Improved MalGAN: Avoiding Malware Detector by Leaning Cleanware Features

Masataka Kawai, Kaoru Ota, Mianxing Dong
Department of Information and Electronic Engineering
Muroran Institute of Technology
Muroran, Japan 050-0071
E-mail:{15024306, ota, mxdong}@mmm.muroran-it.ac.jp

***Abstract*—In recent years, researches on malware detection using machine learning have been attracting wide attention. At the same time, how to avoid these detections is also regarded as an emerging topic. In this paper, we focus on the avoidance of malware detection based on Generative Adversarial Network (GAN). Previous GAN-based researches use the same feature quantities for learning malware detection. Moreover, existing learning algorithms use multiple malware, which affects the performance of avoidance and is not realistic on attackers. To settle this issue, we apply differentiated learning methods with the different feature quantities and only one malware. Experimental results show that our method can achieve better performance than existing ones.**

***Keywords*—deep learning, adversarial machine learning, GAN, malware detection.**

I. Introduction

McAfee Labs announced in threat report of March 2018 that it confirmed on average eight new malware samples per second in the fourth quarter of 2017 [1]. In order to counter those increasing malware, automatic detection by software is indispensable.

In recent years, machine learning has achieved excellent results in the field of pattern recognition, so malware detectors also use it now [2] [3]. However, attackers may try to avoid detection by changing malware's behavior patterns, strings, APIs, and DLLs. Therefore, research on attacks and avoidance against machine learning algorithms are being conducted. These research are called "Adversarial Machine Learning".

As a previous research of Adversarial Machine Learning, Narodytska et al. [4] avoided image classification by Convolutional Neural Network (CNN) by applying microfabrication that is invisible to the human eye to images. Tramer et al. [5] performed Equation-Solving Attacks and Path-Finding Attacks based on multiple input values and corresponding output values for a service using machine learning, and stole the machine learning model such as Logistic Regression (LR), Decision Tree (DT), Multi Layer Perceptron (MLP), and Support Vector Machine (SVM). Hu et al. [6] generated feature quantities of malware to avoid detection by incorporating detection results of malware detectors using machine learning into GAN [7] based neural networks called MalGAN. GAN is a machine learning technique that makes competition of learning of two neural networks, Generator and Discriminator (Figure 1).

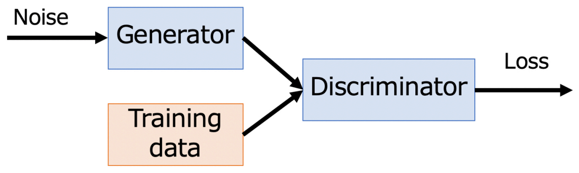

Figure 1. Configuration of GAN.

Generator learns to generate data similar to the training data in order to avoid Discriminator identification. Discriminator learns to accurately identify the training data and the generated. Finally, GAN generates data indistinguishable from the training data.

We are very interested in avoiding malware detection by MalGAN. However, their program [8] has some issues from a realistic viewpoint, so we try to improve them.

II. Avoidance Method by MalGAN

A. Overview

MalGAN consists of two neural networks, Generator and Substitute Detector, and it takes a classification result of a malware detector (Black-Box Detector) (Figure 2). Treating the malware detector as a Black-Box means that attackers can not access each parameter of the machine learning algorithm used for the detector. MalGAN and the detector uses the Windows API for feature quantities, because malware detection researches often use it. Furthermore, Hu et al. have reduced the number of dimensions to 128 by using the variable importance of Random Forest (RF).

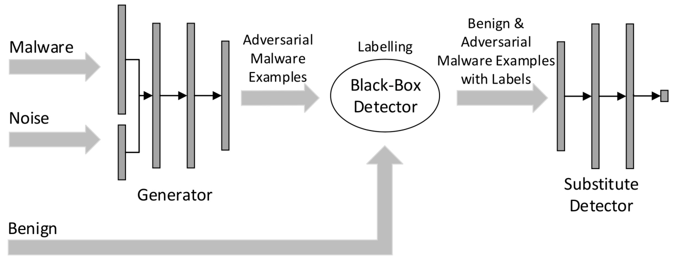
Malware
Noise
Generator
Adversarial Malware Examples
Labelling
Black-Box Detector
Benign & Adversarial Malware Examples with Labels
Substitute Detector
Benign

Figure 2. Configurarion of MalGAN.

