Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN

Weiwei Hu[1,2,3] and Ying Tan[1,2,3](✉)

[1] School of Intelligence Science and Technology, Peking University, Beijing 100871, China
{weiwei.hu,ytan}@pku.edu.cn
[2] Key Laboratory of Machine Perceptron (MOE), Peking University, Beijing 100871, China
[3] Institute for Artificial Intelligence, Peking University, Beijing 100871, China

Abstract. Machine learning has been used to detect new malware in recent years, while malware authors have strong motivation to attack such algorithms. Malware authors usually have no access to the detailed structures and parameters of the machine learning models used by malware detection systems, and therefore they can only perform black-box attacks. This paper proposes a generative adversarial network (GAN) based algorithm named MalGAN to generate adversarial malware examples, which are able to bypass black-box machine learning based detection models. MalGAN uses a substitute detector to fit the black-box malware detection system. A generative network is trained to minimize the generated adversarial examples' malicious probabilities predicted by the substitute detector. The superiority of MalGAN over traditional gradient based adversarial example generation algorithms is that MalGAN is able to decrease the detection rate to nearly zero and make the retraining based defensive method against adversarial examples hard to work.

Keywords: Malware detection · Adversarial examples · Generative adversarial network

1 Introduction

In recent years, many machine learning based algorithms have been proposed to detect malware, which extract features from programs and use a classifier to classify programs between benign programs and malware. For example, Schultz et al. proposed to use dynamic-link libraries (DLL), application programming interfaces (API) and strings as features for classification [24], while Kolter et al. used byte level n-gram as features [10,11].

Most researchers focused their efforts on improving the detection performance (e.g. true positive rate, accuracy and AUC) of such algorithms, but ignored the

This paper is the arXiv version of https://arxiv.org/abs/1702.05983.

robustness of these algorithms. Generally speaking, the propagation of malware will benefit malware authors. Therefore, malware authors have sufficient motivation to attack malware detection algorithms.

Many machine learning algorithms are very vulnerable to intentional attacks. Machine learning based malware detection algorithms cannot be used in real-world applications if they are easily to be bypassed by some adversarial techniques.

Recently, adversarial examples of deep learning models have attracted the attention of many researchers. Szegedy et al. added imperceptible perturbations to images to maximize a trained neural network's classification errors, making the network unable to classify the images correctly [25]. The examples after adding perturbations are called adversarial examples. Goodfellow et al. proposed a gradient based algorithm to generate adversarial examples [6]. Papernot et al. used the Jacobian matrix to determine which features to modify when generating adversarial examples [18]. The Jacobian matrix based approach is also a kind of gradient based algorithm.

Grosse et al. proposed to use the gradient based approach to generate adversarial Android malware examples [7]. The adversarial examples are used to fool a neural network based malware detection model. They assumed that attackers have full access to the parameters of the malware detection model. For different sizes of neural networks, the misclassification rates after adversarial crafting range from 35% to 84%.

In some cases, attackers have no access to the architecture and weights of the neural network to be attacked; the target model is a black box to attackers. Papernot et al. used a substitute neural network to fit the black-box neural network and then generated adversarial examples according to the substitute neural network [17]. They also used a substitute neural network to attack other machine learning algorithms such as logistic regression, support vector machines, decision trees and nearest neighbors [16]. Liu et al. performed black-box attacks without a substitute model [13], based on the principle that adversarial examples can transfer among different models [25].

Machine learning based malware detection algorithms are usually integrated into antivirus software or hosted on the cloud side, and therefore they are black-box systems to malware authors. It is hard for malware authors to know which classifier a malware detection system uses and the parameters of the classifier.

However, it is possible to figure out what features a malware detection algorithm uses by feeding some carefully designed test cases to the black-box algorithm. For example, if a malware detection algorithm uses static DLL or API features from the import directory table or the import lookup tables of PE programs [14], malware authors can manually modify some DLL or API names in the import directory table or the import lookup tables. They can modify a benign program's DLL or API names to malware's DLL or API names, and vice versa. If the detection results change after most of the modifications, they can judge that the malware detection algorithm uses DLL or API features. Therefore, in this paper we assume that malware authors are able to know what features a malware detection algorithm uses, but know nothing about the machine learning model.

