Adversarial Examples Against Visualized Images-based Malware Classification Systems

Bao Ngoc Vi, Huu Noi Nguyen, Cao Truong Tran, Ngoc Tran Nguyen
Le Quy Don Technical University
Hanoi, Vietnam
Email: {ngocvb, noi.nguyen, truongct,ngoctn}@lqdtu.edu.vn

***Abstract*—As the threat of malicious software (malware) becomes urgently serious, automatic malware classifier techniques have received increasing attention recently, where the deep learning (DL) based visualization classifiers plays a significant role. However, this leads to a fundamental problem whether such classifiers can be robust enough against various potential attacks. Even though DL shows superiority to conventional ones in malware classifiers in terms of high efficiency and accuracy, this paper demonstrates that such DL-based malware classifiers are vulnerable to adversarial attacks. We propose the first adversarial attack framework based on the gradient descent method. By introducing perturbations on resources' part of PE files, DL-based malware classifiers completely fail. The experimental results on the Malimg dataset show that a small interference can achieve success attack rate when challenging convolutional neural network malware classifers.**

***Index Terms*—malware classification, adversarial attack, convolution neural network,**

I. Introduction

Malware denotes a particular type of programs that perform malicious tasks and illegal controls on computer systems by breaking software processes to obtain the unauthorized access,interrupt normal operations and steal information on computers or mobile devices. There is a variety kinds of malwares including viruses, Trojans, worms, backdoors, rootkits, spyware, ransomware and panic software, etc. [19].

Previous work on malware classification can be broadly classified into two categories: non-machine learning methods and machine learning-based methods.

Non-Machine Learning Methods: In the past, malware was detected using static or dynamic signature-based techniques. Static analysis uses syntax or structural properties of the program in order to detect malware even before the program under inspection executes [13]. However, malware developers use various encryption, polymorphism and obfuscation techniques to overcome these detection algorithms. In the dynamic approach, malware is executed in a virtual environment and its behavior is analyzed in order to detect harmful actions during or after the program execution. Although dynamic analysis of malware is a promising approach, it is still very complex and time consuming [22].The major drawback of classical signature-based detection is that it is not scalable and its effectiveness can be undermined with the growing variants of malware. Therefore, another approach which is based on intelligent machine learning algorithms is investigated.

Machine Learning-Based Methods: In order to address the limitations of the aforementioned methods and inspired by the fact that variants of malware families typically share similar behavior patterns, anti-malware organizations started to develop more sophisticated classification methods based on data mining and machine learning techniques. These techniques use different feature extraction (i.e. data representation) methods to build more intelligent malware detection systems [11], [24]. Nataraj et al. [22] proposed a strategy to represent a malware as a grayscale image and then use GIST to compute texture features. Next, the grayscale malware images are classified using k-nearest neighbor algorithm. The major drawback of this method is that they use shallow learning techniques which are not very scalable with the growing number of malware samples and also requires to engineer the feature representation by hand. In order to tackle these problems, deep learning architectures that are robust and more general in nature are developed. For example, Drew et al. [7] [6] introduced malware classification using modern gene sequence classification tool. Ahmadi et al. [2] and trained a classifier based on the XGBoost technique. A convolutional neural network is also applied for malware classification [14], they proposed a deep convolutional neural network (CNN) architecture for malware classification. They first convert malware samples to grayscale images and then train a CNN for classification.

However, both DNNs and machine learning models lack robustness to adversarially crafted inputs known as adversarial examples. These inputs are derived from legitimate inputs by adding carefully chosen perturbations that force models to output erroneous predictions. Adversarial examples were originally proposed by Szegedy et al. [25] in 2014. Since the discriminative model is constructed by the dataset without adversarial examples, the classification accuracy can be incredibly reduced due to adversarial examples. A large number of adversarial examples construction methods [25], [9], [23], [5], [21] and related adversarial defence techniques [27], [18], [18], [4], [16] were proposed.

Our contributions are as follows:

- A new method to generating adversarial examples which can fool malware classifier was proposed.
- To evaluate our attack method, a simple CNN architecture is proposed that can classify malware with high accuracy on Malimg dataset.

- Adversarial training to defense again malware adversarial attack is also investigated.

