Malware Evasion Attack and Defense

Yonghong Huang*, Utkarsh Verma*, Celeste Fralick*, Gabriel Infante-Lopez†, Brajesh Kumar‡, Carl Woodward*
*McAfee, USA, †McAfee, Argentina, ‡McAfee, India
{yonghong_huang, utkarsh_verma1, celeste_fralick, gabriel_infante-lopez, brajesh_kumar2, carl_woodward}@mcafee.com

***Abstract*—Machine learning (ML) classifiers are vulnerable to adversarial examples. An adversarial example is an input sample which can be modified slightly to intentionally cause an ML classifier to misclassify it. In this work, we investigate *white-box* and *grey-box* evasion attacks to an ML-based malware detector and conducted performance evaluations in a real-world setting. We propose a framework for deploying *grey-box* and *black-box* attacks to malware detection systems. We compared the defense approaches in mitigating the attacks.**

***Index Terms*—Adversarial machine learning, adversarial examples, evasion attack, defense**

I. Introduction

It has been shown that machine learning (ML) classifiers are vulnerable to adversarial examples. An adversarial example is a small perturbation of the original inputs and carefully crafted to misguide the classifier to produce incorrect output. As the presence of ML increases in malware detection, so does its likelihood of being attacked, hijacked or manipulated. In this work, we study vulnerabilities of ML-based malware detection and defenses. We hope that our findings may motivate better use and safe practices of ML for security applications.

The study of adversarial examples have continued to grow. Szegedy [1] was the first to identify adversarial examples in deep neural networks (DNN). Goodfellow [2] proposed the Fast Gradient Sign Method (FGSM), which is a simple and fast method to generate adversarial examples for practical adversarial training. Carlini [4] proposed to optimize the C&W attacks by minimizing the $L0$, $L2$ and $L\infty$ distance metrics. Papernot [3] proposed the Jacobian-based Saliency Map Approach (JSMA) which minimize the number of features and magnitude to be manipulated. Grosse [6] presented the adversarial examples for Android malware. There are some work on *black-box attack* on images [7], [8] and malware [9]. However they are not in real-world black-box setting.

Traditional techniques for making ML robust such as weight decay or dropout do not provide a practical defense. Adversarial training [2] and defensive distillation [10] are two significant defense methods. Carlini [11] showed that the current defense methods including [2], [10] are not effective.

The goal of this work is to study the vulnerabilities of an ML-based malware detector and develop the solutions resilient to the evasion attack. The main contributions are

- We conducted *white-box* & *grey-box* attacks to an ML-based malware detector with thorough evaluations.
- We proposed a framework for deploying grey-box and black-box attacks in real-world setting.
- We developed defense mechanisms that are effective.

TABLE I
The Datasets

Dataset	Number of Samples
Training Set	57170 (28594 clean and 28576 malware)
Validation Set	578 (280 clean and 298 malware)
Malware Test Set	28874
Clean Test Set	16154

TABLE II
Excerpt of a log file

```
GetStartupInfoW:7FEFDD39C37 ()"61468"
GetFileType:7FEFDD39D0C ()"61468"
GetModuleHandleW:13FBC34C3 ()"61484"
GetProcAddress:13FBC34D6 (76D30000,"FlsAlloc")"61484"
...
GetStartupInfoW:13FBC4539 ()"61484"
GetStdHandle:13FBC46F1 ()"61484"
GetFileType:13FBC4707 ()"61484"
FreeEnvironmentStringsW:13FBC4D49 ()"61484"
GetCPInfo:13FBC263D ()"61484"
```

II. Methodology

A. Data Description

The training data were collected by McAfee Labs in January and February 2018. A subset of the training samples and the trained DNN model (trained with over millions of samples) were provided. Table I shows the datasets. The testing data were collected from VirusTotal, which are independent of the training data. Table II shows an excerpt of a log file. The logs capture the API calls from the source files. Only PE samples (exe and dll) were used to generate logs. The mixed data, which contained API logs generated from Win7, WinXP, Win8, and Win10, were created. The 491 API features were extracted from the log files. The raw counts of the APIs were applied to feature transformation and the values were normalized to [0,1]. Table III shows an excerpt of the 491 API features.

