Information Sciences 460–461 (2018) 83–102

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders

Jin-Young Kim, Seok-Jun Bu, Sung-Bae Cho*

Department of Computer Science, Yonsei University, Seoul, South Korea

ARTICLE INFO

Article history:
Received 22 September 2017
Revised 23 March 2018
Accepted 29 April 2018
Available online 18 May 2018

Keywords:
Malicious software
Zero-day attack
Generative adversarial network
Autoencoder
Transferlearning
Robustness to noise

ABSTRACT

Detecting malicious software (malware) is important for computer security. Among the different types of malware, zero-day malware is problematic because it cannot be removed by antivirus systems. Existing malware detection mechanisms use stored malware characteristics, which hinders detecting zero-day attacks where altered malware is generated to avoid detection by antivirus systems. To detect malware including zero-day attacks robustly, this paper proposes a novel method called transferred deep-convolutional generative adversarial network (tDCGAN), which generates fake malware and learns to distinguish it from real malware. The data generated from a random distribution are similar but not identical to the real data: it includes modified features compared with real data. The detector learns various malware features using real data and modified data generated by the tDCGAN based on a deep autoencoder (DAE), which extracts appropriate features and stabilizes the GAN training. Before training the GAN, the DAE learns malware characteristics, produces general data, and transfers this capacity for stable training of the GAN generator. The trained discriminator passes down the ability to capture malware features to the detector, using transfer learning. We show that tDCGAN achieves 95.74% average classification accuracy which is higher than that of other models and increases the learning stability. It is also the most robust against modeled zero-day attacks compared to others.

1. Introduction

Malicious software, also known as malware, is a generic term for all software products that adversely affect computers. The damage incurred by malware has been recently increasing. For example, malware can attack automated teller machines (ATMs), causing significant loss [12]. Malware has been steadily growing in speed (rapidity of threats), number (growing threat landscape), and discrepancy (introduction of new methods) [15]. Malware developers constantly modify existing malware to avoid detection by antivirus systems. The conventional method to detect modified malware is to store all malware characteristics and detect new malware by matching the patterns of stored malware characteristics. The Symantec Intelligent organization stored 44 million new samples just in 2015 [1]. Automatic detection of malware is needed because storing all malware characteristics is very costly and inefficient.

The methods for malware detection have been extensively studied, but most of them use stored features of malware or detect malware based on several rules, which fail to detect zero-day attacks. Zero-day attacks are undisclosed attacks of computer software that hackers can exploit to adversely affect computer programs, data, or networks of computers. Zero-day

* Corresponding author.
E-mail addresses: seago0828@yonsei.ac.kr (J.-Y. Kim), sjbuhan@yonsei.ac.kr (S.-J. Bu), sbcho@yonsei.ac.kr, sbcho@cs.yonsei.ac.kr (S.-B. Cho).

https://doi.org/10.1016/j.ins.2018.04.092
0020-0255/

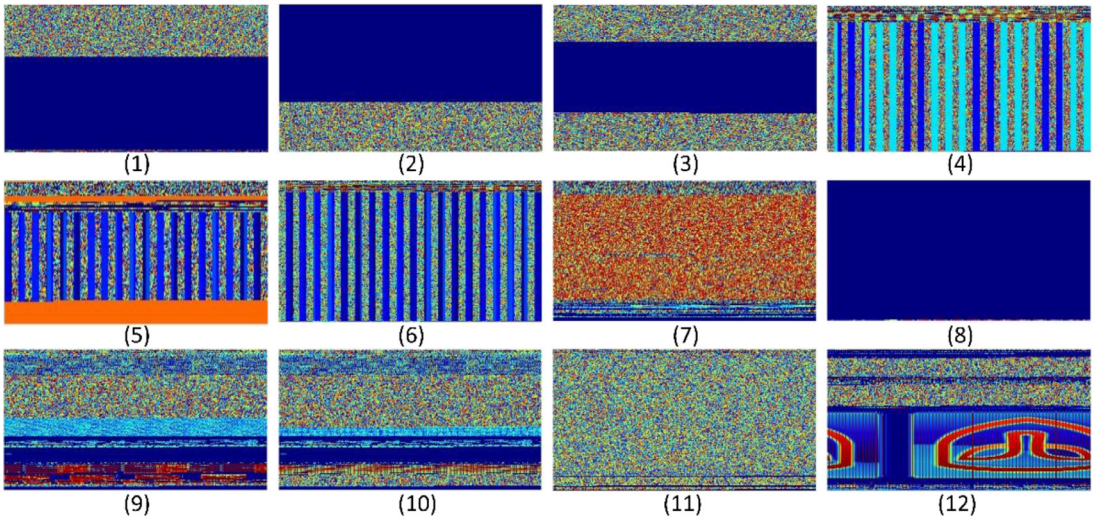


Fig. 1. Twelve typical patterns of malware images. The last image shows malware signed by the developer.

attacks are a critical issue in the field of computer security, with detection of zero-day attacks being the highest priority of malware detection systems (MDSs). Because antivirus software cannot treat zero-day attacks, computers are more vulnerable to them compared with general malware. Research that focuses on zero-day attacks is ongoing [20,2]. Typically, there are two approaches to detect malware of zero-day attacks [10]: One is to detect malware using the log obtained by executing malware directly in virtual environment, which requires time, high resource consumption, and has low scalability. The other is to detect malware using the preprocessed features from raw code, which has limitations in detecting particular variants of malware.

To mitigate both limitations, we propose a deep learning method to detect malware based on raw code without running the malware directly and to generate malware with arbitrarily modified features. We assume that zero-day attacks can be modeled by adding noise to existing malware data, and investigate the method for detecting zero-day attacks. Before generating the data directly, it is important to know the characteristics of the data. A deep autoencoder (DAE) can represent the features of data by unsupervised learning, where the decoder reconstructs the data compressed by the encoder. We exploit the decoder to generate the competitive new data for the generative adversarial network (GAN). Our main contribution is to devise a unique architecture by combining several deep learning methods of DAE, GAN, and transfer learning to construct the final malware detector that is robust to noise and zero-day attacks.

The remainder of this paper is organized as follows. Section 2 reviews the background of malware and the conventional malware detection techniques, and shows how these techniques differ from our proposal. The architecture and background of the proposed method, the model's validity, and finally the malware detection algorithms are presented in Section 3. In Section 4 we demonstrate the performance of the proposed method, compare it with conventional models, and visualize the intermediate outputs of the model. Some conclusions are discussed as well as the future works in Section 5.

2. Background

2.1. Difficulties in malware detection

In this section, we show examples of malware data and some variants of malware. Examples of malware data[1] used in this paper are shown in Fig. 1, which illustrates twelve typical patterns of malware images. The malware code, which is subject to certain rules, generates specific patterns. However, even if images have the same form, the corresponding code may be different, which is done to avoid detection. As shown in Fig. 2, malware is generated with similar-shape but different codes, making it difficult to detect malware using only pattern matching, which uses null value (insignificant value) insertion. In addition, Fig. 3 is another example of obfuscations by reordering of subroutines. Two malware call the same subroutine, but they are located in different positions. In this paper, we propose a generative deep learning model that is resistant to noise, for coping with various and rapidly modified malware.

[1] The details of malware data are presented in Section 4.1. We convert it to image, say *malware image*, using equations (18) and (19).

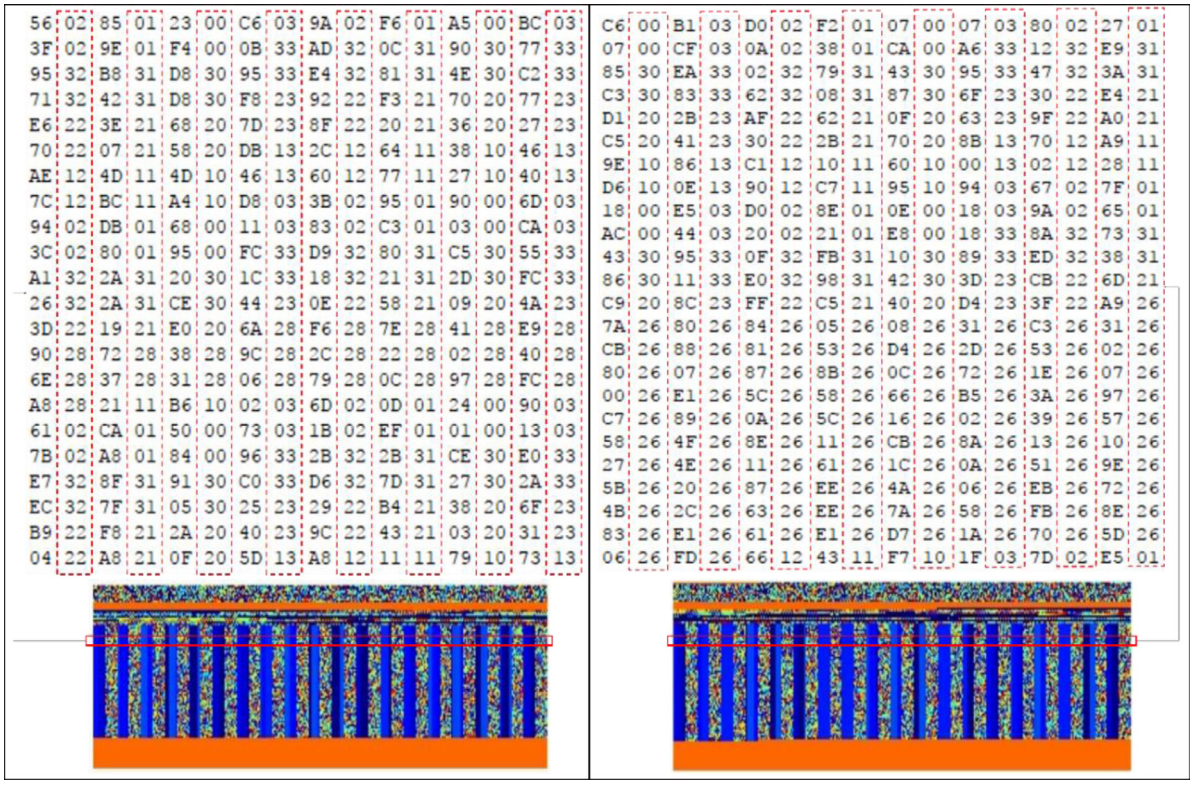


Fig. 2. The same pattern of figures using different hexadecimal codes. Owing to these modifications, a noise-robust model is needed.

