Detecting Intrusive Malware with a Hybrid Generative Deep Learning Model

Jin-Young Kim and Sung-Bae Cho(✉)

Department of Computer Science, Yonsei University, Seoul, South Korea
{seago0828, sbcho}@yonsei.ac.kr

Abstract. A small amount of unknown malware can be analyzed manually, but it is generated with extremely more and more so that automatic detection of them is needed. Malware is usually generated with different features from those of existing ones (e.g., code exchange, null value insertion, or reorganization of subroutines) to avoid detection of antivirus systems. To detect malware with obfuscation, this paper proposes a method called latent semantic controlling generative adversarial networks (LSC-GAN) that learns to generate malware data with *i*-feature from a specific Gaussian distribution which represents *i*-feature and distinguish it from the real. Variational autoencoder (VAE) projects data to latent space for feature extraction and is transferred to generator (G) of LSC-GAN to train it stably. G generates data from Gaussian distribution, so it produces similar data but not identical to the actual data: it includes modified features compared with the real. The detector is inherited with transfer learning in a encoder that learns various malware features using real and modified data generated by the LSC-GAN based on a LSC-VAE. We show that LSC-GAN achieves detection accuracy of 96.97% on average that is higher than those of other conventional models. We demonstrate statistical significance of the performance of the proposed model using t-test. The result of detection is analyzed with confusion matrix and F1-score.

Keywords: Malicious software · Malware detection
Generative adversarial networks · Variational autoencoder · Transfer learning

1 Introduction

Malicious software (malware), which is a generic term for all software products that adversely affect computers, is significant tool used in cyberwar. It is generally used to steal personal, financial, or business information as well as military information. It has been steadily growing in speed (rapidity of threats), number (growing threat landscape), and discrepancy (introduction of new methods) [1]. They usually intrude into a computer and destroy or steal important information. Once infected, it infects additional connected computers in the vicinity and increases the damage. Malware is generally divided into 4 types in terms of its behavior: Virus, worm, PUP, and trojan. When virus is executed, it manipulates malicious behavior that it is doing to other files, such as by inserting code in another medium. Unlike a virus, a worm replicates itself without an intermediary, and the infection proceeds through a vulnerability of the

H. Yin et al. (Eds.): IDEAL 2018, LNCS 11314, pp. 499–507, 2018.
https://doi.org/10.1007/978-3-030-03493-1_52

operating system to another network. If the worm is executed, it will copy itself constantly in the PC, which will adversely affect the memory and the CPU. Potentially unwanted program (PUP) is installed with consent that is not done voluntarily. It may use an implementation that can compromises privacy or weaken the computer's security. Trojan misleads the user's original intention by performing a hidden function if this is executed after the attacker has tricked the target for the intended purpose.

There are two main ways to detect malware: one is static code analysis and the other is dynamic code analysis. Former refers to analyze software without actual execution, but latter dose with it. Static code analysis detects malware based on raw code, so there is no need for additional processors for malware execution and no risk of direct execution, but it is vulnerable to malware deformability. Dynamic code analysis is robust to malware variability because it detects malware based on actual execution, but there are intense time complexity, large resource consumption, and poor scalability. To work out both limitations, we propose a deep learning method to detect malware based on raw code and to generate malware with arbitrary modified features so that detector can learn features of it even with unseen obfuscations. Before generating the data directly, it is important to know the features of them. Because VAE learns to project data with encoder to a specific latent space and reconstructs them with decoder, it can represent the characteristics of data. We exploit the decoder and encoder to generate the competitive new data for GAN and to confirm that G generates appropriate data, respectively.

2 Related Works

Many studies have been conducted to deal with malware because the damage incurred by it has been increasing and the need for malware detection methods is evident. We review the approaches for malware detection in two categories: static code analysis and dynamic code analysis.

In the approach of static code analysis, Nataraj et al. preprocessed and classified malware data written in the binary code into images [2]. However, if the null values inserted in malware changes, this method cannot detect malware well. Grace et al. tried detecting Android malware using first-order and second-order analyses [3]. They used ensemble technique which mixes two modules that detect malware in different ways. Garcia et al. extracted features of malware using image transformation technique proposed by Nataraj et al. and detected the malware using random forest [4]. Wang et al. proposed a method to detect malware using an adversary resistant deep learning model with random feature nullification [5]. They nullified characteristics randomly, resulting in the risk of eliminating meaningful features as well as null values.

