A Framework for Enhancing Deep Neural Networks Against Adversarial Malware

Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu

***Abstract*—Machine learning-based malware detection is known to be vulnerable to adversarial evasion attacks. The state-of-the-art is that there are no effective defenses against these attacks. As a response to the adversarial malware classification challenge organized by the MIT Lincoln Lab and associated with the AAAI-19 Workshop on Artificial Intelligence for Cyber Security (AICS'2019), we propose six guiding principles to enhance the robustness of deep neural networks. Some of these principles have been scattered in the literature, but the others are introduced in this paper for the first time. Under the guidance of these six principles, we propose a defense framework to enhance the robustness of deep neural networks against adversarial malware evasion attacks. By conducting experiments with the Drebin Android malware dataset, we show that the framework can achieve a 98.49% accuracy (on average) against grey-box attacks, where the attacker knows some information about the defense and the defender knows some information about the attack, and an 89.14% accuracy (on average) against the more capable white-box attacks, where the attacker knows everything about the defense and the defender knows some information about the attack. The framework wins the AICS'2019 challenge by achieving a 76.02% accuracy, where neither the attacker (i.e., the challenge organizer) knows the framework or defense nor we (the defender) know the attacks. This gap highlights the importance of knowing about the attack.**

***Index Terms*—Adversarial machine learning, adversarial malware detection, deep neural networks, malware classification**

I. Introduction

MALWARE remains a big threat to cyber security despite communities' tremendous countermeasure efforts. For example, Symantec [2] reports seeing 357019,453 new malware variants in the year 2016, 669974,865 in the year 2017, and 246002,762 in the year 2018. Worse yet, there is an increasing number of malware variants that attempt to undermine anti-virus tools and indeed evade many malware detection systems [3].

In order to cope with the increasingly severe situation, we have to resort to machine learning for automating the detection of malware in the wild [4]. However, machine learning based techniques are vulnerable to adversarial evasion attacks, by which an adaptive attacker perturbs or manipulates malware examples into adversarial examples that would be detected as benign rather than malicious (see, for example, [5], [6], [7], [8], [9], [10], [11]). The state-of-the-art is that there are many attacks, but the problem of effective defense is largely open. For example, adversarial training is known to be able to harden classifiers against adversarial examples, but requires knowing about the attack in terms of (for example) its manipulation set [7]. This is indeed the context in which the AICS'2019 Malware Classification Challenge is proposed. In a broader context, adversarial malware examples are a particular kind of attacks against adversarial machine learning. Although adversarial machine learning has received much attention in application domains such as image processing (see, e.g., [12], [13], [14]), the problem of adversarial malware examples are much less investigated [5], [7], [9], [15].

The AICS'2019 challenge mentioned above is essentially about *whether we can defend against adversarial examples in the wild*. The challenge is characterized as follows. First, we (i.e., any team participating in the Challenge as the defender) are given a training set in the form of anonymized feature representation by the Challenge organizer (i.e., we do not even know what the feature names are), as well as the corresponding ground-truth labels. We are informed by the Challenge organizer that the training data contains *no* adversarial examples. Second, we are given a set of test data (again, in anonymized feature representation) and are told that the test data contains both adversarial examples and non-adversarial examples. We do not know what attacks are used by the

Manuscript received April 15, 2020; revised October 2, 2020; accepted January 7, 2021. Date of publication January 13, 2021; date of current version March 5, 2021. Deqiang Li was supported in part by the China Scholarship Council under Grant 201706840123. Qianmu Li was supported in part by the National Key R&D Program of China under Grants 2020YFB1804604, 2020YFB1804600 and 2020YFB1805503, the 2020 Industrial Internet Innovation and Development Project from Ministry of Industry and Information Technology of China, the 2018 Jiangsu Province Major Technical Research Project Information Security Simulation System, the Fundamental Research Fund for the Central Universities under Grants 30918012204 and 30920041112, the 2019 Industrial Internet Innovation and Development Project from Ministry of Industry and Information Technology of China. Y. Ye and S. Xu were supported in part by NSF Grant #1814825. The opinions expressed in the paper are those of the authors' and do not reflect the funding agencies' policies in any sense. Recommended for acceptance by X. Du.

Deqiang Li is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: lideqiang@njust.edu.cn).

Qianmu Li is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China, and also with the School of Intelligent Manufacturing, Wuyi University, Nanping 529020, China (e-mail: qianmu@njust.edu.cn).

Yanfang Ye is with the Department of Computer and Data Sciences, Case Western Reserve University, Cleveland, OH 44106 USA (e-mail: yanfang.ye@case.edu).

Shouhuai Xu was with the Department of Computer Science, University of Texas, San Antonio. He is now with the Department of Computer Science, University of Colorado Colorado Springs, Colorado Springs, CO 80918 USA (e-mail: sxu@uccs.edu).

Digital Object Identifier 10.1109/TNSE.2021.3051354

2327-4697 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

Challenge organizers. We do not know which examples in the test set are perturbed by adversaries. This means that we neither know which are adversarial examples, nor the attacks that are used to generate them, nor the manipulation set. Third, our task is to accurately classify the test data, including both adversarial and non-adversarial examples. The setting of the Challenge is realistic because in the real world defenders do not know attacker's specifications such as attack methods, manipulation sets, and specific adversarial examples. The importance of the problem in defending against adversarial malware examples and the realistic setting of the Challenge motivate the present study.

A. Our Contributions

In this paper, we make the following contributions. First, we propose, to the best of our knowledge, the first systematic defense framework to enhance the robustness of Deep Neural Network (DNN)-based malware classifiers against adversarial evasion attacks. The framework is designed under the guidance of a set of principles, some of which are known but scattered in the literature (e.g., using an ensemble of classifiers and *minmax* adversarial training), but others are introduced in this paper for the first time, such as the following. We propose (i) using white-box attack, where the attacker knows everything about the defense, to bound the capability of grey-box attacks with respect to the ℓ_p $(p \geq 1)$ norm, where the attacker knows something about the defense; (ii) using adversarial regularization (i.e., adversarial training with small perturbations) when the manipulation set is not available to the defender; (iii) leveraging semantics-preserving representations (realized by Denoising Auto-Encoder or DAE for shorthand).

Second, we empirically validate the effectiveness of the framework against 11 grey-box attacks and 9 white-box attacks (i.e., 20 attacks in total). The 11 grey-box attacks include the Random attack, two Mimicry attacks [16], the Fast Gradient Sign Method (FGSM) attack [13], Grosse attack [5], Bit Gradient Ascent (BGA) attack [7], Bit Gradient Ascent (BCA) attack [7], and four variants of the Projected Gradient Descent (PGD) attacks. The 9 white-box attacks leverage the victim models directly and the attack algorithms are the same as the latter 9 ones mentioned above. Among these attacks, the four variants of the PGD attacks are used to be investigated in other application settings and are adapted to the adversarial malware detection domain for the first time. The variant PGD attacks permit feature addition and feature removal, incurring larger manipulation sets than the Grosse, BGA, and BCA attacks. In these experiments, adversarial malware examples are generated by manipulating regular malware examples while preserving their malicious functionalities. Our empirical findings include: (i) standard DNNs without incorporating defense can be ruined by both grey-box and white-box attacks; (ii) adversarial regularization without considering attacks in the training phase has limited success in terms of improving the robustness of DNNs against adversarial examples; (iii) adversarial training with the Adam optimizer can significantly enhance the robustness of DNNs against multiple grey-box evasion attacks, but not the more capable white-box Grosse, BCA and PGD-ℓ_1 attacks; (iv) DAE provides a degree of extra robustness when used together with adversarial training, which is ineffective in defending against the white-box Grosse, BCA and PGD-ℓ_1 attacks; (v) adding ensembles further improves the robustness of DNNs, at the price of sacrificing a degree of the effectiveness of adversarial training against the white-box PGD-ℓ_2, PGD-ℓ_∞ and PGD-Adam attacks.

Third, we apply the framework to the AICS'2019 adversarial malware classification challenge organized by the MIT Lincoln Lab. According to the Challenge organizers, there were "over 300 participants attempted to download and classify the malware data set" [17] and we win the Challenge by achieving a 73.60% Harmonic mean score (which is the metric the organizer chose to use before making the data available); i.e., we achieve the *highest* score among all of the participating teams.

Fourth, after announcing that we win the Challenge, the organizer makes the ground-truth labels publicly available at http://www-personal.umich.edu/~arunesh/AICS2019/challenge.html. In order to understand why we only achieve a 73.60% Harmonic mean score, we leverage the ground-truth labels to conduct a further study. We show that (i) oversampling benefits adversarial regularization in defending against evasion attacks in term of the Macro-F1 score and (ii) adversarial regularization tends to overfit the perturbed examples while this phenomenon does not occur to the non-adversarial (i.e., original) data.

