Journal of Computer Virology and Hacking Techniques (2021) 17:153–163
https://doi.org/10.1007/s11416-021-00378-y

ORIGINAL PAPER

Enhanced DNNs for malware classification with GAN-based adversarial training

Yunchun Zhang[1] · Haorui Li[1] · Yang Zheng[1] · Shaowen Yao[1] · Jiaqi Jiang[1]

Received: 10 September 2020 / Accepted: 25 February 2021 / Published online: 10 March 2021

Abstract
Deep learning based malware classification gains momentum recently. However, deep learning models are vulnerable to adversarial perturbation attacks especially when applied in network security application. Deep neural network (DNN)-based malware classifiers by eating the whole bit sequences are also vulnerable despite their satisfactory performance and less feature-engineering job. Therefore, this paper proposes a DNN-based malware classifier on the raw bit sequences of programs in Windows. We then propose two adversarial attacks targeting our trained DNNs to generate adversarial malware. A defensive mechanism is proposed by treating perturbations as noise added on bit sequences. In our defensive mechanism, a generative adversary network (GAN)-based model is designed to filter out the perturbation noise and those that with the highest probability to fool the target DNNs are chosen for adversarial training. The experiments show that GAN with filter-based model produced the highest quality adversarial samples with medium cost. The evasion ratio under GAN with filter-based model is as high as 50.64% on average. While incorporating GAN-based adversarial samples into training, the enhanced DNN achieves satisfactory with 90.20% accuracy while the evasion ratio is below 9.47%. GAN helps in secure the DNN-based malware classifier with negligible performance degradation when compared with the original DNN. The evasion ratio is remarkably minimized when faced with powerful adversarial attacks, including $FGSM^r$ and $FGSM^k$.

Keywords Malware classification · Deep learning · Adversarial attack · Robust machine learning · Generative adversarial network

1 Introduction

The pioneering work on modeling malware detection task as statistical testings in [1] is constructive and lays the foundation for never-ending battle between attackers and detectors in the era of virology. Since then, deep learning-based malware classification is proved to be more effective than conventional signature-based mechanisms [2] and many classifiers have been proposed and applied in different commercial anti-virus products [3]. Malware gray-scale image-based classification [4] is the most common technique by training a deep neural network (DNN) model based on image texture features. Beside malware images, other malware features, including API call sequences, operational codes (OpCode) and raw bit-sequences, are widely used for malware classification. Depending on whether the extracted features are collected by running malware programs or not, existing feature extraction algorithms are grouped into either dynamic or static. It is important to notice that both static and dynamic methods need an independent feature extraction procedure and this may introduce high time overhead. Most kinds of features extracted are organized into binary feature vectors, such as, APIs and OpCode where 1 means its appearance and 0 means its absence. As those features share similar data structure, DNNs trained based on them share similar architecture. Therefore, adversarial samples generated by adversarial attack targeting one DNN-based classifier are highly transferable to other DNNs.

In contrast, malware classifier by eating the raw bit-sequence as input without an independent feature extraction procedure is proved to be more effective. Major breakthroughs are represented by the following three papers. First, a deep neural network *MalConv* [5] [6], is trained on raw bits of the malware program. This paper proves that only raw bits are enough to build a malware classifier with satisfactory per-

✉ Shaowen Yao
yaosw@ynu.edu.cn

1 School of Software, Yunnan University, Kunming 650095, China

formance. Second, the authors in [7] designed an adversarial attack targeting *MalConv*. This attack is gradient-based while perturbations are mainly applied on the appended section at the end of the original file. Third, the authors in [8] proved that the vulnerability exploited in [7] succeed because the positional information is not encoded in *MalConv*. An optimized convolutional neural network (CNN) is then proposed based on *MalConv* in [8]. Meanwhile, a slash attack is proposed in [7] to scatter the perturbations evenly in the whole PE file.

As raw program bits are applicable for malware classification, following observations are worth noticing. First, as the beginning section of a PE file is valuable, *MalConv* reads up to 2 MB bytes from a portable executable (PE) file as input. Second, the degree of perturbations that are measured and quantized by the total number of bits changed should be minimized. Third, to preserve the original malicious functionalities and semantic meanings, the perturbations are usually applied on a precisely computed section within PE file, such as the end section in [7]. Fourth, Adversarial attacks targeting binary feature vector-based DNNs are harder to implement because a larger distortion is required than other feature-based classifiers. Meanwhile, adversarial samples with large and infinitesimal perturbations are more easily to be detected during invasion [9]. Fifth, adversarial training by combining adversarial samples with the original training samples has been proven to be helpful in enhancing the robustness. However, the relationships among the quality of adversarial samples, cost and performance improvements by introducing adversarial training are not fully researched.

To secure DNNs from adversarial attacks, some defensive mechanisms are in great demand today. This paper aims at designing a generative adversarial network (GAN)-based adversarial malware filtering model and adversarial training mechanism to secure binary feature-based DNNs for malware classification. Based on the above observations, this paper makes the following contributions:

(1) Instead of eating the whole raw bits of a PE program in Windows, we only use the header 2,000 bytes for analysis by referring [5]. Then, this paper proposes a raw-bit-sequence-based deep neural network. The proposed DNN achieves 97.31% accuracy on average and outperforms other classifiers especially *MalConv* [5] and XGBoost.

(2) Targeting the proposed DNN and *MalConv* [5], adversarial attacks proposed in [7] and [10] are implemented to generate adversarial samples. The redesigned adversarial attack is measured not only by its effectiveness and correctness, but also by its quality and cost on generated adversarial samples.

(3) To improve $FGSM^k$ proposed in [10] and make them suitable for any binary feature-based DNNs for malware classification and detection, this paper designed a generative adversarial network (GAN) for purifying adversarial samples by selecting samples that have higher probability to mislead the target DNNs. By modeling and solving a *minmax* function through redesigning the loss functions of both generator and discriminator of GAN, perturbations computed based on the binary features can be effectively optimized. By input those filtered adversarial samples into DNN, the evasion ratio of adversarial samples increased by 50.64% on average when compared with $FGSM^k$ and $FGSM^r$ [11] by 27.28% and 35.16%, respectively.

