2016 IEEE 40th Annual Computer Software and Applications Conference

Malware Detection with Deep Neural Network Using Process Behavior

Shun Tobiyama*, Yukiko Yamaguchi†, Hajime Shimada†, Tomonori Ikuse‡ and Takeshi Yagi‡
*Graduate school of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan
†Information Technology Center, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan
‡NTT Secure Platform Laboratories
Midori-cho, Musashino-shi, Tokyo, 180-8585, Japan
Email:*tobiyama@net.itc.nagoya-u.ac.jp, †{yamaguchi, shimada}@itc.nagoya-u.ac.jp,
‡{ikuse.tomonori, yagi.takeshi}@lab.ntt.co.jp

***Abstract*—Increase of malware and advanced cyber-attacks are now becoming a serious problem. Unknown malware which has not determined by security vendors is often used in these attacks, and it is becoming difficult to protect terminals from their infection. Therefore, a countermeasure for after infection is required. There are some malware infection detection methods which focus on the traffic data comes from malware. However, it is difficult to perfectly detect infection only using traffic data because it imitates benign traffic. In this paper, we propose malware process detection method based on process behavior in possible infected terminals. In proposal, we investigated stepwise application of Deep Neural Networks to classify malware process. First, we train the Recurrent Neural Network (RNN) to extract features of process behavior. Second, we train the Convolutional Neural Network (CNN) to classify feature images which are generated by the extracted features from the trained RNN. The evaluation result in several image size by comparing the AUC of obtained ROC curves and we obtained AUC= 0.96 in best case.**

***Keywords*—malware infection detection; neural network; process behavior**

I. Introduction

Nowadays, the Internet is an essential part of our work. On the other hand, cyber-attacks based on malware are also a serious problem. The number of malware is increasing year by year and cyber-attacks are more advanced and sophisticated. In these advanced attacks, unknown malware which has not determined by security vendors are often used for evading malware detection system. Moreover, advanced malware is also appeared which evade signature matching by modifying own code dynamically. In this way, it becomes difficult to perfectly defend terminal from attacks and demands for after infection countermeasure are increasing.

Malware infection detection using traffic data is one of the countermeasure after infection. However, finding recent malware traffic is difficult because it imitates benign traffic. Moreover, the attacks are becoming silent and hidden due to the change of an attacker's purpose which earns money by stealing intellectual property. Thus, malware which takes long-term to steal information are also appeared and they can reduce traffic frequency. In this way, detecting malware infection is not easy.

In this paper, we propose a new malware process detection method using process behavior to detect whether a terminal is infected or not. Our proposal uses two types of Deep Neural Network (DNN) to adapt different characteristic of individual operation flows. The one is Recurrent Neural Network (RNN) which is used for feature extraction, and the other one is Convolutional Neural Network (CNN) which is used for feature classification. In the training phase of the method, first of all, we record API call sequences as process behavior and construct the feature extractor by learning language model of API call based on language model with Long Short-Term Memory (LSTM). Then we extract features from the trained RNN and generate feature images. Finally, we train the CNN with the feature images annotated with malware or benign. In the validation phase, we calculate malware probability. Firstly, we obtain a feature image by trained RNN and API call log. Then we obtain probability by trained CNN and the feature image.

II. Related works

In this section, we introduce previous researches of malware detection method and recent researches about DNN.

A. Malware Detection Method

The countermeasures for malware is divided into two types. The one is detecting malware files before they are executed to prevent terminals from infection, and the other one is detecting infected terminals to minimize the expansion of infection.

In the countermeasure of preventing infection, malware files are detected by signature or behavior of malware. While a detection error rate of signature-based detection is very low, it can't adapt to unknown malware because of a necessity of signature in advance. To resolve this problem, behavior-based detection method was proposed. This method finds malware files from behavioral or structural features of programs. Ahmed et al. proposed the method to detect malware files from spatial/temporal features of API calls [1]. They use the

0730-3157/16 $31.00 © 2016 IEEE
DOI 10.1109/COMPSAC.2016.151

features extracted from spatial information of API calls such as arguments and return values, and temporal information such as the sequence of API calls. The spatial information is extracted using statistical analysis or information theoretic method. The temporal information is extracted by the transition matrix which modeled API call sequences with Markov chain.