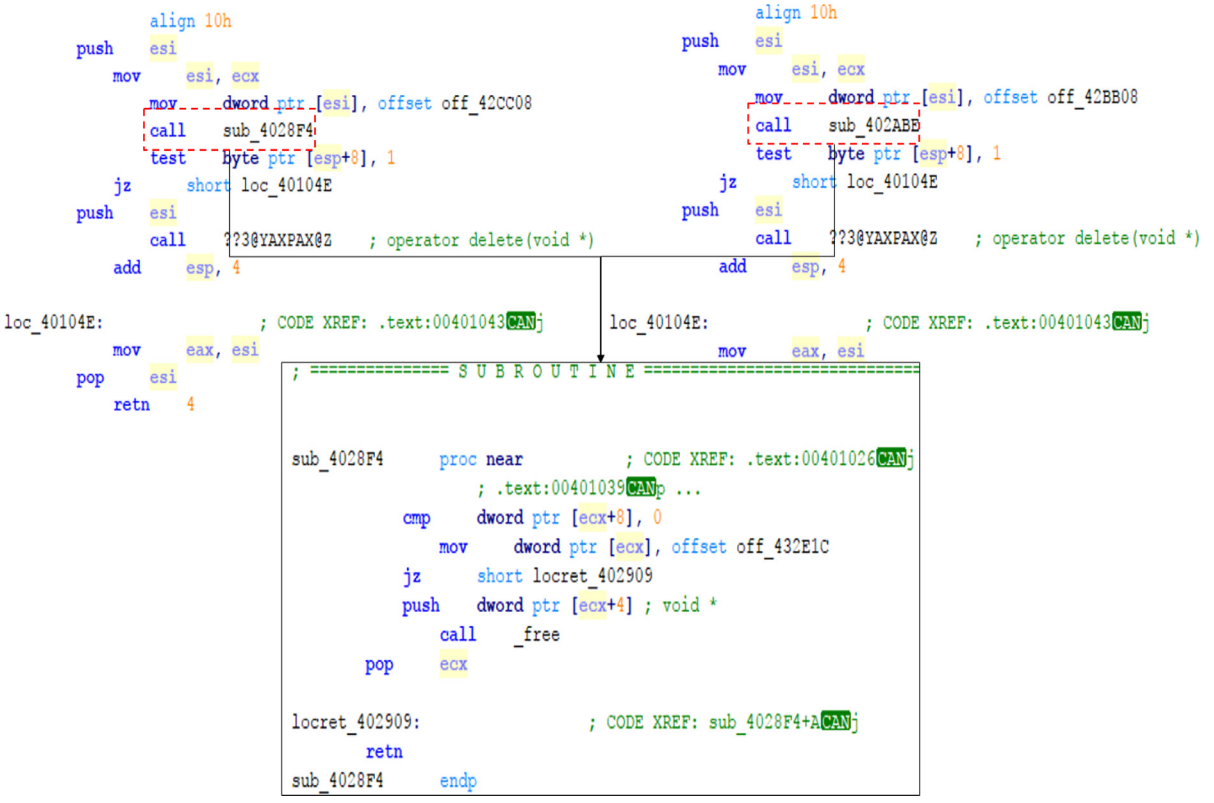


Fig. 3. An example of obfuscations of malware. Left top and right top are assembly codes which call the same subroutine as in the bottom, but they are located in different positions.

2.2. Related work

Many studies have been conducted to address malware detection because the damage incurred by malware has been increasing and the need for malware detection methods is evident. We discuss the approaches for malware detection in three categories: analysis, detection, and zero-day malware detection.

First, Christodorescu analyzed malware data and detected new malware by comparing it with old malware [9]. Analysis of malware data allows some reduction of the amount of stored information, but remains disadvantageous owing to the cost of storing the malware features. Nataraj preprocessed and classified malware data written in the assembly or binary code, into images [33]. This method allows classification of malware using visual characteristics, as shown in Fig. 1. However, as shown in Fig. 2, the method cannot recognize modified features. Mas'ud extracted various features from malware and evaluated each feature set using machine learning methods to select the effective features [30]. Cao proposed a method that discovers hidden suspicious accounts using a forwarding message tree. They exposed the inner relationships among hidden suspicious accounts [7]. They developed methods to extract features efficiently for capturing modification and scalability from simple comparison using machine learning. We utilize not only the method of visualizing malware proposed by Nataraj, but also DAE that can represent the data by unsupervised learning.

Several studies have addressed malware detection and zero-day attacks. Deep learning methods have become popular in this area. Berlin and Ye used Windows API and Windows Audit Log to detect malware [6,42], but the disadvantage of this approach is that only known malware can be detected; thus, the method cannot learn modified features. Some malware detectors utilize machine learning algorithms such as hidden Markov models (HMMs), clustering, random forests, and recurrent neural networks [4,25,17,34]. Wang proposed malware detection using an adversary resistant deep learning model with random feature nullification [40], but, in this approach, malware characteristics are eliminated at random, resulting in the risk of eliminating not only noise but also meaningful features. Singh proposed a distributed framework based on random forest to build a practical system [37]. Chen et al. tried detecting mobile malware including highly imbalanced network traffic by using imbalanced data gravitation-based classification [8]. Because malware developers create new variants with different signatures from existing malware using obfuscation techniques such as dead-code insertion, code interchanging, and subroutine reordering, they attempted to detect malware by using several methods that are robust to modification. However, they deal with a particular variant of malware and may miss other obfuscation methods.

Studies on zero-day attacks were based on the detection of malware content [2] and detection of zero-day Android malware using first-order and second-order analyses [20]. Santos modeled a sequence of opcodes to add the flexibility that captures zero-day attacks [36]. Huda proposed the semi-supervised learning of unlabeled data that automatically integrates the knowledge about unknown malware from already available and cheap unlabeled data into the detection system using k-means clustering algorithm and SVM classifier based on tf-idf features [21]. In these studies, some rules were set, and malware was detected based on these rules such as capturing sequence of opcodes, extracting dynamic features, and searching for common contents in malware. The malware that would not conform to these rules would not be detected. Furthermore, the methods extracted features or detected malware dynamically in virtual space and suffered from time intensity, resource consumption and scalability.

Research on malware analysis and malware detection is summarized in Table 1. However, these studies did not deal with scalability and noise robustness as discussed in Section 1; thus, detection of modified malware is not guaranteed. To detect malware even if it is modified, we propose a method that exploits deep learning with a generative adversarial network based on a deep autoencoder. The proposed method extracts appropriate features with DAE, generates virtual malware data for expanding the range of malware with GAN, and detects malware with the generated and the real data by transferring discriminator to detector.

3. The proposed method

Let $\mathbb{D}$, $\mathbb{C}$ and malware detector ψ be data and class spaces of malware, respectively, and map from $\mathbb{D} \times \mathbb{C}$ to {True, False}, respectively. Figs. 4 and 5 show that malware data can have complex features, which are assessed in terms of correlations and heat maps using structural similarity[2] (SSIM), mean squared error (MSE), peak signal to noise ratio (PSNR) and cosine similarity (CSS) measures. To improve ψ, we use machine learning to understand the complex characteristics of data, and a GAN to make the method robust to noise.

3.1. Overview of the proposed system

Fig. 6 shows the architecture of the malware detection system (MDS) proposed in this paper. The system consists of three parts: 1) data compression and reconstruction, 2) generation of fake malware data, and 3) malware detection. Transfer learning is used to connect the different parts [5]. This method allows learning tasks more effectively by using a similar model that solves the same task but has a different domain, which can be considered as transferring the ability of the trained model to the other model.

[2] The formula details of SSIM are in Section 4.4.

Table 1
Summary of related work on malware detection.

Category	Author	Method	Description
Malware Analysis	M. Christodorescu [9]	Semantic aware	Semantic comparison with example malware data
	L. Nataraj [33]	Visualization of malware	Malware detection through visualization of malware data
	M.Z. Mas'ud [30]	Combination of feature selection methods	Apply basic machine learning methods to find effective features for malware
	J. Cao [7]	Forward message tree	Expose inner relations among hidden suspicious accounts.
Malware Detection	K. Berlin [6]	Logistic regression	Logistic regression with Windows audit logs
	D. Kong [25]	Clustering	Malware detection with clustering method
	R. Pascanu [34]	Recurrent neural network	Detect malware by considering malware data as time series data
	C. Annachhatre [4]	Hidden Markov model	Detect malware using k-means clustering and HMM
	F.C.C. Garica [17]	Random forest	Convert binary to image and use random forest to detect malware
	Q. Wang [40]	Deep neural network	Adversary resistant malware detection through random feature nullification
	Y. Ye [42]	Restricted Boltzmann machine	Malware detection with Windows API and RBM
	K. Singh [37]	Random forest with parallel processing	Build a distributed framework based on random forest for practical system
	Z. Chen [8]	Imbalanced data gravitation-based classification	Detect mobile malware which is highly imbalanced.
Zero-day Attack	P. Akritidis [2]	Content-based detection	Use worm detection method based on four observations
	M. Grace [20]	Ensemble	Two modules which detect malware in different way
	I. Santos [36]	Opcode-based detection	Use various machine learning methods to model the sequence of opcodes
	S. Huda [21]	*k*-means, SVM	Semi-supervised learning with unlabeled data based on tf-idf features

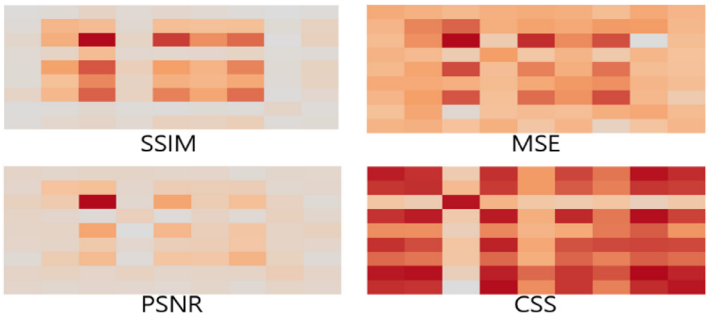


Fig. 4. Comparison of images using the SSIM, MSE, PSNR, and CSS measures. Darker color corresponds to stronger correlation. Different malware types are either fully correlated or completely uncorrelated. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

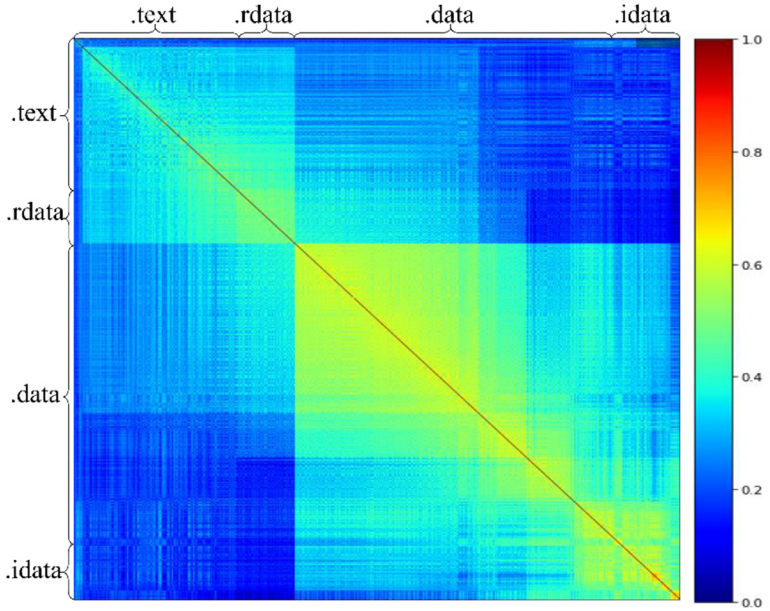


Fig. 5. Correlations between malware features. Blue color corresponds to a weak correlation. Because the variables are not strongly correlated, all features are used for malware detection. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

Before using malware data in the proposed MDS, the data are preprocessed. Then, the preprocessed data are used as the input to the first module of the MDS, which compresses and reconstructs the malware data preprocessed in the first part. To learn a representation of data, we use a DAE in the second part. The decoder of the autoencoder, which is trained to reconstruct malware data, is transferred into the generator of the GAN, which is the second part. The generator produces fake malware data given a known probability distribution and interacts with the discriminator of the GAN to learn the characteristics of malware data. The GAN is described in detail in the next section. Finally, the discriminator of the GAN is transferred to the malware detector of the MDS, and the system is trained to detect malware data.

