not clear whether the training log file contains all process operations, some process operations may only appear in the verification log file during the verification phase. To avoid this problem, some of the operations in the training log are anonymized. An action is selected in each log file that appears less than 10 times in the file, and is replaced with “unknown action”. This replacement is performed for each log file. An epoch learning means that all training log files are entered into the RNN and the order of the training log files is random at each epoch. RNN training is performed on multiple epochs, so a well-trained feature extractor is obtained.

3.4. Feature extraction and image generation

The trained RNN is used to extract the process features and generate feature images. The feature extraction process is shown in Figure 3. The feature extractor trained in the previous section is able to predict the next operation from a previous series of inputs, which means that the last hidden layer $h^3$ contains the previously entered information. Furthermore, local features are learned in layers close to the input layer in DNN, and abstract features are learned in deeper layers in conjunction with local features. Since that, it is expected behavioral features to be included in the deep layers of the feature extractor. Therefore, the whole series of hidden layers $h^3$ is considered as one feature of the process behavior.

To generate a feature image, the operations in the log file are first converted to the same 1-hot vector as the previous bar, and then entered into the RNN in turn. L is set as the length of the operation recorded in the log file. The value of the third hidden layer $h^3$ is extracted for each input, and a series of feature vectors $\{h^3_1, h^3_2, ..., h^3_L\}$ is obtained. Since the feature classifier based on fixed-size images is used in this paper, and the operation lengths between log files are different, it is necessary to convert these vector sequences into fixed-length images. The conversion is performed according to Equation 1 and Equation 2.
\[ \begin{align*}
  p_k &= \begin{cases} 
    0 & (k = 0) \\
    \frac{L + k - 1}{N} + p_{k-1} & (1 \leq k \leq N)
  \end{cases} \\
  f_i &= \frac{1}{p_i - p_{i-1}} \sum_{j=p_{i-1}}^{p_i} h_j^i
\end{align*} \] (1)

Where \( f_i \) is the element of the fixed length vector sequence, \( N \) is the height of the feature image, and \( p_k \) is the last number of the \( k \)-th vector set. The vector sequence is divided into \( N \) groups, and the average value of each group is calculated. \( W \) is set as the dimension of the third hidden layer, the grade of the fixed vector can be described as the matrix \( F \):

\[
F = \begin{pmatrix}
  f_1^1 & f_1^2 & \cdots & f_1^W \\
  f_2^1 & f_2^2 & \cdots & f_2^W \\
  \vdots & \vdots & \ddots & \vdots \\
  f_k^1 & f_k^2 & \cdots & f_k^W
\end{pmatrix}
\] (2)

Each element of \( F \) is mapped to the space \([0,1]\) by the sigmoid function, and multiplied by 255 to form a grade 256 grayscale image. Finally, the output matrix \( F \) is used as a feature image with a resolution of \( W \times N \).

### 3.5. CNN training and malicious program detection

The CNN divides the process local features appearing in the image into malware-specific features and benign features. During the training phase, the CNN is trained with feature images marked as malicious or benign. The structure of CNN is shown in Figure 4. The CNN consists of an input layer, two convolution-pooling layers, a fully-connected layer, and an output layer. The first convolution layer uses 10 core filters \( W_0 \times W_0 \times 1 \) to input an image. The second convolution layer is output with \( W_1 \times W_1 \times 10 \) of the previous layer of 20 core filters. Each pooling layer receives the output of the previous convolutional layer and reduces its size to 1/2 by Max-Pooling, with a stride of 2. Since CNN is a binary classifier, the output layer has a dimension of 2.

In the verification phase, trained CNN is used to calculate the probability that a process is malware. When the CNN receives the feature image of the verification process, the CNN outputs a two-dimensional vector, respectively representing the degree of maliciousness and benignity. If the input image is classified as malicious, the malicious value will be higher than the benign value. The malware probability \( p \) is calculated by the sigmoid function of Equation 4.

\[
p = \text{Sig}(y_i) = \frac{1}{1 + \exp(-y_i)}
\] (4)

### 4. Experimental results and analysis

#### 4.1. Experimental environment and configuration

The test environment is shown in Figure 5. The process monitor will supervise the execution of malware files on Windows. In the virtual server, INetSim is used to simulate the Internet environment. It can simulate common Internet services such as HTTP, SMTP, DNS, FTP. All traffic from Windows XP is
sent to the virtual server, and the server returns a fake response. A process that meets any of the following conditions is considered a malware process:

1) The process of pre-determined malware files (including processes with the same name);
2) a process/child process generated or created by a malicious process in 1);
3) The process of injecting malicious code into the processes in 1) and 2).