B. Generator

Based on the classification rules of the malware detector learned with Substitute Detector, Generator generates feature quantities classified as cleanware (Benign) by adding noise to malware. Noise consists of 20 dimensions, randomly generated in the range [0, 1]. Generator networks consist of two layers of Fully Connected layers (Dense), the Sigmoid function for the activation function, binary cross entropy for loss the function, and Adam with learning rate 0.001 for the optimizer. Detailed Generator model is shown in Table I. Here, Generator adds the feature quantities of malware to the generated ones by using Maximum Layer not to impair the function as malware. Also, 'None' means the number of files used for one learning.

TABLE I
GENERATOR MODEL

Layer Type	**Output Shape**
Input Layer (malware)	(None, 128)
Input Layer (noise)	(None, 20)
Concatenate (malware+noise)	(None, 148)
Dense	(None, 256)
Activation (Sigmoid)	(None, 256)
Dense	(None, 128)
Activation (Sigmoid)	(None, 128)
Maximum	(None, 128)

C. Substitute Detector

Substitute Detector learns classification rules of malware and cleanware classified by the malware detector. Also Substitute Detector networks also consist of two layers of Dense, the Sigmoid function for the activation function, binary cross entropy for the loss function, and Adam with learning rate 0.001 for the optimizer. Detailed Substitute Detector model is shown in Table II.

TABLE II
SUBSTITUTE DETECTOR MODEL

Layer Type	**Output Shape**
Input Layer	(None, 128)
Dense	(None, 256)
Activation (Sigmoid)	(None, 256)
Dense	(None, 1)
Activation (Sigmoid)	(None, 1)

D. Malware Detector

The malware detector classifies the feature quantity generated by Generator, and the cleanware. The classification results are treated as labels on Substitute Detector. The detector uses RF, LR, DT, MLP, SVM, and Voting algorithm (majority voting against prediction probability of these classifiers).

E. Issues

The following issues are seen in the program of the previous research.

The first issue is to import the malware detector in MalGAN, training and predict. This is obviously convenient for attackers, and should be done externally.

The second issue is that the feature quantities used in MalGAN is reduced to 128. This means that the feature quantities of the original malware are not retained.

The third issue is that both MalGAN and the detector use the same API list as feature quantities. When they used function of RF to create API lists, they used not only training data and test data of MalGAN but also those of the detector. This means that attackers know all the malware and cleanware used learning the detector.

The fourth issue is using multiple malware to learn in MalGAN. This may affect the performance of avoidance. And typically, attackers would want to generate that variant for a single malware.

III. PROPOSED METHOD

A. Overview

We improve those issues as follows.

1: Execute the malware detectors externally using Python's subprocess library, and import the detection results into MalGAN.

2: Instead of APIs reduction based on importance of RF, add all the APIs used for the malware to the feature quantities.

3: Create MalGAN and the detector API list from different training data.

4: Only use one malware for MalGAN.

Improved MalGAN is shown in Figure 3.

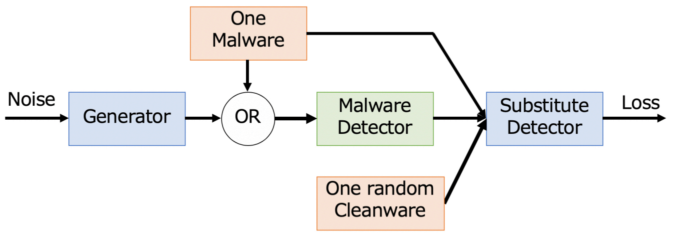

Figure 3. Configurarion of improved MalGAN.

Improved MalGAN first creates an API list from multiple cleanware and one malware. By doing this, MalGAN tries to avoid malware detection by adding cleanware's feature quantities to malware. Noise generates the values in the range of [-1, 1] for the number of APIs in the API list, and inputs to Generator. Generator finally generates the values in the range of (-1, 1) by the Tanh function. The reason for using the Tanh function instead of the Sigmoid function to deal with the value in the range (-1, 1) is to use the Parametric ReLU (PReLU) function [9] described later. Then, replace these in the range of (0, 1), round off, and perform OR operation with the feature quantities of the original malware. This OR operation is to add malware function to the generated data. After the OR operation, write the corresponding API to files. In this experiment, we generated 200 files for each epoch. Then use the subprocess library, input the generated files to the malware detector, and labeling. After labeling, input labeled data in Substitute Detector and learn classification rules of the malware detector. At this time, if all the generated files is classified as malware or cleanware, learning of Substitute Detector fails. Therefore, add to the input the original malware and the cleanware that randomly taken out used to create the API list. Finally, Generator learns to generate cleanware based on Substitute Detector weights. Substitute Detector learning and Generator learning repeat 10 times for each epoch.