3.2. Transferred deep convolutional GAN (tDCGAN)

Background. GANs have led to significant improvements in data generation [19]. They are used for image generation [22,14], video prediction [31], and in several other domains. The basic training process of a GAN is to adversely interact and simultaneously train a discriminator and a generator. The generator produces fake data from a known probability distribution and is trained to make the discriminator unable to distinguish the generated fake data from real data. The discriminator is trained to distinguish the real data from the fake data produced by the generator. Eq. (1) shows the objective function of a GAN. p_{data} is the probability distribution of the real data. $G(z)$ is generated from a probability distribution p_z by the generator G, and it is distinguished from the real data by the discriminator D. The discriminator is trained such that $D(x)$ of the first term is 1 and $D(G(z))$ of the second term is 0, to maximize $V(D, G)$, and G is trained such that $D(G(z))$ of the second

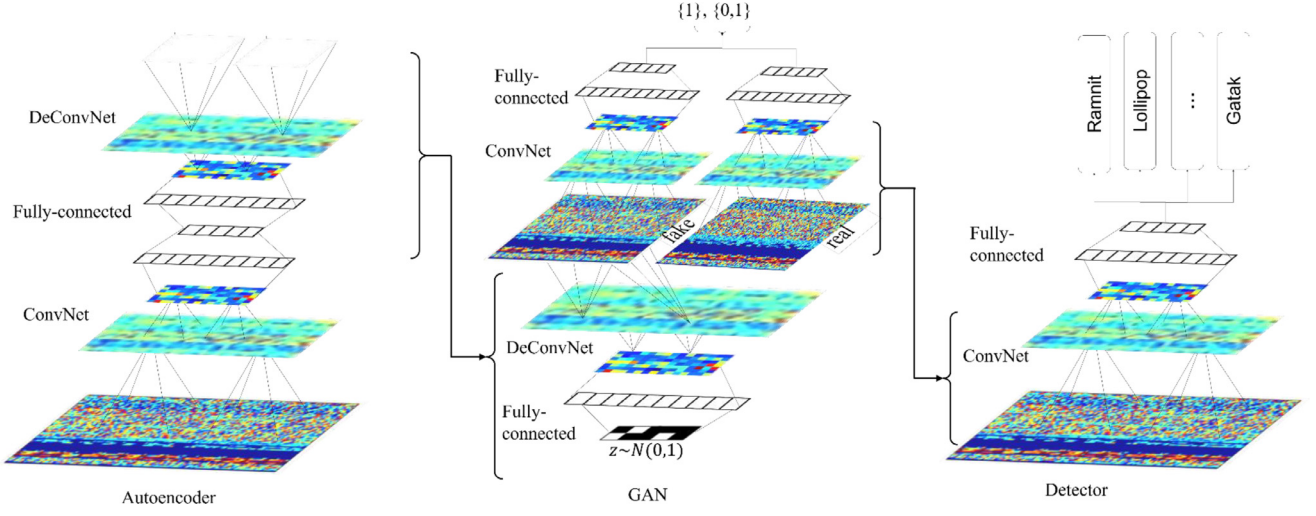


Fig. 6. The architectures of DAE, tDCGAN, and detector. The dropout layers follow the LeakyReLU layers, except for the encoder of the autoencoder.

term is 1, to minimize $V(D, G)$.

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{1}$$

According to the original GAN formulation, $V(D, G)$ converges when the distribution of the data generated by the generator becomes the same as the distribution of real data. However, the problem of the instability associated with convergence remains, and much research is underway to solve this problem [35,44]. Among GANs, deep convolutional GAN (DCGAN) is an approach in which original layers are changed to convolutional layers [26]. A DCGAN generates higher-quality images than images produced by the corresponding original GAN, and the trained system is relatively stable. Nevertheless, the learning process remains unstable, and many experiments are needed to determine optimal parameters. Therefore, in this paper, a DCGAN based on a DAE is proposed for addressing the learning process's instability. The DAE, which is widely used for image processing [26,39,28] to capture image features, is a module that is composed of data compression and reconstruction processes. The encoder of the DAE compresses the input x using Eq. (2), while the decoder uses Eq. (3) to reconstruct y that corresponds to the compressed data x. To transfer them to the GAN, they are stacked with convolutional networks (Eq. (4)), because the GAN is also stacked in a similar manner. We propose a method to transfer the reconstructing part of the DAE to the generator of the DCGAN, for more stable training. The validity of the DAE is described in Appendix C.

$$y = Enc(x) = conv_{enc}(x) \tag{2}$$

$$z = Dec(y) = conv_{dec}(y) \tag{3}$$

$$conv\left(\left(x_{ij}\right)_{\substack{i=1,\dots,n\\ j=1,\dots,n}}\right) = \left(c_{ij}\right)_{\substack{i=1,\dots,n\\ j=1,\dots,n}} = \sum_{a=0}^{m-1}\sum_{b=0}^{m-1} w_{ab}x_{(i+a)(j+b)} \tag{4}$$

Pre-train generator. The original GAN is disadvantageous in that the generated data are insensible because of the unstable learning process of the generator. In this section, we propose a method to solve the instability issue. The goal of the learning process of the generator is the same for Eq. (5) from Eq. (1) and it is equivalent to that of Eq. (6). However, the method is not efficient to pre-train the generator, because it depends on the parameters of the discriminator, as demonstrated in Appendix C. Therefore, we change Eq. (6) to (7), using only by the parameters of the generator. In this paper, to train the generator with Eq. (7), we use a DAE that is composed of an encoder and a decoder, and then transfer the decoder, which is trained by using $Dec(Enc(x)) \approx x$, into the generator of the GAN. In addition, we incorporate batch normalization [27] into the last layer of the encoder, to overcome the difference related to the fact that the generator generates fake data from a random variable z but the decoder generates it from $Enc(x)$.

$$\min_G log(1 - D(G(z))) \tag{5}$$

$$\Leftrightarrow D(G(z)) \approx 1 \tag{6}$$

$$\Leftrightarrow G(z) \approx x \in \chi, \text{ where } \chi \text{ is real dataset} \tag{7}$$

From the training point of view, the learning process of the GAN can be seen as indirect instruction. In comparison, training the GAN based on the DAE can be seen as direct instruction, and it is known that this training method is quick and effective [13].

Optimality of the Solution. From the game theory point of view, the GAN converges to the optimal point when the discriminator and the generator reach the Nash equilibrium. In this section, let p_G be the probability distribution of data created from the generator. We show that if $G(z) \approx x$, i.e., $p_{data} \approx p_G$, the GAN reaches the Nash equilibrium. We define $J(D, G) = \int_x p_{data}(x)D(x)dx + \int_z p_z(z)(1 - D(G(z)))dz$ and $K(D, G) = \int_x p_{data}(x)(1 - D(x))dx + \int_z D(G(z))dz$. We train the discriminator G and generator D to minimize $J(D, G)$ and $K(D, G)$ respectively. Then, we can define the Nash equilibrium of the tDCGAN as a state that satisfies Eq. (8) and (9). Fully trained generator and discriminator are denoted by G^* and D^*, respectively.

$$J(D^*, G^*) \le J(D^*, G) \forall G \tag{8}$$

$$K(D^*, G^*) \le K(D, G^*) \forall D \tag{9}$$

Theorem 1. *If $p_{data} \approx p_G$ almost everywhere, then the Nash equilibrium of the tDCGAN is reached.*

Before proving this theorem, we need to prove the following two lemmas.

Lemma 1. *$J(D^*, G)$ reaches a minimum when $p_{data}(x) \le p_G(x)$ for almost every x.*

Proof. The equality of the last inequality can happen when $p_{data}(x) \le p_{G^*}(x)$ for almost every x.

$$J(D^*, G) = \int_x p_{data}(x)D^*(x)dx + \int_z p_z(z)(1 - D^*(G(z)))dz \tag{10}$$

$$= \int_x p_{data}(x)D^*(x) + p_G(x)(1 - D^*(x))dx \tag{11}$$

$$= \int_x D^*(x)(p_{data}(x) - p_G(x))dx + \int_x p_G(x)dx \tag{12}$$

$$= \int_x 1_{p_{data}(x) > p_G(x)}(p_{data}(x) - p_G(x))dx + \int_x p_G(x)dx \tag{13}$$

$$\geq \int_x p_G(x)dx \tag{14}$$

■

From Eq. (10) to (12), because D outputs 0 or 1 according to the distribution of data, then as $1_{p_{data}(x) > p_G(x)}$. If $p_{data}(x) \leq p_G(x)$ for almost every x, the first term which must be nonnegative should be zero, resulting in Eq. (14).