To verify whether the process satisfies condition 2) or condition 3), the Cuckoo sandbox is used to confirm. Malware files are executed in the Cuckoo sandbox and the sandbox tracks the behavior of the malicious processes to determine which child processes or injected processes the malware files generate.

From the malicious code sample library provided by malwarebenchmark [17], 18,825 cases of malware are selected with hidden functions and their log files are recorded. Then these log files and 2000 benign process log files are used for training and verification. In all malware log files, 10690 files meet condition 1), 7670 files meet condition 2), and 465 files meet condition 3).

During the RNN training phase, 5,000 malicious program process logs and 1500 benign software logs are used for training. 12,000 feature images are generated using the trained RNN, and the CNN is trained and evaluated by fivefold cross validation. Approximately 9,600 images are used for training and the remaining images are used for cross validation.

In order to compare the performance of the classifier under different parameters, the RNN/CNN is trained and verified with different parameter configurations. The parameter configuration used is shown in Table 2 and Table 3. The size of the feature image is first set to be much larger than the amount of data used for training, which is used to verify if condition 1 is true. Then by evaluating with a smaller feature image size, conditions 2 and 3 are verified.

**Figure 5 Test environment**

![Figure 5 Test environment](image)

Table 2 RNN parameter settings

| Condition | Condition 2 | Condition 3 |
|-----------|-------------|-------------|
| Dimension of the hidden layer $\mathbf{h}^i$ | 50 | 50 | 20 |
| Dimensions of LSTM layer $\mathbf{h}^2, \mathbf{h}^3$ | 500 | 40 | 30 |
| Other hyperparameters | The number of Epoch: 10; The size of Minibatch: 40 |

Table 3 CNN parameter settings

| Condition | Condition 2 | Condition 3 |
|-----------|-------------|-------------|
| Conv1 Input size $W_0$ | 350×350 | 30×30 | 20×20 |
| Other hyperparameters | Input channel: 1, Output channel: 10 | Filter size: 5×5 |
| Pool1 Input size $W_1$ | 346×346 | 26×26 | 16×16 |
| Other hyperparameters | Input channel: 10, Output channel: 10 | Filter size: 2×2, Stride: 2 |
4.2. Evaluation

In a multi-class classification, a positive sample means that an object \( x \) belongs to class \( a \) because it exceeds a threshold \( Th \). Otherwise, it becomes a negative sample. This method makes it become a binary problem, so there is only one threshold. \( y \) is set as the result of classification \( x \), then True Positive (TP), False Positive (FP), True Negative (TN), and False Positive (FN) are defined in Table 4. It is shown that the True Positive Rate (TPR) is calculated by \( TP/P \), and the False Positive Rate (FPR) is calculated by \( FP/N \) and its accuracy is calculated by \( (TP+TN)/(P+N) \).

According to the ROC curve, the efficiency of the classifier is evaluated by calculating the area under the curve (AUC). The ROC curve under each condition is calculated by considering the TPR as the detection rate of the malware process and the FPR as the error detection rate of the benign process. In addition, the AUC under each condition is compared to evaluate the efficiency of the classifier.

4.3. Results analysis

Figure 6 shows the ROC curve. The horizontal axis represents the error detection rate and the vertical axis represents the detection rate. The 5 solid lines represent a five-fold cross-validation of a single ROC curve, and the dashed line represents the micro-average of a single ROC curve. The average AUC under conditions 1, 2, and 3 are respectively 0.84, 0.97, and 0.93. Therefore, the method can detect malware processes with high precision.

| Dimension of m | 1000 | 250 | 40 |
|----------------|------|-----|----|
| Other hyperparameters | The number of Epoch: 50; The size of Minibatch: 20 |

| Conv2 | Input size \( W_{1/2} \) | 173×173 | 13×13 | 8×8 |
|-------|-------------------------|---------|-------|-----|
| Other hyperparameters | Input channel: 10, Output channel: 20 | Filter size: 5×5 |
| Pool2 | Input size \( W_2 \) | 169×169 | 9×9 | 4×4 |
| Other hyperparameters | Input channel: 20, Output channel: 20 | Filter size: 2×2, Stride: 2 |

Table 4 Classification problem

| category | \( y=a \) (positive) | \( y\neq a \) (negative) |
|----------|----------------------|-------------------------|
| True \( x \in a \) | TP | FN | \( P=TP+FN \) |
| \( x \notin a \) | FP | TN | \( N=FP+TN \) |