B. Improved Generator model and Substitute Detector model

Improved Generator model and Substitute Detector model are shown in Table III-IV. These models are referenced to the model of Deep Convolutional GAN (DCGAN) [10]. DCGAN is a GAN model proposed by Radford et al. [11] for image generation using CNN. However, since the data dealt with in this research is not image data, we replace the Convolutional Layer with the Dense Layer. DCGAN usually uses the ReLU function or the Leaky ReLU function as the activation function, but we use the PReLU function.

In Generator, we use binary cross entropy for the loss function, and Adam with learning rate 0.0002 and β_1 = 0.5 for the optimizer. And in Substitute Detector, we use binary cross entropy for the loss function, and Adam with learning rate 0.00001 and β_1 = 0.1 for the optimizer.

TABLE III
IMPROVED GENERATOR MODEL

Layer Type	Output Shape
Input Layer	(None, 774[a])
Dense	(None, 512)
Activation (PReLU)	(None, 512)
Dense	(None, 256)
Batch Normalization	(None, 256)
Activation (PReLU)	(None, 256)
Dropout (0.5)	(None, 256)
Dense	(None, 128)
Batch Normalization	(None, 128)
Activation (PReLU)	(None, 128)
Dropout (0.5)	(None, 128)
Dense	(None, 64)
Batch Normalization	(None, 64)
Activation (PReLU)	(None, 64)
Dropout (0.5)	(None, 64)
Dense	(None, 774[a])
Activation (Tanh)	(None, 774[a])

[a]The number of dimensions depends on the number of APIs of malware and cleanware.

C. PReLU function

The PReLU function is represented by Equation (1). Where y is the input data matrix and a is a coefficient that controls the slope of the negative part.

TABLE IV
IMPROVED SUBSTITUTE DETECTOR MODEL

Layer Type	Output Shape
Input Layer	(None, 774[a])
Dense	(None, 64)
Activation (PReLU)	(None, 64)
Dense	(None, 128)
Batch Normalization	(None, 128)
Activation (PReLU)	(None, 128)
Dense	(None, 256)
Batch Normalization	(None, 256)
Activation (PReLU)	(None, 256)
Dense	(None, 512)
Batch Normalization	(None, 512)
Activation (PReLU)	(None, 512)
Dense	(None, 1)
Activation (Sigmoid)	(None, 1)

[a]The number of dimensions depends on the number of APIs of malware and cleanware.

$$PReLU(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases} \tag{1}$$

The PReLU function has the characteristics that it can self-learn the slope of the negative part of the Leaky ReLU function. GAN is known to be unstable in learning, but we thought that this characteristics is effective. Therefore, it adopts to improved MalGAN.

D. Malware Detector

We adopt the following algorithms for the malware detector.

- RF (n_estimators = 1000)
- MLP (hidden_layer_sizes = (64,))

For the API list of the detector, using the APIs of all files makes the dimension too large. Therefore, we restrict the number of files used to create the API list, and create two patterns. In addition, we create two patterns of API list with dimensions reduced to 128 using the variable importance function of RF as Hu et al. did.

IV. EXPERIMENTS

A. Experimental Setup

Program language is used Python 3.5.0, and Tensorflow 1.11.0 and Keras 2.2.4 use to create MalGAN, and Scikit-learn 0.20.0 uses to create the malware detector.

Dataset used FFRI Dataset 2018. This data set is provided by FFRI, Inc. as a part of research data set of anti Malware engineering WorkShop (MWS) [12]. And, it consists of hash value of malware and cleanware and surface analysis data [13]. Of this data set, 36265 malware and 26127 cleanware are used as training data of the malware detector. Also, 36047 malware and 26590 cleaware are used as test data of the detector. For training data of MalGAN, one malware and up to 44 cleanware are used.

First, from the training data used in the malware detector, we create an API list with the four patterns in Table V. Pattern Ⅲ and Ⅳ use the variable importance of RF.

