

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 is slightly modified to induce misclassification in an ML classifier. In this work, we investigate *white-box* and *grey-box* evasion attacks to an ML-based malware detector and conduct performance evaluations in a real-world setting. We compare the defense approaches in mitigating the attacks. We propose a framework for deploying *grey-box* and *black-box* attacks to malware detection systems.

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 [1]–[6], so does its likelihood of being attacked, hijacked or manipulated. In this work, we examine the potential vulnerabilities of ML-based malware detectors in adversarial environments. We hope that our findings may motivate better use and safe practices of ML for security applications.

The study of adversarial examples has continued to grow. Biggio provided an overview of adversarial ML for the past 10 years [7]. There have been extensive studies of adversarial examples in computer vision [8]–[14]. Szegedy [8] first identified adversarial examples in deep neural networks (DNN) and proposed L-BFGS attack. Goodfellow [9] proposed the Fast Gradient Sign Method (FGSM), which is a simple and fast method to generate adversarial examples for practical adversarial training. Carlini [11] proposed the C&W attacks by minimizing the L_0 , L_2 and L_∞ distance metrics. It is one of the strongest attacks. Papernot [10] proposed the Jacobian-based Saliency Map Approach (JSMA) which minimize the number of features and magnitude to be manipulated. Athalye [12] proposed the expectation over transformation (EOT) algorithm to generate adversarial examples which remain adversarial over a chosen transformation distribution.

Recent research of adversarial examples in malware detection has drawn attention in both ML and security communities [16]–[19]. Grosse [16] presented the adversarial examples for Android malware. Demetrio [19] used an interpretable ML approach to identify the most influential features and proposed an attack to generate adversarial malware examples by manipulating the file header. There are some work on *black-box attack* on images [20], [21] and malware [22]. However, they are not in real-world black-box setting.

TABLE I
THE DATASET

Dataset	Number of Samples
Training Set	57170 (28594 clean and 28576 malware)
Validation Set	578 (280 clean and 298 malware)
Test Set	45028 (16154 clean and 28874 malware)

Traditional techniques for making ML robust, such as weight decay or dropout, do not provide a practical defense. Adversarial training [9] and defensive distillation [23] are two significant defense methods. Carlini [24] shows that the current defense methods including [9], [23] are not effective.

The goal of this work is to study the vulnerabilities of an ML-based malware detector and develop a solution 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 developed defense mechanisms and compared their effectiveness.
- We propose a framework for deploying *grey-box* and *black-box* attacks in a real-world setting.

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 millions of samples) were provided. Table I shows the dataset including training set, validation set and testing set. The testing data were collected from VirusTotal, which are independent of the training data.

The 491 API features were extracted from log files. Table II shows an excerpt of a log file. The logs capture the API calls from the source files. Portable executable (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 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 API features.

B. Attack Experimental Setup

We design 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

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"
```

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
```

ML model, but knowledge of the features. In *black-box attack*, the attacker has no knowledge of the system.

1) *JSMA for Adversarial Examples*: In this work, we aim to develop an attack with the perturbations using minimum number of features, so we selected the JSMA [10]. The success rate and transfer rate for the JSMA are relatively high.

The JSMA allows us to select 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 F 's output. A perturbation of X with maximal positive gradient into the target class 0 (clean) is chosen. For $F(X) = F_0(X), F_1(X)$,

$$\nabla F = \frac{\partial F(X)}{\partial X} = \left[\frac{\partial F_i(X)}{\partial X_j} \right]_{i \in \{0,1\}, j \in [1,M]}, \quad (1)$$

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 [15] was used for the JSMA implementation. The θ controls the perturbations introduced to modified features. 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 [26]. The transferability of adversarial examples makes the *grey-box* and *black-box* attacks become

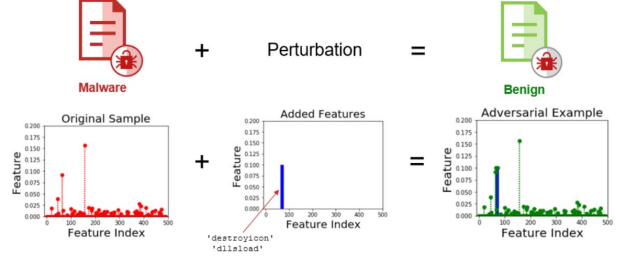


Fig. 1. Generate an adversarial example for malware. The left is one malware sample with 491 features. The middle one is adding two API calls - 'destroyicon' and 'dllsload'. The right is the adversarial example.