(4) While GAN with filter based adversarial attack outperforms other attacks, as shown in our experiments in Sect. 5, the generated high quality adversarial malware samples are selected. We combine those high quality adversarial samples with the original samples and retrain the DNN. This adversarial training DNN is evaluated by both accuracy and evasion ratio under three conditions. The results show that adversarial training with samples generated by GAN with filter achieves 88.33% accuracy on average and is only 6.69% less than the original DNN classifier without being attacked.

This paper is organized as follows. The background and major contributions are introduced in Sect. 1. The related works in the area of binary feature-based DNNs for malware classification, adversarial attacks and robust defensive mechanisms are surveyed in Sect. 2. Section 3 provides more details on our designed DNNs by eating the beginning section of a program or API features are proposed. What follows are FGSM-based adversarial attack and improved GAN-based adversarial sample enhancement mechanism to further optimize existing attacks in Sect. 4. The experiments and results are fully analyzed in Sect. 5. A conclusive mark and possible future research directions are presented in Sects. 6 and 7.

2 Related works

When input raw bit-sequence vectors to train a deep neural network for malware classification, those vectors share similar characteristics with binary feature vectors, such as API, operational codes (OpCode), etc. Therefore, this paper focuses on recent advances on deep learning-based malware classification with binary feature vectors as inputs. Existing works on other malware features, including malware grayscale images [12], API call sequences, OpCode N-gram and dynamic behaviors, are not included here.

2.1 Binary feature-based malware classification

When conventional machine learning algorithms are applied on malware classification, an independent feature extraction process is required. The classifiers trained can be grouped into two categories: API call-based and raw bit sequence-based.

(1) API call-based malware classification

As API call sequences are frequently used to detect malware from benign programs, some well trained models are available [13–15]. Instead of analyzing the API call sequence patterns that are indicative for both malware and benign program, API-based features are usually organized into binary feature vector. As each attribute takes either 0 or 1, the conventional gradient-based deep learning models are not applicable directly. In [10], a feed-forward neural network with *ReLU* activation function and *LogSoftMax* function for the output layer is proposed by input binary API feature vectors. There are 22,761 unique API calls for their dataset. By modelling high-level semantic relationships among API calls, HinDroid [16] is proposed for malware classification.

(2) Raw bit sequence-based malware classification

The pioneering work on using raw bit sequence for malware classification is proposed in 2015 Microsoft Malware Challenge.[1] The beginning section of a *.ASM* file is retrieved as inputs to build a classifier. In [5], malware classification is done by eating the whole *.exe* to train *MalConv*. In *MalConv*, the original whole bytes, together with padding bytes on the end section of a *.exe* file, are embedded into a single vector and then feed into a CNN for training.

2.2 Adversarial attacks against binary feature-based malware classifiers

When input binary features, attackers aim at leading the classifier to give a wrong predicted label while minimizing the number of bits that are perturbed. As most adversarial attacks are gradient-based, input feature vectors are required to be continuous and derivative. However, binary feature vectors are discontinuous and thus non-derivative. Consequently, most existing adversarial attacks cannot be directly applied on binary feature vector-based DNNs.

As *MalConv* is the first widely acknowledged classifier based on raw bit sequences, most adversarial attacks are targeting it. In [7], an adversarial attack that performs on the appended section at the end of a binary file is proposed. This gradient-based attack is compared with a method that apply random perturbation on the same section. While this attack is proved to be effective, it is, however, changed the file size and could be detected when comparing its file size with the original program file. Kreuk et al. [17] designed an adversarial attack on one-hot vector representing the binary file. The attack is FGSM (Fast Gradient Sign Method)-based [18] and adversarial samples are generated by reconstructing the new binary file based on embedding matrix with a revised surrogate loss function. However, this work heavily relies on the learned embedding and is shown to be with low transferability. Hu, et al. [19] proposed *MalGAN* which works by generating adversarial samples under black-box attack based on binary malware features. The substitute detector is used to fit the black-box detector and provides gradient information to train the generator. However, *MalGAN* is unstable and a stabilized training method is still missing. Chen et al. [20] proposed an automated tool for constructing adversarial Android malware by improving C&W [21] and JSMA (Jacobian-based Saliency Map Attack) [22] attacks. They proved that a 99% mis-classification rate was possible by perturbing 3.5 features on average. Wang et al. [23] proposed an adversarial attack against deep neural networks for malware detection by nullification of binary features indicating whether an event is appeared or not.

While existing adversarial attacks are successful and effective, it is widely acknowledged that most of them failed to preserve the original malicious functions. Therefore, the authors in [24] combined C&W attack with code replacement techniques, including In-place randomization (IPR) [25] and Disp [26], to generate practical adversarial malware. Meanwhile, conventional code randomization and binary manipulation techniques are employed to generate adversarial malware while preserving their original functionalities [27].

2.3 Secure and robust machine learning

The reasons why deep neural networks are vulnerable against adversarial attacks are not fully researched yet. However, some defensive mechanisms have been proposed for malware detection. First, some machine learning-based malware classifiers against evasion attacks are proposed [28,29]. Besides, the recent progress on adversarial malware detection is surveyed in [30]. However, most of the existing mechanisms are proved to be either with low accuracy or targeting linear classifiers only. Second, the authors in [10] insist that it is the model's blind spots that are responsible for a successful attack. By modeling malware adversarial learning as a saddle point model, a new blind spots coverage metric is proposed to measure the models' effectiveness against four adversarial attacks, including $dFGSM^k$, $rFGSM^k$, BGA^k and BCA^k. Third, Chen et al. [31] proposed *DroidEye* to find the best trade-off between security and accuracy for malware classifiers. Meanwhile, *SecMD* [32] and *SecureDroid* [33] are proposed to improve the system security against multiple adversarial attacks. While some defensive mechanisms are proposed for securing machine learning models, it is still a challenging problem to secure DNN-based Malware classifiers.