TABLE V
CREATED API LIST FOR THE MALWARE DETECTORS

Pattern	Malware	Cleanware	APIs
I	1104	826	16337
Ⅱ	75	60	4045
Ⅲ	1104	826	128
Ⅳ	75	60	128

The classification results of the test data in each algorithm of the malware detector are shown in Table VI.

TABLE VI
CLASSIFICATION RESULTS OF THE MALWARE DETECTORS

	RF		MLP	
Pattern	TPR(%)	FPR(%)	TPR(%)	FPR(%)
I	96.9	3.8	97.4	4.5
Ⅱ	96.9	3.9	97.4	4.6
Ⅲ	96.0	8.2	95.9	9.2
Ⅳ	95.6	8.5	95.5	8.9

The TPR and FPR are derived by Equations (2-3).

$$TPR = \frac{TP}{TP + FN} \tag{2}$$

$$FPR = \frac{FP}{FP + TN} \tag{3}$$

Second, from the training data used in MalGAN, we create an API list. Also in MalGAN, we create the two patterns (Table VII). Generator generates data from these API lists.

TABLE VII
CREATED API LIST FOR MALGAN

Pattern	Malware	Cleanware	APIs
1	1	44	2452
2	1	4	774

B. Experimental Results

Figure 4-7 shows execution results of MalGAN for each pattern. The Y-axis shows the number of files classified as cleanware, and the X-axis shows epochs. In this experiment, we generated 200 files for each epoch, so the maximum value on the Y-axis is 200.

First, focus on the malware detectors, MalGAN can more easily avoid if the number of feature quantities used for the malware detectors increases (Figure 4-5). On the other hand, difficulty avoid if calculated importance from more feature quantities (Figure 6-7). Moreover, RF is seem an algorithm which is difficult to avoid for MalGAN than MLP. This means that RF is a more robust algorithm.

Second, focus on MalGAN, MalGAN can more easily avoid if the number of feature quantities used for MalGAN increases. However, this is a natural result and also causes another issue. Its issue is the number of APIs of the generated data. For example, when calculating the average API number of the generated data classified as cleanware at the 30 epochs of Pattern Ⅱ MLP-1 and Pattern Ⅱ MLP-2, it is as shown in Table VIII. It is too much than the number of APIs of the original malware, so it is not realistic.

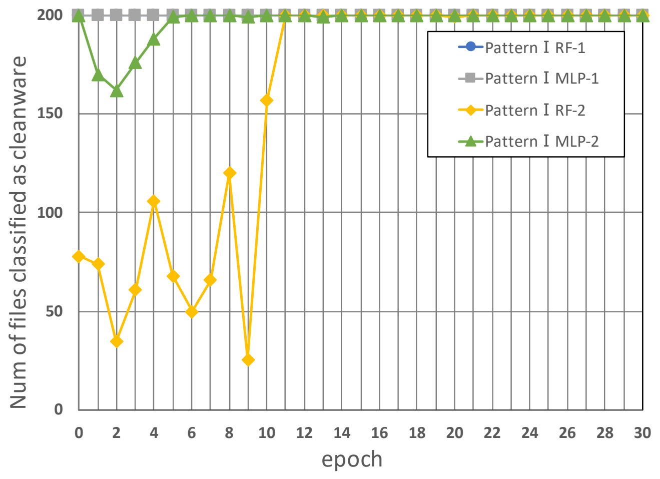

Figure 4. Result in Pattern I.

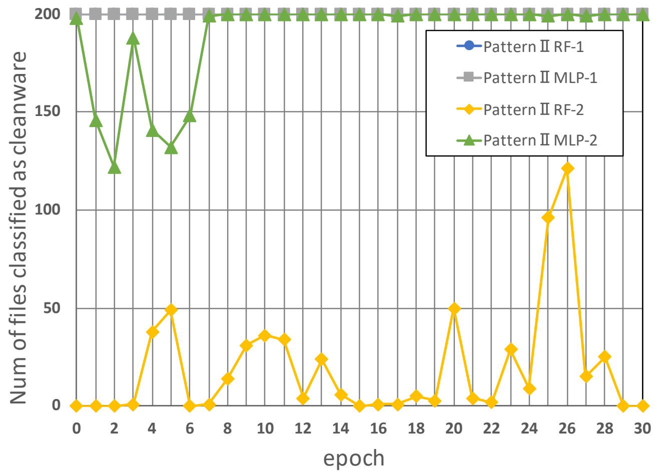

Figure 5. Result in Pattern Ⅱ.