In the countermeasure for after infection, there are many researches focused on traffic data. Otsuki et al. detected malware infection using appearance frequency of ASCII code in traffic payload and the length of HTTP request [2]. They also mention that the sequential feature is different between malicious and normal traffic. However, the behavior-based detection method has higher detection error rate. Moreover, using only traffic-data-based method is insufficient because recent benign traffic is more diversified and malware traffic becomes less detectable.

B. Deep Neural Network

Neural Network (NN) is a mathematical model which simulates a network structure of a brain. NN is consisted of many neuron layers. DNN is a NN which has many hidden layers. It is attracting various fields such as image recognition and language processing, because it can automatically learn features and abstract features.

Hinton et al. proposed Dropout which improves DNN performance [3]. This method reduces a dependency between neurons by dropping some outputs of neurons to avoid overfitting. The dropped neurons is chosen randomly for every input. Thus, each training is performed with different structural network and this reduces the dependency between neurons.

In CNN, it has two special layers such as a convolution layer and a pooling layer. In the convolution layer, features are extracted by convoluting filter to inputs. In the pooling layer, an input is down-sampled to decrease the effect of small position shifting. CNN is consisted of some sets of this two types of special layers and normal NN. It is mainly used in image recognition because it can recognize features regardless of the appeared position. Krizhevsky et al. amazingly decreased the error rate of object recognition dataset by using CNN [4].

In RNN, it has a special loop structure or memory units, which retain the information of previous inputs or the state of hidden layer. RNN can train sequential data because the output depends on previous inputs. There is a problem in the training phase of RNN that the error is often vanished while long backpropagating input sequence. To avoid this problem, Gers et al. proposed LSTM [5]. It avoids the error vanishing problem by fixing weight of hidden layers to avoid error decay and retaining not all information of input but only selected information which is required for future outputs. RNN shows good results in various fields which use sequential data such as language processing or speech recognition. Mikolov et al. proposed Recurrent Neural Network Language Model (RNNLM) which is the language model using a simple RNN [6]. The purpose of language modeling is predicting a next word from previous inputs. In the RNNLM, words are converted to potential vectors and the next word is predicted by the potential vector of current input word and the history of previous inputs. Semantic information of the word is stored in the potential vector, and the relation between words can calculate using this potential vector. Pascanu et al. proposed malware detection method using RNN [7]. They first train RNN to construct API call language model and then generate fixed length feature vector which is converted from the obtained hidden vector at each time. Feature vectors are then classified by logistic regression or multi-layer perceptron.

III. Proposed Method

In this section, we propose the malware process detection method for discovering possible infected terminal. The proposal applies DNN in 2 stages. The first stage extracts process activities by RNN and conclude them to feature vectors. The feature vectors is treated as an image and classified with CNN based image classification.

A. Overview

The overview of proposed method is shown in Fig. 1. Behavior of the process is composed of various activities such as file management, and these activities consist of multiple operations. When we record process behavior as an API call sequence, the individual API call represents the operation of the process, multiple API calls represent the activity, and whole recorded API calls represent behavior of the process. This hierarchical architecture is similar to the construction of writings. One writings is composed of various sentences, and these sentences are consisted of multiple words. For that reason, we presume that we can extract the feature of process behavior by using language model and we can train the feature of process behavior to RNN. After training, we extract features as feature vectors from the process for validation using trained RNN.

The feature vectors extracted from the process behavior log is converted to an image. This image potentially contains various local features which represent process activities. We apply CNN to classify these images by training local features. The CNN trains imaged features of malware and benign processes to create a classifier. In this way, our proposal divides process classification to two parts and use suitable DNN.

The training flow is divided to four parts as shown in Fig. 1. First, behavior of processes is monitored and generated log files. Second, the RNN is trained to construct behavioral language model by using log files. Third, features are extracted from the log file by trained RNN and convert features to a feature image. Finally, the CNN is trained with training feature images which has malware or benign label.