All in all, both adversarial attacks and defensive mechanisms targeting deep learning-based malware classifiers gain momentum recently. Adversarial attacks are not only helpful in evaluating the models robustness, but also deepen our understanding of how DNNs work and why adversarial attacks succeed.

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

3 Raw bit sequence-based DNN for malware classification

When given a collection of malware programs, the following four steps as shown in Fig. 1 are designed to train a robust and secure DNN for malware classification. First, feature extraction process is applied to derive a binary feature vector for each sample. Generally speaking, each sample x is converted into a binary feature vector. While programs are varied in length, this paper picks the beginning 2,000 bytes for analysis. Files longer than predefined length are truncated. Second, adversarial attacks in [10] are introduced to generate binary adversarial sample x'. Third, clean sample x and adversarial sample x' are combined and then feed into a GAN model for data augmentation and enhancement. Fourth, a secured DNN model for malware classification is implemented by *adversarial training* [22] with adversarial samples generated by GAN-based filter.

3.1 Data preprocessing and feature extraction

While malware is popular in both Windows and Android systems, variants in different systems vary remarkably. While Android program shares different file format and architecture with Windows PE file, we restrict our work on Windows malware only. However, our proposed models and algorithms are also applicable on Android programs with minor revisions on data preprocessing step as shown in Fig. 1. In Windows, a PE file is chosen because its header part contains the most distinctive features for malware classification. When given *.exe* programs for analysis, data preprocessing method is applied to extract either raw bit sequences or binary feature vectors. Thus, the models proposed in this paper also applicable on API calls, OpCodes and other binary features.

Generating the raw bit sequence from a program is simple. It is also important to notice, however, that programs are with varied length and cannot be directly input into DNNs. Considering the input size of a malware program, we made the following observations. First, the authors in [5] has already proven that neural network architecture need no major redesign to take the whole raw bit stream of a program as input. However, local patterns contribute more than global patterns for malware classification. Second, one of the wining solutions in 2015 Microsoft Malware Challenge builds a malware classifier based on the beginning 1,000 bytes of the *ASM* and *BYTE* files. Third, the authors in [10] use only 22,761 bits for malware classification. Based on the above analyses, this paper chose 8,000 bits from the beginning section of a program and organized them into a binary feature vector. Each feature vector is saved as a *.cvs* file with a unique label derived by VirusTotal.[2]

[2] https://www.virustotal.com/gui/.

3.2 Deep neural network-based malware classification

A deep neural network for malware classification is defined as a function $y_i = f_\theta(x_i)$, where $y_i \in [1, C]$ and C represents the total number of malware families, θ is the weighted parameter that are initialized randomly and optimized through the training process. A typical DNN is composed of an input layer, multiple convolutional layers, multiple pooling layers, a fully-connected layer and an output layer. Let $f : x \rightarrow z$ be a function that takes $x \in \mathbb{Z}^d$ (where d is the number of bits for a given sample) as input and outputs the logit vector $z \in \mathbb{Z}^k$, where $k \in C$ is the number of classes. Given a labeled malware sample x_i and corresponding label y_i, we define $\mathcal{L}(\theta, x, y)$ as the loss function of our classifier with parameter θ on given (x, y). The malware classification task is accomplished through finding the global optimized solution for θ defined as:

$$\theta^\star \in arg \underset{\theta \in \mathbb{Z}^p}{MIN} \mathbb{E}_{(x,y)\in\mathcal{D}} \mathcal{L}(\theta, x, y) \tag{1}$$

in which, $\mathcal{L}()$ measures the deviations of the prediction $f_\theta(x)$ from its true label y. The training process stops when Eq. 1 converges.

Considering the differences between discrete binary feature vectors and continuous features derived from conventional samples (such as images, texts, etc.), the network architecture for malware classification need to be redesigned. Specifically, the architecture of DNN with discrete binary feature vectors as inputs is as bellow:

(1) **Input layer** accept binary bit vector $x \in \mathbb{Z}^d$ as input, where d is the number of bits for a given samples and is set as $d = 8 \times 1{,}000$ in this paper. Each input feature vector is required to be with the same length. While our DNN also applicable on other binary features, the shorter one is padded with 0 on its rear end.

(2) **Hidden layer** is composed of multiple convolutional layers and pooling layers. Each convolutional layer takes the outputs from the previous layer as inputs. To solve linearly inseparable problems from real applications, neural networks with perception is proposed. The perception is defined as:

$$y = \sum_{i=1}^{K} W_i x_i + b \tag{2}$$

where W_i is a weighted coefficient matrix. We implement 3 hidden layers and each layer is configured with 1,200 neurons. The weighted coefficient matrix is a 1,200 $\times$ 1,200 matrix. To learn non-linear decision boundaries precisely,

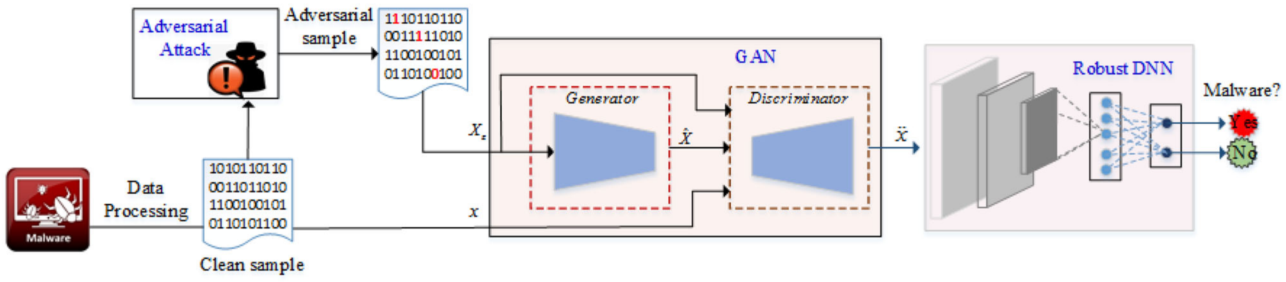

Fig. 1 Major steps for secure malware classification against GAN-based adversarial attacks