Lemma 2. *$K(D, G^*)$ reaches a minimum when $p_{data}(x) \geq p_{G^*}(x)$ for almost every x.*

Proof. The equality of the last inequality can occur when $p_{data}(x) \geq p_{G^*}(x)$ for almost every x.

$$K(D^*, G) = \int_x p_{data}(x)(1 - D(x))dx + \int_z p_z(z)D(G^*(z))dz \tag{15}$$

$$= \int_x p_{data}(x)(1 - D(x))dx + \int_x p_{G^*}(x)D(x)dx \tag{16}$$

$$= \int_x D(x)(p_{G^*}(x) - p_{data}(x))dx + \int_x p_{data}(x)dx \tag{17}$$

$$= \int_x 1_{p_{data}(x) > p_{G^*}(x)}(p_{G^*}(x) - p_{data}(x))dx + \int_x p_{data}(x)dx \tag{18}$$

$$\geq \int_x 1_{p_{data}(x) > p_{G^*}(x)}(p_{G^*}(x) - p_{data}(x))dx \tag{19}$$

$$\geq -M \tag{20}$$

■

Eqs. (15)–(19) are rearranged similarly to Lemma 1. The second term of Eq. (19) should achieve a value as high as possible to minimize Eq. (19). If $p_{data}(x) \geq p_{G^*}(x)$ for almost every x, then $1_{p_{data}(x) > p_{G^*}(x)}$ in the second term of Eq. (19) should be 1 and $(p_{G^*}(x) - p_{data}(x))$ be positive, resulting in Eq. (20).

Proof of Theorem 1. We assume that $p_{data} \approx p_G$. From Lemma 1 to Lemma 2, if $p_{data} \approx p_G$, then $J(D, G)$ and $K(D, G)$ both reach minima. Therefore, the tDCGAN reaches the Nash equilibrium and converges to optimal points. ■

3.3. Malware detection

As discussed in Section 3.2, it is the malware detector of the MDS that finally detects malware. In the process of generating fake data and distinguishing it from real data, the convolution layers of the discriminator can sufficiently learn the characteristics of the real data. We transfer this ability to the malware detector of the MDS, imparting the malware detector with a good ability to observe the characteristics of malware data. This process is similar to the process of category GAN, which performs category classification using a GAN [38]. The learning process of the MDS is described in Algorithm 1.

4. Experiments

4.1. Dataset

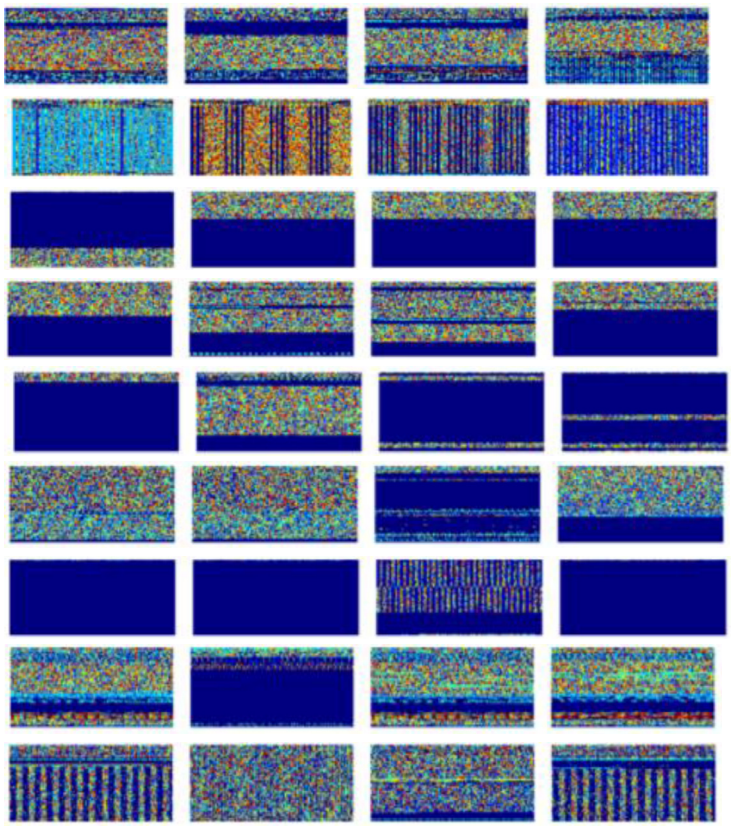
To validate the performance of fake data generation and the detection performance of the tDCGAN, we used the malware dataset from the Kaggle Microsoft Malware Classification Challenge.[4] The malware data were written in assembly and binary codes, and we used the form of the binary code. The malware data codes were converted into deformed-size malware

[4] Kaggle: Microsoft Malware Classification Challenge (BIG 2015). https://www.kaggle.com/c/malware-classification.

Algorithm 1 Learning algorithm for MDS.

Input: Preprocessed data $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ (:data,:label), encoder of DAE A_E, decoder of DAE A_D, generator of tDCGAN G_G, discriminator of tDCGAN G_D
Output: Malware detector D
for $epochs_{DAE}$ **do**
  Train A_E, A_D with $T_A = \{x_1, \ldots, x_n\}$ using Eqs. (2)–(4) and (7);
end for
Transfer A_D to G_G;
for $epochs_{tDCGAN}$ **do**
  Train G_G with $T_{G_G} = \{(G(z)_1, 1), \ldots, (G(z)_n, 1)\}$, where $z \sim N(0, 1)$;
  Train G_D with $T_{G_D} = \{(x_1, 1), \ldots, (x_n, 1), (G(z)_1, 0), \ldots, (G(z)_n, 0)\}$
  using Eq. (1);
end for
Transfer G_D to D;
Train D with ;
Return D;

Table 2
Summary of malware data and sample images of preprocessed malware code. Same types of malware yield similar images.

Type	The # of data (The # of test data)	Sample images of each type
Ramnit	1539(145)	
Lollipop	2459(234)	
Kelihos_ver3	2942(296)	
Vundo	451(44)	
Simda	42(5)	
Tracur	744(75)	
Kelihos_ver1	391(43)	
Obfuscator.ACY	1221(132)	
Gatak	1011(106)	

images when the malware code was compiled, for use with the tDCGAN. Then, because malware images were too large, the images were reduced to 0.1 times their original sizes.

If k is the length of the binary code, C is the size of the transformed column, and R is the size of the transformed row, then the method for calculating the transformed column and row size is given by Eq. (21) and (22).

$$C = 2^{\frac{\log\sqrt{16k}}{\log 2}+1} \tag{21}$$

$$R = \frac{16k}{C} \tag{22}$$

This transformation yielded standardized data, with values ranging from 0 to 1. Table 2 lists the used malware data types, the number of malware data for each malware type, and sample malware images for each malware. In the malware images, the red color corresponds to 1, while the blue color corresponds to 0.

The number of malware data was 10,800, and the original set was split into a training set (9720 instances) and a test set (1080 instances). For each malware type, the number of instances in the test set is shown in parentheses in Table 2. The final malware images were 63×135.

4.2. Experimental details

The operating system of the computer used in our experiments was Ubuntu 16.04.2 LTS and the central processing unit of the computer was an Intel Xeon E5-2630V3. The random access memory of the computer was Samsung DDR4 16 GB $\times$ 4, and the graphic processing unit of the computer was GTX Titan X D5 12 GB $\times$ 4.

When constructing the tDCGAN, we used the Keras library. LeakyReLUs were used between the layers of the tDCGAN, and a batch normalization layer was added after MaxPooling of the DAE encoder. We added LeakyReLUs and dropouts with the rate of 0.25 to the DAE decoder and to the tDCGAN layers. DeConv was the deconvolutional layer [43] that was used to rescale the images to match them to the actual size. We used AveragePooling instead of MaxPooling to enable differentiation when training the tDCGAN. An example of this is global average pooling, which has been utilized in state of the art image classification models [3].

- Deep AutoEncoder
- Encoder: 3×3Conv@4 – 3×3MaxPooling – 3×3Conv2D@8 – Fully connected layer of size 100
- Decoder: Fully connected layer of size 13,440 – 3×3DeConv@128 – 3×3Conv@64 – 3×3DeConv@64 – 3×3Conv@32 – 3×3Conv@1
- GAN
- Generator: Same as the DAE decoder
- Discriminator: 3×3Conv@4 – 3×3AveragePooling – 3×3Conv@8 – 3×3AveragePooling – Fully connected layer of size 128–2
- Malware Detection System
- Detector: 3×3Conv@4 – 3×3AveragePooling – 3×3Conv@8 – 3×3AveragePooling – Fully connected layer of size 128–9

The batch size was set to 64 and increased to 512 in the tDCGAN training. The optimizer used for training was the Adam optimizer, with the learning rate of 10^{-4} and $\beta_1 = 0.5$ [24]. However, for stable learning, the optimizers of the generator and discriminator of the tDCGAN were the Adam optimizers with the learning rates of 10^{-5} and 10^{-4}. To avoid memorizing real images, the DAE was trained 100 times, the tDCGAN and GAN were trained 2000 times, and the tGAN and GAN were trained 1000 times. To train a malware detector of the MDS, we used the Adam optimizer with the default value in the Keras package, and iteratively trained it 100 times. All weight parameters were initialized with the Glorot initializer [18].

4.3. Malware detection

Accuracy. The malware detector was transferred from the discriminator of the tDCGAN. All layers except the fully-connected layer of the discriminator were transferred, and the fully-connected layer of this system was trained anew. In this section, the performance of the malware detector is compared with other machine learning algorithms, to validate the malware detector of the MDS using tDCGAN. We compared the proposed method with the k nearest neighbors (k-NN), naïve Bayes, random forest, decision trees, AdaBoost, support vector machine (SVM) with a polynomial kernel, SVM with a radial basis function (RBF) kernel, linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA) methods, which are provided in the scikit-learn library. For all of these algorithms we used default values, except for $k = 3$ in the k NN method, the penalty parameter $C = 0.025$ in the polynomial kernel SVM, kernel coefficient $\gamma = 2$ in the RBF SVM, maximal depth $= 5$ in the decision tree and random forest methods, and maximal number of features $= 1$ in the random forest method.

We compared the proposed method not only to those machine learning algorithms provided in the scikit-learn library, but also to a multi-layer perceptron (MLP) (which has the same structure as the discriminator of GAN), a convolutional neural network (CNN) (which has the same structure as the discriminator of DCGAN), a GAN (which does not use convolutional layers), a tGAN (which is a GAN based on a DAE), a DCGAN (which uses convolutional layers), and the proposed tDCGAN (which is a DCGAN based on a DAE). Ten-fold cross-validation was used for all algorithms in the malware detection experiment. The results of these experiments are summarized in Fig. 7. The averaged correct classification achieved by the proposed model was 95.74%. Compared with the conventional models, the proposed model performs much better. The numerical results are shown in Table 3. We can observe that there is statistical significance in the difference between MLP and tGAN but not between GAN and tGAN. However, the results of the t-test with CNN, DCGAN, and tDCGAN show that tDCGAN is statistically better than others. We conducted the experiments 10 times for each model to get statistical significance. Fig. 8 shows the performance according to the number of iterations for the top-six models.

Some studies of malware detection used the same dataset that we used here [32,16,23]. However, in the present work, we detected no significant difference in the malware detection ability after reducing the scale of the malware data, and we modeled and tested zero-day attacks by investigating malware detection problems.