The rest of this paper is organized as follows: Section II details the related work of malware visualization classification methods associated with the adversarial example techniques. Our malware adversarial attack method is introduced in Section III. We show the experimental designs and results in Section IV, we also gives a defensive strategy against adversarial attack in this section . Finally, we make a conclusion and provide some future research directions in section V.

II. Related Work

A. Malware Visualization

1) PE file overview: PE (Portable Executable) is Win32's own format [20]. Most of the executable files on Win32 are in PE format (except VxDs and 16bit DLL files).

Why need to understand the PE file structure? To be able

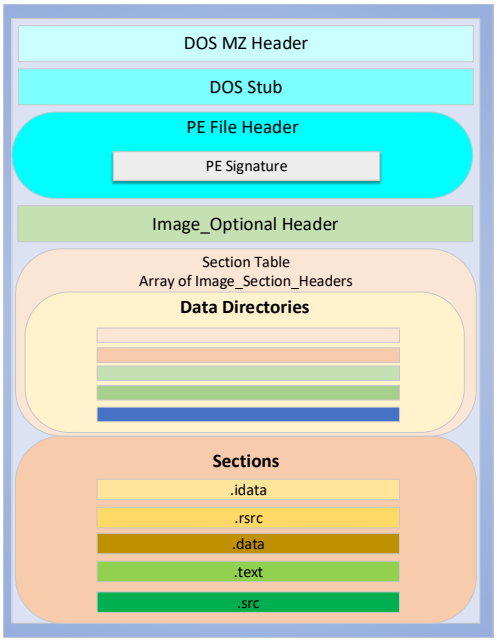

Fig. 1. The PE file structure.

to execute on a computer, the PE file content is divided into components and has a close relationship with each other. Understanding the PE structure will help us understand the execution mechanism of a program, from organizing to loading to memory, using resources,....

Moreover, when we want to modify a file, such as adding some code, editing some components but still want the program to execute normally. Therefore, it is necessary to understand the PE file structure, the relationship between the components in the file to be able to quickly change the file and satisfy the requirements.

PE Structure can consist of multiple sections, of which at least two sections: data and code.

Some commonly used sections are found in programs:

Executable	Description
.text	Contains the executable code
.rdata	Holds read-only data that is globally accesible within the program
.data	Stores global data accessed throughout the program
.idata	Sometimes present and stores the import function information; if this section is not present, the import function information is stored in .rdata section
.edata	Sometimes present and stores the export function information; if this section is not present, the export function information is stored in .rdata section
.pdata	Present only in 64-bit executatbles and stores exception-handling information
.rsrc	Stores resources needed by the executatble
.reloc	Contains information for relocation of library files

TABLE I
PE COMMON SECTIONS

1) The Executable Code Section, named .text (Microsoft) is CODE (Borland).
2) Data Sections, with names like .data, .rdata hoc .bss (Microsoft) or DATA (Borland)
3) Resources Section, named .rsrc
4) Export Data Section, named .edata
5) Import Data Section. named .idata
6) Debug Information Section, named .debug

If we inject to .text section, the functionality of the PE file may change. Other parts, such as resources, their changes will not affect to the functionality of the PE file.

In Table II-A1 we see sections of a PE file for a Windows Executable [20].

2) PE file to image: A PE file can be read as a vector of 8-bit unsigned integers and then organized into a 2D array [22] [1].

Each 8-bit value will be in interval [0, 255]. Therefore, from this two-dimensional array, we can convert into gray scale image, with 0-black, 255-white color values, respectively.

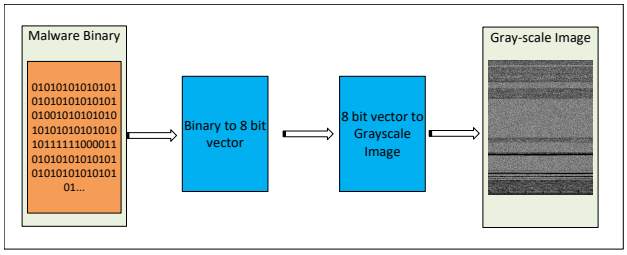

Fig. 2. PE file to image visualization.

The width of each photo depends on the size of the PE file and is fixed. The height of the image is calculated according to the size of the file and the width of the image.

In the table II-A2, is the corresponding width of the image with the corresponding file size [22].

For example, with a binary file, we can split into the following sections: .text, .rsrc, .reloc.