ReLU is used as activation function on neurons defined as:

$$\sigma(z) = ReLU(x) = \begin{cases} x, & if x > 0 \\ 0, & if x \le 0 \end{cases} \tag{3}$$

The proposed DNN uses feed-forward back propagation and the output a_j^L of the j-th neuron on the L-th layer is computed as:

$$a_j^L = \sigma(Z_j^L) = \sigma(\sum_{k=1}^{m} w_{jk}^L a_k^{L-1} + b_j^L) \tag{4}$$

in which, w_{jk}^L is the weighted parameter and b_j^L is the bias added. Then, the output from the L-th layer is derived as:

$$a^L = \sigma(z^L) = \sigma(w^L a^{L-1} + b^L) \tag{5}$$

The weights on each layer are optimized and updated by solving the above objective function with stochastic gradient descent (SGD) as follows:

$$\theta \leftarrow \theta - \varepsilon \cdot \nabla_\theta (\sum_{(x,y)\in\mathbb{B}} \ell(f_\theta(x), y)) \tag{6}$$

where, $\mathbb{B}$ is a subset of randomly selected samples and adjustable through batch size parameter. The learning rate ε is used to control the magnitude of how weights θ should be adjusted.

(3) **Output layer** receives the outputs from the last pooling layer for final classification. The output layer performs classification based on a $1 \times C$ vector, indicating its probability to each specific malware family. To further reduce the value domain, *LogSoftMax* function is designed and then output a probabilistic distribution as:

$$LogSoftmax(z_i) = log\left(\sum_j a_j^L\right)$$
$$= log\left(exp(z_i)/\sum_j exp(z_j)\right)$$
$$= log\left(e^{(z_i^L)}/\sum_j e^{z_j^L}\right) \tag{7}$$

where $\sum_j e^{z_j^L}$ is the summed outputs from the L-th layer. Then, by computing the derivative we got:

$$\nabla LogSoftmax(z_i) = \left(log\left(\frac{exp(z_i)}{\sum_j exp(z_j)}\right)\right)' \tag{8}$$
$$= 1 - \left(\frac{exp(z_i)}{\sum_j exp(z_j)}\right) \tag{9}$$

Finally, the one with the highest probability is assigned as the predicted label. The above trained DNN is also applicable on malware detection task with minor modifications on the number of output layer nodes.

4 GAN-based defensive mechanism against FGSM-based adversarial attacks

4.1 FGSM-based attack for discrete binary features

Adversarial attacks are common in the context of malware attacks with the aim of wrongly classify malware as benign or vice versa. Here, we define adversarial attack in deep learning-based malware detection as:

Definition: Adversarial Attack. *For a given deep neural network $y = f_\theta(x)$, the adversarial attack aims at changing input x with perturbation δ* into *$x' = x + \delta$, where $\delta \le \epsilon$ with the maximum perturbation as ϵ. A successful adversarial attack is defined as finding at least one x' that satisfies $y' = F_\theta(x') \ne F_\theta(x)$.*

As conventional DNNs and adversarial attacks are gradient-based, the feature domain should be continuous and derivative. For attacks targeting the gray-scale image-based CNNs, many gradient-based adversarial sample generation methods are applicable, including *FGSM* [18], *FGSM*^{r} [11], *rFGSM*^{k} and *dFGSM*^{k} [10]. However, adversarial attacks against binary feature-based DNNs face the following challenges.

First, as binary features are discrete, gradient-based attacks cannot be applied directly. Second, malware functions should not be changed by any adversarial perturbation attacks which means only adding binary feature is allowed while deletion is prohibited. Third, the magnitude of perturbation quantified by distance-based metrics is usually constrained by ℓ_0, ℓ_2 and ℓ_∞. However, malware belonging to the same family may have longer distance than those from different families when measured by API [34]. Fourth, as adding an API call into an application without affecting its original functionality is a non-trivial task, it is thus critical to restrict the number of bits changed. The above restriction also holds on bit sequence-based features.

To propose an effective adversarial attack targeting previously trained malware classifiers when faced with the above challenges, this paper redesigned *FGSM*^{r} [11] algorithm as following. Given a selected malware sample x with family label c_x and is correctly classified by classifier as $y = F(x)$, the perturbed sample x' is constructed by maximizing the loss function $L(\theta, x, y)$ as:

$$x' \in s^*(x) = arg \max_{\overline{x} \in S(X)} L(\theta, \overline{x}, y) \tag{10}$$

in which, $x' \in S^*(X)$ is the newly constructed adversarial sample, $S(X) \subseteq X$ is a set of binary feature vectors generated by each sample in X, $S^*(X) \subseteq S(X)$ is a set of binary feature vectors generated by X' that maximizing the loss function.

In *FGSM*^{r} [11], perturbations are applied on a set of computed positions by changing 0s to 1s indicating an added API call. To minimize the total number of bits perturbed and introducing *FGSM*^{r} on raw bit sequence targeting our previously designed DNN, the perturbations applied are constrained by

$$S(X) = \overline{X} \in 0, 1 | X \wedge \overline{X} = X and |S(x)| = 2^{(m - x^T 1)} \tag{11}$$

in which, x^T means transpose operation to x. Finally, a set of adversarial malware samples are generated. While $rFGSM^k$ is proved to be the best in [10], adversarial malware binaries generated by it are also included in our experiments and analysis.

4.2 GAN-based adversarial sample enhancement

When adversarial samples are mingled with clean samples in training and/or testing phases, it is required that adversarial samples should be detected. However, not all adversarial samples have the same probability to defeat the victim classifiers. With the aim of choosing high quality adversarial samples and serving the purpose of enhancing DNN robustness, a GAN-based adversarial sample filter component is designed. Our design is inspired by the intuition that generator in APE-GAN [9] is used to sanitize the input before passing it to the classifier. All samples are required to be filtered before forwarding them to the target DNN for classification. The working mechanism and major components in this GAN-based filter mechanism are as shown in Fig. 2. As shown in Fig. 2, the GAN-based filter mechanism is composed of the following two major steps.