Fifth, we show that the framework is effective in resisting grey-box attacks via the widely-used Drebin Android malware dataset (with a 98.49% accuracy on average), where the attacker knows some information about the defense and the defender knows some information about the attack. When applied to the AICS'2019 challenge dataset but *only* considering the adversarial examples (for the sake of fair comparison with the experiment on the Drebin dataset), the framework only achieves a 76.02% accuracy on average, where it is still true that neither the attacker knows the defense nor the defender knows the attacks. This highlights that the defender should always strive to know as much information as possible about the attacks. In order to avoid any confusion, we reiterate that the aforementioned experiment result (i.e., 73.60% in the Harmonic mean score) considers both adversarial and non-adversarial examples (as required by the challenge organizer); whereas the 76.02% accuracy disregards of the non-adversarial examples (for fair comparison with the experiment with the Drebin dataset). Another difference is that in the new experiment achieving a 76.02% accuracy we use an ensemble of 5 building-block models, whereas in the experiment achieving 73.60% Harmonic mean score we use 10 building-block models.

Last but not the least, we made our the code of our models publicly available at https://github.com/deqangss/aics2019_challenge_adv_mal_defense.

B. Related Work

Since the present paper focuses on defense against adversarial malware examples, we review related prior studies in four

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

categories: *ensemble learning*, *input prepossessing*, *adversarial training/regularization*, and *DAE-based representation learning*.

Ensemble learning can reduce the generalization error by diversifying the building-block models. Biggio *et al.* [18], [19] show how the *bagging* and *random subspace* techniques can enhance the robustness of linear models against evasion attacks. Smutz and Stavrou [20] propose using the confidence score produced by random forest classifiers to detect adversarial malware. Stokes *et al.* [21] investigate the resilience of ensemble DNNs against evasion attacks. In this paper, we diversify the building-block models via randomly initialized parameters and the random subspace algorithm.

Input prepossessing transforms the input to a different representation, aiming to reduce the degree of perturbations applied to the original input. For example, Random Feature Nullification (RFN) randomly nullifies features both in the training and test phases [22]; HashTran [23] reduces small perturbations using a locality-sensitive hashing; DroidEye [24] quantizes binary representation via count featurization. In our framework, inspired by the idea of feature squeezing [25], we use binarization to reduce the perturbations.

Adversarial training augments the training data with adversarial examples. Various kinds of *heuristic* training strategies have been proposed (see, e.g., [5], [12], [13], [26], [27]. However, these strategies typically deal with specific evasion methods and are not effective against others. Furthermore, researchers propose considering adversarial training with the optimal attack, which in a sense corresponds to the worst-case scenario and therefore could lead to classifiers that are robust against the non-optimal attacks [7], [28]. In our framework, we seek the optimal attack via a gradient descent method, while meeting the requirement of discrete inputs via a nearest neighbor search.

Adversarial regularization is an adversarial training method that aims to train a model with *slightly* perturbed examples, which may or may not be functionality-preserving. Intuitively, small perturbations benefit the generalization of DNN models [13], [27], [29], [30], [31]. This approach may be useful because in the context of malware detection, the defender may not know the manipulation set of the attacker.

DAE facilitates robust representation learning [32], [33]. Li *et al.* [23] propose detecting adversarial malware examples using DAE. In our framework, we use DAE to learn the robust representation that is insensitive to perturbations.

C. Paper Outline

The rest of the paper is organized as follows. Section II presents the adversarial malware evasion attacks, including four attacks that are adapted to the domain of adversarial malware detection for the first time. Section III describes our defense framework. Section IV validates our defense framework with a real-world dataset. Section V presents the results when applying the framework to the AICS'2019 Challenge *without* knowing anything about the attack. Section VI presents our further study after winning the AICS'2019 Challenge and being given the ground-truth labels of the test data. Section VII concludes the present paper.

TABLE I
MAIN NOTATIONS USED IN THE PAPER

Notation	Meaning
$z \in \mathcal{Z}$	z is a software example; $\mathcal{Z}$ is the example space
$(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$	$\mathbf{x}$ is feature representation of z; $\mathcal{X}$ is the feature space; y is the label of $\mathbf{x}$; $\mathcal{Y}$ is the label space
$\delta_{\mathbf{x}} \in \mathcal{M}_{\mathbf{x}}$	$\delta_{\mathbf{x}}$ is a perturbation vector of $\mathbf{x}$; $\mathcal{M}_{\mathbf{x}}$ is the manipulation set of $\mathbf{x}$
$\mathbf{x}' \in \mathcal{S}(\mathbf{x}, \mathcal{M}_{\mathbf{x}})$	$\mathbf{x}'$ is perturbed from $\mathbf{x}$; $\mathcal{S}$ is the set of perturbed representations derived from $\mathbf{x}$ and $\mathcal{M}_{\mathbf{x}}$; $\mathcal{S} \subseteq \mathcal{X}$
o	o is the number of classes
dim	dim is the number of dimension of $\mathbf{x}$
$f : \mathcal{X} \to \mathcal{Y}$	f is the classifier
$\mathbf{F} : \mathcal{X} \to \mathbb{R}^o$	$\mathbf{F}$ denotes a neural network
θ	θ denotes parameters of neural network $\mathbf{F}$
$L : \mathbb{R}^o \times \mathcal{Y} \to \mathbb{R}$	L is cross-entropy loss function

II. Adversarial Malware Evasion Attacks

A. Notations

Given a non-adversarial malware example $z \in \mathcal{Z}$, its feature representation $\mathbf{x} \in \mathcal{X}$ can be obtained via some *feature extraction* methods, where $\mathcal{Z}$ denotes the *example space* (i.e., the set of all possible software examples) and $\mathcal{X}$ denotes the *feature space* (typically discrete). A classifier $f : \mathcal{X} \to \mathcal{Y}$ takes $\mathbf{x}$ as input and outputs its label $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the *label space*.

We focus on a classifier f that is learned as a neural network model $\mathbf{F} : \mathcal{X} \to \mathbb{R}^o$, whose output (*softmax*) is treated as the probability mass function over $o = |\mathcal{Y}|$ classes [1], [5], [7], [34]. That is $f = arg\,max_{j \in \mathcal{Y}} \mathbf{F}$, where $arg\,max$ returns the index of the maximum element in a o-dimension vector. Let $L : \mathbb{R}^o \times \mathcal{Y} \to \mathbb{R}$ be a loss function. The parameters of $\mathbf{F}$, denoted by θ, are optimized via

$$\min_{\theta} \mathbb{E}_{(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}} [L(\mathbf{F}(\mathbf{x}), y)]. \tag{1}$$

Specifically, the cross-entropy is leveraged $L(\mathbf{F}(\mathbf{x}), y) = -\mathbb{1}_y^\top \log(\mathbf{F}(\mathbf{x}))$, where $\mathbb{1}_y$ is the one-hot encoding vector for the label y. For simplifying notations, we use $\mathbf{F}$ (rather than $\mathbf{F}_\theta$) to denote a neural network. Table I summarizes the main notations used in the paper.

B. Basic Idea

With regard to the feature space $\mathcal{X}$, given the representation-label pair $(\mathbf{x}, y)$, the adversarial evasion attack attempts to perturb $\mathbf{x}$ into $\mathbf{x}'$, such that

$$f(\mathbf{x}') \neq y \quad \text{s.t.} \quad \mathbf{x}' \in \mathcal{S}(\mathbf{x}, \mathcal{M}_{\mathbf{x}}) \tag{2}$$

where $\mathcal{S}(\mathbf{x}, \mathcal{M}_{\mathbf{x}})$ is the set of perturbed representations derived from the non-adversarial feature representation $\mathbf{x}$ and a *manipulation set* $\mathcal{M}_{\mathbf{x}}$ (i.e. the set of manipulations that can preserve the malicious functionality of malware examples). The *perturbation vector* is denoted by $\delta_{\mathbf{x}} = \mathbf{x}' - \mathbf{x}$ with $\delta_{\mathbf{x}} \in \mathcal{M}_{\mathbf{x}}$. Since the manipulation is conducted in the feature space, the attacker needs to map $\mathbf{x}'$ back into the example space $\mathcal{Z}$ in order to obtain an executable adversarial malware example $z' \in \mathcal{Z}$.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

This is a requirement that distinguishes adversarial malware detection from adversarial machine learning in other application domains, which induces the problem of generating adversarial examples in the discrete space. It is worth mentioning that an attacker tends to modify malware examples by exploiting one or multiple feasible manipulations [5], [10], [16].

C. Threat Model

The threat model against malware classifiers and detectors can be specified by *what the attacker knows*, *what the attacker can do*, and *how the attacker wages the attack*.

1) What the Attacker Knows: There are three kinds of models from this perspective. A *black-box* attacker knows nothing about classifier f except what is implied by f's responses to the attacker's queries. A *white-box* attacker knows all kinds of information about f, including its learning algorithms, model parameters, defenses strategies, etc. A *grey-box* attacker knows an amount of information about f that resides in between the preceding two extremes. For example, the attacker may know the feature set.