B. Attack Experimental Setup

We designed the *white-box*, *grey-box* and *black-box* attack experiments. In *white-box attack*, the attacker has complete knowledge of the system, including training data, features, and ML models (i.e. DNN architecture and parameters). In *grey-box attack*, the attacker has no knowledge of training data and ML model, but knowledge of the features. In *black-box attack*, the attacker has no knowledge of the system. Black-box attack is ongoing work.

TABLE III
EXCERPT OF THE API FEATURES

475	waitmessage
476	windowfromdc
477	winexec
478	writeconsolea
479	writeconsolew
480	writefile
481	writeprivateprofilestringa
482	writeprivateprofilestringw
483	writeprocessmemory
484	writeprofilestringa

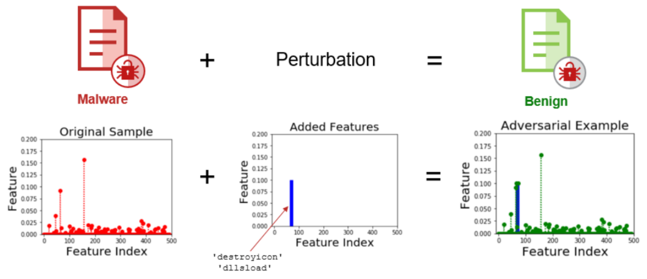


Fig. 1. Generate an adversarial example for malware. The left is one malware sample with 491 features. The middle one is adding two API calls - 'destroyicon' and 'dllsload'. The right is the adversarial example.

1) JSMA for Adversarial Examples: We used the JSMA method [3] to generate the adversarial examples. The JSMA selects the most important feature associated with the maximum gradient based on the saliency map, and perturb the feature to increase the likelihood of the input as the target class. We compute the gradient of F with respect to X to estimate the direction in which a perturbation in X would change Fs output. A perturbation of X with maximal positive gradient into the target class 0 (clean) is chosen. For $F(X) = F_0(X), F_1(X)$,

$$\bigtriangledown_F = \frac{\partial F(X)}{\partial X} = [\frac{\partial F_i(X)}{\partial X_j}]_{i \in 0,1, j \in [1,M]}, \tag{1}$$

Where M is the number of features, $i = 0, 1$ is clean and malware. We ensure that only API calls are added and not deleting any existing features. We make sure the functionality of malware unchanged. CleverHans [5] was used for JSMA implementation. The θ controls the perturbations introduced to modified components. It can be positive or negative. The γ controls the maximum percentage of perturbed features, which is associated with the number of perturbed features. Figure 1 shows how to generate an adversarial example for malware. The adversarial example evades the malware detector and is recognized as benign.

2) Transferability of Adversarial Examples: Adversarial examples generated from one ML model can be used to misclassify another model, even if both models have different architectures and training data. This property is called transferability. This property holds because a contiguous subspace with a large dimension in the adversarial space is shared among different models [13]. The transferability of adversarial examples makes the grey-box and black-box attacks become

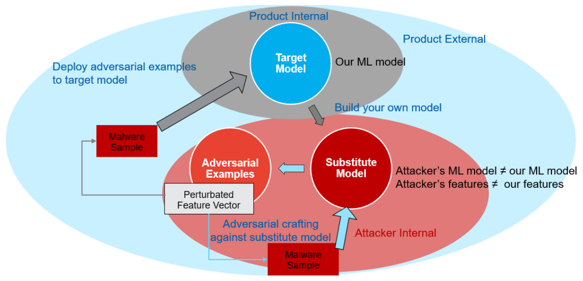

Product Internal
Product External
Target Model
Our ML model
Deploy adversarial examples to target model
Build your own model
Malware Sample
Adversarial Examples
Substitute Model
Attacker's ML model ≠ our ML model
Attacker's features ≠ our features
Perturbated Feature Vector
Adversarial crafting against substitute model
Attacker Internal
Malware Sample

Fig. 2. A framework for *grey-box* and *black-box* attacks in real-world setting

TABLE IV
THE SUBSTITUE MODEL

57170 balanced training data	
5-layer DNN	
1st layer	491 nodes
2nd layer	1200 nodes
3rd layer	1500 nodes
4th layer	1300 nodes
5th layer	2 nodes

possible. Papernot [8] shows that multiple ML algorithms are vulnerable due to the transferability of adversarial examples. Figure 2 shows the framework for *grey-box* and *black-box* attacks in real-world testing and transferability of malware detection. The attacker can train a substitute model to craft adversarial examples and deploy them to the target model to evade the detection. The target model is 4-layer fully connected DNN. Table IV summarizes the architectures of the substitute model - a 5-layer DNN.