Existing algorithms mainly use gradient information and hand-crafted rules to transform original samples into adversarial examples. This paper proposes a generative neural network based approach which takes original samples as inputs and outputs adversarial examples. The intrinsic non-linear structure of neural networks enables them to generate more complex and flexible adversarial examples to fool the target model.

The learning algorithm of our proposed model is inspired by generative adversarial networks (GAN) [5]. In GAN, a discriminative model is used to distinguish between generated samples and real samples, and a generative model is trained to make the discriminative model misclassify generated samples as real samples. GAN has shown good performance in generating realistic images [4,15].

The proposed model in this paper is named as MalGAN, which generates adversarial examples to attack black-box malware detection algorithms. A substitute detector is trained to fit the black-box malware detection algorithm, and a generative network is used to transform malware samples into adversarial examples. Experimental results show that most of the adversarial examples generated by MalGAN successfully bypass the detection algorithms and MalGAN is very flexible to fool further defensive methods of detection algorithms.

2 Architecture of MalGAN

2.1 Overview

The architecture of the proposed MalGAN is shown in Fig. 1.

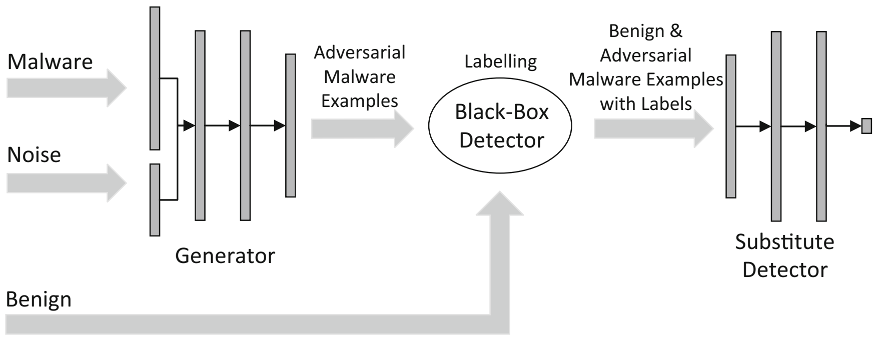
Malware
Noise
Generator
Benign
Adversarial Malware Examples
Labelling
Black-Box Detector
Benign & Adversarial Malware Examples with Labels
Substitute Detector

Fig. 1. The architecture of MalGAN.

The black-box detector is an external system which adopts machine learning based malware detection algorithms. We assume that the only thing malware authors know about the black-box detector is what kind of features it uses. Malware authors do not know what machine learning algorithm it uses and do not have access to the parameters of the trained model. Malware authors are able to get the detection results of their programs from the black-box detector. The whole model contains a generator and a substitute detector, which are both feed-forward neural networks. The generator and the substitute detector work together to attack a machine learning based black-box malware detector.

In this paper we only generate adversarial examples for binary features, because binary features are widely used by malware detection researchers and are able to result in high detection accuracy. Here we take API feature as an example to show how to represent a program. For malware detection on the Microsoft Windows operating systems, APIs are the most used features. If M APIs are used as features, an M-dimensional feature vector is constructed for a program. If the program calls the d-th API, the d-th feature value is set to 1, otherwise it is set to 0. For Android malware detection, additional features such as permissions from the manifest file of an APK are also used by many researchers [3,20,22]. In such case, if an Android program requires a permission, the corresponding feature value is set to 1.

The main difference between this model and existing algorithms is that the adversarial examples are dynamically generated according to the feedback of the black-box detector, while most existing algorithms use static gradient based approaches to generate adversarial examples.