2) What the Attacker Can Do: In evasion attack, the attacker only can manipulate malware examples in the test set using various kinds of manipulations, while obeying some constraints. One constraint is to preserve the malicious functionality of malware. A simplifying assumption is to consider insertion only (e.g., flipping a feature value from '0' to '1' [5], [7], [9], [22], [35], [36], [37], [38], while noting that attackers can manipulate a malware example by inserting and deleting operations [39], [40]. Since a larger manipulation set gives the attacker more freedom, we permit the attacker to conduct both *feature addition* and *feature removal*. The other constraint is to maintain the relation between features. Using the AICS'2019 malware classification challenge as an example, we note that n-gram (uni-gram, bi-gram, and tri-gram) features reflect sequences of Windows system API calls. This means that when the attacker inserts an API call into a malware example, several features related to this API call will need to be changed according to the definition of n-gram features.

3) How the Attacker Wages the Attack: Researchers generate adversarial malware examples using various machine learning-based techniques such as genetic algorithms, reinforcement learning, and generative networks [26], [40], [41], [42]. In order to generate adversarial malware examples effectively and efficiently, attackers often exploit the gradient-based methods [13], [35], [36], [43], [44]. We here briefly describe multiple types of attacks, some of which were introduced in the context of malware detection but the others were introduced in the context of image classification and then adapted to the context of malware detection.

Random Attack. We introduce this method as a baseline attack in the adversarial malware detection domain. In this attack, the attacker randomly modifies a feature at each iteration until a predefined step is reached or no more features can be manipulated.

Mimicry Attack. This attack was introduced in [16], [35], [36], [45] for studying adversarial malware detection. In this attack, the attacker perturbs or manipulates a malware example such that the resulting adversarial version mimics a chosen benign example as much as possible. In order to reduce the degree of perturbations, the attacker may select the benign example to be close to the malware example that is to be modified.

FGSM Attack. This attack was introduced in the context of image classification [13] and then adapted to the malware detection [7], [24]. FGSM perturbs a feature vector $\mathbf{x}$ in the direction of the ℓ_∞ norm of the gradients of the loss function with respect to the input, namely:

$$\mathbf{x}' = \mathrm{Proj}_{\mathcal{S}}(\mathbf{x} + \varepsilon \cdot \mathrm{sign}(\nabla_{\mathbf{x}} L(\mathbf{F}(\mathbf{x}), y))),$$

where $\varepsilon > 0$ is a scalar known as *step size*, $\nabla_{\mathbf{x}}$ indicates the derivative of the loss function $L(\mathbf{F}(\mathbf{x}), y)$ with respect to $\mathbf{x}$, and $\mathrm{Proj}_{\mathcal{S}}(\cdot)$ projects an input into $\mathcal{S}$ that denotes the shorthand of $\mathcal{S}(\mathbf{x}, \mathcal{M}_{\mathbf{x}})$.

Grosse Attack. This attack was introduced by Grosse *et al.* [5] in the context of malware detection. The attack considers sensitive features, namely the features have large positive gradients as far as the softmax output is concerned. The attack is to manipulate the absence of a feature (e.g., not making a certain API call) into the presence of the feature (i.e., making the API call), while preserving their malicious functionalities. These sensitive features can be identified by leveraging the gradients of the *softmax* output of a malware example with respect to the input.

BGA Attack and BCA Attack. In the context of malware detection, Al-Dujaili *et al.* [7] proposed two separate methods, dubbed BGA and BCA, aiming to solve:

$$\max_{\mathbf{x}' \in \mathcal{S}(\mathbf{x}, \mathcal{M}_{\mathbf{x}})} L(\mathbf{F}(\mathbf{x}'), y). \tag{3}$$

In addition, the authors considered malware examples in the binary space and restricted $\mathcal{M}_{\mathbf{x}}$ to API calls addition. Both attack methods iterate multiple steps. In each iteration, BGA increases the feature value from '0' to '1' if the corresponding partial derivative of the loss function with respect to the input is greater than or equal to the gradient's ℓ_2 norm divided by $\sqrt{dim}$, where dim is the input dimension. In contrast, BCA flips '0' to '1' for a component at the iteration corresponding to the maximum gradient of the loss function with respect to the input.

PGD Attack. We adapt the PGD method to the context of malware detection, by accommodating discrete input spaces. In contrast to the Grosse, BGA, and BCA attacks, the adapted PGD attacks permit both feature addition and feature removal. Specifically, PGD finds perturbations via an iterative procedure

$$\delta_{\mathbf{x}}^{i+1} = \mathrm{Proj}_{\hat{\mathcal{M}}_{\mathbf{x}}}\big(\delta_{\mathbf{x}}^{i} + \alpha \cdot \nabla_{\delta_{\mathbf{x}}} L(\mathbf{F}(\mathbf{x} + \delta_{\mathbf{x}}^{i}), y)\big), \tag{4}$$

where $\alpha > 0$ is the step size, $\nabla_{\delta_{\mathbf{x}}}$ is the derivative of the loss function $L(\mathbf{F}(\mathbf{x} + \delta_{\mathbf{x}}^{i}), y)$ with respect to $\delta_{\mathbf{x}}$, and $\mathrm{Proj}_{\hat{M}}\mathbf{x}$ projects perturbations into a predetermined space $\hat{M}_{\mathbf{x}}$. We set $\hat{M}_{\mathbf{x}} = [\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ for $\underline{\mathbf{u}} = minimum(\mathcal{M}_{\mathbf{x}})$ and $\overline{\mathbf{u}} = maximum(\mathcal{M}_{\mathbf{x}})$, where $minimum$ returns the component-wise minimum vector (i.e., each component of the vector corresponding to the minimum of the corresponding component values of the vectors in $\mathcal{M}_{\mathbf{x}}$) and $maximum$ returns the component-wise maximum vector.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1. PGD attack in the feature space.

Input: The feature representation-label pair $(\mathbf{x}, y)$, manipulation set $\mathcal{M}_{\mathbf{x}}$, number of iterations T, step size α
Output: Perturbed example $\mathbf{x}'$
Initialize perturbation vector $\delta_{\mathbf{x}}^0 = \mathbf{0}$;
Derive the continuous space $\hat{\mathcal{M}}_{\mathbf{x}}$ and the perturbed representation set $\mathcal{S}$;
for $i = 0$ to $T - 1$**do**
  Obtain the derivatives $\triangledown_{\delta_{\mathbf{x}}} L$ and normalize them using ℓ_p-norm where $p = 1, 2, \infty$ or the Adam method;
  Calculate $\delta_{\mathbf{x}}^{i+1}$ via the Eq.(4);
end
Obtain $\mathbf{x}'$ by mapping $\tilde{\mathbf{x}}' = \mathbf{x} + \delta_{\mathbf{x}}^T$ via Eq.(5);
return $\mathbf{x}'$.

When solving Eq.(4), we encounter two issues that need to be addressed: (i) small derivatives of $\mathbf{g} = \triangledown_{\delta_{\mathbf{x}}} L$ and (ii) mapping perturbations into discrete space $\mathcal{M}_{\mathbf{x}}$. To see issue (i), we note that by writing $\mathbf{F}$ as $\mathbf{F}(\mathbf{x}) = softmax(\mathbf{Z}(\mathbf{x}))$, we have $\partial L/\partial \delta_{\mathbf{x}} = (\mathbf{F} - \mathbb{1}_y) \cdot \partial \mathbf{Z}/\partial \delta_{\mathbf{x}}$, meaning that the derivatives approach zero when $\mathbf{F}$ predicts $\mathbf{x}$ as y with high confidence. To cope with this, researchers [28], [46] have proposed to "normalize" the derivatives using ℓ_p-norm and leveraging the steepest direction as follows:

- For $p = 1$, the direction is $\text{sign}(g_i)\mathbf{1}_i$, where i is the index corresponding to the maximum absolute value of $\mathbf{g} = (g_1, \ldots, g_{dim})$ with dim being the number of input dimension, $\mathbf{1}_i$ has the same dimension as $\mathbf{g}$ and has value 1 at the ith component and value 0 at the other components, and sign returns 1 when the input > 0, -1 when the input < 0, and 0 when the input $= 0$.
- For $p = 2$, the direction is $\mathbf{g}/\parallel \mathbf{g} \parallel_2$.
- For $p = \infty$, the direction is sign of gradients, i.e., $\text{sign}(\mathbf{g})$.

We call these variant PGD attacks PGD-ℓ_1, PGD-ℓ_2 and PGD-ℓ_∞, respectively. Note that PGD-ℓ_1 degrades to the BCA attack when only feature addition is permitted. In addition to these ℓ_p-norm based attacks, we observe that the attacker can use the Adam optimizer to accelerate the process of gradient descent (the "normalized" gradients are approximate to ± 1) [47], leading to a new variant of the PGD attack, which we call PGD-Adam.