After training DNNs, we evaluate the process for validation with the trained DNNs. First, feature images are generated from the process behavior log file by the trained RNN. Next, these images are classified whether malware process or not by the trained CNN. Then we calculate malware probability of the process using the classifier output.

The detail of each procedure is explained in following subsections.

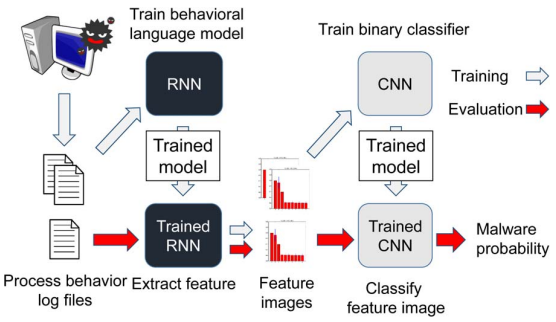


Fig. 1. Overview of proposed method.

TABLE I
LOGGED INFORMATION.

Time	Time when the Operation is executed
Process Name	Process name of the Operation
PID	Process ID of the Operation
Event	Name of the Operation
Path	Current directory when the Operation is executed
Result	Result of the Operation
Detail	More information about the Operation

B. Logging Process Behavior

We record behavior of all executed processes in predefined length and times under predefined intervals. We use Process Monitor[1] to log behavior such as `ReadFile`, `RegSetValue`, `Thread Start`, and so on. Behavior items which we logged by Process Monitor is shown in Table I. The Result shows a result code of the Operation like `SUCCESS`, `ACCESS DENIED`, `FILE NOT FOUND`, and so on. The Detail shows some parts of information about arguments. In this study, we use the logged Operations as process behavior instead of API call sequence.

We performed 5 minutes logging and 5 minutes interval set for 10 times. Log files are created for individual processes which has different PID. Each recorded log contains 7 items shown in Table I. We use input context of Event and Result for malware classification and reminder becomes identifier of the process. Since this section, we define the Operation as a set of Event and Result.

C. Training RNN

Based on the Operations, we construct behavioral language model. We use RNN with LSTM units for the model. The RNN consists of an input layer $\mathbf{x}$, a normal hidden layer $\mathbf{h}^1$, two LSTM layers $\mathbf{h}^2$ and $\mathbf{h}^3$, and an output layer $\mathbf{y}$. In training phase, we also use Dropout [3] for non-recurrent connection. The flow of the RNN is shown in Fig. 2. The input vector $\mathbf{x}$ is the converted 1-hot vector which represents an individual Operation $\mathtt{OP}_t$. The conversion is performed as follows.

1) Creating a dictionary in which IDs and Operations are associated each other

[1]https://technet.microsoft.com/ja-jp/sysinternals/processmonitor.aspx

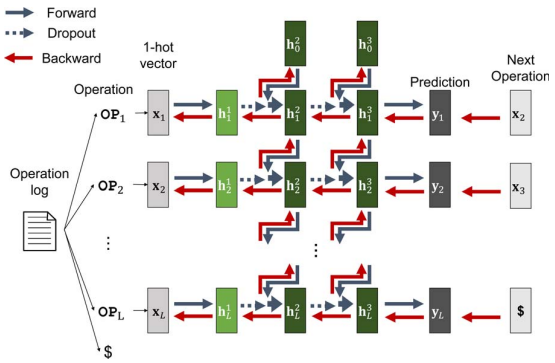


Fig. 2. Flow of RNN training.

2) Converting Operations to 1-hot vectors which is filled with 0s excepting a position associated with the Operation ID (give 1 for this position).

The RNN is trained by repeatedly using log files. First, we choose one log file and convert Operations = $\{\mathtt{OP}_1, \mathtt{OP}_2, ..., \mathtt{OP}_L\}$ to 1-hot vectors = $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$. Each 1-hot vector $\mathbf{x}_t$ is sequentially inputted to the RNN and it outputs prediction $\mathbf{y}_t$. Then we calculate loss function by comparing $\mathbf{y}_t$ with correct answer $\mathbf{x}_{t+1}$. After input T Operations, weights are updated by backpropagation.