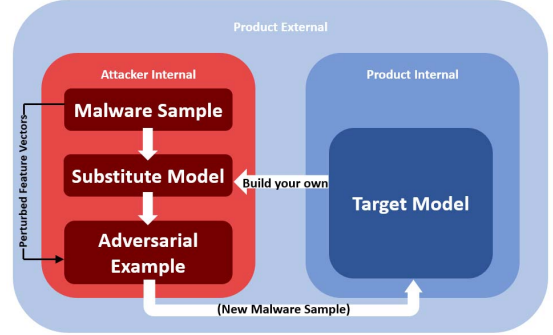


Fig. 2. A framework for *grey-box* and *black-box* attacks in real-world setting

possible. Papernot [21] 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 the transferability of adversarial examples in 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 attacker's model, features and training data are different from the target model, features and training data. In this work, the target model is 4-layer fully connected DNN (The target model is proprietary, so we cannot release the detail information.) Table IV summarizes the architectures of the substitute model, an 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 [8], [9] is to inject adversarial examples into training process and train the model 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 performance decreases for different attack methods.

2) *Defensive Distillation*: There are two models in defensive distillation [23]. 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

TABLE IV
THE SUBSTITUTE 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

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 [25] 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 [27]. Instead of training a classifier on the original data, it reduces the features from the n -dimension to k ($k \ll n$), 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 in terms of transfer rate, and perturbation measured by L_2 distance norm between the adversarial examples and the original examples. For defense evaluation, we use *confusion matrix* - true positive rate (TPR), true negative rate (TNR), false positive rate (FPR), and false negative rate (FNR).

III. RESULTS AND DISCUSSION

A. White-box Attack

The *white-box* attack on 28,874 malware samples is shown in Figure 3, where (a) $\theta = 0.1$ and $\gamma = [0 : 0.005 : 0.030]$ adding $[0 : 2 : 14]$ features, and (b) $\gamma = 0.025$ and $\theta = [0 : 0.0125 : 0.15]$. For both (a) and (b), the detection rates drop significantly when attack strength increases either by adding more API calls or increasing the frequency of the API calls. Randomly adding features does not decrease the detection rates. The results show that the *white-box* evasion

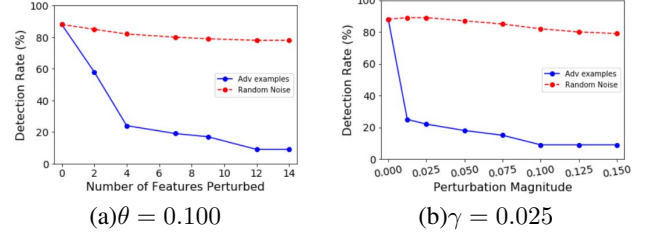


Fig. 3. Security evaluation curve for *white-box* attack. γ is associated with the number of perturbed features. θ is the magnitude of perturbed features.