The probability distribution of adversarial examples from MalGAN is determined by the weights of the generator. To make a machine learning algorithm effective, the samples in the training set and the test set should follow the same probability distribution or similar probability distributions. However, the generator can change the probability distribution of adversarial examples to make it far from the probability distribution of the black-box detector's training set. In this case the generator has sufficient opportunity to lead the black-box detector to misclassify malware as benign.

2.2 Generator

The generator is used to transform a malware feature vector into its adversarial version. It takes the concatenation of a malware feature vector $\boldsymbol{m}$ and a noise vector $\boldsymbol{z}$ as input. $\boldsymbol{m}$ is a M-dimensional binary vector, where M represents the number of features. Each element of $\boldsymbol{m}$ corresponds to the presence or absence of a feature. $\boldsymbol{z}$ is a Z-dimensional vector, where Z is a hyper-parameter. Each element of $\boldsymbol{z}$ is a random number sampled from a uniform distribution in the range $[0, 1)$. The effect of $\boldsymbol{z}$ is to allow the generator to generate diverse adversarial examples from a single malware feature vector.

The input vector is fed into a multi-layer feed-forward neural network with weights θ_g. The output layer of this network has M neurons and the activation function used by the last layer is sigmoid which restricts the output to the range

$(0, 1)$. The output of this network is denoted as $\boldsymbol{o}$. Since malware feature values are binary, binarization transformation is applied to $\boldsymbol{o}$ according to whether an element is greater than 0.5 or not, and this process produces a binary vector $\boldsymbol{o'}$.

When generating adversarial examples for binary malware features we only consider to add some irrelevant features to malware. Removing a feature from the original malware may crack it. For example, if the "WriteFile" API is removed from a program, the program is unable to perform normal writing function and the malware may crack. The non-zero features in the binary vector $\boldsymbol{o'}$ which have zero feature values in $\boldsymbol{m}$ act as the irrelevant features to be added to the original malware. The final generated adversarial example can be expressed as $\boldsymbol{m'} = \boldsymbol{m}|\boldsymbol{o'}$ where "|" is element-wise binary OR operation.

To make the adversarial example executable, malware authors need to add the irrelevant features to the source code of the original malware. For example, if a malware detection algorithm uses API features, malware authors should intentionally call the irrelevant APIs in the source code. Then the modified source code should be compiled into the final adversarial malware program. The adversarial malware will have the whole malicious functions of the original malware. The source code should be modified carefully, to make sure that adding the irrelevant features does not influence the existing functions of the original malware. Malware authors can also develop some automatic tools for adding irrelevant features, in order to generate a large number of adversarial malware examples.

For Android malware features extracted from the manifest files of APKs, it is more easy to insert the irrelevant features. For example, if an Android malware detection algorithm uses permission features, the irrelevant permissions can be easily inserted into the manifest file without influencing the original function of the malware.

$\boldsymbol{m'}$ is a binary vector, and therefore the gradients are unable to back propagate from the substitute detector to the generator. A smooth function G is defined to receive gradient information from the substitute detector, as shown in Formula 1.

$$G_{\theta_g}(\boldsymbol{m}, \boldsymbol{z}) = \max(\boldsymbol{m}, \boldsymbol{o}). \tag{1}$$

$\max(\cdot, \cdot)$ represents element-wise max operation. If an element of $\boldsymbol{m}$ has the value 1, the corresponding result of G is also 1, which is unable to back propagate the gradients. If an element of $\boldsymbol{m}$ has the value 0, the result of G is the neural network's real number output in the corresponding dimension, and gradient information is able to go through. It can be seen that $\boldsymbol{m'}$ is actually the binarization transformed version of $G_{\theta_g}(\boldsymbol{m}, \boldsymbol{z})$.

2.3 Substitute Detector

Since malware authors know nothing about the detailed structure of the black-box detector, the substitute detector is used to fit the black-box detector and provides gradient information to train the generator.