There are some possibility that some Operations are only appears in validation log files in validation phase because it is not clear that training log files contain all Operations. To avoid this problem, we anonymize a part of the Operations in training log. We select a Operation in each log file which is appeared less than 10 times in the file and replace them to "Unknown Operation". This Operation replacement is performed for every log file.

One epoch of learning means that all training log files are inputted to RNN. The order of training log files is randomized in every epoch. The RNN training is executed for multiple epochs and we can obtain the trained feature extractor.

D. Feature Extraction and Imaging

We extract the feature of processes using trained RNN and generate feature images. The flow of feature extraction is shown in Fig. 3. The feature extractor we trained in Section III-C can project a next Operation from previous series of inputs. It means that the last hidden layer $\mathbf{h}^3$ contains information of previous inputs. Moreover, in DNN, local features are learned in layers which close to the input layer, and abstracted features are learned in deeper layer with combining local features. For these reasons, we expect that a behavioral feature is contained in the deep layer of the feature extractor. Accordingly, we regard a series of $\mathbf{h}^3$ as a feature of process behavior.

To generate a feature image, we first convert the Operations in the log file to 1-hot vectors same as Section III-C and input them to RNN sequentially. Let L be the length of Operations recorded in the log file. We extract the value of 3rd hidden layer $\mathbf{h}^3$ for every input and obtain series of feature

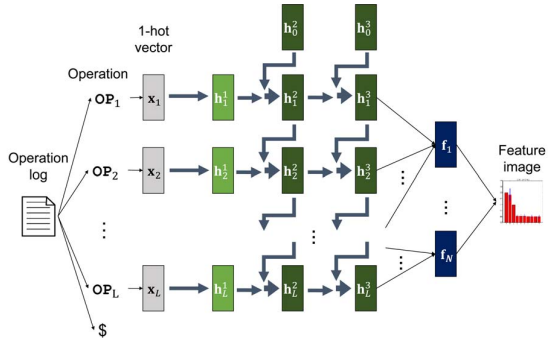


Fig. 3. Flow of feature extraction.

vector $\{\mathbf{h}_1^3, \mathbf{h}_2^3, ..., \mathbf{h}_L^3\}$. We designed feature classifier (Section III-E) to accept fixed size images so that we need to convert these series of vector to fixed length one because the length of Operations differs between log files. The conversion is described by following equations:

$$p_k = \begin{cases} 0 & (k = 0) \\ \lfloor \frac{L+k-1}{N} \rfloor + p_{k-1} & (1 \leq k \leq N) \end{cases} \quad (1)$$

$$\mathbf{f}_k = \frac{1}{p_k - p_{k-1}} \sum_{j=p_{k-1}+1}^{p_k} \mathbf{h}_j^3, \quad (2)$$

where $\mathbf{f}_k$ is the element of the fixed length series of vector, N is the height of the feature image, and p_k is the last number of kth vector set. The series of vector is divided into N sets and calculate the average of each set. Let W be the dimension of 3rd hidden layer, then the series of fixed vector can be described as a matrix F.

$$F = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_N \end{pmatrix} = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1W} \\ f_{21} & f_{22} & \cdots & f_{2W} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N1} & f_{N2} & \cdots & f_{NW} \end{pmatrix} \quad (3)$$

We map each element of F to the $[0,1]$ space by sigmoid function and multiply 255 to form 256 level gray scale image. Finally, the matrix F is outputted as the feature image which resolution is $W \times N$.

E. Training CNN and Perform Malware Process Detection

There are many features of activities in a feature image. Therefore, we use CNN to classify process local features appeared in feature images into malware specific features and benign features.