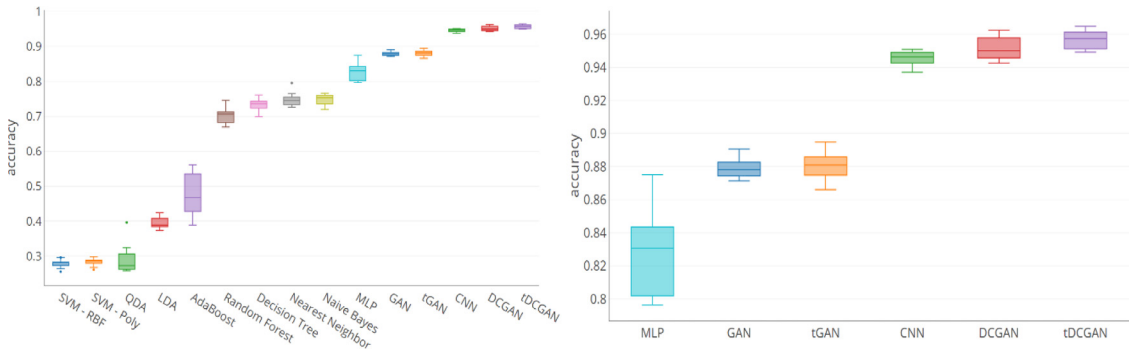


Fig. 7. All results (left) and top six results (right), for ten-fold cross-validation of malware detection.

Table 3
Numerical result for the top 6 models. We show the median value of accuracy in the second row. We conduct a *t*-test for statistical significance of difference of accuracy. The *p*-values of MLP and GAN are results of the *t*-tests with tGAN and those of CNN and DCGAN are results of the *t*-tests with tDCGAN.

Model	MLP	GAN	tGAN	CNN	DCGAN	tDCGAN
Accuracy (%)	83.06	87.81	88.10	94.63	95.01	95.74
Std. dev.	7.54e-04	3.44e-05	8.05e-05	2.12e-05	4.60e-05	3.03e-05
p-value	0.0002	0.3473	–	4.85e-05	0.0414	–

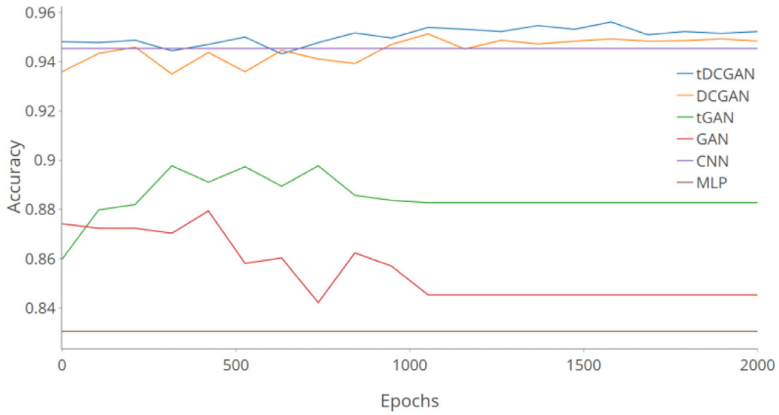


Fig. 8. Top six accuracies for each model according to the training epochs.

Table 4
Precision, recall, and F1 scores, for different malware types.

	R	L	K3	V	S	T	K1	O	G
Precision	0.950	0.983	0.997	0.840	1.000	0.854	1.000	0.944	0.928
Recall	0.910	0.966	0.997	0.955	0.600	0.933	1.000	0.902	0.972
F1-score	0.930	0.974	0.997	0.894	0.750	0.898	1.000	0.923	0.949

Analysis. The confusion matrix for malware detection is shown in Fig. 9 to validate the performance of the proposed malware detector. The matrix confirms that the MDS cannot distinguish well (Ramnit, Obfuscator.acy), (Lollipop, Obfuscator.acy) and (Ramnit, Tracur). The malware that confuses malware detector has similar pattern throughout the sample images in Table 2. As different malware types appear with different frequencies, Table 4 lists the calculated precision, recall, and F1 scores, for each malware type.

We extract the intermediate values based on the F1 scores to evaluate the algorithm of the malware detector. Because the amount of Simda malware data is small, we used two sample data that are Kelihos ver1 malware data and are detected correctly, and two sample data that are Tracur malware data but are categorized into Tracur and Ramnit. The first columns illustrated in Fig. 10 are malware data images, the second and third columns are generated by the first convolutional and LeakyReLU layers, and the fourth and fifth columns are generated by the second convolutional and LeakyReLU layers.

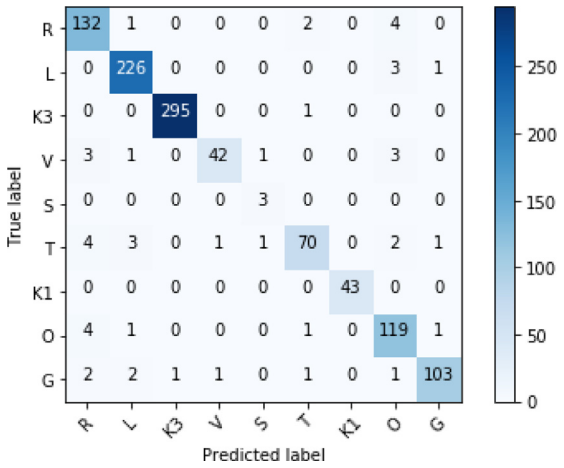


Fig. 9. Confusion matrix for malware detection.

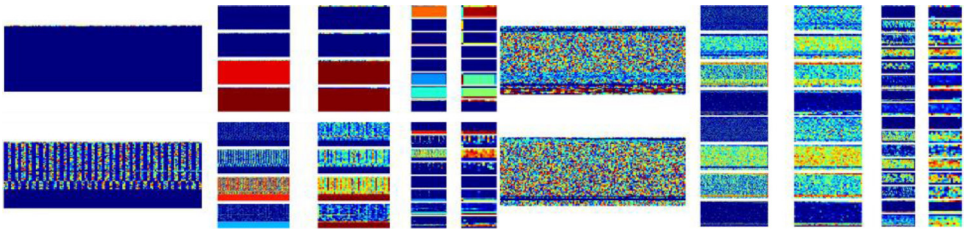


Fig. 10. Intermediate outputs of the malware detection system. Left: the two malware images are Kelihos ver1 malware images. Right: the two malware images are Tracur malware images, but the last image is detected as Ramnit malware data.

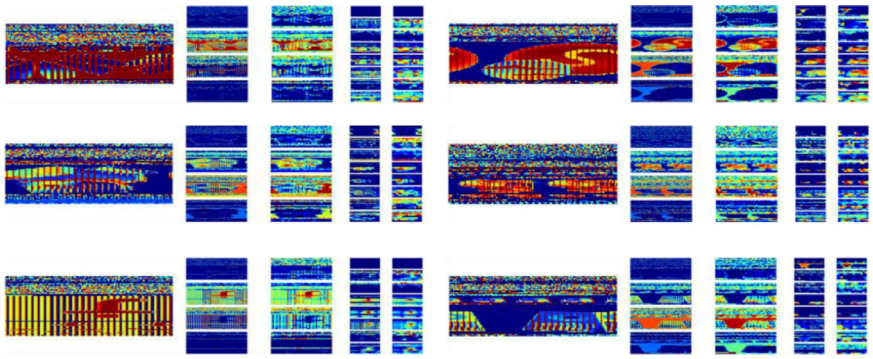


Fig. 11. Intermediate outputs for special case of malware images.

From the intermediate images for each malware image, convolutional layers recognized empty space, vertical and horizontal patterns in the images, empty space between lines, and reversed the pixel values. Fig. 11 illustrates the particular case of malware images embedded with the malware developer's signature. By analyzing the intermediate output of these data, we see that the malware detector is well aware of the malware developer's signature.

The results of these experiments indicate that the proposed malware detector can extract and recognize the characteristics of malware images. Fig. 12 reveals clustering patterns in the malware data by applying the t-SNE algorithm [29], which groups data points according to their similarity. It can be seen that the malware data are sufficiently modified for the clustering pattern to become apparent to the malware detector.

4.4. Zero-day attacks

In Section 1, we assumed that zero-day attacks can be modeled by introducing noise into existing malware data. To model zero-day attacks, a systematic method of noise generation is required. The sample images in Table 2 have similar structures for malware of the same type; thus, we generated noise using the SSIM method, which utilizes the structural similarity of images. The method for calculating the SSIM value for a pair of images x, y involves computing μ_x, μ_y as the

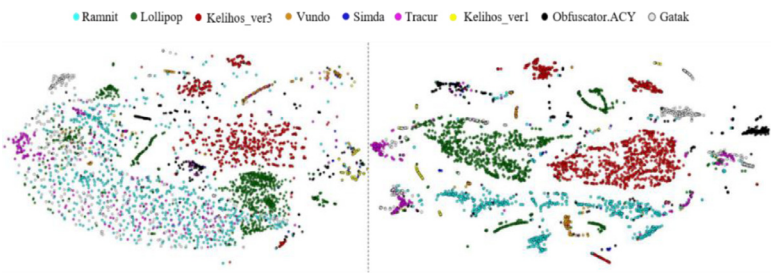


Fig. 12. Distributions of the raw data of malware (left) and output of the malware detector (right). The detector changes the values so that similar data points are clustered together.

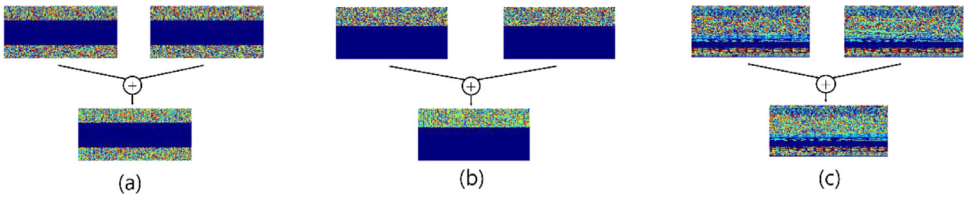


Fig. 13. Examples of modeled zero-day malware whose SSIM values are (a) 0.7, (b) 0.64 and (c) 0.61 They have similar patterns but different details.

Table 5
Experimental results of zero-day attack detection. The values are expressed as a percentage. Our model has the best overall performance compared to other conventional models.