The substitute detector is a multi-layer feed-forward neural network with weights θ_d which takes a program feature vector $\boldsymbol{x}$ as input. It classifies the program between benign program and malware. We denote the predicted probability that $\boldsymbol{x}$ is malware as $D_{\theta_d}(\boldsymbol{x})$.

The training data of the substitute detector consist of adversarial malware examples from the generator, and benign programs from an additional benign dataset collected by malware authors. The ground-truth labels of the training data are not used to train the substitute detector. The goal of the substitute detector is to fit the black-box detector. The black-box detector will detect this training data first and output whether a program is benign or malware. The predicted labels from the black-box detector are used by the substitute detector.

3 Training MalGAN

To train MalGAN malware authors should collect a malware dataset and a benign dataset first.

The loss function of the substitute detector is defined in Formula 2.

$$\begin{aligned} L_D = &- \mathbb{E}_{\boldsymbol{x} \in BB_{Benign}} \log\left(1 - D_{\theta_d}(\boldsymbol{x})\right) \\ &- \mathbb{E}_{\boldsymbol{x} \in BB_{Malware}} \log D_{\theta_d}(\boldsymbol{x}). \end{aligned} \tag{2}$$

BB_{Benign} is the set of programs that are recognized as benign by the black-box detector, and $BB_{Malware}$ is the set of programs that are detected as malware by the black-box detector.

To train the substitute detector, L_D should be minimized with respect to the weights of the substitute detector.

The loss function of the generator is defined in Formula 3.

$$L_G = \mathbb{E}_{\boldsymbol{m} \in S_{Malware}, \boldsymbol{z} \sim \boldsymbol{p}_{\text{uniform}[0,1)}} \log D_{\theta_d}\left(G_{\theta_g}(\boldsymbol{m}, \boldsymbol{z})\right). \tag{3}$$

$S_{Malware}$ is the actual malware dataset, not the malware set labelled by the black-box detector. L_G is minimized with respect to the weights of the generator.

Minimizing L_G will reduce the predicted malicious probability of malware and push the substitute detector to recognize malware as benign. Since the substitute detector tries to fit the black-box detector, the training of the generator will further fool the black-box detector.

The whole process of training MalGAN is shown in Algorithm 1.

In line 2 and line 4, different sizes of minibatches are used for malware and benign programs. The ratio of $\boldsymbol{M}$'s size to $\boldsymbol{B}$'s size is the same as the ratio of the malware dataset's size to the benign dataset's size.

Algorithm 1. The Training Process of MalGAN

```
1: while not converging do
2:     Sample a minibatch of malware $\boldsymbol{M}$
3:     Generate adversarial examples $\boldsymbol{M'}$ from the generator for $\boldsymbol{M}$
4:     Sample a minibatch of benign programs $\boldsymbol{B}$
5:     Label $\boldsymbol{M'}$ and $\boldsymbol{B}$ using the black-box detector
6:     Update the substitute detector's weights $\theta_d$ by descending along the gradient
       $\nabla_{\theta_d} L_D$
7:     Update the generator's weights $\theta_g$ by descending along the gradient $\nabla_{\theta_g} L_G$
8: end while
```

4 Experiments

4.1 Experimental Setup

The main dataset used in this paper was crawled from a program sharing website[1]. We downloaded 180 thousand PC programs in Microsoft Windows operating systems from this website and about 70% of them are malware. API features are used for this dataset. An 160-dimensional binary feature vector is constructed for each program, based on 160 system level APIs.

We will also report the results on the Drebin Android malware dataset[2] when comparing MalGAN with the algorithm proposed by Grosse et al., since Grosse et al. used this Android dataset [7]. The Drebin dataset contains 8 kinds of features, such as hardware components, requested permissions and API calls. After removing the features which appear less than 5 times in the dataset, we got 44942 features and used these features to train MalGAN. However, the dataset only contains 5560 malware samples, which is too small for normal deep learning applications. Therefore, we only used this dataset as a supplemental dataset for comparison. Most experiments and analyses were conducted on the crawled 180 thousand programs.