(1) Adversarial sample filtering

The trained DNN for malware classification takes the original sample x and adversarial sample $x^{'}$ generated by $FGSM^k$ and $rFGSM^k$ as inputs. Instead of output the predicted labels, feature maps derived from the last fully-connected layer are extracted as $Z = \dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n$. We define $\dot{x} = a^{L-1}$ as the output feature maps extracted from the last-fully connected layer in DNN. We set $\dot{x}_{max} = MAX(Z)$ and $\dot{x}_{mean} = MEAN(Z - \dot{x}_{max})$ as the maximum sample in Z and their averaged value, respectively. Meanwhile, $\overline{Z} = \dot{x}_i$ where $\dot{x}_i$ meet $\|\dot{x}_i - \dot{x}_{mean}\| \leq \theta$ and θ is a predefined hyper-parameter. $\overline{Z}$ is the filtered set of samples that have the highest probability to fool the target DNN classifier and is then forwarded to the GAN for data augmentation.

(2) GAN-based adversarial malware construction

When $\dot{x}_i \in \overline{Z}$ is computed, it is forwarded to this GAN network that responsible for data augmentation. For filtered dataset $\overline{Z}$ and the original dataset X, we define $G(\overline{Z})$ and $D(X)$ as the generator and discriminator in GAN, respectively. The data augmentation works as following. First, the discriminator D is trained by solving the following optimization function as

$$\overline{V} = \frac{1}{m} \sum_{i=1}^{m} log D(\dot{x}_i) + \frac{1}{m} \sum_{i=1}^{m} log(1 - D(\dot{x}_i)) \tag{12}$$

To optimize the objective function as shown in Eq. 12, θ_d is adjusted by Eq. 13 as

$$\theta_d \leftarrow \theta_d + \eta \nabla \tilde{V}(\theta_d) \tag{13}$$

where θ is adjusted using gradient descending method.

Second, the GAN's generator is G trained by optimizing the objective function defined in Eq. 14 as

$$\overline{V} = \frac{1}{m} \sum_{i=1}^{m} log(D(G(\dot{z}_i))) \tag{14}$$

in which, $\dot{z}_i \in \overline{Z}$ is the filtered sample. Meanwhile, θ_g is adjusted based on Eq. 15 as

$$\theta_g \leftarrow \theta_g + \eta \nabla \overline{V}(\theta_g) \tag{15}$$

To summarize, the designed GAN achieves data augmentation by solving a *minmax* optimization problem defined in

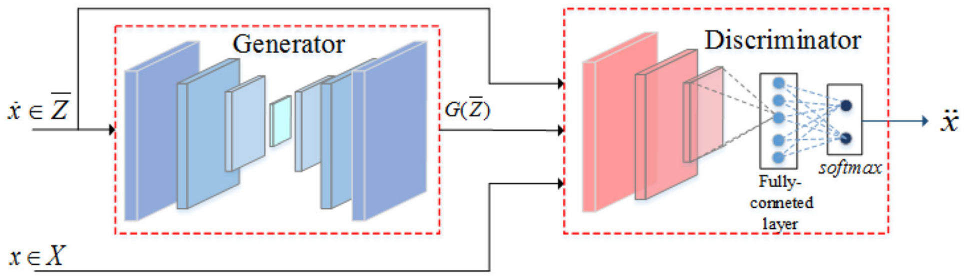

Fig. 2 GAN-based data augmentation for adversarial malware filtering

Table 1 Malware samples used from BIG 2015 dataset

Malware family	Types	No. of samples
Ramint	Worm	1,541
Lollipop	Adware	2,478
Kelihos_ver3	Backdoor	2,942
Obfuscator.ACY	Obfuscated malware	1,228

Eq. 17 as

$$\ddot{x} = min_G max_D \quad = E_{x\ P_{data}(x)}[log(D(x))] \tag{16}$$

$$+E_{z\ p_z(z)}[log(1 - D(G(z)))] \tag{17}$$

in which, a set of $\ddot{x}$s computed by GAN are also adversarial samples but with higher probability to succeed. Those enhanced adversarial samples are mingled with the original clean samples xs for adversarial training. This adversarial training-based classifier is supposed to be more robust and secure than the original DNN.

5 Experiments and analyses

5.1 Datasets and running environments

To make sure that our proposed models work well on Windows programs, two datasets are used: BIG 2015[3] and Windows PE files in [35]. Each sample is converted into a raw bit sequence and the beginning 8,000 bits are preserved and saved in *.csv* format. As samples contained in BIG 2015 is unbalanced and some families are not common in Windows, we chose only 4 major types in our experiments as shown in Table 1. Besides, we include more than 19,000 malware and 19,000 benign programs in [35] because its wide popularity in malware domain. All PE files in [35] contain all necessary information including API call sequences when reloading *.exe* program. We use 80% samples for training and 20% samples for model validation.

When all samples are collected, an independent *program-to-bit-sequences* program written in Python is applied. Each binary feature vector, together with a malware family label, is stored in a *.csv* file. The label is either assigned by professional experts or assigned by tools, such as VirusTotal.[4] When different labels are given by various anti-virus engines, the one with the highest probability is chosen. All programs, including the pre-processing program, DNN, XGBoost, $FGSM^r$, $rFGSM^k$ and GAN, are tested under the same operating system and computing configurations. The running operating systems are configured with Intel Core i7 7500, Nvidia GeForce GTX1060 with 8G memory and run on Ubuntu 16.04. All programs are running on Python 3.6 and TensorFlow 1.7. To accelerate the running speed, both DUDA 9.1 and CUDNN 7 are applied simultaneously. When configured with GPU or with more memory resources, all models would be accelerated.