The .text section contains the executable code. From the figure 2, we can see that the first part of the .text section contains the code whose texture is fine grained. Next, the .rsrc

TABLE II
IMAGE WIDTH FOR DIFFERENT FILE SIZE.

PE file size (KB)	Image width
¡10	32
10 - 30	64
30 - 60	128
60 - 100	256
100 - 200	384
200 - 500	512
500 - 1000	768
⩾1000	1024

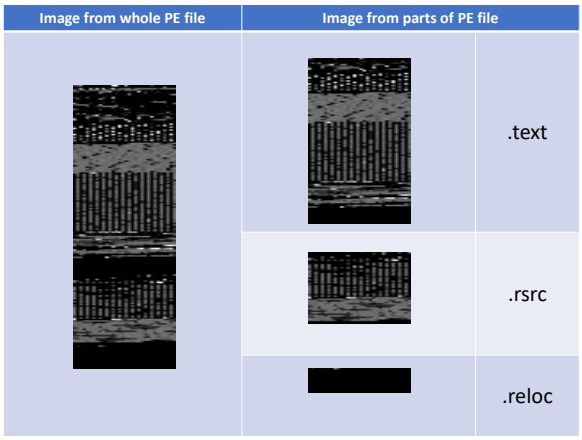

Fig. 3. Image segmentation by PE section.

section which contains all the resources of the module. These may also include icons that an application may use. Finally, the .reloc section is used for image storing and contains entries for all base relocation in the image.

3) PE file injection: For PE files in general and malware file in particular, the .text (the PE part stores compiled code) will determine the functionalities of those files. The resource used to store information such as icons, images, strings, languages will not affect the function of the file. Therefore, we can process the PE file, change the values of resource and the functionalities of PE file are kept.

Steps to alter the resource:

- Load the data of the PE file;
- Retrieve the data which contains resources. In this data section, we will consider two values, offset, indicating the starting position of resource and length, indicating the length of the data array;
- Convert the byte values of resources to 0 or to random values rather than original values;
- Write data back to the PE file.

Before and after replacement, we conducted to test by scanning PE file using an engine ([26]). Both files are considered malware.

B. Adversarial examples in Deep Neural Networks

Adversarial examples work for crafting subtle perturbations on original datasets for image classification problems and are able to fool the state-of-the-art deep neural networks (DNN) with high probability. In general, transferability and robustness are two key features for adversarial examples. The transferability refers to the degree to which a particular adversarial example attack can be transferred to function as other types of adversarial examples for different attacks. The robustness represents the ability to withstand or overcome adversarial example attacks. Prior work can be classified into untargeted and targeted adversarial examples. Un-targeted adversarial examples can cause the classifier to produce any incorrect output without specifying a predicted category, while targeted adversarial examples cause the classifier to produce a specific incorrect output. In the past three years, various adversarial example crafting methods were proposed, such as L-BFGS [25], FGSM [9], JSMA [23], C&Ws attack [5], DeepFool [21] as well as Generative Adversarial Network (GAN) [8]. Among them, researches have shown that FGSM [9] are the most popular choices for generating adversarial examples [28], [3]. We hence proposed a FGSM-based method in this paper.

C. Adversarial Malware Examples

Adversarial malware examples are used to bypass the detection of malicious codes. Gross et al. [10] used adversarial examples to interfere with the binary features of Android malware. They aimed to attack DNN-based detection for Android malware and retain malicious features in the Apps. But this kind of attack is only adapted to some Android malware samples processed by binary features. As for the end-to-end static code detection techniques using convolutional neural networks, F. Kreuk et al. [15] used adversarial examples to extract the feature information from binary code and disguised malware as a benign sample by injecting a small sequence of bytes (payload) in the binary file. This method not only retains the original function of malware but also achieves the purpose of deceiving global binary malware detector. However, it is extremely sensitive to discrete input dataset such as executable bytes. Minor modifications to the bytes of the file may lead to significant changes in its functionality and validity. Hu et al. [12] proposed a method of generating sequential adversarial examples by improving the robustness of its adversarial pertaining process, which can be used to attack the sequential API features-based malware in RNN detection system. The drawbacks of this method are time-consuming and large overhead in the whole processing. Different from the existing studies using adversarial examples to escape the malware detection, Liu et al. [17] proposed the method using adversarial examples to attack ML-based visualization detectors, named ATMPA. The ATMPA uses the gradient-based FGSM method and L-norm based C&Ws attack method to generate adversarial examples on the converted image dataset. Our proposed method also use adversarial examples to attack visualization classifications, but instead of generate adversarial examples on the converted image dataset, our method genarate adversarial examples on PE file.