To address the issue (ii), we introduce a mapping method to consider two settings as follows. In order to follow the direction of "normalized" gradients, let the perturbation $\delta_{\mathbf{x}}$ be continuous during the optimization process. We map the perturbed representation obtained at the last iteration, denoted by $\tilde{\mathbf{x}}' = (\tilde{x}'_1, \ldots, \tilde{x}'_{dim})$, into $\mathcal{S}$ by selecting its closest neighbor $\mathbf{x}' = (x'_1, \ldots, x'_{dim})$ such that

$$\mathbf{x}' = \underset{\mathbf{x}' \in \mathcal{S}}{arg\,min} \parallel \mathbf{x}' - \tilde{\mathbf{x}}' \parallel_1 = \underset{\mathbf{x}' \in \mathcal{S}}{arg\,min} \sum_{i=1}^{dim} |x'_i - \tilde{x}'_i|. \quad (5)$$

Geometrically speaking, Eq.(5) says that for the ith dimension, x'_i is the feasible scalar closest to $\tilde{x}'_i$. Algorithm 1 summarizes the PGD attacks in the feature space.

III. Adversarial Malware Defense

A. Guiding Principles

These principles are geared to neural network classifiers, which are chosen as our focus because deep learning techniques are increasingly employed in the domain of malware detection/classification, but their vulnerability to adversarial evasion attack has yet to be tackled [48].

1) Principle 1: Knowing the Enemy: This principle says that the defender should strive to extract useful information about the attacks as much as possible as the information will offer insights on designing countermeasures. Threat models are a standard approach to modeling attacks. Moreover, it is possible to design some indicators of adversarial examples. On the other hand, the attack method and manipulation set may not be known to the defender. This means that whenever possible, the defender has to simulate them.

2) Principle 2: Bridging Grey-Box Vs. White-Box Gap: In grey-box attacks, the attacker knows some information about the feature set and therefore can train a surrogate classifier $\hat{f} : \mathcal{X} \rightarrow \mathcal{Y}$ from a training set (where the realization of $\hat{f}$ is a neural network $\hat{\mathbf{F}}$), leveraging the transferability from $\hat{f}$ to f to generate adversarial examples. Consider an input $\mathbf{x}$ for which a grey-box attacker generates perturbations using

$$\hat{\delta}_{\mathbf{x}} = \underset{||\delta_{\mathbf{x}}|| \leq \eta \,\wedge\, \delta_{\mathbf{x}} \in \mathcal{M}_{\mathbf{x}}}{arg\,max} L(\hat{\mathbf{F}}(\mathbf{x} + \delta_{\mathbf{x}}), y),$$

where η is an upper bound and possibly large. Based on the degree of perturbations, we consider two cases: (i) η is small and (ii) η is large. We further assume that the optimal perturbation vector $\delta_{\mathbf{x}}$ of $\mathbf{F}$ exists.

In case (i) or when η is small, the change to the loss of f incurred by $\hat{\delta}_{\mathbf{x}}$ is

$$\begin{aligned} |\Delta L| &= \left| L(\mathbf{F}(\mathbf{x} + \hat{\delta}_{\mathbf{x}}), y) - L(\mathbf{F}(\mathbf{x}), y) \right| \\ &\approx \left| \triangledown L(\mathbf{F}(\mathbf{x}), y)^\top \hat{\delta}_{\mathbf{x}} \right| \leq \max_{||\delta|| \leq \eta} \left| \triangledown L(\mathbf{F}(\mathbf{x}), y)^\top \delta \right| \\ &= \eta ||\triangledown L||_*, \end{aligned}$$

where the approximation is derived using the first-order Taylor expansion at point $\mathbf{x}$, $\triangledown$ is the operator for computing partial derivatives of the loss function with respect to the input of neural network $\mathbf{F}$, and "$||\cdot||_*$" is the dual norm of $||\cdot||$.

In case (ii) or when η is large, we derive

$$\begin{aligned} |\Delta L| &= \left| L(\mathbf{F}(\mathbf{x} + \hat{\delta}_{\mathbf{x}}), y) - L(\mathbf{F}(\mathbf{x}), y) \right| \\ &= \left| \int_0^{\hat{\delta}_{\mathbf{x}}} \triangledown L(\mathbf{F}(\mathbf{x} + \delta), y) d\delta \right| \\ &= \left| \int_0^1 \triangledown L(\mathbf{F}(\mathbf{x} + t\hat{\delta}_{\mathbf{x}}), y)^\top \hat{\delta}_{\mathbf{x}} dt \right| \\ &\leq \eta \sup_{||\delta|| \leq \eta} \|\triangledown L(\mathbf{F}(\mathbf{x} + \delta), y)\|_*. \end{aligned}$$

The preceding observation shows that corresponding to the same perturbation upper bound η, the loss incurred by grey-box attacks is upper bounded by the loss incurred by white-

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

box attacks. This suggests us to focus on the robustness of classifier f against the optimal white-box attack.

3) Principle 3: Not Putting All Eggs in One Basket: This is suggested by the observation that no single classifier may be effective against all kinds of attacks. An ensemble can be built by many methods (e.g., bagging, boosting, or stacking) [49]. For example, *random subspace* [50] is seemingly particularly suitable for formulating malware classifier ensembles owing to the high dimensional feature vector of malware, which indicates a high vulnerability of classifiers to adversarial malware examples [16], [51].

Formally, an ensemble $f_{en} : \mathcal{X} \to \mathcal{Y}$ contains a set of neural networks $\{\mathbf{F}_i\}_{i=1}^{l}$, namely $\{\mathbf{F}_i : \mathcal{X} \to \mathbb{R}^o\}$ for $1 \le i \le l$. Given a test example $\mathbf{x}$, we treat the base model equally, as suggested by the study [18], [52], and the voting method is

$$\mathbf{F}_{en}(\mathbf{x}) = \frac{1}{l}\sum_{i=1}^{l}\mathbf{F}_i(\mathbf{x}).$$

We obtain the predicted label by $f_{en} = arg\,max_{j\in\mathcal{Y}}\mathbf{F}_{en}$.

4) Principle 4: Using Transformation Against Perturbation: In typical applications, the defender does not know what kinds of evasion attacks are waged by the attacker. These attacks can produce a spectrum of perturbations, from manipulating a few features (e.g., the PGD-ℓ_1 attack) to manipulating a large number of features (e.g., the FGSM attack). Moreover, we may give higher weights to the transformation techniques that can simultaneously reduce the degrees of multiply types of perturbations such as ℓ_∞ norm, ℓ_1 norm, or ℓ_2 norm. This suggests us to use the *binarization* technique [25], [53]: When the feature value of the ith feature, denoted by x_i, is smaller than a threshold $\mathbf{\Theta}_i$, we binarize x_i to 0; otherwise, we binarize x_i to 1.

5) Principle 5: Using Vaccine: We harden a model incorporating the known *minmax* adversarial training:

$$\min_{\theta}\ \mathbb{E}_{(\mathbf{x},y)\in\mathcal{X}\times\mathcal{Y}}\left[L(\mathbf{F}(\mathbf{x}),y) + \max_{\mathbf{x}'\in\mathcal{S}} L(\mathbf{F}(\mathbf{x}'),y)\right]. \quad (6)$$

Al-Dujaili *et al.* [7] instantiate this method by using attacks with feature addition solely (e.g., BGA). In order to accommodate more manipulations, we solve the problem of inner maximization using the Adam optimizer (see Section II-C3). Given the issue of local minima, we run the inner maximizer several times, each with a random initial point near the training data, and then select the point that maximizes the loss function of L.

It is worth mentioning that in the AICS'2019 Challenge, the defender does not know the manipulation set $\mathcal{M}_{\mathbf{x}}$ and thus cannot derive $\mathcal{S}$. In this case, we propose training malware classifiers by applying small perturbations to the feature representations of malware examples (*without* necessarily preserving their malicious functionalities). This would benefit model generalization [13], [29]. Let a norm $||\cdot||$ measure the perturbation $\delta_{\mathbf{x}}$ with η bounded. We have $\max_{||\delta_{\mathbf{x}}||\le\eta} L(\mathbf{F}(\mathbf{x}+\delta_{\mathbf{x}}),y) \approx L(\mathbf{F}(\mathbf{x}),y) + \eta||\triangledown L(\mathbf{F}(\mathbf{x}),y)||_*$, leading to $|L(\mathbf{F}(\mathbf{x}'),y) - L(\mathbf{F}(\mathbf{x}),y)| \le \eta||\triangledown L(\mathbf{F}(\mathbf{x}),y)||_*$. Therefore, adversarial regularization assures that small perturbations do not change the prediction significantly.