To evaluate and compare the designed DNNs, the following performance metrics, including accuracy, precision, recall and F1, are computed as shown in Eqs. 18–21.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{18}$$

$$Recall = TP/(TP + FN) \tag{19}$$

$$Precision = TP/(TP + FP) \tag{20}$$

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall} \tag{21}$$

5.2 Classification performance and analysis

To evaluate DNN's classification performance, we also implemented XGBoost for malware classification. In BIG 2015 (Microsoft Malware Challenge) competition, XGBoost is proved to be the best and achieves 98.0% accuracy on maximum when massive samples are provided. Therefore, the comparisons among XGBoost and our proposed DNN are sound and persuasive. Both classifiers are trained with the same training and testing samples. The results are obtained by running the model for at least 10 times and averaged. The experimental results are summarized in Table 2.

It is observed that existing deep learning-based malware classifiers generally achieve 90.0%–95.0% accuracy on average [3]. As shown in Table 2, both the proposed DNN and

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

[4] https://www.virustotal.com/gui/.

Table 2 Performance under Different Classification Models

Classifier	Accuracy	Precision	Recall	F1	Specificity
DNN	**97.31%**	98.00%	96.50%	24.31%	94.36%
XGBoost	93.89%	94.00%	91.50%	23.18%	92.33%

Bold value indicates the best value for each performance measure among different models

Table 3 Performance comparison under GAN-based attack proposed in this paper

Victim classifier	Accuracy	Precision	Recall
Net1 (1000, 1000, 1000)	46.66%	**30.44%**	**42.33%**
Net2 (2000, 1000, 1000)	46.31%	37.66%	42.44%
Net3 (2000, 2000, 1000)	**46.00%**	37.63%	42.68%
Net4 (2000, 2000, 2000)	47.57%	37.54%	43.88%

Bold values indicate the best value for each performance measure among different models

Table 4 Performance comparison under $FGSM^r$ attack proposed in [11]

Victim Classifier	Accuracy	Precision	Recall
Net1 (1000, 1000, 1000)	**62.15%**	**54.05%**	49.81%
Net2 (2000, 1000, 1000)	63.55%	54.91%	47.74%
Net3 (2000, 2000, 1000)	63.96%	54.82%	**47.14%**
Net4 (2000, 2000, 2000)	63.67%	55.52%	47.53%

Bold values indicate the best value for each performance measure among different models

Table 5 Performance comparison for three adversarial attacks under accuracy

Victim Classifier	GAN-based	$FGSM^r$	$FGSM^k$
Net1 (1000, 1000, 1000)	**46.66%**	62.15%	70.03%
Net2 (2000, 1000, 1000)	47.57%	63.67%	70.94%
Net3 (2000, 2000, 1000)	48.47%	66.40%	74.63%
Net4 (2000, 2000, 2000)	48.79%	67.77%	77.81%

Bold value indicates the best value for each performance measure among different models

Table 6 Evasion cost and the number of generated adversarial malware samples under three adversarial attacks

Attack	Cost	Adversarial samples
$FGSM^r$	41.44	3,040
GAN-based attack	37.10	2,134
Filter+$FGSM^r$	**34.10**	1,274

Bold value indicates the best value for each performance measure among different models

XGBoost achieve quite satisfactory accuracy. Both classifiers are applicable and can be implemented in real network environments.

5.3 Performance analyses under different adversarial attacks

While all classifiers achieve satisfactory performance, it is thus critical to evaluate their robustness against adversarial attacks. The adversarial attack, either targeted or non-targeted, aims at reducing the classifiers' accuracy remarkably. As this paper focuses on binary feature-based malware classification, we compare the proposed filter-based GAN attack with $FGSM^r$ [11]. While $FGSM^r$ works directly on binary features with fixed dimension, our proposed GAN-based attack is applicable on varied-length binary feature vectors. All models are accelerated by Adam while hyper parameter Z is set to be 15 representing the dimensions of the noise vector z. The generator of GAN designed is a 3-layer and configured with 180, 256 and 160 neurons in each layer, respectively. The discriminator is also a 3-layer neural network with 160, 256 and 1 neurons in each layer. The learning rate for both generator and discriminator is set as 0.001. The epoch is set as 100 and each model is trained 100 times on the training datasets. The attack performances under the above two algorithms, evaluated under 4 different network settings, are as shown in Tables 3 and 4, respectively.

Based on the results shown in Tables 3 and 4, following observations are made: (1) As adversary attack aims at defeating the victim classifier with accuracy decline, GAN-based attack outperforms $FGSM^r$ with 16.70% accuracy decline on average. To consolidate our observations, we took another adversarial attack $FGSM^k$ [10] for reference. As shown in Table 5, the GAN-based attack achieves 17.125% and 25.48% accuracy decline on average than $FGSM^r$ and $FGSM^k$, respectively. (2) When configured with varied number of neurons in each convolutional layer, the accuracy under all adversarial attacks only changed slightly. While convolutional layers are responsible for feature extraction, perturbations applied by adversarial attacks are preserved and forwarded to the final output layer for classification. (3) Both precision and recall under GAN-based attack are lower than other attacks. Therefore, the evasion ratio under GAN-based attack is higher than $FGSM^r$ and $FGSM^k$.

The GAN-based attack works by filtering generated adversarial samples while those that have the highest probability to successfully mislead the target model are preserved and enhanced. Therefore, we included $FGSM^r$ and $FGSM^r$ with filter (abbreviated as $Filter + FGSM^r$) attacks for comparison. First, we applied $FGSM^r$ to generate 3,040 adversarial samples derived from the original 15,200 training samples.