In order to validate the transferability of adversarial examples generated by MalGAN, we tried several different machine learning algorithms for the black-box detector. The used classifiers include random forest (RF), logistic regression (LR), decision trees (DT), support vector machines (SVM), multi-layer perceptron (MLP), and a voting based ensemble of these classifiers (VOTE).

We adopted two ways to split the dataset. The first splitting way regards 80% of the dataset as the training set and the remaining 20% as the test set. MalGAN and the black-box detector share the same training set. MalGAN further picks out 25% of the training data as the validation set and uses the remaining training data to train the neural networks. Some black-box classifiers such as MLP also need a validation set for early stopping. The validation set of MalGAN cannot be used for the black-box detector since malware authors and antivirus vendors do

[1] https://malwr.com/.
[2] https://www.sec.cs.tu-bs.de/~danarp/drebin/index.html.

not communicate on how to split dataset. Splitting validation set for the black-box detector should be independent of MalGAN; MalGAN and the black-box detector should use different random seeds to pick out the validation data.

The second splitting way picks out 40% of the dataset as the training set for MalGAN, picks out another 40% of the dataset as the training set for the black-box detector, and uses the remaining 20% of the dataset as the test set.

In real-world scenes the training data collected by the malware authors and the antivirus vendors cannot be the same. However, their training data will overlap with each other if they collect data from public sources. In this case the actual performance of MalGAN will be between the performances of the two splitting ways.

Adam [9] was chosen as the optimizer. We tuned the hyper-parameters on the validation set. For the dataset with 180 thousand programs, 10 was chosen as the dimension of the noise vector $\boldsymbol{z}$. The generator's layer size was set to 170-256-160, the substitute detector's layer size was set to 160-256-1, and the learning rate 0.001 was used for both the generator and the substitute detector. For the Drebin dataset, we used the same network structures as Grosse et al.[7]. The maximum number of epochs to train MalGAN was set to 100. The epoch with the lowest detection rate on the validation set is finally chosen to test the performance of MalGAN.

4.2 Experimental Results

We first analyze the case where MalGAN and the black-box detector use the same training set. For malware detection, the true positive rate (TPR) means the detection rate of malware. After adversarial attacks, the reduction in TPR can reflect how many malware samples successfully bypass the detection algorithm. TPR on the training set and the test set of original samples and adversarial examples is shown in Table 1. The datasets with 180 thousand programs is used here.

Table 1. True positive rate (in percentage) on original samples and adversarial examples when MalGAN and the black-box detector are trained on the same training set. "Adver." represents adversarial examples.

	Training set		Test set	
	Original	Adver.	Original	Adver.
RF	97.62	0.20	95.38	0.19
LR	92.20	0.00	92.27	0.00
DT	97.89	0.16	93.98	0.16
SVM	93.11	0.00	93.13	0.00
MLP	95.11	0.00	94.89	0.00
VOTE	97.23	0.00	95.64	0.00

For random forest and decision trees, the TPRs on adversarial examples range from 0.16% to 0.20% for both the training set and the test set, while the TPRs on the original samples are all greater than 93%. When using other classifiers as the black-box detector, MalGAN is able to decrease the TPR on generated adversarial examples to zero for both the training set and the test set. That is to say, for all of the backend classifiers, the black-box detector can hardly detect any malware generated by the generator. The proposed model has successfully learned to bypass these machine learning based malware detection algorithms.

The structures of logistic regression and support vector machines are very similar to neural networks and MLP is actually a neural network. Therefore, the substitute detector is able to fit them with a very high accuracy. This is why MalGAN can achieve zero TPR for these classifiers. While random forest and decision trees have quite different structures from neural networks so that MalGAN results in non-zero TPRs. The TPRs of random forest and decision trees on adversarial examples are still quite small, which means the neural network has enough capacity to represent other models with quite different structures. The voting of these algorithms also achieves zero TPR. We can conclude that the classifiers with similar structures to neural networks are in the majority during voting.