TABLE VIII
NUMBER OF AVERAGE API THE GENERATED DATA CLASSIFIED AS CLEANWARE

Original Malware	Pattern Ⅱ MLP-1	Pattern Ⅱ MLP-2
148	1270	451

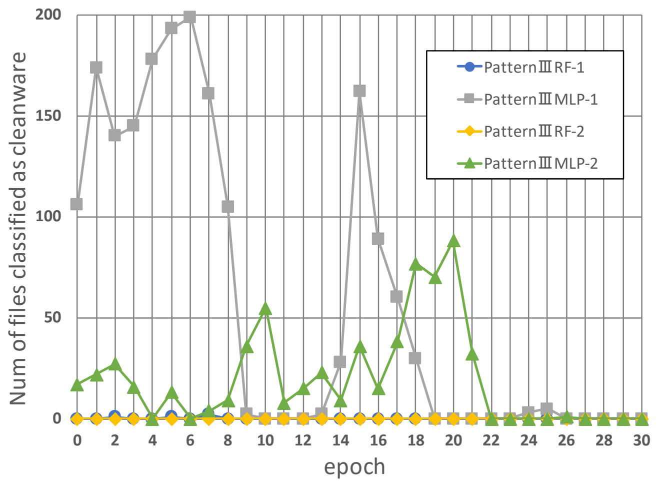

Figure 6. Result in Pattern Ⅲ.

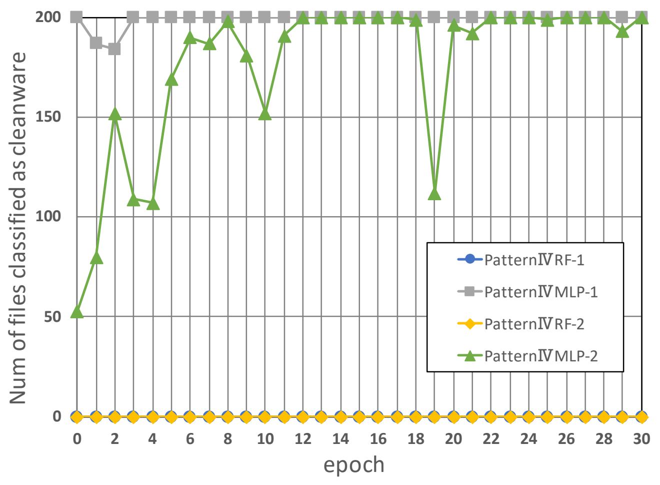

Figure 7. Result in Pattern Ⅳ.

To solve this issue, we have created a Layer that calculate only losses based on the implementation of Variational Autoencoder [14].This Layer is appended to the end of the Generator model. And calculates the customized Root Mean Square Error (RMSE) (Equation (4)) between the output data replaced by the range of (0, 1) and the feature quantities of the original malware. We created this equation so that it approaches the number of feature quantities of the original malware.

$$customized\ RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{m}(\hat{x}_{ij} - x_{ij})^2} \quad (4)$$

n is the number of files generated each epoch, m is the number of feature quantities, $\hat{x}$ is the output data replaced, and x is the original malware.

Here, if the calculated customized RMSE is directly added to the loss, the power to reduce the feature quantities is too strong, learning of MalGAN fails. Therefore, we added it multiplied by 0.05. Along with that, we changed binary cross entropy to that multiplied by 0.95.

Figure 8-11 shows the number of the generated data classified as cleanware and the number of average API for each epoch of Pattern Ⅱ MLP-1, Pattern Ⅱ MLP-2, Pattern Ⅳ MLP-1, and Pattern Ⅳ MLP-2. The number of average API of Pattern Ⅱ MLP-1 and Pattern Ⅱ MLP-2 has decreased to about 250 at the 30 epochs. But, the number of average API of Pattern Ⅳ MLP-1 and Pattern Ⅳ MLP-2 have larger than the previous two patterns.

From these results show that customized RMSE can reduce the APIs required to avoid the malware detectors. However, the learning becomes unstable by reducing the API, and more unstable by using importance of RF. Also, using the importance hinders the reduction of the number of APIs.