There are also several studies that detect malware based on dynamic code analysis. This could recognize malware behavior, but it limited to detect malware which matches with stored templates. Ye et al. used Windows Audit Log to detect malware [6], but they have disadvantage that the malware which has only pre-defined features can be detected; thus, they cannot capture modified features. Lin et el. proposed a virtual time control (VCT) mechanics to detect malware in a short time [7].

These studies proposed methods that extract features and detect malware based on them, or execute it directly and capture it with logs, but they did not deal with obfuscation; thus, detection of modified malware is not guaranteed. To detect them, we propose a method that exploits deep learning with GAN based on VAE. The proposed model extracts appropriate features with VAE, generates virtual malware for expanding the range of malware with GAN, and finally detects malware with the generated and the real data by transferring encoder to detector.

3 The Proposed Method

3.1 Overview

Whole process of the proposed method is illustrated in Fig. 1, which is divided into three parts: (1) feature extraction, (2) data generation, and (3) detection.

Before using binary codes of malware in the proposed method, they are preprocessed, and used as the input to the first part where the data is projected to a specific latent space according to its features and reconstructed. We use LSC-VAE in the first part, and reuse encoder of LSC-VAE and decoder of LSC-VAE as G in the second part. G generates fake malware data from a specific latent space with D to learn the characteristics of malware data while encoder projects it back to specific latent space. Finally, the encoder is transferred to the detector, and it is trained to detect malware data.

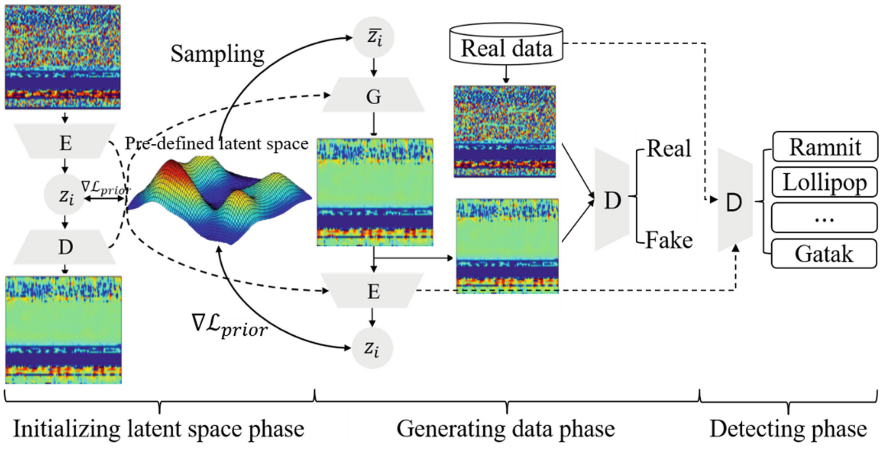

Fig. 1. The architecture of the proposed method. The process under the dashed line is the detection phase and all process is conducted separately.

3.2 Feature Extraction with VAE

Autoencoder has traditionally been used to learn representation of data without supervision. VAE, one of autoencoders, is one of the most popular approaches to unsupervised learning of complicated distributions [8]. The basic process of autoencoder is as follows:

$$z = Enc(x) \sim Q(z \mid x), \tag{1}$$

$$x = Dec(z) \sim P(x \mid z), \tag{2}$$

where x is a data and z is a latent variable. We project x to z with an encoder Q, and reconstruct z back to x with a decoder P. It can learn a representation of data, but it is hidden to us. To show which features are projected and reconstructed, we project and reconstruct the data x_i with i-feature into a specific space $Q(z_i \mid x_i)$. The changed autoencoder process is as follows.

$$z_i = Enc(x_i) \sim Q(z_i \mid x_i), \tag{3}$$

$$x_i = Dec(z_i) \sim P(x_i \mid z_i), \tag{4}$$

where index i means a feature which is included in data x or latent variable z. The encoder is regularized by imposing a prior over the latent distribution $P(z)$. In general, $z \sim N(0, I)$ is chosen, but we choose $z_i \sim \mathcal{N}(\mu_i, I)$ for dealing with a specific feature, where μ_i is a prototype vector of data with i-feature. We call VAE that goes through the process of Eqs. (3) and (4) as LSC-VAE. The loss of LSC-VAE is sum of loss of VAE and prior regularity as in Eq. (5).