The convergence curve of TPR on the training set and the validation set during the training process of MalGAN is shown in Fig. 2. The black-box detector used here is random forest, since random forest performs very well in Table 1.

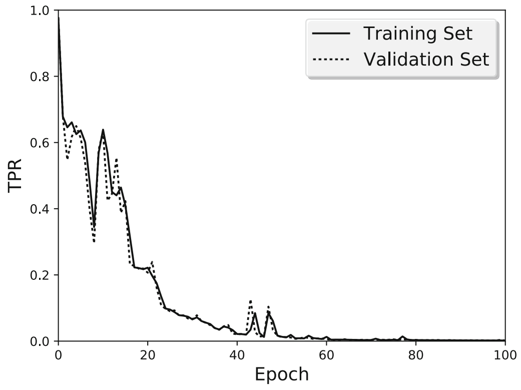

Fig. 2. The change of the true positive rate on the training set and the validation set over time.

TPR converges to about zero near the 40th epoch, but the convergence curve is a bit shaking, not a smooth one. This curve reflects the fact that the training of GAN is usually unstable. How to stabilize the training of GAN has attracted the attention of many researchers [1,21,23].

Now we will analyze the results when MalGAN and the black-box detector are trained on different training sets. Fitting the black-box detector trained on a different dataset is more difficult for the substitute detector. The experimental results are shown in Table 2.

Table 2. True positive rate (in percentage) on original samples and adversarial examples when MalGAN and the black-box detector are trained on different training sets. "Adver." represents adversarial examples.

	Training set		Test set	
	Original	Adver.	Original	Adver.
RF	95.10	0.71	94.95	0.80
LR	91.58	0.00	91.81	0.01
DT	91.92	2.18	91.97	2.11
SVM	92.50	0.00	92.78	0.00
MLP	94.32	0.00	94.40	0.00
VOTE	94.30	0.00	94.45	0.00

For SVM, MLP and VOTE, TPR reaches zero, and TPR of LR is nearly zero. These results are very similar to Table 1. TPRs of random forest and decision trees on adversarial examples become higher compared with the case where MalGAN and the black-box detector use the same training data. For decision trees the TPRs rise to 2.18% and 2.11% on the training set and the test set respectively. However, 2% is still a very small number and the black-box detector will still miss to detect most of the adversarial malware examples. It can be concluded that MalGAN is still able to fool the black-box detector even trained on a different training set.

4.3 Comparison with the Gradient Based Algorithm to Generate Adversarial Examples

Existing algorithms of generating adversarial examples are mainly for images. The difference between image and malware is that image features are continuous while malware features are binary.

Grosse et al. modified the traditional gradient based algorithm to generate binary adversarial malware examples [7]. They did not regard the malware detection algorithm as a black-box system and assumed that malware authors have full access to the architecture and the weights of the neural network based malware detection model. The misclassification rates of adversarial examples range from 35% to 84% under different hyper-parameters.

We applied MalGAN on the Drebin dataset used by Grosse et al. with a malware ratio of 0.5 to attack a black-box random forest. The TPRs on the test set are 5.63% and 6.87% respectively when MalGAN and random forest are trained on the same training set and on different training sets. MalGAN is able to make more malware undetected than the gradient based approach. This gradient based approach is under the white-box assumption, while MalGAN produces better results with a harder black-box assumption. In the following experiments we will continue to use the dataset with 180 thousand programs since it has much more malware examples than the Drebin dataset.

The algorithm proposed by Grosse et al. uses an iterative approach to generate adversarial malware examples. At each iteration the algorithm finds the feature with the maximum likelihood to change the malware's label from malware to benign. The algorithm modifies one feature at each iteration, until the malware is successfully classified as a benign program or there are no features available to be modified.

We tried to migrate this algorithm to attack a random forest based black-box detection algorithm. A substitute neural network is trained to fit the black-box random forest. Adversarial malware examples are generated based on the gradient information of the substitute neural network.