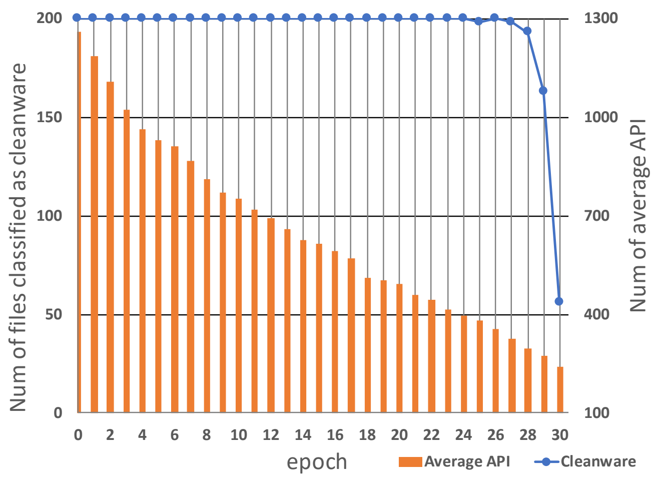

Figure 8. Result of Pattern Ⅱ MLP-1.

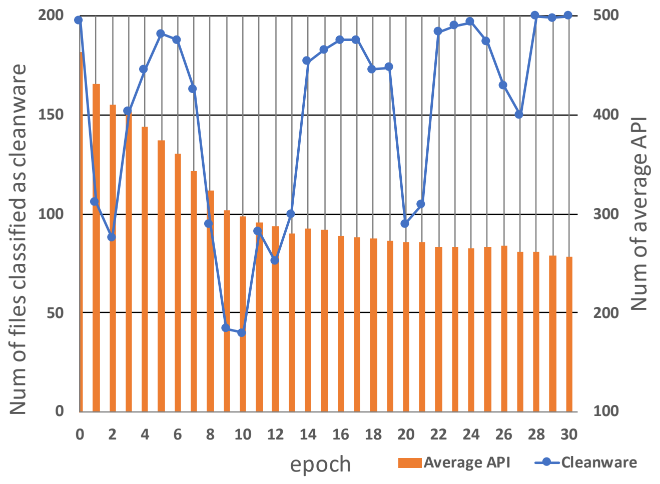

Figure 9. Result of Pattern Ⅱ MLP-2.

V. Discussion and Conclusion

In this paper, we point out issues of MalGAN proposed by Hu et al., and improved it from a more realistic viewpoint.

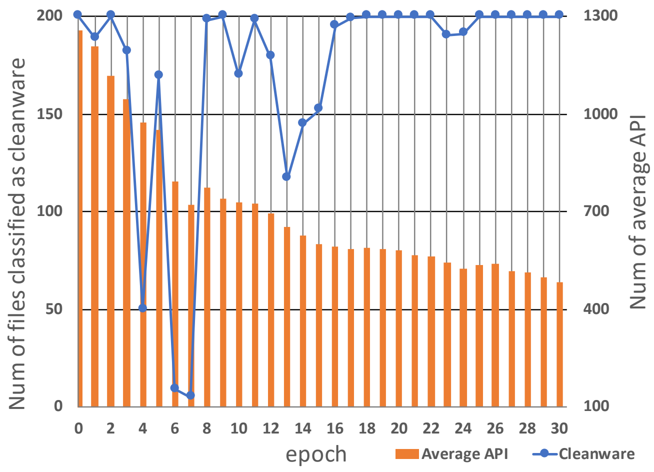

Figure 10. Result of Pattern IV MLP-1.

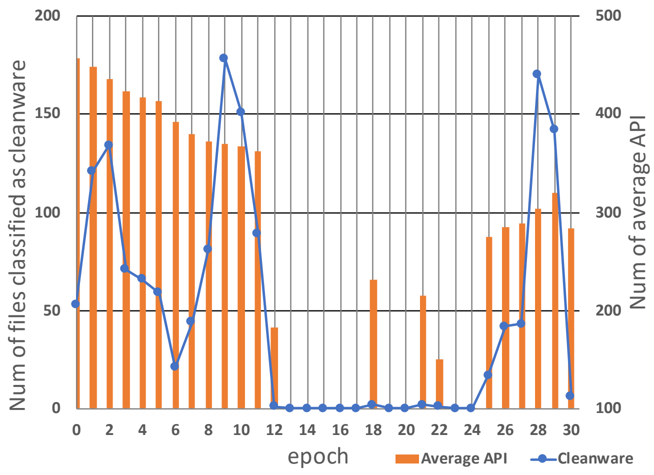

Figure 11. Result of Pattern IV MLP-2.