$$\mathcal{L}_{\text{LSC-VAE}} = \mathbb{E}_{z_i \sim Q(z_i|x_i)}[\log P(x_i \mid z_i)] + \mathcal{D}_{KL}[Q(z_i \mid x_i)||P(z_i)] = \mathcal{L}_{VAE} + \mathcal{L}_{prior}, \tag{5}$$

where $\mathcal{D}_{\text{KL}}$ is the Kullback-Leibler divergence. LSC-VAE is also used in training GAN, which is discussed more in details in the next section. We use deconvolution layers to increase the size of latent variables to that of malware data [9]. We put batch normalization layer [10], leaky ReLU activation layer and dropout layer after every layer in above except the last layer.

3.3 Generating Data Using LSC-GAN

GAN has led to significant improvements in data generation [11]. The basic training process of GAN is to adversely interact and simultaneously train G and D. Equation (6) shows the objective function of a GAN. p_d is the probability distribution of the real data. $G(z)$ is generated from a probability distribution p_z by the G.

$$\min_{\text{G}} \max_{D} V(D, G) = \min_{G} \max_{D} E_{x \sim p_d(x)}[logD(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{6}$$

Since original GAN has disadvantage that the generated data are insensible because of the unstable learning process of the generator, we pre-train G with decoder of VAE. The result of VAE is that $\left|\mathrm{p}_{\mathrm{d}}^{LSC-VAE} - p_G^{LSC-VAE}\right| \leq |p_d - p_G|$ which yields a goal of GAN $(\mathrm{p}_{\mathrm{d}} \approx p_G)$ stably.

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., $p_d \approx p_G$, the GAN reaches the Nash equilibrium. We define $J(D, G)$ and $K(D, G)$ as in [12]. Then, we can define the Nash equilibrium of the GAN as a state that satisfies Eqs. (7) and (8). Fully trained generator and discriminator are denoted by G^* and D^*, respectively.

$$J(D^*, G^*) \leq J(D^*, G) \; \forall \; G \tag{7}$$

$$K(D^*, G^*) \leq K(D, G^*) \; \forall \; D \tag{8}$$

Theorem 1. If $p_d \approx p_G$ almost everywhere, then the Nash equilibrium of the GAN is reached.[1]

GAN used in this paper is based on Wasserstein GAN. It defines the distance between distributions as Wasserstein, not Jensen-Shannon used in original GAN [13]. For verifying that the data generated from the space representing the *i*-feature really has the *i*-characteristic, we project it back to latent space with encoder Q. Objective function of LSC-GAN is in Eq. (9).

$$\max V_{LSC-GAN}(D, G) = \max V(D, G) - \mathcal{D}_{KL}[Q(z_i \mid G(z_i)) || \mathcal{N}(\mu_i, I)]] \tag{9}$$

Reason for using GAN is to classify new data with surrounding noise with model learned from existing data. Because GAN generates data from a random distribution, new data has some variants compared to existing data. It expands a knowledge space of data. Encoder and D are trained with expanded space. Therefore, encoder and D are robust to deformation [14].

4 Experiments

4.1 Dataset and Experimental Setting

To validate the performance of generating fake data and detecting malware data, we used the malware dataset from the Kaggle Microsoft Malware Classification Challenge[2]. Ramnit, Kelihos ver 3 (K3), Simda, and Kelihos ver1 (K1) are a botnet which can be used to perform distributed denial-of-service attack (DDos attack), steal data,

[1] The proof of Theorem 1 was discussed by Kim et al. [14].

[2] https://www.kaggle.com/c/malware-classification.

and allows the attacker to access the device and its connection. Vundo, Simda, Tracur, and Gatak are a trojan horse malware which is misleads user's true intentions. It appears to be a normal software, but it will run malicious code if run. Lollipop is an adware that generates revenue for its developer by automatically generating online advertisements in the user interface of the software. Obfuscator, ACY (O.ACY) is a combination of several malicious methods.