In the training phase, we train the CNN by using feature images that are labeled malicious or benign. The structure of the CNN is shown in Fig. 4. The CNN consists of an input layer, two convolution-pooling layers, a fully-connected layer, and an output layer. The first convolutional layer filters the $W_0 \times W_0 \times 1$ input image with 10 kernels. The second convolutional layer filters the $W_1 \times W_1 \times 10$ output of previous layer with 20 kernels. Each pooling layer receives the output of the previous convolutional layer and reduced their size into

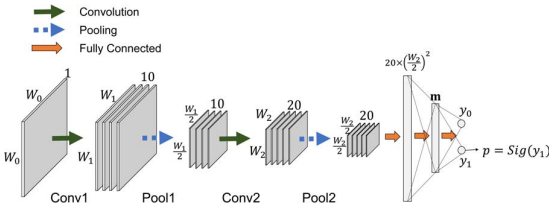


Fig. 4. Structure of the CNN.

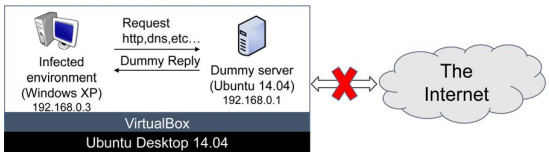


Fig. 5. Logging environment.

1/2 by Max-Pooling with stride of 2. The dimension of the output layer is two because the CNN is binary classifier.

In the validation phase, we use the trained CNN and calculate malware probability of the process. When the CNN receives a feature image of validation process, the CNN outputs a two dimensional vector which members represent malicious and benign degrees. If the input image is classified as malicious, then the value of the malicious is higher than the value of the benign. Malware probability p is calculated by applying following sigmoid function to the malicious class value.

$$p = Sig(y_1) = \frac{1}{1 + \exp(-y_1)} \quad (4)$$

IV. Experiment

To evaluate our proposal, we recorded behavior of malware and benign processes at the virtual environment which separated from the Internet and investigate the efficiency of proposed method with them.

A. Experiment Configuration

The logging environment is shown in Fig. 5. Malware files are executed on the Windows under monitoring of Process Monitor. The imitated environment of the Internet is simulated by INetSim[2], which is executed in dummy server. INetSim simulates common Internet services, such as HTTP, SMTP, DNS, FTP, and so on. All traffic from Windows XP are sent to the dummy server, and the server returns dummy responses. We regarded a process as a malware process which satisfies any of the following conditions:

1) The process of the predetermined malware file (same name)
2) The process which generated from the process 1)
3) The process which is injected malicious code from 1) and 2)

We used Cuckoo Sandbox to confirm whether the process satisfies condition 2) or condition 3) or not. Malware files are

[2]http://www.inetsim.org/

TABLE II
PARAMETER SETTINGS OF THE RNN.

Item	Cond. 1	Cond. 2	Cond. 3
Dimension of $\mathbf{h}^1$	30	30	10
Dimension of $\mathbf{h}^2, \mathbf{h}^3$	350	30	20
Other parameters	Epoch num: 5, Minibatch size: 20		

executed in the Cuckoo Sandbox and traced malware process behavior to determine generated and injected processes.

We used 81 malware process log files and 69 benign process log files for training and validation, which were logged and determined by the above method. Malware process log files were obtained from 26 malware files which are collected by NTT Secure Platform Laboratory from April 2014 to October 2014. Those malware files were classified into 11 families by Symantec. In the malware log files, 46 files were satisfied condition 1), 33 files were satisfied condition 2), and 2 files were satisfied condition 3).

In the RNN training phase, we used 44 malware process logs and 39 benign logs for training. We selected those files so that the total Operation length in files of malware and benign classes become almost same. Types of the Operation appeared in all files were 81.

We also trained and evaluated the CNN by 5-fold cross validation using 150 feature images generated from the trained RNN. Approximately 120 images were used for training, and the remaining images were used for validation in each cross validation.

To compare the performance of the classifier in different parameters, we trained and validated the RNN/CNN with several parameters. The used parameters are shown in Table II and Table III. We first set the size of feature image exceedingly large compared with the amount of data for training. This is the validation Condition 1. Then we evaluated with much smaller feature image size and it becomes Condition 2 and 3. We set the first RNN parameter by referring the settings of [8] which is using RNN with LSTM [5] for language modeling. The CNN parameters was also set by referring LeNet [9], which is used for recognizing hand written digit in 28×28 pixel image.