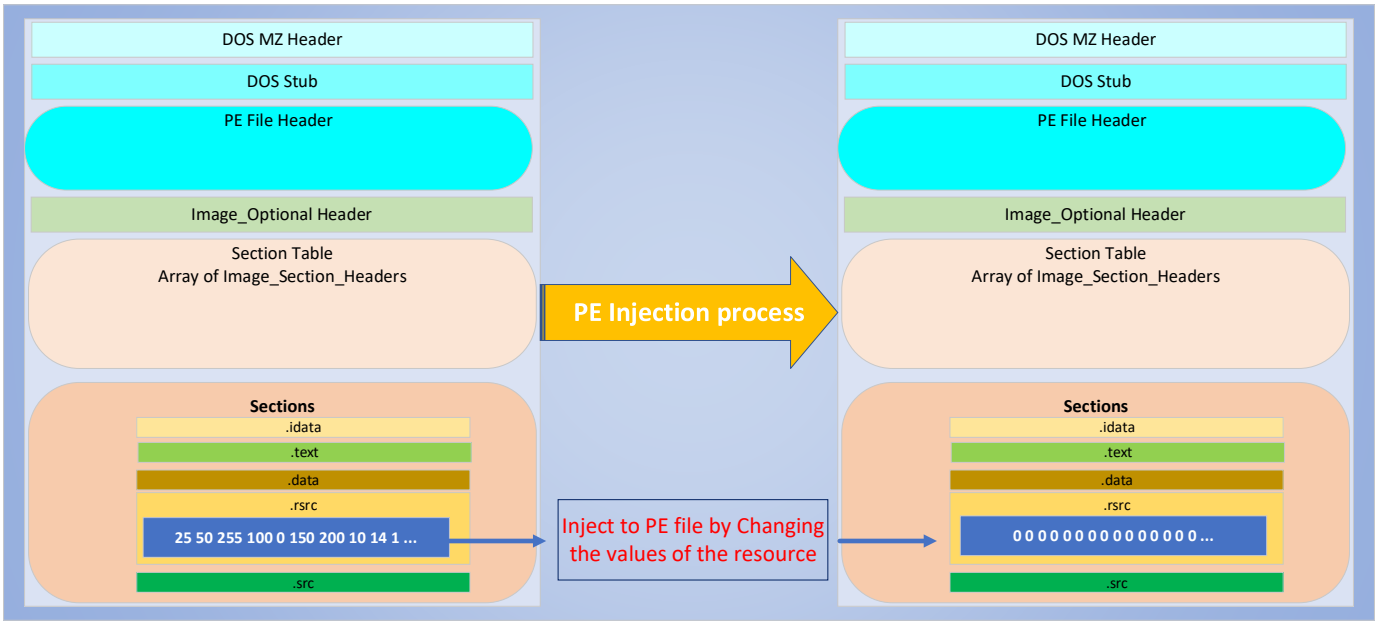

Fig. 4. Inject the resource area of the PE file.

III. The Proposed Method

In this section, we introduce a novel algorithm to generate adversarial malware examples to fool malware classifier f based on FGSM [9]. FGSM is a fast method to generate adversarial examples [9] and selects a perturbation parameter by differentiating this cost function with respect to the input itself. Along the direction of the gradients sign at each pixel point, FGSM only performs one step gradient update. The perturbation can be expressed as:

$$\sigma = \epsilon * \nabla_x J_\theta(x, y_{true}) \tag{1}$$

where ϵ is a small scalar value that restricts the magnitude of the perturbation and represents the distortion between adversarial examples and original sample. $sign()$ denotes the sign function,$\nabla_x J_\theta(x, y_{true})$ computes the gradient of the cost function J around the current value x of the model parameters . y_{true} is the label of x.

Then, the adversarial is computed by

$$x^* = x + \sigma \tag{2}$$

However, with malware classifier, x is a malware image (resized image), generating adversarial example by Eq.2 due to crack PE files.