Principle 6: Preserving Semantics: This suggests us to strive to learn neural network models that are sensitive to malware semantics, but not the perturbations because adversarial examples must retain the malicious functionality of original malware. Specifically, we propose using denoising auto-encoder to learn semantics-preserving representations, rendering neural network less sensitive to perturbations. A DAE $ae = d \circ e$ unifies two components: an encoder $e : \mathcal{X} \to \mathcal{H}$ that maps an input $M(\mathbf{x})$ to a latent representation $\mathbf{r} \in \mathcal{H}$ and a decoder $d : \mathcal{H} \to \mathcal{X}$ that reconstructs $\mathbf{x}$ from $\mathbf{r}$, where the $\mathcal{H}$ is the latent representation space and M refers to some operations applied to $\mathbf{x}$ (e.g., adding Gaussian noises to $\mathbf{x}$). Vincent *et al.* [32] showed that the lower bound of the *mutual information* between $\mathbf{x}$ and $\mathbf{r}$ is maximized when the reconstruction error is minimized. In the case of Gaussian noise $\epsilon \sim \mathcal{N}(0,\sigma^2)$ and reconstruction loss

$$\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)}\|ae(\mathbf{x}+\epsilon) - \mathbf{x})\|_2^2, \quad (7)$$

Alain and Bengio [54] showed that the optimal $ae^*(\mathbf{x})$ is

$$ae^*(\mathbf{x}) = \frac{\mathbb{E}_\epsilon[p(\mathbf{x}-\epsilon)(\mathbf{x}-\epsilon)]}{\mathbb{E}_\epsilon[p(\mathbf{x}-\epsilon)]}, \quad (8)$$

where $p(\cdot)$ is the probability density function. Eq.(8) says that representations of a well-trained DAE are insensitive to $\mathbf{x}$ because of the weighted average from the neighborhood of $\mathbf{x}$, which is reminiscent of the *attention* mechanism [55]. This means that DAE can handle certain types of small perturbations. To learn a DAE model, we leverage two kinds of noise: (i) *Salt-and-pepper noise* ϵ: A small fraction of elements of original example $\mathbf{x}$ are randomly selected, and then set their values as their respective minimum or maximum. (ii) *Adversarial perturbation* $\delta_{\mathbf{x}}$: A perturbation $\delta_{\mathbf{x}}$ is added to $\mathbf{x}$ such that classifier f or base classifier f_i misclassifies $\mathbf{x}' = \mathbf{x} + \delta_{\mathbf{x}}$. Given a training example $\mathbf{x}$ over the feature space $\mathcal{X}$, the risk of a denoising auto-encoder is

$$\min_{\tilde{\theta},\xi}\mathbb{E}_{\mathbf{x}\in\mathcal{X}}[L_{ae}(\mathbf{x}, ae(\mathbf{x}+\epsilon)) + L_{ae}(\mathbf{x}, ae(\mathbf{x}'))], \quad (9)$$

where $L_{ae} : \mathcal{X}\times\mathcal{X}\mapsto\mathbb{R}$ calculates the mean-square error, the learnable parameters $\tilde{\theta}$ and ξ respectively belongs to the encoder and decoder.

B. Turning Principles Into a Framework

The principles discussed above guide us to propose a framework for adversarial malware classification and detection, which is highlighted in Figure 1 and elaborated below. Specifically, we first examine whether the attacks have some useful information that could to be incorporated via a proper preprocessing (according to Principle 1). We propose using an ensemble f_{en} of classifiers $\{f_i\}_{i=1}^{l}$ (according to Principle 3), which are trained from random subspace of the original feature space. Each classifier f_i is hardened by three countermeasures: input transformation via binarization (according to Principle 4); adversarial training/regularization models on the attacks using Adam optimizer (dot arrows in Figure 1, according to Principle 2 and 5); semantics-

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

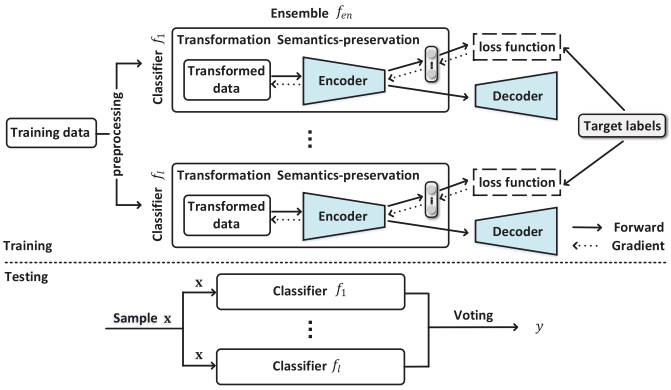


Fig. 1. Overview of the adversarial malware defense framework. In the training phase, an ensemble of l neural network classifiers are trained, with each classifier hardened by three countermeasures (i.e., input transformation, semantics-preserving, and adversarial training on the transformed data). .

preservation is achieved via an encoder and a decoder (according to Principle 6). In order to attain adversarial training and at the same time semantics-preservation, we learn classifier f_i via block coordinate descent to optimize different components of the model.

Algorithm 2 integrates all pieces for training individual classifiers. The training procedure consists of the following steps. (i) Given a training set (X, Y), we randomly select a ratio Λ of sub-features to the feature set, and then transform X into $\overline{X}$ via the binarization discussed above. (ii) We sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^N$ from $(\overline{X}, Y)$, and calculate adversarial examples $\mathbf{x}'_i$ for $\mathbf{x}_i \in \{\mathbf{x}_i\}_{i=1}^N$ according to Lines 5-9 in Algorithm 2. (iii) We pass the $\{\mathbf{x}'_i\}_{i=1}^N$ through the denoising auto-encoder to compute the reconstruction loss with respect to the target $\{\mathbf{x}_i\}_{i=1}^N$ via Eq.(9), and update the parameters of the denoising auto-encoder. (iv) We pass both the $\{\mathbf{x}_i + \delta_{\mathbf{x}_i}\}_{i=1}^N$ and $\{\mathbf{x}_i\}_{i=1}^N$ through the neural networks to compute the classification error with respect to the ground-truth label $\{y_i\}_{i=1}^N$ via Eq.(6), and update the parameters of the classifier via backpropagation. Note that Steps (ii)-(iv) are performed in a loop. The output of the training algorithm is a neural network classifier.

Algorithm 2. Training classifier f_i

Input: The training set (X, Y), number of repeat times K, epoch N_{epoch} and mini-batch size N.

1 Select a ratio Λ of sub-features from the feature set;
2 Transform input X to $\overline{X}$ via binarization;
3 **for** $epoch = 1$ to N_{epoch} **do**
4 Sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^N$ from the $(\overline{X}, Y)$;
5 **for** $k = 1$ to K **do**
6 Apply slight salt-and-pepper noises to $\{\mathbf{x}_i\}_{i=1}^N$;
7 Derive the perturbed representation $\{\mathbf{x}'^k_i\}_{i=1}^N$ via Algorithm 1 using Adam method;
8 **end**
9 Select $\mathbf{x}'_i$ from $\{\mathbf{x}'^k_i\}_{k=1}^K$ for $\mathbf{x}_i$ $(i = 1, \ldots, N)$ to maximize the cross-entropy loss;
10 Calculate the reconstruction loss via Eq.(9);
11 Backpropagate the loss and update the denoising autoencoder parameters;
12 Calculate the adversarial training loss via Eq.(6);
13 Backpropagate the loss and update classifier parameters;
14 **end**

IV. Validating Framework Via Drebin Dataset

We validate the effectiveness of the framework using the Drebin dataset of Android malware [56], while considering 11 grey-box attacks and 9 white-box attacks. This dataset also applied by former studies in the domain of adversarial malware detection [5], [16], [23], [57].

A. Data Pre-Processing

Dataset. The Drebin dataset [56] contains 5615 malicious Android packages (APKs), and also provides features of 123453 benign examples, together with their SHA256 values but not the examples themselves. All samples were labeled using the VirusTotal service [58] before the year 2015. An example was treated as malicious if there are at least two scanners say it is malicious, and is treated as benign if no scanners detect it [56]. Because the VirusTotal may update the detection result along with the time [59], we consider relabeling the APKs. We download benign applications corresponding to the given SHA256 values from the APK markets (e.g., Google Play, AppChina, etc.), and collect 54829 APKs in total. We send all of these examples (i.e., malicious and benign alike) to the VirusTotal service again. Surprisingly, 12496 benign APKs are detected as malicious (rather than benign) by at least one scanners, and most of them are detected as *Adware* or *Trojan*; this suggests that the original Drebin training set has been contaminated by the *poisoning attack*. This may be caused by some of the following reasons: (i) Virustotal updates the scanners over the time; (ii) Virustotal updates the report of a file when a user requires to rescan the file; (iii) after an update, the previous report cannot be obtained anymore. We thus remove these 12496 benign examples from the original benign dataset, leaving 42333 benign APKs. The resulting dataset contains 5615 malicious APKs and 42333 benign APKs, namely 47948 examples in total. We randomly split the dataset into three disjoint sets for training (60%), validation (20%), and test (20%), respectively.