The data are given in form of assembly and binary code, and we only used the binary code. The values in hexadecimal are converted to decimal numbers from 0 to 255, which are normalized into values of 0 to 1. As Nataraj did, we convert the malware code to image, called *malware image*.[3] Then, because malware images were too large, the images were reduced to 0.2 times their original sizes, resulting in the size of 256 × 128. The number of malware for each type is shown in Table 1. We set the size of latent space as 270 dimension and assign 90 dimension for each malware class.

Table 1. Summary of malware data.

Type	Train (test)	Type	Train (test)	Type	Train (test)
Ramnit	1387 (153)	Vundo	435 (40)	K1	358 (40)
Lollipop	2249 (229)	Simda	35 (7)	O.ACY	1110 (118)
K3	2620 (322)	Tracur	688 (63)	Gatak	898 (115)

4.2 Result of Detection

In this section, the performance of the proposed model is compared with other conventional machine learning algorithms, a convolutional neural network (CNN) (which has the same architecture to that of detector), and a GAN (which is not based on VAE). When going from the generation phase to the detection phase, D of GAN is transferred because there is no encoder to be transferred to the detector. Architectures of CNN and D of GAN are same to detector of our proposed model. Parameters of other machine learning methods is set to default value in scikit-learn library.

The results of these experiments are summarized in Fig. 2 and Table 2. The averaged accuracy is 96.97% which is better than other conventional method. Some studies of malware detection used the same dataset that we used here [15–17]. However, there is no significant difference in the malware detection ability after reducing the scale of the malware data. We also verify the statistical meaning of difference among CNN, GAN, and the proposed model with *t*-test, resulting in meaningful difference.

[3] We use the 'jet' colormap to represent the values between 0 and 1 in color.

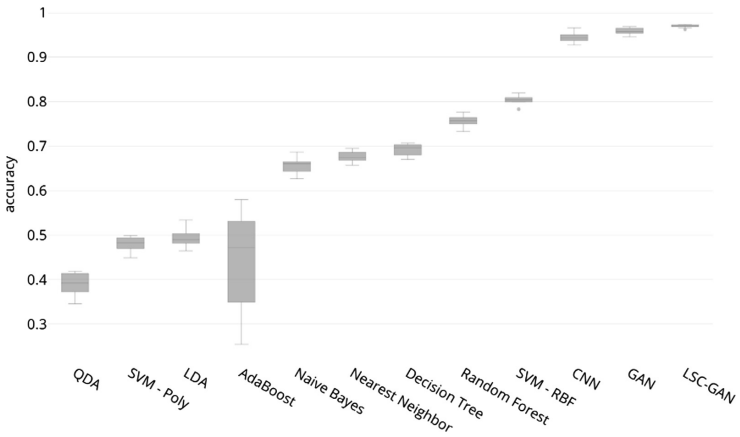

Fig. 2. The results of experiments in box plot.

Table 2. The numerical results of experiments.

	CNN	GAN	LSC-GAN
Accuracy	94.49	95.79	96.97
Std. dev	1.34e−04	5.78e−05	1.22e−05
p-value	4.17e−06	3.11e−04	–

4.3 Analysis of Result

Generated malware images are illustrated in Fig. 3 with the real images. They look very similar to the actual image, but we cannot return the image to the code because the image was reduced in the preprocessing step. Data similar to the generated data is found in the training dataset using structural similarity index (SSIM). The SSIM value and standard deviation of malware data containing real and generated data are 0.1019 and 0.3308, respectively, but those of only real data are 0.1068 and 0.3224, respectively. It shows that the generated data increase the diversity of malware.

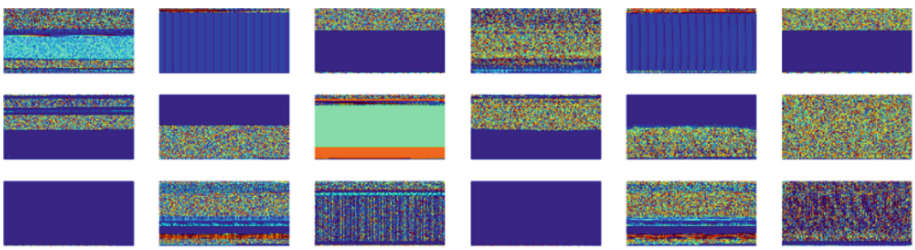

Fig. 3. The actual malware images (left 9) and the generated malware images (right 9). The malware image to the left of the three spaces of the generated malware image is the actual image with the largest SSIM value.