As shown in section ..., altering byte values of resources do not influence the function of PE files. Therefore, our approach for generating adversarial malware examples is adding perturbation to the pixels which corresponding with resource part of PE files instead of all pixels.We assume that all malware images are reshaped using Nearest-neighbor interpolation. Therefore, the value of pixel (i, j) in 128x128 malware image is equal to the value of pixel $([i * w/128], [j * h/128])$ in the original malware image with size wxh. Then, we can define what pixels in malware images can be added perturbation.

Let x_{org} and x^*_{org} denote original malware image and adversarial example of original malware image respectively, S be set of pixels in x_{org} which represent the byte values of resources and σ be perturbation x gained by Eq.1.

Algorithm 2 illustrates the pseudo-code of Adding perturbation algorithm.

Algorithm 1 Adding perturbation

Input: x, σ , x_{org}, S
Output: x^*, x^*_{org}

```
for i = 0 to 127 do
    for j = 0 to 127 do
        i_org = [i * w/128]
        j_org = [j * h/128]
        if (i_org, j_org) in S then
            x(i, j) += σ(i, j)
            x_org(i_org, j_org) += σ(i, j)
        end if
    end for
end for
```

Moreover, there is no limitation of the maximum values we can change to these pixels, we also modify the FSGM method to gain a better adversarial examples. Perturbation is iterative generated and added to x with small ϵ for N times. At each iteration, the value of pixel is clipped to so that the value is in range $[0, 1]$.

Algorithm 2 generating Adversarial Example for Malware image with FGSM-based method

Input: x, x_{org}, S, ϵ, N, J, θ, y_{true}
Output: x^*, x^*_{org}

$x^* = x$
$x^*_{org} = x_{org}$
for $i = 0$ to N **do**
  $\sigma = \epsilon * \nabla_x J_\theta(x^*, y_{true})$
  x^*, x^*_{org} = AddPerturbation (x, σ, x_{org}, S)
  $clip\{x^*, 0, 1\}$
  $clip\{x^*_{org}, 0, 1\}$
end for

x^*_{org} is then used to reconstruct adversarial malware examples as shown in section II-A3.

IV. Experimental

A. Experimetal setup

Experimental evaluation is conducted in terms of the effectiveness. We firstly introduce the setup of the experiment. A visualization detectors based on Convolutional Neural Network (CNN) is used as case studies to evaluate the effectiveness of our attack methods.

1) Dataset: We have used the Malimg Dataset that had been used in [22]. It consists of 9339 grayscale images of 25 malware families among which 70% of the total data is used for training and 30% is used for testing. Table IV-A1 describes the datasets. In our methods, we alter the value on resources' part only, then in this table we also summarize the number of PE files having resources with different size (in percentage).

2) CNN classifier: We have used CNN because it is reliable and it can be applied to the entire image at a time and then we can assume they are best to use for feature extraction. CNN is a feed-forward neural network where the connectivity pattern between neurons is inspired by the structure of an animal visual cortex and that has proven great value in the analysis of visual imagery. Before feeding to CNN model, all malware images are reshaped into a size of 128 128 and pixels' value are standardized to range $[0, 1]$ by divided by 255 .

Our CNN model is described in Figure IV-A2.Cross entropy loss function that is commonly used for multi class classification was used for this work as well as Adam optimizer for optimization task.

CNN classifier was trained with learning rate = 0.001, batch size = 128, and number of epochs = 20. The experiment shows a test accuracy of 96.9% and a F1 score of 96.38%.

3) Generating adversarial examples: The effectiveness of our generate adversarial examples methods is evaluated by attacking CNN classifiers. We conduct five different experiments:

1) Generating adversarial examples of all samples in test set which are correctly classified.
2) Generating adversarial examples of all samples in test set which are correctly classified and have resources in corresponding PE files.
3) Generating adversarial examples of all samples in test set which are correctly classified and have resources occupy $< 5\%$ in corresponding PE files.
4) Generating adversarial examples of all samples in test set which are correctly classified and have resources occupy from 5% to 10% in corresponding PE files.
5) Generating adversarial examples of all samples in test set which are correctly classified and have resources occupy $\geqslant 10\%$ in corresponding PE files.