Feature Extraction. APK is an archive file containing *AndroidManifest.xml*, *classes.dex*, and others (e.g., *res*, *assets*). The *manifest* file describes an APK's information, such as the name, version, announcement, library files used by the application. The source code is compiled to build the *.dex* file which is understandable by the Java Virtual Machine. The Drebin dataset has eight subsets of features, including four subsets of features extracted from *AndroidManifest.xml* (denoted by S_1, S_2, S_3, S_4, respectively) and four subsets of features extracted from the disassembled dexcode (denoted by S_5, S_6, S_7, S_8, respectively). More specifically, (**i**) S_1 contains features corresponding to the access of an APK to the hardware of a smartphone (e.g., camera, touchscreen, or GPS module); (**ii**) S_2 contains features corresponding to the permissions requested by the APK in question; (**iii**) S_3 contains features corresponding to the application components (e.g., *activities*, *service*, *receivers*, etc.); (**iv**) S_4 contains features corresponding to the APK's communications with the operating system; (**v**) S_5 contains features corresponding to the

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

critical system API calls, which cannot run without appropriate permissions or the *root* privilege; (**vi**) S_6 contains features corresponding to the used permissions; (**vii**) S_7 contains features corresponding to the API calls that can access sensitive data or resources on a smartphone; (**viii**) S_8 contains features corresponding to IP addresses, hostnames and URLs found in the disassembled code.

In order to extract applications' features, we use Androgurad 3.3.5, a reverse engineering toolkit for APK analysis [60]. Note that 141 APKs cannot be analyzed. Moreover, a feature selection is conducted to remove those low-frequency features for the sake of computational efficiency. As a result, we keep 10000 features with high frequencies. The APK is mapped on the feature space as a binary feature vector, where '1' ('0') represents the presence (absence) of a feature in the APK.

B. Training Classifiers

Classifiers. In order to validate the defense framework, we use and compare five classifiers: (**i**) the basic DNN with no effort made to defend adversarial examples; (**ii**) hardened DNN incorporating adversarial training with known manipulation set (dubbed Adversarial Training), which manifests Principle 2 (grey-box attacks can be bounded by the worst-case white-box attack) and Principle 5 (min-max adversarial training); (**iii**) hardened DNN incorporating adversarial regularization because the defender may know nothing about the manipulation set, which is true in the case of AICS'2019 Challenge (dubbed Adversarial Regularization); (**iv**) Denoising Auto-Encoder (DAE) based-classifier, which manifests Principle 6 (semantics-preserving representations); (**v**) classifier hardened by both Adversarial Training and DAE (dubbed AT+DAE); (**vi**) ensemble of AT+DAE classifiers in the random subspace (manifesting Principle 3, dubbed Ensemble AT+DAE). For Principle 1 (i.e., knowing your enemy), we will simulate attacks in the next subsection. Since we use binary feature vector, Principle 2 (binarization) is not applicable.

Hyper-Parameters Setting. We use DNNs with two fully-connected hidden layers (each layer having 160 neurons) with ReLU activation function. All classifiers are optimized by using Adam with epochs 150, mini-batch size 128, and learning rate 0.001. For Adversarial Training, the inner maximization is optimized by using Adam with learning rate 0.02 and iteration steps $T = 100$ to search adversarial examples as many as possible. For Adversarial Regularization, we set the learning rate as 0.01 for Adam and conduct preliminary experiments to tune a proper iteration step T. Finally, we set $T = 60$. We use an ensemble of 5 base classifiers. Our preliminary experiments suggest us to learn base classifiers from the entire training set and the entire feature set. Unless with special mentioning, all classifiers that require to solve the inner maximization are trained without random starting points so as to ease the analysis (i.e., $K = 0$).

C. Attack Experiments and Classification Results

We present threat models specified by whether the attacker wages grey-box or white-box attacks, and constraints on the attacker's manipulation set.

Grey-box vs. White-box Attacks. We consider two attack scenarios. (**i**) *Grey-box attacks:* In this setting, we simulate the attack scenario of the AICS'2019 Challenge organizers. That is, the attacker knows the dataset, feature set, but not the defender's learning algorithm. The attacker generates adversarial examples from a surrogate classifier. We consider a surrogate model of two fully-connected hidden layers (200 neurons each layer) and learn the model on the training set. (**ii**) *White-box attacks:* In this setting, the attacker knows everything about the malware detector. Therefore, the adversarial examples are directly generated from the corresponding malware detector.

Manipulations Constraints. Given an APK, we consider both *incremental* and *decremental* manipulations. The incremental manipulation allows the attacker to insert some objects (e.g., *activity*) into an APK example to avoid detection. The decremental manipulation allows the attacker to hide some objects (e.g., *activity*) to avoid detection. In any case, the adversarial example should preserve the malicious functionality of the malware from which the adversarial example is generated.

When the attacker uses incremental manipulations, the attacker can insert some manifest features (e.g., request extra permissions and hardware, state additional *services*, *Intent-filter*, etc.). However, some elements are hard to insert, such as *content-provider*, because the absence of URI will corrupt an APK example. With respect to the *.dex* file, a dead function or class (which is never called) containing specified system APIs can be injected without destroying the APK example. The similar means can be performed for the *string* injection (e.g., IP address), as well.

When the attacker uses decremental manipulations, the APK's information in the *xml* files can be changed (e.g., package name). However, it is impossible to remove *activity* entirely because an *activity* may represent a class implemented in the *.dex* code. Nevertheless, we can rename an *activity* and change its relevant information (e.g., *activity label*), while noting that the related components in the *.dex* should be modified accordingly. The other components (e.g., *service*, *provider* and *receiver*) also can be modified in the similar fashion, and the resource files (e.g., images, icons) can be manipulated as well. In terms of *dexcode*, the method names and class names that are defined by developers could be modified, too. Note that the corresponding statement, instantiation, reference, and announcements should be changed accordingly. Moreover, user-specified *strings* can be obfuscated using encryption and the cipher-text will be decrypted at running time. Further, the attacker can hide *public* and *static* system APIs using Java reflection and encryption together. This is shown by the example in List 1. All of the modifications mentioned above only obfuscate an APK without changing its functionalities.

One challenge is that the attacker needs to perform fine-grained manipulations on compiled files automatically at scale, while preserving the functionalities of malware examples. This important because a small change in a malware example can render the file unexecutable. Since Android APIs have upgraded multiple times in the past 5 years, the attacker has to inject compatible APIs into an APK when manipulating

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

TABLE II
OVERVIEW OF MANIPULATIONS ON FEATURE SPACE, WHERE ✓(✘) INDICATES THAT THE FEATURE ADDITION OR REMOVAL OPERATION CAN (CANNOT) BE PERFORMED ON FEATURES IN THE CORRESPONDING SUBSET

Feature sets		Addition	Removal
manifest	S_1 Hardware	✓	✗
	S_2 Requested permissions	✓	✗
	S_3 Application components	✓	✓
	S_4 Intents	✓	✗
dexcode	S_5 Restricted API calls	✓	✓
	S_6 Used permission	✗	✗
	S_7 Suspicious API calls	✓	✓
	S_8 Network addresses	✓	✓

a malware example. The preservation of malicious functionalities may be estimated by using a dynamic malware analysis tool, (e.g., Sandbox).

Mapping Manipulations to Feature Space. The aforementioned manipulations modify static Android features, such as API calls and components in the manifest file. Two kinds of perturbations can be applied to the Drebin feature space. (i) *Feature addition.* The attacker can increase the feature values (e.g., flipping '0' to '1') of appropriate objects, such as components (e.g., *activity*), system APIs, and IP address. (ii) *Feature removal.* The attacker can flip '1' to '0' by removing or hiding objects (e.g., *activity*, APIs.) Table II summarizes our manipulations in the Drebin feature space. We observe that neither feature addition nor feature removal can be applied to S_6 because these features depend on S_2 and S_5, meaning that modifications on S_2 or S_5 may cause changes to features in S_6.

Evasion Attacks Setting. We randomly select 800 malware examples from the test set to wage evasion attacks by using the attack algorithms described in Section II-C. In the settings of Random, Grosse, BGA, BCA, and ℓ_1-PGD attacks, we iterate these algorithms until reaching a predefined maximum number of steps, while noting that Grosse, BGA, and BGA attacks leverage feature addition only. For waging the Mimicry attack, in order to increase its effectiveness, we use N_b benign examples to guide the perturbation of a single malware example, leading to N_b perturbed examples; then, we select a resulting example such that it causes the mis-classification with the smallest perturbation. Therefore, we denote this attack as Mimicry$\times N_b$. For other attacks, we set $\varepsilon = 1.0$ for the FGSM attack. In ℓ_∞ norm and Adam based PGD attacks, the step size is $\alpha = 0.01$ with iterative times 100. The ℓ_2 norm PGD attack is performed for 100 iterations with step size 1.0.

Experimental Validation of Functionality. In order to test whether or not perturbations in the feature space render to executable files in the example space, we use Cuckoodroid [61] to install and run APKs in an Android emulator. Owing to efficiency considerations, we randomly select 10 malware APKs and generate their perturbed APKs using the PGD-Adam attack against the Basic DNN model. Among the 10 original (i.e., unperturbed) APKs, all of them can be loaded but 2 cannot run in the Android emulator. Among the 10 perturbed examples, all of them can be loaded but 5 of them cannot run (and 2 of these 5 correspond to the 2 original APKs that cannot run). This means that more research is needed in order to systematically assure that perturbation can indeed preserve the functionalities of malware examples, which is unique to adversarial malware detection [9], [11].

TABLE III
EFFECTIVENESS OF THE DEFENSE FRAMEWORK WHEN THERE ARE NO ADVERSARIAL ATTACKS.