C. Defense Methods

We applied the four defense approaches for the ML malware engine. Our considerations in defense are low impact on model architecture and model speed, and maintain model accuracy.

1) Adversarial Training: The basic idea of adversarial training [1], [2] is to inject adversarial examples into training process and train the model either on adversarial examples or on a mix of original and adversarial examples so that the model can generalize to defense against the attacks. The approach has been shown effective with only a small loss of accuracy. However, it needs adversarial examples to train the model and the defense capability is divergent for different attack methods.

2) Defensive Distillation: There are two models in defensive distillation [10]. The first model is trained as usual. The second model (compressed model) is trained with the probabilities (soft labels) learned from the first one. The benefit of using soft class labels lies in the additional knowledge in probabilities compared to hard class labels. Defensive distillation prevents models from fitting too tightly to the data and contributes to better generalization. Distillation is controlled by the softmax temperature T. High T force DNN to produce probabilities with large values for each class.

3) Feature Squeezing: Feature squeezing [12] reduces the degrees of freedom for the attacker to construct adversarial examples by squeezing out unnecessary input features. We used $L1$ norm to measure the distance between the model's prediction on the original sample and the prediction on the sample after squeezing. If the distance is larger than a threshold, then the input sample is an adversarial example. Otherwise, it is a legitimate sample. The assumption is that the squeezing significantly impacts the classifier's prediction on the adversarial examples while it has no significant impact on the classifier's prediction on the legitimate samples.

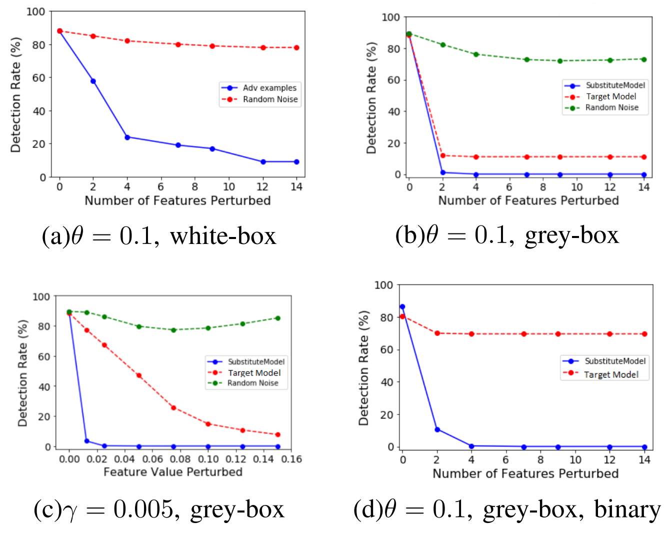
4) Dimensionality Reduction: The approach uses Principal Component Analysis (PCA) for dimensionality reduction [14]. Instead of training a classifier on the original data, it reduces the features from the n-dimension to k ($k << n$), and trains the classifier on the reduced input. The defense restricts the attacker to the first k components. The assumption is that adversarial examples rely on principal components. Therefore, restricting the attack to the first k components should increase the required distortion to produce adversarial examples.

D. Evaluation Metrics

For attack evaluation, we used the security evaluation curve (detection rate as a function of attack strength), transferability (transfer rate) and perturbation measured by L_2 distance norm between the adversarial examples and the original examples. For defense evaluation, we use the confusion matrix - true positive rate (TPR), true negative rate (TNR), false positive rate (FPR) and false negative rate (FNR).

III. Results and Discussions

A. White-box Attack



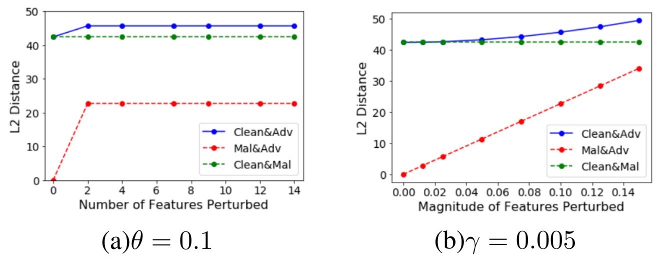
(a)$\theta = 0.1$, white-box (b)$\theta = 0.1$, grey-box

(c)$\gamma = 0.005$, grey-box (d)$\theta = 0.1$, grey-box, binary

Fig. 3. Security evaluation curves for white-box and grey-box attacks. γ is the number of perturbed features and θ is the magnitude of perturbed features.