We calculate precision, recall, and F1 scores for each malware type and lists them in Table 3. Most of malware data have high F1-score except for Simda, whose type of malware is scarce.

Table 3. Precision, recall, and F1 scores for different malware types.

	R	L	K3	V	S	T	K1	O	G
Precision	0.987	0.987	1.000	0.886	0.500	0.836	1.000	0.966	0.974
Recall	0.955	0.978	0.997	0.975	0.286	0.968	0.950	0.949	0.983
F1-score	0.970	0.982	0.998	0.929	0.364	0.897	0.974	0.957	0.978

5 Conclusions

In this paper, we raise the problems caused by malware and attempt to solve them. The newly generated malware has some obfuscations compared with existing malware, e.g., code exchange, null value insertion, and reorganization of subroutine. To deal with theses modification and detect modified malware, we propose a LSC-GAN which extracts features with VAE and generates virtual data to expand a range of knowledge space. The proposed detector inherits the ability of the encoder, so it can know a wider knowledge space. It achieves 96.97% of accuracy which is a better performance than other conventional machine learning algorithms.

In the future, we will address the issue of different lengths of various malware, since the malware code was converted into malware images through crop and operations in this study. After this issue, we will convert generated malware data to malware code. We plan to build a complete system for detecting malware and use it for the real world.

Acknowledgment. This work was supported by Air Force Defense Research Sciences Program funded by Air Force Office of Scientific Research.

References

1. Dhammi, A., Singh, M.: Behavior analysis of malware using machine learning. In: IEEE International Conference on Contemporary Computing, pp. 481–486 (2015)
2. Nataraj, L., Karthikeyanm, S., Jacob, G., Manjunath, B.S.: Malware images: visualization and automatic classification. In: Conference on Visualizing for Cyber Security, pp. 1–7 (2011)
3. Grace, M., Zhou, Y., Zhang, Q., Zou, S., Jiang, X.: Riskranker: scalable and accurate zero-day android malware detection. In: Proceedings of International Conference on Mobile Systems, Applications, and Services, pp. 281–294 (2012)
4. Garcia, F.C.C., Muga, I.I., Felix, P.: Random Forest for Malware Classification. arXiv preprint arXiv:1609.07770 (2016)

5. Wang, Q., et al.: Adversary resistant deep neural networks with an application to malware detection. In: International Conference on Knowledge Discovery and Data Mining, pp. 1145–1153 (2017)
6. Ye, Y., Chen, L., Hou, S., Hardy, W., Li, X.: DeepAM: a heterogeneous deep learning framework for intelligent malware detection. Knowl. Inf. Syst. **54**, 1–21 (2017)
7. Lin, C.H., Pao, H.K., Liao, J.W.: Efficient dynamic malware analysis using virtual time control mechanics. Comput. Secur. **73**, 359–373 (2018)
8. Kingma, D.P., Welling, M.: Auto-encoding Variational Bayes. arXiv preprint arXiv:1312.6114 (2013)
9. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8689, pp. 818–833. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10590-1_53
10. loffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv preprint arXiv:1502.03167 (2015)
11. Goodfellow, I. et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
12. Kim, J.Y., Bu, S.J., Cho, S.B.: Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. Inf. Sci. **460–461**, 83–102 (2018)
13. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein generative adversarial networks. In: International Conference on Machine Learning, pp. 214–223 (2017)
14. Radford, A., Metz L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
15. Kim, J.Y., Bu, S.J., Cho, S.B.: Malware detection using deep transferred generative adversarial networks. In: Liu, D., Xie, S., Li, Y., Zhao, D., El-Alfy, E.S. (eds.) Neural Information Processing. ICONIP 2017. Lecture Notes in Computer Science, vol. 10634, pp. 556–564. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70087-8_58
16. Drew, J., Moore, T., Hahsler, M.: Polymorphic malware detection using sequence classification methods. In: Security and Privacy Workshops, pp. 81–87 (2016)
17. Narayanan, B.N., Djaneye-Boundjou, O., Kebede, T.M.: Performance analysis of machine learning and pattern recognition algorithms for malware classification. In: Aerospace and Electronics Conference on and Ohio Innovation Summit, pp. 338–342 (2016)