TPR on the adversarial examples over the iterative process is shown in Fig. 3. Please note that at each iteration not all of the malware samples are modified. If a malware sample has already been classified as a benign program at previous iterations or there are no modifiable features, the algorithm will do nothing on the malware sample at this iteration.

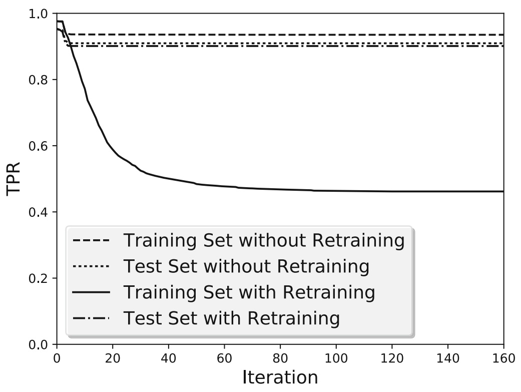

Fig. 3. True positive rate on the adversarial examples over the iterative process when using the algorithm proposed by Grosse et al..

On the training set and the test set, TPR converges to 93.52% and 90.96% respectively. In this case the black-box random forest is able to detect most of the adversarial examples. The substitute neural network is trained on the original training set, while after several iterations the probability distribution of adversarial examples will become quite different from the probability distribution of the original training set. Therefore, the substitute neural network cannot approximate the black-box random forest well on the adversarial examples. In this case the adversarial examples generated from the substitute neural network are unable to fool the black-box random forest.

In order to fit the black-box random forest more accurately on the adversarial examples, we tried to retrain the substitute neural network on the adversarial examples. At each iteration, the current generated adversarial examples from the whole training set are used to retrain the substitute neural network. As shown in Fig. 3, the retraining approach make TPR converge to 46.18% on the training set, which means the black-box random forest can still detect about half of the adversarial examples. However, the retrained model is unable to generalize to the test set, since the TPR on the test set converges to 90.12%. The odd probability distribution of these adversarial examples limits the generalization ability of the substitute neural network.

MalGAN uses a generative network to transform original samples into adversarial samples. The neural network has enough representation ability to perform complex transformations, making MalGAN able to result in very low TPRs on both the training set and the test set. While the representation ability of the gradient based approach is too limited to generate high-quality adversarial examples.

4.4 Retraining the Black-Box Detector

Several defensive algorithms have been proposed to deal with adversarial examples. Gu et al. proposed to use auto-encoders to map adversarial samples to clean input data [8]. An algorithm named defensive distillation was proposed by Papernot et al. to weaken the effectiveness of adversarial perturbations [19]. Li et al. found that adversarial retraining can boost the robustness of machine learning algorithms [12]. Chen et al. compared these defensive algorithms and concluded that retraining is a very effective way to defend against adversarial examples, and is robust even against repeated attacks [2].

In this section we will analyze the performance of MalGAN under the retraining based defensive approach. If antivirus vendors collect enough adversarial malware examples, the can retrain the black-box detector on these adversarial examples in order to learn their patterns and detect them. Here we only use random forest as the black-box detector due to its good performance. After retraining the black-box detector, it is able to detect all adversarial examples, as shown in the middle column of Table 3.

Table 3. True positive rate (in percentage) on the adversarial examples after the black-box detector is retrained.

	Before retraining MalGAN	After retraining MalGAN
Training set	100	0
Test set	100	0

However, once antivirus vendors release the updated black-box detector publicly, malware authors will be able to get a copy of it and retrain MalGAN to attack the new black-box detector. After this process the black-box detector can hardly detect any malware again, as shown in the last column of Table 3. We found that reducing TPR from 100% to 0% can be done within one epoch during retraining MalGAN. We alternated retraining the black-box detector and retraining MalGAN for ten times. The results are the same as Table 3 for the ten times.