The result of the white-box attack on $28,874$ malware is shown in Figure 3 (a), where $\theta = 0.1$ and $\gamma = [0 : 0.005 : 0.030]$ (adding $[0 : 2 : 14]$ features). The detection rate drops significantly when attack strength increases by adding more API calls. We observed similar detection rate drops for increasing the frequency of the API calls. The results show that the white-box evasion attack is effective. The perturbation is different from random noise. In the operating point $\theta = 0.1$ and $\gamma = 0.025$ (adding 12 features), the detection rate drops to 0.099, which indicates $26,015$ malware evasion.

B. Grey-box Attack



(a)$\theta = 0.1$ (b)$\gamma = 0.005$

Fig. 4. L2 distances in gray-box attack using original features

We designed three experiments for grey-box attacks. For the first one, we assume the attacker knows the exact 491 features. The substitute model is used to craft the adversarial examples, which are then deployed to the target model. We trained the substitute model with 1000 epoch, batch size = 256 and learning rate = 0.001 and Adam optimizer. We used $28,874$ malware samples to craft adversarial examples. The grey-box attack is shown in Figure 3, where (b)$\theta = 0.1$ and (c)$\gamma = 0.005$ (i.e. adding 2 features). The results show that the grey-box attack is effective. The detection rate of the target model significantly drops with a small number of perturbed features or with small amount magnitude changes. It indicates that the attacker can craft the adversarial examples using the substitute model to evade the DNN malware detector. In the operating point $\theta = 0.1$ and $\gamma = 0.005$, which associated with adding 2 features, the detection rate of the target model drops to 0.147. The transferability is 0.853. It indicates that $24,630$ malware can evade the malware detector. The grey-box attack is less effective than the white-box attack because the attacker has less knowledge about the target system.

For the second experiment, we assume the attacker does not have knowledge of the DNN model and the feature transformation but knows API calls as features. We create a substitute model using binary features - when the API appears, the feature value equals one, otherwise equals to zero. Figure 3 (d) shows the detection rate drops significantly when the attack strength increases by adding more API calls for the substitute model, but does not significantly impact the target model. The detection rate is 0.3049. The transferability is 0.6951.

To understand adversarial examples across the decision boundary, we evaluated the L2 distances for (1) malware and adversarial examples (2) malware and clean (3) clean and adversarial examples. The L2 results are shown in Figure 4, where (a)$\theta = 0.1$ and (b)$\gamma = 0.005$ (i.e. adding 2 features). Both plots show consistent results. The results show that the distance for malware and adversarial examples is the shortest and the distance for clean to adverarial examples is the largest.

TABLE V
ADVERSARIAL TRAINING DATASET

Dataset	Number of Samples
Training Set	53482(26118 clean, 27364 malware and advExa)
Malware Test	5252
Clean Test	5090
advExample	16218

TABLE VI
DEFENSE TESTING RESULTS

	Dataset Name	TPR	TNR
No Defense	Clean Test	nan	0.964
	Malware Test	0.883	nan
	AdvExamples	0.304	nan
AdvTraining	Clean Test	nan	0.995
	Malware Test	0.888	nan
	AdvExamples	0.931	nan
Distillation	Clean Test	nan	0.428
	Malware Test	0.573	nan
	AdvExamples	0.577	nan
FeaSqueezing	Clean Test	nan	0.586
	Malware Test	nan	0.438
	AdvExamples	0.554	nan
DimReduct	Clean Test	nan	0.674
	Malware Test	0.914	nan
	AdvExamples	0.913	nan

The distance increases when The attack strength increase. From the results, we can see that the adversarial examples in high dimension feature space are in the blind spot far away from clean and not lie in the decision boundary between malware and clean, which give us the insight for defense.

For the third experiment, we conducted the live grey-box testing. We were provided a source file and an associated log file. We used the substitute model to generate an adversarial example. We asked the security team to add one single API call multiple times in the source code and ran the DNN engine to detect this sample. The DNN engine originally detects this sample as malware with 98.43% confidence. After adding this API once into the source code, the detection rate drops to 88.88% confidence. When adding the same API eight times to the source code, the detection rate drops to 0%. It indicates the adversarial example successfully evaded the DNN malware detector in a real-world grey-box setting.