Table 7 Performance comparison under three conditions, including the original DNN without attack (Original-DNN), DNN when attacked by GAN with filter based attack (GAN-DNN) and DNN with adversarial training (Adv-DNN)

Networks	Original-DNN	GAN-DNN		Adv-DNN	
	Accuracy	Accuracy	Evasion Ratio	Accuracy	Evasion Ratio
Net1 (1000, 1000, 1000)	95.03%	46.66%	24.77%	87.44%	16.67%
Net2 (2000, 2000, 2000)	94.74%	47.57%	21.64%	87.07%	14.76%
Net3 (3000, 3000, 3000)	95.00%	48.47%	21.80%	88.60%	13.77%
Net4 (4000, 4000, 4000)	**95.31%**	48.79%	20.77%	**90.20%**	**9.47%**

Bold values indicate the best value for each performance measure among different models

Then, we apply $Filter + FGSM^r$ and GAN-based filter on those adversarial samples and output 2,134 and 1,274 samples, respectively. It is observed that almost 41.91% of the generated adversarial samples are chosen by GAN-based attack. While the number of adversarial samples are halved by GAN, the cost to generate adversarial samples also matters. As far as binary features are concerned, the attack cost is defined as the number of changed bits between the original clean sample and computed adversarial sample. One unit of cost is defined as the change from 0 to 1 in each feature vector. Based on the above definition, we compared three attacks under evasion cost. As shown in Table 6, $FGSM^r$ requires the highest cost to successfully craft an adversarial sample while $Filter + FGSM^r$ is the best with the lowest cost. All in all, changing 42 bits on average in a feature vector is trivial and thus acceptable for adversarial attackers.

5.4 Secure defense by adversarial training

We tested four DNNs with different number of neurons in the range of $[1000, 4000]$. Each DNN is evaluated under three conditions, including the original DNN without attack (abbreviated as Original-DNN), DNN when attacked by GAN with filter based attack (GAN-DNN) and DNN with adversarial training (abbreviated as Adv-DNN). The results are shown in Table 7.

It can be seen in Table 7 that adversarial training with samples generated by GAN with filter based attack algorithms secures our DNN classifier. The detection accuracy achieves more than 90.00% on average. The robustness of adversarial training based DNN classifier achieves at the cost of slight accuracy decline. This cost is affordable especially when evasion ratio drops below 10.0%. All in all, the GAN with filter based adversarial attack is not only effective in producing high quality adversarial samples, but also effective in enhancing the classifiers' robustness.

6 Discussions and limitations

First, the raw bit sequence-based DNNs for malware classification are popular. The proposed DNNs are capable of detecting packed malware and PDF malware with minor revision while the malicious codes are usually located in the scripting code section. However, Android malware is not exploited in this paper because their different file format (*apk* file).

Second, the proposed DNNs are also applicable to any binary features, including API calls and OpCode n-gram. The extracted feature vectors are required to be with fixed length. Retraining DNN is required while network architecture and parameter initialization are left unchanged.

Third, most of the existing adversarial attacks, including $dFGSM^k$ and $rFGSM^r$ [10], are proved successful. However, functionality-preserving attacks are not thoroughly analyzed. The biggest challenge is how to bridging the gap between the adversarial samples and the original programs on their malicious functionalities.

Fourth, this paper further proves that GAN is effective on adversarial malware detection. It is unquestionable that GAN's involvement in both adversarial machine learning and secure machine learning can be deeper and wider.

Last, a larger DNN with adversarial training is more robust than a shallow one, as shown in Table 7. However, the linear parts of a DNN are still problematic. To design a robust and secure DNN classifier, the linear parts should be changed in essence.

7 Conclusion

A GAN-based adversarial sample filtering mechanism is introduced in this paper to secure DNNs for malware classification. This paper is meaningful in the following two aspects. First, various of features are employed for malware classification. While static features are proved to be less accurate, dynamic features are with high cost and complexity. Thus, we proved that raw-bit-sequence-based DNN is effective and possible. Second, while GAN is introduced for data augmentation and adversarial sample sanitizing, this paper proves that GAN also succeeds in choosing high quality adversarial samples. Those high quality adversarial malware are used in adversarial training to further enhance the DNN's robustness.

The battle between adversarial attack and robust defense will last forever. In malware classification, the problem is even worse. First, new malware variants are generated with higher quantity and quality. Therefore, existing DNNs for malware classification need to be updated incrementally. Second, more powerful adversarial attacks will be published. Protecting all DNNs from adversarial attacks is challenging. Recent progress on DNN verification and compressed DNN may provide some possibilities.

Acknowledgements This work has been supported by the Open Foundation of Key Laboratory in Software Engineering of Yunnan Province under Grant Nos. 2020SE401, 2020SE306 and 2020SE305.