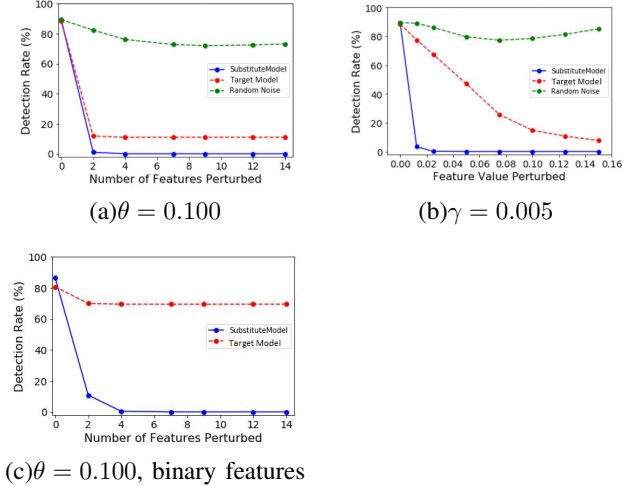


Fig. 4. Security evaluation curves for *gray-box* attacks. γ is the number of perturbed features and θ is the magnitude of perturbed features.

attack is effective. The JSMA 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

We design three experiments for *gray-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 train the substitute model with 1000 epoch, batch size = 256 and

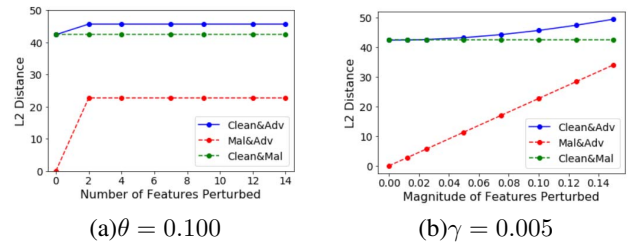


Fig. 5. L_2 distances in *gray-box* attack using original features

learning rate = 0.001 and Adam optimizer. We use 28,874 malware samples to craft adversarial examples. The *gray-box* attack is shown in Figure 4, where (a) $\theta = 0.1$ and (b) $\gamma = 0.005$ (i.e. adding 2 features). The results show that the *gray-box* attack is effective. The detection rates of the target model significantly decrease 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 transfer rate is 0.853. It indicates that 24,630 malware can evade the malware detector. The *gray-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 4 (c) shows the detection rates decrease 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 of the target model is 0.6951. The transfer rate is 0.3049. It shows the attack is less effective when attacker has less knowledge about the features.

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 5, 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 adversarial examples is the largest. 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 provides us the insight for defense.

For the third experiment, we conducted the live *gray-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 a security researcher 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 in the source code, the detection rate drops to 88.88% confidence. When adding the same API eight times in the source code, the detection rate drops to 0%. It indicates the adversarial example successfully evaded the DNN malware detector in a real-world *gray-box* setting.

C. Defense

We applied the four defense methods in Section II-C. The results of the defense approaches are shown in Table VI. In adversarial training, we did **sanity check on the data to reduce**

TABLE V
ADVERSARIAL TRAINING DATASET

Dataset	Number of Samples
Training Set	53482(26118 clean, 27364 malware and advEx)
Test Set	26560(5090 clean, 5252 malware and 16218 advEx)

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
DimReduce	Clean Test	nan	0.674
	Malware Test	0.914	nan
	AdvExamples	0.913	nan

the duplicated samples. A subset of the adversarial examples from the *gray-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 without compromising the detection rate for the 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 may consider ensemble adversarial training and dimension reduction. For both defensive distillation ($T = 50$) and feature squeezing, the detection rate of adversarial examples increases. However, the TNR for detecting clean and TPR for detecting malware drop.

IV. CONCLUSION

The JSMA 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 building the real-world *black-box* testing framework as proposed in Figure 2 using open source data with different features and models. We will study the interpretability of adversarial examples to develop more effective defenses.

ACKNOWLEDGMENT

We would like to thank Vaisakh Shaj and Xiang Xu for their engineering contributions. Additional thanks to Esteban Prospero, Muralivardhan Pannala, Sidney Gomindes for their support to the project.