TABLE III
Summary of Malimg dataset

	Family name	**No. of samples**	**No. of samples with resources occupy <5%**	**No. of samples with resources occupy from 5% to 10%**	**No. of samples with resources occupy >10%**
1	Adialer.C	122	122		
2	Agent.FYI	116			
3	Allaple.A	2949			
4	Allaple.L	1591			
5	Alueron.gen!J	198	193		
6	Autorun.K	106		106	
7	C2LOP.gen!g	200	181	12	2
8	C2LOP.P	146	22	57	67
9	Dialplatform.B	177			
10	Dontovo.A	162			
11	Fakerean	381		5	331
12	Instantaccess	431	431		
13	Lolyda.AA1	213			
14	Lolyda.AA2	184			
15	Lolyda.AA3	123			2
16	Lolyda.AT	159	5		
17	Malex.gen!J	136		1	
18	Obfuscator.AD	142	142		
19	Rbot!gen	158	157		
20	Skintrim.N	80	80		
21	Swizzor.gen!E	128	99	28	1
22	Swizzor.gen!I	132	74	42	16
23	VB.AT	408	10	209	189
24	Wintrim.BX	97	95	2	
25	Yuner.A	800		800	
		9339	**1611**	**1262**	**608**

All experiments are use $\epsilon = 0.01$ and $N = 200$. To measure the effectiveness of the attacks using the Success Rate (SR): the percentage of adversarial samples that successfully evaded classifier

B. Results and Discussion

1) The effectiveness of the attack using adversarial examples: The figure 6shows the SR through each iteration of five experiments conducted as described in section IV-A3 and the SR of adversarial examples generated is shown in table IV-B1.It can be seen that the highest SR is gained when PE files have more than 10% capacity of resources.

2) Adversarial Training: Training with adversarial examples is one of the countermeasures to make neural networks more robust [28]. Adversarial examples of all samples in

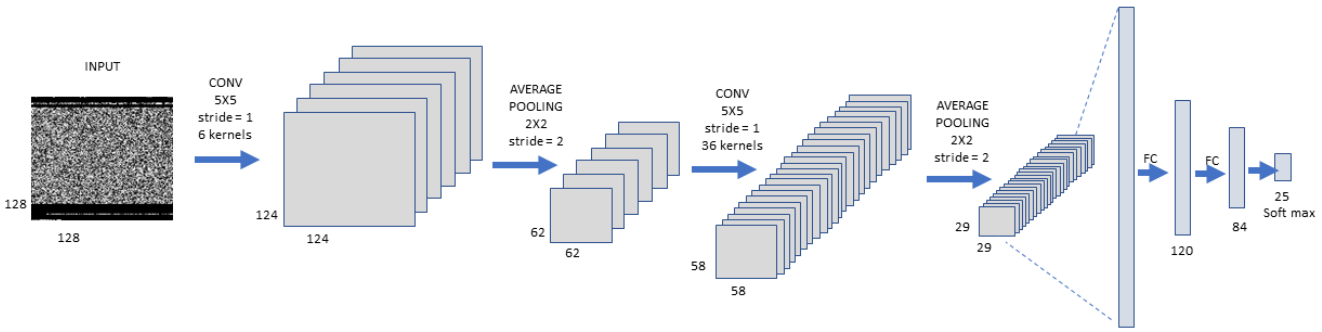

Fig. 5. An overview of our CNN model

TABLE IV
SR OF ADVERSARIAL EXAMPLES IN 5 EXPERIMENTS CONDUCTED AS DESCRIBED IN IV-A3

Experiments of generating adversarial examples	Success Rate
Experiment 1	8.84%
Experiment 2	24.87%
Experiment 3	4.56%
Experiment 4	19.88%
Experiment 5	87.01%

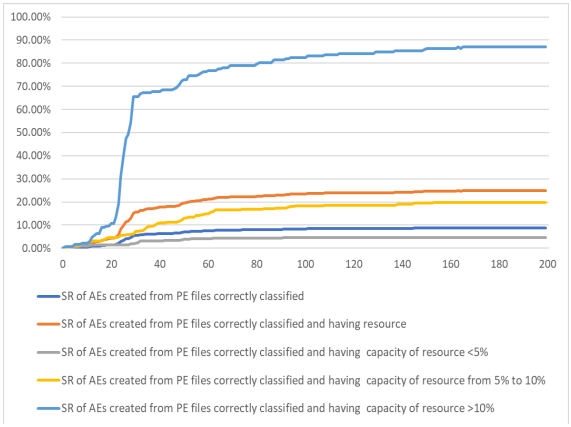

Fig. 6. Success rate of adversarial examples created from different sets

training set are created and classifier will be retrained with these samples.