B. Evaluation Method

In multi-class classification problem, Positive means that an object x belongs to the class a due to exceeding a threshold (Th), otherwise it becomes Negative. In this method, the problem becomes classification into two classes so that the number of the threshold becomes only one. Let y be the result of classification of x. In this situation, True Positive (TP), False Positive (FP), True Negative (TN), False Positive (FN) are defined as shown in Table IV. True Positive Rate (TPR) is delivered by TP/P, and False Positive Rate (FPR) is delivered by FP/N. We also use Accuracy Rate which is delivered by (TP+TN)/(P+N). We evaluate the efficiency of the classifier by Area Under the Curve(AUC) which is calculated from ROC curve. ROC curve is a graph which indicates a relation between

TABLE III
PARAMETER SETTINGS OF THE CNN.

Item		Cond. 1	Cond. 2	Cond. 3
Conv1	Input size W_0	350×350	30×30	20×20
	Other parameters	Input channel: 1, Output channel: 10 Filter size: 5×5		
Pool1	Input size W_1	346×346	26×26	16×16
	Other parameters	Input channel: 10, Output channel: 10 Filter size: 2×2, Stride: 2		
Conv2	Input size $W_1/2$	173×173	13×13	8×8
	Other parameters	Input channel: 10, Output channel: 20 Filter size: 5×5		
Pool2	Input size W_2	169×169	9×9	4×4
	Other parameters	Input channel: 20, Output channel: 20 Filter size: 2×2, Stride: 2		
Dimension of $\mathbf{m}$		1000	250	40
Other parameters		Epoch num: 50, Minibatch size: 20		

TABLE IV
CLASS CLASSIFICATION PROBLEM.

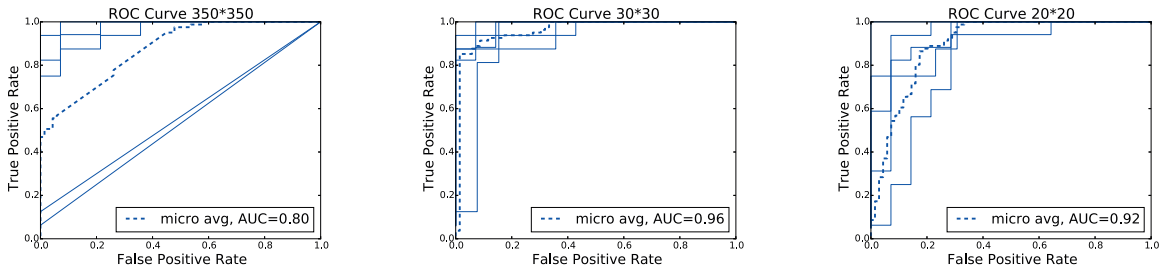
		Classified class		
		$y = a$(Positive)	$y \neq a$(Negative)	
Real class	$x \in a$	TP	FN	P=TP+FN
	$x \notin a$	FP	TN	N=FP+TN

TPR and FPR under threshold values. In our method, processes are classified malware or benign with malware probability p delivered by (4). The range of p is $[0,1]$, thus the range of threshold value is also $[0,1]$. We calculated ROC curve for each condition by regarding TPR as detection rate of malware processes and FPR as error detection rate of benign processes. Moreover, we compared the AUC in each condition to evaluate classifier efficiency.