C. Defense

We applied four defense approaches. The results of the four defense approaches are shown in Table VI. In adversarial training, we did sanity check on the data to reduce the duplicated samples. A subset of the adversarial examples from the grey-box attack ($\theta = 0.1$ and $\gamma = 0.02$) and a subset of testing malware were added to the training set. In order to make the training set balance, we added a subset of clean samples into the training set. Table V shows the dataset. The testing results show that adversarial training significantly increases the detection rate for adversarial examples from 0.304 to 0.931. At the same time, it also increases the TNR from 0.964 to 0.998 with compromising the detection rate for original malware. In dimension reduction, we selected $K = 19$. The testing results show that both detection rates of adversarial examples and malware significantly increases to 0.913 and 0.914. However, the TNR decreases from 0.964 to 0.674. The results suggest we should consider ensemble adversarial training and dimension reduction. For both defensive distillation ($T = 50$) and feature squeezing, the detection rate of adversarial examples increases. However, TNR for detecting clean and TPR for detecting malware drop.

IV. CONCLUSION

The current attack shows that modifying one bit in the feature vector can bypass the real-world ML-based malware detector. The attack can be done by modifying the malware source code by adding one API call. We show some preliminary results on building attack resiliency to API additions for mitigation. It is an open challenge to design a defense against a powerful adaptive attack. For future work, we are working on creating the substitute model using open source data with different features and models for real-world black-box testing as proposed in Figure 2. We will study the intepretability of adversarial examples to develop more effective defenses.

ACKNOWLEDGMENT

Thanks to Vaisakh Shaj and Xiang Xu for their technical work. Thanks to Esteban Prospero, Muralivardhan Pannala, Sidney Gomindes for their support.

REFERENCES

[1] C. Szegedy, W. Zaremba, I. Sutskever, D. Erhan, I. Goodfellow and R. Fengus, *Intriguing properties of neural networks*, in Proc of ICLR, 2013.

[2] I.J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, in Proc of ICLR, 2015.

[3] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, *The limitations of deep learning in adversarial settings*, in Proc of IEEE European Symposium on Security & Privacy, 2016.

[4] N. Carlini and D. Wagner, *Towards evaluating the robustness of neural networks*, in Proc of IEEE Symposium on Security and Privacy, 2017.

[5] N. Papernot et al., *Technical report on the CleverHans v2.1.0 adversarial examples library*, arXiv:1610.00768, 2018.

[6] K. Grosse, N. Papernot, P. Manoharan, M. Backes, P. D. McDaniel, *Adversarial examples for malware detection*, ESORICS (2), Vol. 10493 of LNCS, Springer, pp. 6279, 2017.

[7] Y. Liu, X. Chen, C. Liu, and D. Song, *Delving into transferable adversarial examples and black-box attacks*, arXiv:1611.02770, 2016.

[8] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, A. Swami, *Practical black-Box attacks against machine learning*, in Proc of ACM on Asia Conference on Computer and Communications Security, 2017.

[9] W. Hu and Y. Tan, *Generating adversarial malware examples for black-Box attacks based on GAN*, arXiv:1702.05983, 2017.

[10] N. Papernot, P. McDaniel, X. Wu, S. Iha and A. Swami, *Distillation as a defense to adversarial perturbations against deep neural networks*, In Proc of IEEE Symposium on Security & Privacy, 2016.

[11] N. Carlini and D. Wagner, *Adversarial examples are not easily detected: bypassing ten detection methods*, arXiv:1705.07263v2, 2017.

[12] W. Xu, D. Evans, and Y. Qi, *Feature squeezing: Detecting adversarial examples in deep neural networks*, In Proc of Network and Distributed Systems Security Symposium, 2018.

[13] F. Tramr, N. Papernot, I. Goodfellow, D. Boneh, P. McDaniel,*The space of transferable adversarial examples*, arxiv:1704.03453, 2017.

[14] A. N. Bhagoji, d. Cullina, C. Sitawarinn, P. Mittal, *Enhancing robustness of machine learning systems via data transformations*, arxiv:1704.02654v4, 2017.