To evaluate defence strategy by adversarial retraining, CNN classifier is trained with clean examples and adversarial examples created from samples in training set simultaneously. The new model has a test accuracy of 96.06% and a F1 score of 95.58%. Table IV-B2 compare success rate of adversarial examples generated via original model on the new model (with adversarial training) and the original model. The results show that the new model blocks the transferability of adversarial examples.

TABLE V
COMPARISON OF SR ADVERSARIAL EXAMPLES GENERATED VIA ORIGINAL MODEL ON THE NEW MODEL (WITH ADVERSARIAL TRAINING) AND THE ORIGINAL MODEL

Experiments of generating adversarial examples	SR on original model	SR on new model with adversarial training
Experiment 1	8.84%	3.61%
Experiment 2	24.87%	9.53%
Experiment 3	4.56%	4.77%
Experiment 4	19.88%	6.42%
Experiment 5	87.01%	28.25%

V. CONCLUSION AND FUTURE WORK

This work proposed a novel algorithm to generate adversarial malware examples challenge visualized malware classifier. Our method uses the gradient-based FGSM method to generate adversarial examples on the converted image dataset, then these image are used to recontruct adversarial malware examples. The tested CNN classifier is unable to determine the malware correctly. Experimental results demonstrate that adversarial examples of PE files with more than 10% capacity of resources can achieve ... successful attack rate. Moreover, experimental results of adversarial retraining shows that training with adversarial examples created by our method can improve the robustness of malware classifier.

In future, attack transferability will be explored with other malware classifiers.

REFERENCES

[1] A. F. Agarap and F. J. H. Pepito. Towards building an intelligent anti-malware system: A deep learning approach using support vector machine (svm) for malware classification. *arXiv preprint arXiv:1801.00318*, 2017.

[2] M. Ahmadi, G. Giacinto, D. Ulyanov, S. Semenov, and M. Trofimov. Novel feature extraction, selection and fusion for effective malware family classification. In *CODASPY*, 2016.

[3] N. Akhtar and A. S. Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[4] N. Akhtar, S. P. Munagala, and A. S. Mian. Defense against universal adversarial perturbations. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3389–3398, 2018.

[5] N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[6] J. Drew, M. Hahsler, and T. Moore. Polymorphic malware detection using sequence classification methods and ensembles. *EURASIP J. Information Security*, 2017:2, 2017.

[7] J. Drew, T. Moore, and M. Hahsler. Polymorphic malware detection using sequence classification methods. *2016 IEEE Security and Privacy Workshops (SPW)*, pages 81–87, 2016.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2015.

[10] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, pages 62–79, 2017.

[11] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li. Dl 4 md : A deep learning framework for intelligent malware detection. 2016.

[12] W. Hu and Y. Tan. Black-box attacks against rnn based malware detection algorithms. In *AAAI Workshops*, 2018.

[13] N. C. Idika and A. P. Mathur. A survey of malware detection techniques. 2007.

[14] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal. Malware classification with deep convolutional neural networks. *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2018.

[15] F. Kreuk, Y. Adi, M. Cissé, and J. Keshet. Fooling end-to-end speaker verification with adversarial examples. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1962–1966, 2018.

[16] H. Lee, S. Han, and J. Lee. Generative adversarial trainer: Defense to adversarial perturbations with gan. *CoRR*, abs/1705.03387, 2017.

[17] X. Liu, Y. Lin, H. Li, and J. Zhang. Adversarial examples: Attacks on machine learning-based malware visualization detection methods. *arXiv preprint arXiv:1808.01546*.

[18] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *CoRR*, abs/1706.06083, 2018.

[19] A. Makandar and A. Patrot. Malware class recognition using image processing techniques. In *2017 International Conference on Data Management, Analytics and Innovation (ICDMAI)*, pages 76–80, 2017.

[20] Microsoft. https://docs.microsoft.com/en-us/windows/desktop/debug/pe-formatgeneral-concepts.

[21] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.

[22] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, page 4. ACM, 2011.

[23] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. *2016 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 372–387, 2016.

[24] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and classification of malware behavior. In *DIMVA*, 2008.

[25] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2014.

[26] VirusTotal. https://www.virustotal.com.

[27] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille. Adversarial examples for semantic segmentation and object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1378–1387, Oct 2017.

[28] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 2019.