Model\SSIM	0.6	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69
RBF SVM	66.74	70.73	73.42	76.18	79.45	80.06	80.11	80.17	77.39	77.52
Poly SVM	66.51	70.73	73.42	75.92	79.45	80.06	79.83	80.17	77.39	77.18
QDA	59.63	63.17	64.81	68.06	69.86	70.91	70.99	71.23	74.84	75.50
LDA	78.21	81.46	83.04	85.34	87.40	88.09	88.67	88.27	89.17	90.27
AdaBoost	77.29	81.95	84.30	87.43	90.96	91.41	91.16	91.90	90.45	91.28
Random Forest	91.28	94.39	95.19	92.15	92.88	92.80	95.58	96.37	91.40	93.29
Decision Tree	95.64	96.59	96.46	96.07	96.71	96.68	96.41	96.93	96.18	96.64
Nearest Neighbors	97.71	97.80	97.72	**98.16**	98.36	98.34	98.34	98.32	98.09	**98.99**
Naive Bayes	90.60	91.22	90.89	90.84	91.51	91.41	91.16	91.62	90.45	90.94
MLP	96.78	96.59	96.46	96.85	97.26	97.23	97.23	97.21	96.82	97.31
GAN	96.32	97.07	96.96	96.85	96.99	96.95	96.95	96.93	96.50	96.97
tGAN	97.24	97.07	96.96	97.38	97.81	97.78	97.78	97.77	97.45	97.98
CNN	98.16	98.29	98.23	**98.16**	**98.63**	**98.61**	**98.61**	**98.60**	**98.41**	**98.99**
DCGAN	98.16	98.29	98.23	**98.16**	98.36	98.34	98.34	98.32	98.09	98.65
tDCGAN	**98.39**	**98.78**	**98.73**	**98.16**	**98.63**	**98.61**	**98.61**	**98.60**	**98.41**	**98.99**

means over the pixels of images x, y, computing σ_x^2, σ_y^2 as the variances over the pixels of images x, y, and σ_{xy} as the covariance over the pixels of images x, y.

$$\mathrm{SSIM}(x, y) = (2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)/\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right) \tag{23}$$

where, $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$, $L = 2^{\#\ bits\ per\ pixel} - 1$, $k_1 = 0.01$, $k_2 = 0.03$

We confirm that the existing malware image is structurally similar to the other and has some noise when we combine the malware images with an 8:2 ratio and the SSIM value falls between 0.6 and 0.7.[5] The modeled zero-day malware $N_k(x, y)$ is then calculated as follows.

$$N_k(x, y) = (1 - \xi)x + \xi y, \ \ where\ ssim(x, y) > k, \ \ \xi = 0.2 \tag{24}$$

The accuracy of the malware detector as a function of the SSIM value tends to decrease with decreasing SSIM values. We show that the performance of the malware detector that is transferred from the GAN based on the DAE is good, even for zero-day attacks, which were simulated in this paper. Fig. 13 shows examples of modeled zero-day malware. As in Table 5, the detector based on tDCGAN has better performance than other conventional models.

[5] When the SSIM values is set to 0.6 or less, the number of data required is reduced to less than half. A blurred image results for a SSIM value of about 0.7 [41].

5. Conclusions

In this paper, we raised the problems presented by malware and attempted to solve them. As zero-day malware is created with modifications, we use GAN to generate virtual data and train to make the detector robust to data deformation. To solve the training instability problem of GANs and detect the modified malware (zero-day attacks), we stabilized the training process by applying the DAE to the GAN, and used the discriminator's ability to extract meaningful features for malware detection. The proposed tDCGAN exhibited better stability than other GANs, and performed the best on malware detection. We did not simply demonstrate the performance of the GAN and MDS, but also performed a thorough analysis. We showed that the proposed model can recognize modified features based on a SSIM value comparison experiment. The malware detector transferred from the tDCGAN achieved an average classification accuracy of 95.74%, outperforming other machine learning algorithms.

In the future, we will expand the types of malware with a range of malware datasets, and address the issue of different lengths of various malware, because the malware code was converted into malware images through crop and pad operations in this study. Moreover, we need to implement full control of the type of generated malware because the proposed model simply generates malware from a random distribution. We plan to develop a model to generate data from specific distributions, so that we can create relatively more data for different types of malware. The proposed model can also be applied to one-shot learning which involves only a little data. The proposed method can be generically applied to any problems that must be solved with sparse data.

Acknowledgment

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract. (UD160066BD)

Appendix A. Performance of the Generator

Generated images. The malware images that were created by the tDCGAN generator are analyzed in this section. Fig. 14 shows the compressed and reconstructed malware images obtained by training the DAE. By analyzing the compressed and reconstructed images of the actual malware images, it is clear that the DAE cannot reconstruct the details of the malware images but does reconstruct empty spaces, vertical lines and filled spaces.

We also generated malware images using the DAE, from random values ($z \sim N(0, 1)$). These images are shown in Fig. 15. They are quite similar to real malware images from the pattern viewpoint. Because the difference in these images from real malware images can be considered as weak noise, the tDCGAN can learn as noise-robustness discriminator. As a result, we can transfer this ability of the discriminator to the malware detector. In addition, the decoder of the DAE generates malware images from a random value z; thus, the decoder of the DAE can improve the performance of the tDCGAN generator by transferring the ability of the decoder to it.

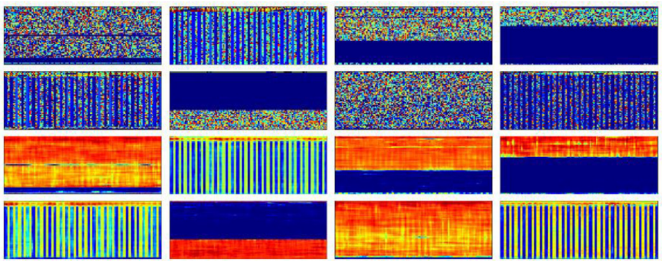


Fig. 14. Inputs (top eight) and outputs (bottom eight) of the autoencoder.

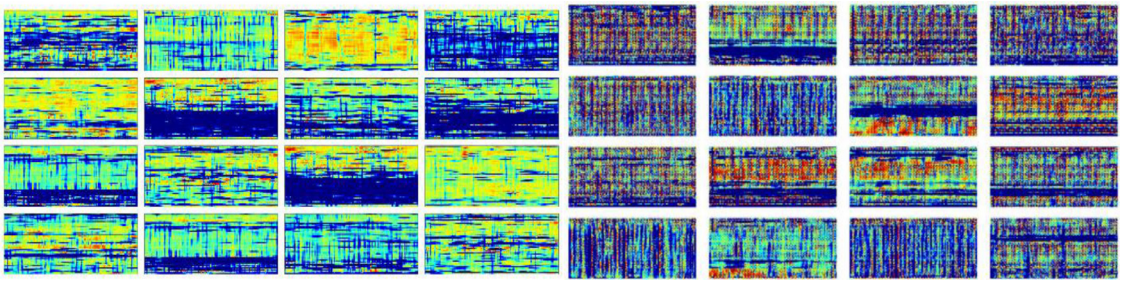


Fig. 15. The outputs of the autoencoder (left) and the tDCGAN (right), for random inputs $z \sim N(0, 1)$.

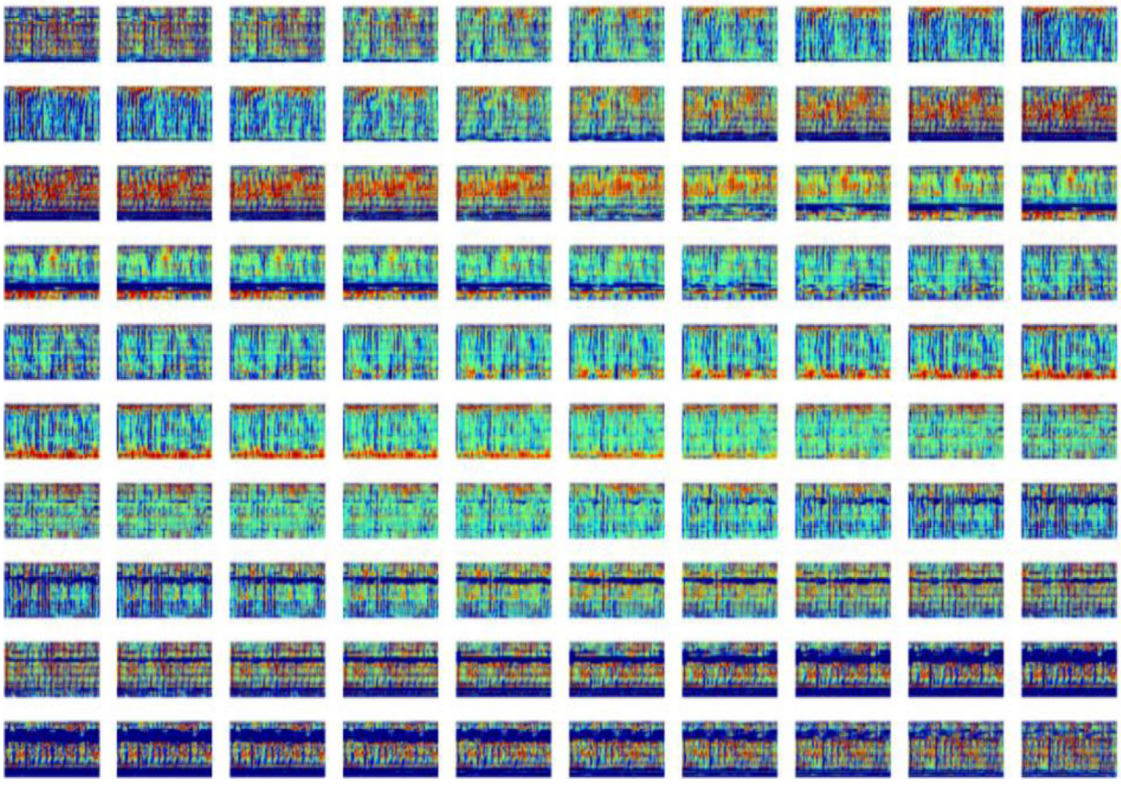


Fig. 16. Results of noise-walking. There are continuous changes in vertical and horizontal lines.