Defense	FNR (%)	FPR (%)	Accuracy (%)
Basic DNN	3.684	0.320	**99.28**
Adversarial Training	3.246	1.777	98.05
Adversarial Regularization	4.737	**0.190**	99.27
DAE	3.246	0.450	99.22
AT+DAE	3.246	1.694	98.12
Ensemble AT+DAE	**2.456**	2.464	97.54

D. Experimental Results

The Case of No Adversarial Attacks. Table III summarizes the classification results on the test set, which are measured with the standard metrics of False Negative Rate (FNR), False Positive Rate (FPR), and classification Accuracy (i.e., the percentage of the test examples that are classified correctly) [62]. We observe that when compared with the Basic DNN, Adversarial Training achieves a lower FNR (0.438% lower) but a higher FPR (1.457% higher). A similar tendency is exhibited by DAE, AT+DAE and Ensemble AT+DAE. This can be explained as follows: by injecting adversarial malware examples into the training set, the learning process makes the model search for malware examples in a bigger space, explaining the drop in FNR and increase in FPR and therefore a slightly drop (1.74%) in the classification accuracy. Adversarial Regularization achieves a comparable classification accuracy as Basic DNN, but the highest FNR among the classifiers we considered. This is caused by the fact that DNN is regularized using small perturbations applied to both benign and malicious samples. In summary, we draw:

Insight 1. In the absence of adversarial attacks, Adversarial Training and DAE can detect more malware examples than the Basic DNN (because of their smaller FNR), at the price of a small side-effect in the FPR and therefore the classification accuracy; Adversarial regularization achieves comparable accuracy as the Baisc DNN while increasing the FNR.

The Case of Grey-box Attacks. Table IV reports the classification results of the defense framework against grey-box attacks. We make the following observations. First, Basic DNN cannot defend against evasion attacks and is completely ruined by attacks that include Mimicry, FGSM, Grosse, BGA, BCA, PGD-ℓ_1, and PGD-ℓ_∞. Second, Adversarial Training significantly enhances the robustness of DNN, achieving the accuracy of 86.13% and 85.63% against the Mimicry$\times 1$ and Mimicry$\times 10$ attack respectively and a 100% accuracy against the other 6 attacks (i.e., BGA, BCA and 4 variants of PGD). Third, Adversarial Regularization, without seeing any adversarial examples, can defend against FGSM, PGD-ℓ_∞, PGD-ℓ_2 and PGD-Adam attacks, but are not effective against attacks such as Grosse, BCA, and PGD-ℓ_1. A similar phenomenon is observed for DAE. Nevertheless, when using Adversarial Training and

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
EFFECTIVENESS OF THE DEFENSE FRAMEWORK AGAINST GREY-BOX ADVERSARIAL MALWARE EVASION ATTACKS

Attack	Accuracy (%)					
	Basic DNN	Adversarial Training (AT)	Adversarial Regularization	DAE	AT+DAE	Ensemble AT+DAE
No Attack	96.63	97.00	95.63	96.88	96.50	**97.75**
Random Attack	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Mimicry×1	53.88	86.13	52.75	56.88	91.50	**96.13**
Mimicry×10	35.25	85.63	34.88	52.63	85.13	**89.88**
FGSM [13]	4.00	97.50	95.88	96.88	96.75	**98.00**
Grosse [5]	1.13	97.00	11.75	65.13	97.63	**99.38**
BGA [7]	0.25	**100.0**	71.13	**100.0**	**100.0**	**100.0**
BCA [7]	0.25	**100.0**	49.50	58.00	**100.0**	**100.0**
PGD-ℓ_1	0.25	**100.0**	43.88	53.88	**100.0**	**100.0**
PGD-ℓ_2	58.63	**100.0**	99.75	**100.0**	**100.0**	**100.0**
PGD-ℓ_∞	0.25	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
PGD-Adam	52.50	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

TABLE V
EFFECTIVENESS OF THE DEFENSE FRAMEWORK AGAINST WHITE-BOX ADVERSARIAL MALWARE EVASION ATTACKS

Attack	Accuracy (%)					
	Basic DNN	Adversarial Training (AT)	Adversarial Regularization	DAE	AT+DAE	Ensemble AT+DAE
Mimicry×10	11.63	68.25	14.88	40.88	69.13	**79.75**
FGSM [13]	0.00	97.00	95.00	96.88	96.50	**97.75**
Grosse [5]	0.00	60.75	16.63	35.50	81.13	**91.75**
BGA [7]	0.00	97.00	91.50	74.00	96.50	**97.50**
BCA [7]	0.00	61.13	16.63	35.38	81.50	**91.75**
PGD-ℓ_1	0.00	69.50	21.88	51.00	81.25	**88.50**
PGD-ℓ_2	3.00	**93.63**	82.13	89.75	91.13	91.63
PGD-ℓ_∞	0.00	**90.38**	89.75	35.38	85.50	73.63
PGD-Adam	1.13	**95.13**	89.63	88.25	92.88	90.00

DAE together, namely AT+DAE, the defense achieves the highest robustness against evasion attacks than using Adversarial Training and DAE individually, except for the Mimicry×10 attack and FGSM attack (encountering a ∼1% decrease). Fourth, the Ensemble AT+DAE consists of five AT+DAE classifiers and achieves the highest robustness among the considered defenses against the attacks investigated. In summary, we draw:

Insight 2. Under grey-box attack scenario, Adversarial Training is an effective defense against evasion attacks; DAE offers some defense capability that may not be offered by Adversarial Training; using an ensemble of five AT+DAE classifiers is more effective than using a single AT+DAE classifier against evasion attacks; Without knowing the attacker's manipulation set, Adversarial Regularization enhances the robustness of Basic DNN but cannot defend attacks such as Grosse.

The Case of White-box Attacks. Table V presents the classification results against white-box attacks. We make the following observations. (i) All attacks can almost completely evade Basic DNN, but the Mimicry attack is, relatively speaking, less effective because this attack leverages less information about the classifiers than what the other attacks do. (ii) Adversarial Training is effective against the FGSM attack, BGA attack and PGD-Adam attack, but not effective against the Grosse attack, BCA attack, and PGD-ℓ_1 attack because these attacks manipulate a few features when generating adversarial examples and these manipulations are unlikely perceived by Adversarial Training (owing to the fact that Adversarial Training penalizes adversarial spaces searched by Adam optimizer). (iii) As expected, Adversarial Regularization is less effective than Adversarial Training. Adversarial Regularization achieves a 91.50% accuracy against the white-box BGA attack, in contrast to the 71.13% accuracy against the grey-box BGA attack. This is counter-intuitive but can be attributed to the fact that Adversarial Regularization may render some gradient-based methods, such as BGA, useless, which is a phenomenon known as *gradient-masking* [63], [64], [65]. (iv) AT+DAE achieves considerable robustness against those attacks, with at least an 81.13% accuracy except for the Mimicry×10 attack, which defeats the AT +DAE defense because Mimicry can make adversarial malware examples similar to benign ones [16]. (v) The ensemble of AT +DAE defense achieves the highest accuracy against the Mimicry×10, the Grosse attack and the BCA attack than the other defenses, with about 10% higher accuracy when compared with the AT+DAE defense. However, the ensemble of AT+DAE achieves lower accuracy than AT+DAE against the PGD-ℓ_2 attack, the PGD-ℓ_∞ attack, and the PGD-Adam attack. This may be caused by the fact that the base model AT+DAE cannot effectively mitigate these attacks. In summary, we draw:

Insight 3. Adversarial Training cannot effectively defend against white-box attacks that were not considered in the training phase; DAE can be useful when adversarial training is not effective; employing ensembles can further improve the robustness against certain white-box attacks. That is, no defenses can defeat all white-box attacks effectively.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
ACCURACY (%) AND MACRO F1 SCORE (%) ARE REPORTED WITH A 95% CONFIDENCE INTERVAL WITH RESPECT TO THE RATIO PARAMETER (%), WHERE '—' MEANS LEARNING A CLASSIFIER USING THE ORIGINAL TRAINING SET

Ratio (%)	Accuracy (%)	Macro F1 (%)
—	93.20±1.04	85.52±1.12
30	92.86±0.75	85.47±1.04
40	92.38±1.00	84.87±1.07
50	92.21±0.60	84.87±1.00
60	92.48±1.12	84.62±1.01

V. APPLICATION TO AICS'2019 CHALLENGE WHEN KNOWING NOTHING ABOUT ATTACKS

The challenge is in the context of adversarial malware classification, namely devising evasion-resistant, machine learning based malware classifiers. The dataset, including both the training set and the test set, consists of feature vectors extracted from Windows malware examples, each of which belongs to one of the following five classes: *Virus*, *Worm*, *Trojan*, *Packed malware*, and *AdWare*. For each example, the features are collected by the challenge organizer via dynamic analysis, including the Windows API calls and further processed unigram, bigram, and trigram API calls. The feature names (e.g., API calls) and the class labels are "obfuscated" by the challenge organizer as integers, while noting the obfuscation preserves the mapping between the features and the integers representation of them. For example, three API calls are represented by three unique integers, say 101, 102, and 103; then, a trigram API call "101;102;103" means a sequence of API calls 101, 102, and 103. In total there are 106428 features.