To retrain the black-box detector antivirus vendors have to collect enough adversarial examples. It is a long process to collect a large number of malware samples and label them. Adversarial malware examples have enough time to propagate before the black-box detector is retrained and updated. Once the black-box detector is updated, malware authors will attack it immediately by retraining MalGAN and our experiments showed that retraining MalGAN takes much less time than the first-time training. After retraining MalGAN, new adversarial examples remain undetected. This dynamic adversarial process lands antivirus vendors in a passive position. Machine learning based malware detection algorithms can hardly work in this case.

5 Conclusions

This paper proposed a novel algorithm named MalGAN to generate adversarial examples from a machine learning based black-box malware detector. A neural network based substitute detector is used to fit the black-box detector. A generator is trained to generate adversarial examples which are able to fool the substitute detector. Experimental results showed that the generated adversarial examples are able to effectively bypass the black-box detector.

Adversarial examples' probability distribution is controlled by the weights of the generator. Malware authors are able to frequently change the probability distribution by retraining MalGAN, making the black-box detector cannot keep up with it, and unable to learn stable patterns from it. Once the black-box detector is updated malware authors can immediately crack it. This process making machine learning based malware detection algorithms unable to work.

Acknowledgment. This work is supported by the Science and Technology Innovation 2030 - 'New Generation Artificial Intelligence' Major Project (Grant Nos.: 2018AAA0100302), and partially supported by National Natural Science Foundation of China (Grant No. 62076010 and 62276008).

References

1. Arjovsky, M., Bottou, L.: Towards principled methods for training generative adversarial networks. In: NIPS 2016 Workshop on Adversarial Training. In review for ICLR, vol. 2016 (2017)
2. Chen, X., Li, B., Vorobeychik, Y.: Evaluation of defensive methods for DNNs against multiple adversarial evasion models (2016). https://openreview.net/forum?id=ByToKu9ll

3. Daniel, A., Michael, S., Malte, H., Hugo, G., Konrad, R.: Drebin: efficient and explainable detection of android malware in your pocket. In: Proceedings of 21th Annual Network and Distributed System Security Symposium (NDSS) (2014)
4. Denton, E.L., Chintala, S., Fergus, R., et al.: Deep generative image models using a Laplacian pyramid of adversarial networks. In: Advances in Neural Information Processing Systems, pp. 1486–1494 (2015)
5. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
6. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
7. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435 (2016)
8. Gu, S., Rigazio, L.: Towards deep neural network architectures robust to adversarial examples. arXiv preprint arXiv:1412.5068 (2014)
9. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
10. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. **7**, 2721–2744 (2006)
11. Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 470–478. ACM (2004)
12. Li, B., Vorobeychik, Y., Chen, X.: A general retraining framework for scalable adversarial classification. arXiv preprint arXiv:1604.02606 (2016)
13. Liu, Y., Chen, X., Liu, C., Song, D.: Delving into transferable adversarial examples and black-box attacks. arXiv preprint arXiv:1611.02770 (2016)
14. Microsoft: Microsoft portable executable and common object file format specification (2013). https://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/pecoff_v83.docx
15. Mirza, M., Osindero, S.: Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 (2014)
16. Papernot, N., McDaniel, P., Goodfellow, I.: Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv preprint arXiv:1605.07277 (2016)
17. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against deep learning systems using adversarial examples. arXiv preprint arXiv:1602.02697 (2016)
18. 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)
19. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 582–597. IEEE (2016)
20. Peiravian, N., Zhu, X.: Machine learning for android malware detection using permission and API calls. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 300–305. IEEE (2013)
21. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)

22. Sahs, J., Khan, L.: A machine learning approach to android malware detection. In: 2012 European Intelligence and Security Informatics Conference (EISIC), pp. 141–147. IEEE (2012)
23. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training GANs. In: Advances in Neural Information Processing Systems, pp. 2226–2234 (2016)
24. Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: 2001 IEEE Symposium on Security and Privacy, 2001. S&P 2001. Proceedings, pp. 38–49. IEEE (2001)
25. Szegedy, C., et al.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)