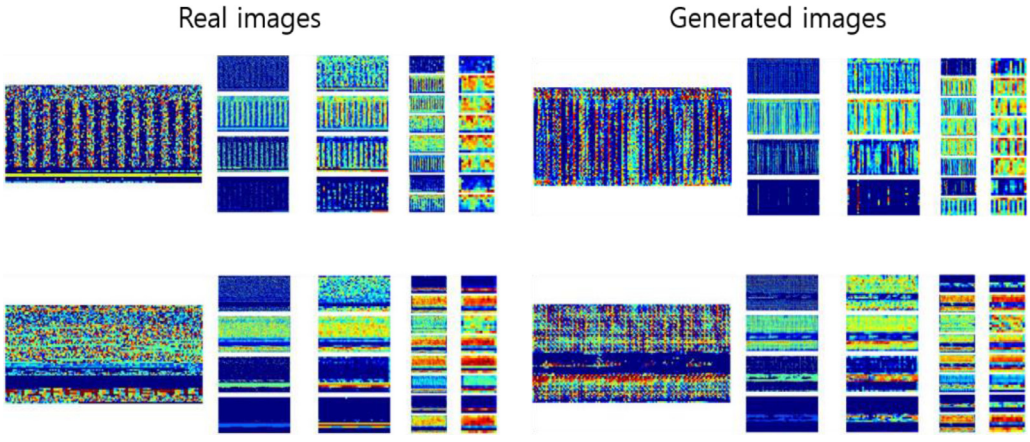


Fig. 17. Intermediate outputs of the discriminator.

The images created from the generator, which was transferred from the DAE decoder and was trained well in the GAN training process, are shown in Fig. 15. Unlike the images generated by the DAE decoder, the images in Fig. 15 are more similar to real malware images. In addition to the malware image features such as vertical lines, the inner details of lines are captured, and the data values are distributed similarly to real malware images.

Analysis of Generator. On the surface, it appears that the tDCGAN generator memorizes malware images rather than analyzes them. To demonstrate that this is not true, we used a technique, called noise-walking, to generate fake malware images with continuous inputs from a known random source. A total of ten inputs generated by a known random source were combined with adjacent input, as shown in Eq. (25), to generate malware images with combined input. The parameter ξ was varied from 0 to 1 in 10 steps.

$$Combined\ value = (1-\xi)N_k + \xi N_{k+1},\ \ where\ k = 1,\ 2, \ldots, 9,\ 10 \tag{25}$$

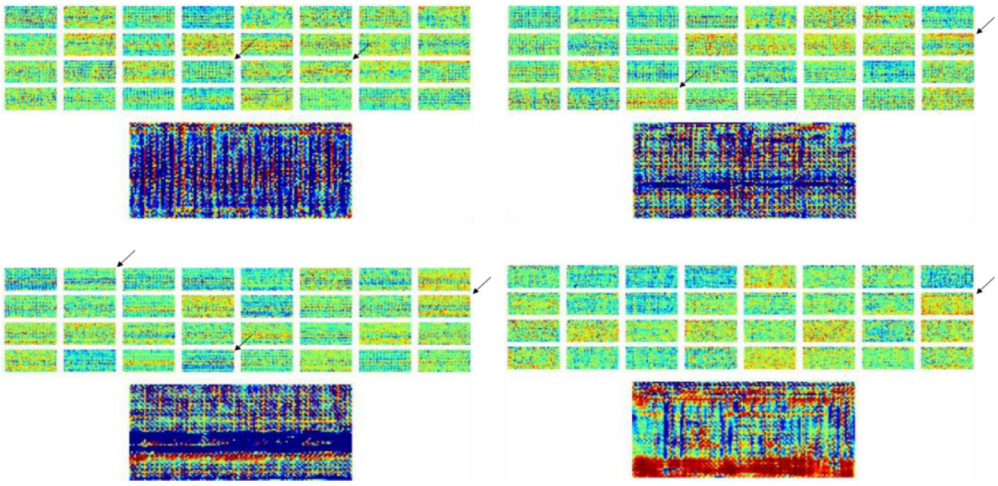


Fig. 18. Intermediate outputs of the generator.

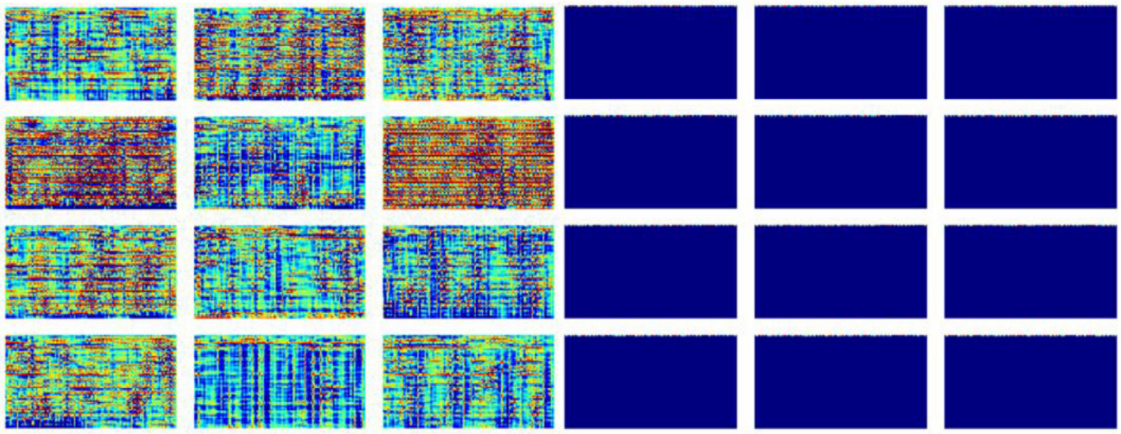


Fig. 19. Generated images by DCGAN that reach different optimal points. Left: twelve malware images are generated by the discriminator, which converge to the optimal point that generates a complex malware image. Right: twelve malware images are generated by the discriminator, which converge to the optimal point that generates a simple malware image.

If the generator does not understand malware images but rather memorizes them, the images generated by noise-walking will become discrete lists. However, the generated malware images by noise-walking in Fig. 16 change continuously.

To analyze the performance of the tDCGAN, we extracted intermediate images from the generator and the discriminator. The intermediate output of the discriminator depicted in Fig. 17 shows the characteristics of the intermediate output of the malware detector of the MDS. It indicates that transfer learning from the discriminator to the malware detector is satisfactory. The bottom two filters in the second and third columns act to reverse the characteristics of malware images, and the top two filters in the second and third columns act to emphasize the features. However, although it is possible to determine the characteristics of images, but it is difficult to determine the ability to discriminate between the actual image and the virtual image, through the middle output of the convolutional layers by a naked eye.

The intermediate output of the generator is extracted from the last two convolutional layers among the three convolutional layers, owing to the quality of images. Fig. 18 shows four fake malware images and the convolutional layer output just before creating fake data. Thirty-two intermediate prints are combined to create one fake malware image. Arrows indicate where 32 of the intermediate figures have features of a fake malware image.

Image Complexity. An interesting result is obtained from the experiment in which tDCGAN and DCGAN are compared. We observe that even if the experiments are performed in the same environment, tDCGAN generates more complex malware images directly, while DCGAN often generates only simple malware images. Because GAN converges to the saddle point owing to the characteristic of the objective function (which is of the min-max type), DCGAN will often converge to the optimal point which generates relatively simple data because it is trained without knowing the distribution of all of the malware images. There can be several saddle points for a given optimization problem [11]. A problem will arise if the discriminator cannot learn complex features of malware images and will adversely affect the performance of the malware

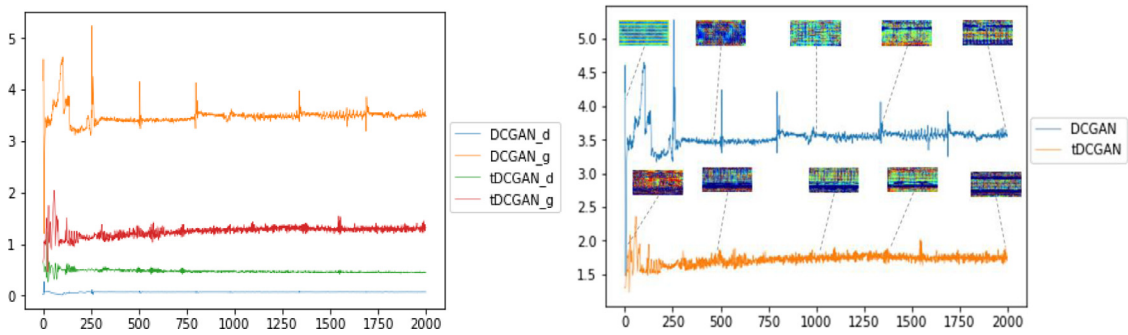


Fig. 20. Loss of the generator and the discriminator of tDCGAN and DCGAN, by epochs (left). The sum of the loss of the generator and the discriminator of tDCGAN and DCGAN, by epochs (right).

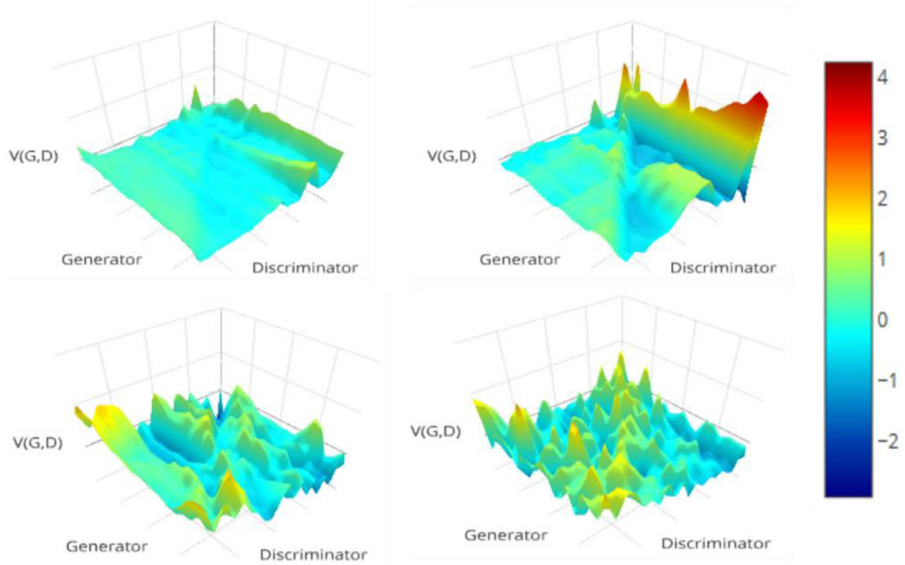


Fig. 21. The degree of change of the loss sum of discriminator and generator by epochs in 3D showing the loss sum of tDCGAN, DCGAN, GAN, tGAN from left-top in the clockwise order.

detector of the MDS which is transferred from the discriminator later. Therefore, experimental results show that pre-training with the DAE improves not only the generation ability of GAN but also the malware detection ability of MDS. Fig. 19 shows the images generated by DCGAN, which converge to different optimal points.

Appendix B. Transferred performance of DAE visualization

To construct the MDS, a generative model is required, i.e., the GAN. However, a problem with GANs is training instability. Therefore, a DCGAN based on a DAE is proposed in this paper to stabilize the GAN training process. In this section, we show that a GAN based on a DAE exhibits more stable training than a GAN without DAE. Fig. 20, which shows the loss for tDCGAN and DCGAN vs. the iteration number, shows that the DCGAN stability loss occurs later than for the tDCGAN generator. The graph in Fig. 20, showing the sum of the loss of the discriminator and the generator, indicates that tDCGAN is stabilized with a lower loss than DCGAN.