C. The Experimental Result and Discussion

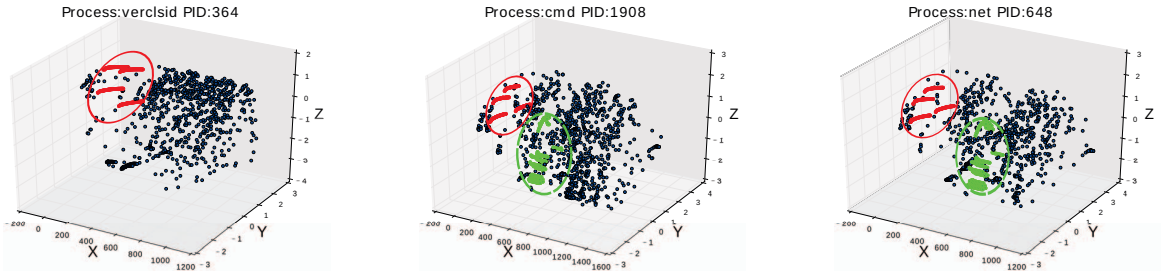
Fig 6 shows ROC curves. The horizontal axis means the error detection rate, and the vertical axis means the detection rate. Five Solid lines represent the individual ROC curve of 5-fold cross validation, and the broken line represents the micro average of individual ROC curve. The average AUC of Condition 1, 2, 3 were 0.80, 0.96, 0.92 respectively. Thus, our proposal can detect malware processes in high precision. In our proposed method, the feature of process behavior is trained first, and then classify the process using the feature image. In this section, we discuss validity of training the feature extractor.

If the RNN is trained well, some kind of regularity should be appeared in the extracted features. Therefore, we analyzed the series of feature vector which is extracted in Condition 2. We converted individual vector to two dimensional vector by principal component analysis. Then we plotted them to three dimensional graph. The example of graphs are shown in Fig. 7. `Verclsid.exe` (Fig. 7(a)) is benign process, `cmd.exe`



(a) Condition 1 (b) Condition 2 (c) Condition 3

Fig. 6. ROC curves of each condition.



(a) `verclsid.exe` (benign) (b) `cmd.exe` (Trojan.Zbot) (c) `net.exe` (Trojan.Zbot)

Fig. 7. Example of analyzed feature vectors.

and `net.exe` (Fig. 7(b), (c)) are different malware processes but belong to Trojan.Zbot family. X axis means the vector sequence, Y and Z axis means the value of vector element. As in `cmd.exe` and `net.exe`, distribution of some vector members are resemble each other (green part) even if the process binary is different. Moreover, partially similar points are also seen around the sequence 0 to 200 among three of them (red part). The Operation sequence of those processes are also resemble. Thus, we can say that the extracted features represents the behavior of process. Hence we can also say that the feature extractor is properly trained. On the other hand, in Condition 1, the amount of data we used for training and validation may not be large enough compared with the complexity of DNNs so that it shows small AUC. Thus, we will be able to classify malware processes with much more preciseness by using larger amount of data.

V. Conclusion

In this paper, we proposed the malware process detection method with two stage DNNs for infection detection. Our proposal detects malware process by classifying the feature images by the CNN. The feature image is generated from extracted behavioral features by behavioral language model which is constructed with RNN. We validated the classifier with 5 fold cross validation using 150 process behavior log files. We compared the result of validation which was performed in several conditions and got best result AUC= 0.96 when the feature image size was 30×30. We also analyzed the feature which is extracted from trained RNN with principal component analysis to proof effectiveness of the proposal. On the other hand, we couldn't utilize large scale DNN due to the small amount of data so that increasing data amount becomes future work. Increasing the amount of data to validate dataset is also future work.

References

[1] 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 *Proc. of the 2nd ACM workshop on Security and artificial intell.*, 2009, pp. 55–62.

[2] Y. Otsuki, M. Ichino, S. Kimura, M. Hatada, and H. Yoshiura, "Evaluating payload features for malware infection detection," *J. of Inform. Process.*, vol. 22, no. 2, pp. 376–387, 2014.

[3] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural inform. process. syst.*, 2012, pp. 1097–1105.

[5] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[6] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *11th Annu. Conf. of the Int. Speech Commun. Assoc. 2010 (INTERSPEECH)*, vol. 2, 2010, pp. 1045–1048.

[7] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *2015 IEEE Int. Conf. on, Acoust., Speech and Signal Process. (ICASSP)*, 2015, pp. 1916–1920.

[8] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[9] Theano Development Team, "Convolutional Neural Networks(LeNet)," http://deeplearning.net/tutorial/lenet.html.