References

1. Filiol, E., Josse, S.: A statistical model for undecidable viral detection. J. Comput. Virol. Tech. **3**(2), 65–74 (2007). https://doi.org/10.1007/s11416-007-0041-5
2. Gavrilut, D., Cimpoesu, M., Anton, D., Ciortuz, L.: Malware detection using machine learning. In: International Multiconference on Computer Science & Information Technology, pp. 735–741. IEEE (2010). https://ieeexplore.ieee.org/document/5352759
3. Gibert, D., Mateu, C., Planes, J.: The rise of machine learning for detection and classification of malware: research developments, trends and challenges. J. Netw. Comput. Appl. **153**, 102536 (2020). https://doi.org/10.1016/j.jnca.2019.102526
4. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath B.S.: Malware images: visualization and automatic classification. In: VizSec 11 Proceedings of the 8th International Symposium on Visualization for Cyber Security, pp. 1–7. ACM (2011). https://doi.org/10.1145/2016904.2016908
5. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.: Malware detection by eating a whole exe (2017). arXiv:1710.09435
6. Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., Tracy, A., McLean, M., Nicholas, C.: An investigation of byte n-gram features for malware classification. J. Comput. Virol. Hacking Tech. **14**(1), 1–20 (2018). https://doi.org/10.1007/s11416-016-0283-1
7. Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G.: Adversarial malware binaries: evading deep learning for malware detection in executables. In: 2018 26th European Signal Processing Conference (EUSIPCO), pp. 533–537. IEEE (2019)
8. Suciu, O., Coull, S.E., Johns, J.: Exploring adversarial examples in malware detection. In: 2019 IEEE Security and Privacy Workshop (SPW), pp. 8–14. CEUR-WS (2019). https://doi.org/10.1109/SPW.2019.00015
9. Jin, G., Shen, S., Zhang, D., Dai, F., Zhang, Y.: APE-GAN: adversarial perturbation elimination with GAN. In: 2019 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp. 3842–3846. IEEE (2019). https://doi.org/10.1109/ICASSP.2019.8683044
10. Al-Dujaili, A., Huang, A., Hemberg, E., O'Reilly, U.M.: Adversarial deep learning for robust detection of binary encoded malware. In: 2018 IEEE Security and Privacy Workshops (SPW), pp. 76–82. IEEE (2018). https://doi.org/10.1109/SPW.2018.00020
11. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations (ICLR 2018), pp. 1–28 (2018)
12. Mercaldo, F., Santone, A.: Deep learning for image-based mobile malware detection. J. Comput. Virol. Hacking Tech. **16**(6), 1–15 (2020). https://doi.org/10.1007/s11416-019-00346-7
13. Aafer, Y., Du, W., Yin, H.: DroidAPIMiner: mining API-level features for robust malware detection in android. In: International Conference on Security and Privacy in Communication Systems, pp. 86–103. Springer (2013). https://doi.org/10.1007/978-3-319-04283-1_6
14. Jerlin, M.A., Marimuthu, K.: A new malware detection system using machine learning techniques for API call sequences. J. Appl. Secur. Res. **13**(1), 45–62 (2018)
15. Zhang, M., Duan, Y., Yin, H., Zhao, Z.: Semantics-aware android malware classification using weighted contextual API dependency graphs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1105–1116. ACM (2014). https://doi.org/10.1145/2660267.2660359
16. Hou, S., Ye, Y., Song, Y., Abdulhayoglu, M.: HinDroid: an intelligent android malware detection system based on structured heterogeneous information network. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1507–1515. ACM (2017). https://doi.org/10.1145/3097983.3098026
17. Kreuk, F., Barak, A., Aviv-Reuven, S., Baruch, M., Pinkas, B., Keshet, J.: Adversarial examples on discrete sequences for beating whole-binary malware detection (2018). arXiv:1802.04528v1
18. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations (ICLR 2015), pp. 1–11 (2015)
19. Hu, W., Tan, Y.: Generating adversarial malware examples for black-box attacks based on GAN (2017). arXiv:1702.05983
20. Chen, X., Li, C., Wang, D., Wen, S., Zhang, J., Nepal, S., Xiang, Y., Ren, K.: Android HIV: a study of repackaging malware for evading machine-learning detection. IEEE Trans. Inform. Forens. Secur. **15**, 987–1001 (2020)
21. Carlini, N., Wagner, D.: Adversarial examples are not easily detected: bypassing ten detection methods. In: Proceedings of the 10th ACM workshop on artificial intelligence and security, pp. 3–14. ACM (2017). https://doi.org/10.1145/3128572.3140444
22. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 372–387. IEEE (2016). https://doi.org/10.1109/EuroSP.2016.36
23. Wang, Q., Guo, W., Zhang, K., Ororbia, II., Alexander, G., Xing, X., Liu, X., Giles, C.L.: Adversary resistant deep neural networks with an application to malware detection. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1145–1153. ACM (2017). https://doi.org/10.1145/3097983.3098158
24. Mahmood, S., Keane, L., Lujo, B., Michael, K.R., Saurabh, S.: Optimization-guided binary diversification to mislead neural networks for malware detection 2019. arXiv:1912.09064
25. Pappas, V., Polychronakis, M., Keromytis, A.D.: Smashing the gadgets: hindering return-oriented programming using in-place code randomization. In: 2012 IEEE Symposium on Security and Privacy, pp. 601–615. IEEE (2012). https://doi.org/10.1109/SP.2012.41
26. Koo, H., Polychronakis, M.: Juggling the gadgets: binary-level code randomization using instruction displacement. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pp. 23–34. ACM (2016). https://doi.org/10.1145/2897845.2897863
27. Song, W., Li, X., Afroz, S., Garg, D., Kuznetsov, D., Yin, H.: Automatic generation of adversarial examples for interpreting malware classifiers (2020). arXiv:2003.03100
28. Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., Giacinto, G., Roli, F.: Yes, machine learning can

be more secure! A case study on android malware detection. IEEE Trans. Depend. Secure Comput. **16**(4), 711–724 (2019). https://doi.org/10.1109/TDSC.2017.2700270

29. Incer, I., Theodorides, M., Afroz, S., Wagner, D.: Adversarially robust malware detection using monotonic classification. In: The Fourth ACM International Workshop, pp. 54–63. ACM (2018). https://doi.org/10.1145/3180445.3180449
30. Maiorca, D., Biggio, B., Giacinto, G.: Towards adversarial malware detection: lessons learned from PDF-based attacks. ACM Comput. Surv. (CSUR) **52**(4), 1–36 (2019). https://doi.org/10.1145/3332184
31. Chen, L., Hou, S., Ye, Y., Xu, S.: DroidEye: fortifying security of learning-based classifier against adversarial android malware attacks. In: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp. 782–789. IEEE (2018). https://doi.org/10.1109/ASONAM.2018.8508284
32. Chen, L., Ye, Y.: SecMD: make machine learning more secure against adversarial malware attacks. In: AI 2017: Advances in Artificial Intelligence, pp. 76–89. Springer (2017). https://doi.org/10.1007/978-3-319-63004-5_7
33. Chen, L., Hou, S., Ye, Y.: SecureDroid: enhancing security of machine learning-based detection against adversarial android malware attacks. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 362–372. ACM (2017). https://doi.org/10.1145/3134600.3134636
34. Yang, W., Kong, D., Xie, T., Gunter, C.A.: Malware detection in adversarial settings: exploiting feature evolutions and confusions in android apps. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 288–302. ACM (2017). https://doi.org/10.1145/3134600.3134642
35. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. **7**(4), 2721–2744 (2006)