To show that a GAN based on a DAE is more stable than a GAN without DAE, the loss sum of the generator and the discriminator combination by epochs is shown in Fig. 21. The red color in Fig. 21 corresponds to a high loss sum, while the blue color corresponds to a low loss sum. The GAN based on the DAE exhibits a stabilized surface without a large change of loss, but the GAN without DAE exhibits a severe change in loss. When either the discriminator or the generator is fixed, loss variation can also reach the optimal point more quickly if the DAE is used to pre-train the generator.

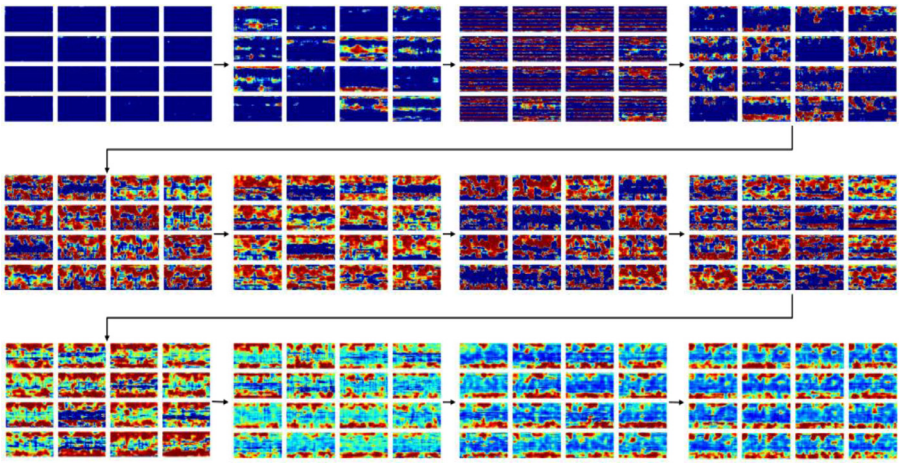


Fig. 22. Difference between images, generated according to the training epochs of the discriminator.

Appendix C. The need for DAE

We do not use Eq. (3) and instead use Eq. (4) for pre-training the GAN generator, because the former contains the parameters of the discriminator. In this section, we show that pre-training the generator using Eq. (3) yields poor performance. An experiment was conducted in which only the generator was trained using a trained discriminator by the number of iterations. As a result, more complicated and complete malware images were created by the generator with progressing training of the discriminator. However, this shows that pre-training the generator is effective only when the discriminator is fully trained. Thus, the generator can be pre-trained when the discriminator is fully trained, however, this is the same as training the GAN twice, which is ineffective pre-training. Therefore, pre-training the generator with Eq. (4) rather than Eq. (3) is valid and does not require a fully-trained discrimimator. Fig. 22 shows the images generated by the generator that was trained with the discriminator, by the number of iterations.

Reference

[1] M. Ahmadi, D. Ulyanov, S. Semenov, M. Tromov, G. Giacinto, Novel feature extraction, selection and fusion for effective malware family classification, in: Conf. Data and Application Security and Privacy, 2016, pp. 183–194.
[2] P. Akritidis, K. Anagnostakis, E.P. Markatos, Efficient content-based detection of zero-day worms, IEEE Int. Conf. Commun. 2 (2005) 837–843.
[3] M. Alexander, O. Christopher, T. Mike, Inceptionism: going deeper into neural networks, 2015. . http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into-neural.html, Accessed 17 June 2015.
[4] C. Annachhatre, A.H. Thomas, S. Mark, Hidden Markov models for malware classification, J. Hacking Tech. 11 (2015) 59–73.
[5] A. Arnold, R. Nallapati, W. Cohen, A comparative study of methods for transductive transfer learning, IEEE Int. Conf. Data Mining (2007) 77–82.
[6] K. Berlin, S. David, S. Joshua, Malicious behavior detection using windows audit logs, Artif. Intell. Secur. (2015) 35–44.
[7] J. Cao, Q. Fu, Q. Li, D. Guo, Discovering hidden suspicious accounts in online social networks, Inf. Sci. 394 (2017) 123–140.
[8] Z. Chen, Q. Yan, H. Han, S. Wang, L. Peng, L. Wang, B. Yang, Machine Learning based mobile malware detection using highly imbalanced network traffic, Inf. Sci. 433-434 (2018) 346–364.
[9] M. Christodorescum, S. Jha, S.A. Seshia, D. Song, R.E. Bryant, Semantics-aware malware detection, Secur. Privacy (2005) 32–46.
[10] A. Damodaran, F. Di Troia, C.A. Visaggio, T.H. Austin, M. Stamp, A Comparison of static, dynamic, and hybrid analysis for malware detection, J. Comput. Virol. Hacking Tech. 13 (1) (2017) 1–12.
[11] Y.N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, Adv. Neural Inf. Process. Syst. (2014) 2933–2941.
[12] S. David, H. Numaan, ATM malware on the rise, 2017 Accessed 08 Aug. https://blog.trendmicro.com/trendlabs-security-intelligence/atm-malware-on-the-rise. Seshia.
[13] D. Dean Jr., D. Kuhn, Direct instruction vs. discovery: the long view, Sci. Edu. 91 (2007) 384–397.
[14] L.E. Denton, S. Chintala, R. Fergus, Deep generative image models using a laplacian pyramid of adversarial networks, Adv. Neural Inf. Process. Syst. (2015) 1486–1494.
[15] A. Dhammi, M. Singh, Behavior analysis of malware using machine learning, IEEE Int. Conf. Contemp. Comput. (2015) 481–486.
[16] J. Drew, T. Moore, M. Hahsler, Polymorphic malware detection using sequence classification methods, Secur. Privacy Workshops (2016) 81–87.
[17] F.C.C. Garcia, II. Muga and P. Felix, Random forest for malware classification, arXiv preprint arXiv:1609.07770, 2016.
[18] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, Int. Conf. Artif. Intell. Stat. (2010) 249–256.
[19] I. Goodfellow, J. Pouget-Abadie, M. Mirze, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, Adv. Neural Inf. Process. Syst. (2014) 2672–2680.
[20] M. Grace, Y. Zhou, Q. Zhang, S. Zou, X. Jiang, Riskranker: scalable and accurate zero-day android malware detection, in: Proceeding International Conference on Mobile Systems, Applications, and Services, 2012, pp. 281–294.
[21] S. Huda, S. Miah, M.M. Hassan, R. Islam, J. Yearwood, M. Alrubaian, A. Almogren, Defending unknown attacks on cyber-physical systems by semi-supervised approach and available unlabeled data, Inf. Sci. 379 (2017) 211–228.
[22] D.J. Kim, C.D.H. Jiang and R. Memisevic, Generating images with recurrent adversarial networks, arXiv preprint arXiv:1602.05110, 2016.
[23] J.-Y. Kim, S.-J. Bu, S.-B. Cho, Malware detection using deep transferred generative adversarial networks, in: ICONIP, In Int. Conf. on Neural Information Processing, 2017, pp. 556–564.
[24] D. Kingma and J. Ba, Adam: a method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.

[25] D. Kong, Y. Guanhua, Discriminant malware distance learning on structural information for automated malware classification, in: Conf. Knowledge discovery and Datamining, 2013, pp. 1357–1365.
[26] A. Krizhevsky, G.E. Hinton, Using very deep autoencoders for content-based image retrieval, in: European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2011, pp. 489–494.
[27] S. Loffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Int. Conf. Machine Learning, 2015, pp. 448–456.
[28] X. Lu, Y. Matsuda, C. Hori, Speech enhancement based on deep denoising autoencoder, Interspeech (2013) 436–440.
[29] L. Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (2008) 2579–2605.
[30] M.Z. Mas' ud, S. Sahib, M.F. Abdollah, S.R. Selamat, R. Yusof, Analysis of feature selection and machine learning classifier in android malware detection, in: Information Science and Applications, 2014 Int. Conf. on IEEE, 2014, pp. 1–5.
[31] M. Mathieu, C. Couprie and Y. Lecun, Deep multi-scale video prediction beyond mean square error, arXiv preprint arXiv:1511.05440, 2015.
[32] B.N. Narayanan, O. Djaneye-Boundjou, T.M. Kebede, Performance analysis of machine learning and pattern recognition algorithms for malware classification, in: Aerospace and Electronics Conf. and Ohio Innovation Summit, 2016, pp. 338–342.
[33] L. Nataraj, S. Karthikeyanm, G. Jacob, B.S. Manjunath, Malware images: visualization and automatic classification, in: Conf. Visualizing for Cyber Security, 2011, pp. 1–7.
[34] R. Pascanu, J.W. Stokes, H. Sanossian, M. Marinescu, A. Thomas, Malware classification with recurrent network, Acoust. Speech Signal Process. (2015) 1916–1920.
[35] A. Radford, L. Metz and S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv preprint arXiv:1511.06434, 2015.
[36] I. Santos, F. Brezo, X. Ugarte-Pedrero, P.G. Bringas, Opcode sequences as representation of executables for data-mining-based unknown malware detection, Inf. Sci. 231 (2013) 64–82.
[37] K. Singh, S.C. guntuku, A. Thakur, C. Hota, Big data analytics frameword for peer-to-peer botnet detection using random forests, Inf. Sci. 278 (2017) 488–497.
[38] J.T. Springenberg, Unsupervised and semi-supervised learning with categorical generative adversarial networks, arXiv preprint arXiv:1511.06390, 2015.
[39] P. Vincent, H. Larochelle, Y. Bengio, P.A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Int. Conf. on Machine Learning, 2008, pp. 1096–1103.
[40] Q. Wang, W. Guo, K. Zhang, G. Alexander, II. Ororbia, X. Xinyu, C. Lee Giles, L. Xue, Adversary resistant deep neural networks with an application to malware detection, in: Int. Conf. Knowledge Discovery and Data Mining, 2017, pp. 1145–1153.
[41] Z. Wang, A.C. Bovil, H.R. Sheikh, E.P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process. vol. 13 (2004) 600–612.
[42] Y. Ye, L. Chen, S. Hou, W. Hardy, X. Li, DeepAM: a heterogeneous deep learning framework for intelligent malware detection, Knowl. Inf. Syst. (2017) 1–21.
[43] J. Zhao, M. Mathieu, and Y. Lecun, Energy-based generative adversarial network, arXiv preprint arXiv:1609.03126, 2016.
[44] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: European Conf. on Computer Vision, 2014, pp. 818–833.