REFERENCES

- [1] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, *Large-scale malware classification using random projections and neural networks*, in Proc of IEEE Intl. Conf on Acoustics, Speech and Signal Processing (ICASSP), 2013.
- [2] J. Saxe and K. Berlin, *Deep neural network based malware detection using two dimensional binary program features*, arXiv preprint arXiv:1508.03096, 2015.
- [3] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, *Malware classification with recurrent networks*, in Proc of ICASSP, 2015.
- [4] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, *Malware detection by eating a whole exe*, arXiv preprint arXiv:1710.09435, 2017.
- [5] B. Athiwaratkun and J.W. Stokes, *Malware classification with LSTM and GRU language models and a character-level CNN*, in Proc of ICASSP, 2017.
- [6] H. S. Anderson and P. Roth, *EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models*, arXiv preprint arXiv:1804.04637, 2018.
- [7] B. Biggio and F. Roli, *Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning*, arXiv preprint arXiv:1712.03141, 2018.
- [8] C. Szegedy, W. Zaremba, I. Sutskever, D. Erhan, I. Goodfellow and R. Fengus, *Intriguing properties of neural networks*, in Proc of Intl Conf on Learning Representations (ICLR), 2013.
- [9] I.J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, in Proc of ICLR, 2015.
- [10] 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.
- [11] N. Carlini and D. Wagner, *Towards evaluating the robustness of neural networks*, in Proc of IEEE Symposium on Security and Privacy, 2017.
- [12] A. Athalye, L. Engstrom, A. Ilyas, K. Kwok, *Synthesizing Robust Adversarial Examples*, in Proc of Intl Conf on Machine Learning (ICML), 2018.
- [13] S. Moosavi-Dezfooli, A. Fawzi and P. Frossard, *Synthesizing robust adversarial examples*, in Proc of IEEE Conf on Computer Vision and Pattern Recognition (CVPR), 2016.
- [14] A. Madry, A. Makelov, L. Schmidt, D. Tsipra and A. Vladu, *Towards Deep Learning Models Resistant to Adversarial Attacks*, arXiv preprint arXiv:1706.06083, 2017.
- [15] N. Papernot et al., *Technical report on the CleverHans v2.1.0 adversarial examples library*, arXiv preprint arXiv:1610.00768, 2018.
- [16] 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.
- [17] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, *Adversarial malware binaries: Evading deep learning for malware detection in executables*, arXiv preprint arXiv:1803.04173, 2018.
- [18] F. Kreuk, A. Barak, S. Aviv, M. Baruch, B. Pinkas, and J. Keshet., *Deceiving end-to-end deep learning malware detectors using adversarial examples*, arXiv preprint arXiv:1802.04528, 2019.
- [19] L. Demetrio, B. Biggio, G. Lagorio, F. Roli and A. Armando, *Explaining Vulnerabilities of Deep Learning to Adversarial Malware Binaries*, arXiv preprint arXiv:1901.03583, 2019.
- [20] Y. Liu, X. Chen, C. Liu, and D. Song, *Delving into transferable adversarial examples and black-box attacks*, arXiv preprint arXiv:1611.02770, 2016.
- [21] 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.
- [22] W. Hu and Y. Tan, *Generating adversarial malware examples for black-Box attacks based on GAN*, arXiv preprint arXiv:1702.05983, 2017.
- [23] N. Papernot, P. McDaniel, X. Wu, S. Jha and A. Swami, *Distillation as a defense to adversarial perturbations against deep neural networks*, In Proc of IEEE Symposium on Security & Privacy, 2016.
- [24] N. Carlini and D. Wagner, *Adversarial examples are not easily detected: bypassing ten detection methods*, arXiv preprint arXiv:1705.07263v2, 2017.
- [25] 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.
- [26] F. Tramr, N. Papernot, I. Goodfellow, D. Boneh, P. McDaniel, *The space of transferable adversarial examples*, arXiv preprint arxiv:1704.03453, 2017.
- [27] A. N. Bhagoji, d. Cullina, C. Sitawarinn, P. Mittal, *Enhancing robustness of machine learning systems via data transformations*, arXiv preprint arxiv:1704.02654v4, 2017.