(a) Condition 1 (b) Condition 2
Next, the effectiveness of training feature extraction is discussed. If the RNN is well trained, the extracted features should share some regularity. Thereby, a series of feature vectors extracted by condition 2 are analyzed. A single vector is converted into two-dimensional by principal component analysis and they are drawn into a three-dimensional graph, as shown in Figure 7. Csrss.exe (Figure 7(a)) is a benign process, while cmd.exe and svchost.exe (Figure 7(b), (c)) are different malware processes but belong to the same Trojan horse family. Zbot. Both of them adopt the behavior of hiding its process.
The $x$-axis represents the vector sequence, and the $y$ and $z$ axes represent the values of vector elements. According to Fig. 7(b)(c), even if the binary process files are different, the distribution of some vector members is similar (green part). Additionally, there are some similar points around the sequence 0 to 200 of the three of them (red part), which indicates that the order of operations shares some similarity. It can be said that the extracted features represent the process behavior, implying that the training experienced by the feature extraction in this chapter is productive. On the other hand, in Condition 1, since the DNN is more complex, the amount of data used for training and verification in this paper may not be large enough compared to massive application software and malicious code that it shows a low AUC. Consequently, malware processes can be classified in a more precise manner with a larger amount of data.

5. Conclusion
This paper proposes a two-phase DNN based malware process detection method. The method detects the malware process by classifying the feature images with the help of CNN. The classifier is cross-validated fivefold with 12,000 process behavior log files. The verification results in several cases are compared. When the feature image size is 30×30, $AUC=0.97$ gives the best results. The principal component analysis is used to analyze the trained RNN features, proving this method effective. Alternatively, since the amount of data is relatively small compared to mass applications, increasing the amount of data also needs to be done in the future.

Reference
[1] CNCERT/CC. China Internet Network Security Report 2018 [EB/OL]. (2019-08-17) [2019-09-08]. https://www.cert.org.cn/publish/main/46/2019/20190717075521797368911/20190717075521797368911_.html
[2] Charan P V S, Kumar T G, Anand P M. Advance Persistent Threat Detection Using Long Short-Term Memory (LSTM) Neural Networks[A]. In: Proceedings of International Conference on Emerging Technologies in Computer Engineering. Springer, Singapore, 2019: 45-54.
[3] Abou-Assaleh T, Cercone N, Keselj V, et al. N-gram-based detection of new malicious code[A]. In: Proceedings of the 28th Annual International Computer Software and Applications Conference[C]. 2004. COMPSAC 2004. IEEE, 2004, 2: 41-42.
[4] Reddy D K S, Dash S K, Pujari A K. New malicious code detection using variable length n-grams[A]. In: Proceedings of International Conference on Information Systems Security[C], Springer, Berlin, Heidelberg, 2006: 276-288.
[5] Khreich W, Khosravifar B, Hamou-Lhadj A, et al. An anomaly detection system based on variable N-gram features and one-class SVM[J]. Information and Software Technology, 2017, 91: 186-197.
[6] Wang Weihong, Zhu Yuchen. Malicious code detection based on N-Gram integration with weighted classifier[J]. Journal of Zhejiang University of Technology, 2017, 45(6): 604-609.
[7] Feng H H, Kolesnikov O M, Fogla P, et al. Anomaly detection using call stack information[A]. In: Proceedings of 2003 Symposium on Security and Privacy[C], 2003. IEEE, 2003: 62-75.
[8] Li Peng, Wang Ruchuan, Gao Dehua. Research on joint Rootkit dynamic detection technology based on fuzzy recognition and support vector machine[J]. Chinese Journal of Electronics, 2012, 40(1): 115-120.
[9] Zhao Binglin, Meng Wei, Han Jin, et al. Analysis of homology of malicious code based on graph structure[J]. Journal on Communications, 2017, 38(Z2): 86-93.
[10] Cosovan D, Lita C V. Practical aspects related to using Hidden Markov Models for detecting metamorphic file infectors[A]. In: Proceedings of 2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)[C], IEEE, 2017: 275-278.
[11] Alqurashi S, Batarfi O. A comparison of malware detection techniques based on hidden markov model[J]. Journal of Information Security, 2016, 7(03): 215.
[12] Otsuki Y, Ichino M, Kimura S, et al. Evaluating payload features for malware infection detection[J]. Journal of information processing, 2014, 22(2): 376-387.

[13] Hinton G E, Srivastava N, Krizhevsky A, et al. Improving neural networks by preventing co-adaptation of feature detectors[J]. arXiv preprint arXiv:1207.0580, 2012.

[14] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[A]. In: Proceedings of Advances in neural information processing systems[C], 2012: 1097-1105.

[15] Mikolov T, Karafiát M, Burget L, et al. Recurrent neural network-based language model[A]. In: Proceedings of Eleventh annual conference of the international speech communication association[C], 2010, 1045–1048.

[16] Pascanu R, Stokes J W, Sanossian H, et al. Malware classification with recurrent networks[A]. In: Proceedings of 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)[C], IEEE, 2015: 1916-1920.

[17] Open Malware Benchmark. [EB/OL]. [2019-09-08]. http://www.malwarebenchmark.org/.