Improved MalGAN tries to avoid malware detectors by adding some cleanware features (APIs) to the original malware. Experimental results show that improved MalGAN can generate feature quantities, and avoid them.

These results also show that the number of feature quantities used for the detectors and MalGAN is closely related to generation capacity of MalGAN. Both the detectors and MalGAN tended to avoid more easily as the number of feature quantities increased. Therefore, as a countermeasure to MalGAN, it may seem that it is better to learn a malware detector with less feature quantities, such as using importance of RF. However, with fewer feature quantities, FPR tends to be large as shown in Table VI. In security areas, FPR rather than TPR is often more important, and it is not necessarily become a direct countermeasure.

Furthermore, we found issue of generating an enormous the number of APIs to avoid the detectors, in improved MalGAN. We solved it by newly adding loss calculation Layer to Generator. However, decreasing the number of APIs generated makes it difficult to avoid them, learning became unstable.

In this experiment, because the feature quantity used for the detectors is only API, it may seem very limited. However, there are other feature quantities that can be easily generated besides APIs, such as strings. In addition, the generation technology by machine learning will improve in the future, there is also the possibility to generate the binary itself. Therefore, security vendors need to keep secret that malware detectors what to be using as the feature quantity.

MalGAN is a dangerous technology, but we believe it is also useful for the creator of the detectors. Like RF and MLP in this experiment, the ease of avoidance by MalGAN depends on the algorithm of the detectors. Therefore, it is effective for testing the robustness of them. In addition, we believe it is effective for amplifying training data for detecting subspecies of malware.

As a next step, we would like to consider a machine learning method to disturb MalGAN learning. Specifically, we are planning to apply Adversarial Training, which is attracting attention as a way to disturb learning of GAN, to the detectors.

ACKNOWLEDGMENT

This work is partially supported by JSPS KAKENHI Grant-Number JP16K00117, and KDDI Foundation.

REFERENCES

[1] McAfee, LLC, “McAfee Labs Threat Report: March, 2018”, https://www.mcafee.com/jp/resources/reports/rp-quarterly-threats-mar-2018.pdf (accessed 2018-5-15).
[2] Ryota Nakamura, and Yoshihiro Oyama, “Comparative Evaluation of Online Machine Learning Algorithm for Behavior Based Malware Detection”, J-STAGE, computer software, Vol.34, No.4, pp.156-177, 2017.
[3] FFRI, Inc., “FFRI yarai”, https://www.ffri.jp/products/yarai/index.htm (accessed 2018-5-15).
[4] Nina Narodytska, and Shiva Kasiviswanathan, “Simple Black-Box Adversarial Perturbations for Deep Networks”, IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp.1310-1318, 2017.
[5] Florian Tramer, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart, “Stealing Machine Learning Models via Prediction APIs”, 25th USENIX Security Symposium, pp.601-618, 2016.
[6] Weiwei Hu, and Ying Tan, “Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN”, arXiv preprint arXiv:1702.05983, 2017.
[7] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, “Generative Adversarial Nets”, NIPS 2014, pp.2672-2680, 2014.
[8] yanminglai, “Malware-GAN”, https://github.com/yanminglai/Malware-GAN (accessed 2018-12-1).
[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, arXiv:1502.01852, 2015.
[10] jacobgil, “keras-dcgan”, https://github.com/jacobgil/keras-dcgan (accessed 2018-12-1).
[11] Alec Radford, Luke Metz, and Soumith Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, arXiv:1511.06434, 2015.
[12] Yuta Takata, Masato Terada, Takahiro Matsuki, Takahiro Kasama, Shoko Araki, and Mitsuhiro Hatada, “Datasets for Anti-Malware Research MWS Datasets 2018 ”, Information Processing Society of Japan, Vol.2018-CSEC-82, No.38, 2018.
[13] FFRI, Inc., “Introduction of FFRI2018dataset”, https://www.iwsec.org/mws/2018/20180530/FFRI_Dataset_2018.pdf (accessed 2018-12-1).
[14] Diederik P. Kingma, and Max Welling, “Auto-Encoding Variational Bayes”, arXiv:1312.6114, 2014.