The test set consists of adversarial examples and non-adversarial examples (i.e., unperturbed malware examples). Adversarial examples are generated by a variety of perturbation methods, which are not known to the participating teams. However, the ground-truth labels of the test examples are not given to the participating teams. This means that the participating teams cannot calculate the accuracy of their detectors by themselves. Instead, they need to submit their classification results (i.e., labels on the examples in the test set) to the challenge organizer, who will calculate the classification score of each participating team. The Challenge organizer decided to use the Macro F1 score as the classification accuracy metric. The Macro F1 score is the unweighted mean of the F1 score [66] for each class of objects in question (i.e., type of malware in this case). The final score is the Harmonic mean upon the two Macro F1 scores, namely the one for the adversarial examples in the test data and the other for the non-adversarial examples in the test data. Given these two numbers, say a_1 and a_2, their harmonic mean $\frac{2a_1a_2}{a_1+a_2}$.

A. Basic Analysis of the AICS'2019 Challenge

Is the Training Set Imbalanced? The training set consists of 12536 instances, and the test set consists of 3133 instances. The training set contains 8678 instances in class '0,' 1883 instances in class '1,' 771 instances in class '2,' 692 instances in class '3,' and 512 instances in class '4'. We can calculate the maximum ratio between the number of instances in different classes is 16.95, indicating that the training set is highly imbalanced. In order to cope with the imbalance in the training set, we use the *Oversampling* method to replicate randomly selected feature vectors from a class with a small number of feature vectors. The replication process ends until the number of feature vectors is comparable to that of the largest class (i.e., the class with the largest number of feature vectors), where "comparable" is measured by a predefined ratio. In order to see the effect of this ratio, we use a 5-fold cross validation on the training set to investigate the impact of this ratio. The classifier consists of neural networks with two fully-connected layers (each layer having 160 neurons with the ReLU activation function), which are optimized via Adam with epochs 50, mini-batch size 128, learning rate 0.001. The model is selected when achieving the best Macro F1 score on the validation set. Table VI shows that the Macro F1 score decreases as the oversampling ratio of minority classes increases. In order to make each mini-batch of training data contain examples from all classes, which would be critical in multiclass classification, our experience suggests us to select the 30% ratio.

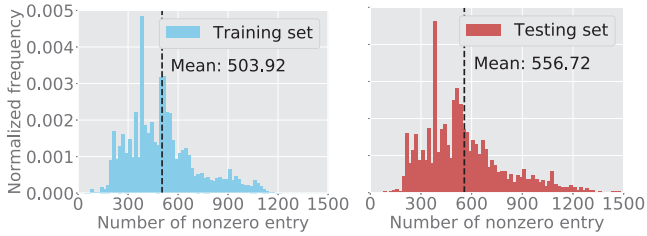


Fig. 2. Histogram of the normalized frequency of the number of nonzero entries of feature vectors in the training set vs. test set. The dashed line represents the mean value.

Are There Simple Indicators of Adversarial Examples? In the first test set published by the challenge organizer, we see negative values for some features. These negative values would indicate that they are adversarial examples. In the revised test set provided by the challenge organizer, there are no negative feature values, meaning that there are no simple ways to tell whether an example is adversarial or not. In spite of this, we can speculate on the count of perturbed features by comparing the number of nonzero entries corresponding to feature vectors in the training set and feature vectors in the test set. Figure 2 shows the normalized frequency of the number of nonzero entries of feature vectors in the training set vs. test set. We observe that their normalized frequencies are similar except that some test examples have more nonzero entries. Their mean values are close and are much smaller than the input dimension (106428), suggesting that the average degree of perturbed features may be small.

B. Classification Results: Challenge Winner

We train 10 deep neural network classifiers to formulate an ensemble model, including 4 classifiers using the binarization, adversarial regularization, and semantics-preservation techniques discussed in the framework, and the other 6 classifiers using the binarization and adversarial regularization techniques because examples are perturbed without preserving their

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
CLASSIFIERS ACCURACY (%) AND MACRO F1 SCORE (%) WITH NO ATTACKS VS. USING GREY-BOX ADVERSARIAL EVASION ATTACKS RESPECTIVELY, AND THE HARMONIC MEAN (%) OF THE TWO MACRO F1 SCORES

Classifiers	No attacks (%)		Attacks (%)		Harmonic mean (%)
	Accuracy	Macro F1	Accuracy	Macro F1	
Basic DNN	**96.24**	**88.91**	63.46	35.00	50.23
Binarization	95.80	87.99	63.79	35.47	50.56
Adversarial Regularization (AR)	95.66	87.98	72.02	58.93	70.58
Binarization+AR	95.62	87.87	75.22	59.87	71.22
Ensemble Binarization+AR	95.93	88.58	**76.02**	**62.95**	**73.60**

malicious functionality in the training. Since we do not have access to the malware examples, we cannot tell whether a feature perturbation preserves the malware functionality or not. The inner maximization performed by using gradient descent with respect to the transformed input iterates $T = 55$ times via the Adam optimizer [47] with learning rate 0.01. We leverage the random start points and $K = 5$. The ratio for ensemble of random subspace method is set as $\Lambda = 0.5$. Each base classifier has two fully-connected hidden layers (each layer having neurons 160), uses the ELU activation function, and is optimized by Adam. The ensemble achieves a Macro F1 score of 88.30% upon non-attack dataset, a 63.0% Macro F1 score under attacks, and a Harmonic mean on both Macro F1 scores of 73.60%. This is the highest Harmonic Mean score among the participating teams. Although this score is not ideal, this may be inherent to the fact that we as the defender do not know any information about the attack. This leads to:

Insight 4. The information "barrier" that the defender does not know the attacker's manipulation set is a fundamental one because the attacker may use adversarial malware examples that are far away from what the defender would use to train its defense model.

VI. APPLICATION TO AICS'2019 CHALLENGE AFTER KNOWING GROUND TRUTH

After the Challenge organizer announced that we won the Challenge, the ground-truth labels of the test set are released so that we can conduct further study. We stress that we still do not know the attacks that were used by the Challenge organizer.

A. Training Classifiers

Classifier. We consider and compare five classifiers: (**i**) Basic DNN without incorporating any defense; (**ii**) hardened DNN incorporating the binarization technique [25] (dubbed Binarization); (**iii**) hardened DNN incorporating adversarial regularization (dubbed Adversarial Regularization); (**iv**) hardened DNN incorporating Binarization and Adversarial Regularization (dubbed Binarization+AR); (**v**) an ensemble of Binarization +AR classifiers (dubbed Ensemble Binarization+AR).

Hyper-Parameter Settings. All of the DNNs we use have two fully-connected hidden layers (each layer having 160 neurons), use the ReLU activation function, and are optimized by Adam with epochs 30, mini-batch size 128, and learning rate 0.001. For Adversarial Regularization, we perform the inner maximization via Adam (with learning rate 0.01). Our preliminary experiments suggest us to set iterations $T = 60$. The starting point is chosen from $K = 5$ initialized points with salt-and-pepper noises, which have a noise ratio ϵ^r chose uniformly at random from 0 to $\epsilon^r_{max} = 10\%$. This means at most 10% of the features can be changed by salt-and-pepper noises in each training round. For the ensemble, we train 5 Binarization+AR classifiers, each of which is learned from an 80% data randomly selected from the training set, with a $\Lambda = 0.5$ fraction of features. We augment the training set for the last three classifiers as described in Section V-A.

TABLE VIII
ACCURACY (%) AND MACRO F1 SCORE (%) OF ADVERSARIAL REGULARIZATION IN THE ABSENCE VS. PRESENCE OF ADVERSARIAL EVASION ATTACKS, WITH RESPECT TO THE MAXIMUM SALT-AND-PEPPER NOISE RATIO ϵ^r_{max}, WHERE $*$ MEANS THAT A CLASSIFIER IS LEARNED USING OVERSAMPLING

Noise Ratio (%)	No attacks (%)		Attacks (%)	
	Accuracy	Macro F1	Accuracy	Macro F1
$\epsilon^r_{max} = 0$	96.11	88.43	69.68	49.87
$\epsilon^r_{max} = 0.1$	95.93	88.10	74.00	50.52
$\epsilon^r_{max} = 1$	**96.24**	**89.14**	73.11	55.98
$\epsilon^r_{max} = 10$	96.19	88.46	**77.11**	56.22
$\epsilon^r_{max} = 20$	96.06	88.14	75.11	51.23
$\epsilon^r_{max} = 10^*$	95.66	87.98	72.02	**58.93**

B. Classification Results

Table VII presents the results with and without adversarial attacks. We make three observations. (i) Adversarial Regularization significantly improves the Macro F1 score against the attacks when compared with the Basic DNN (a 23.93% higher Macro F1 score). The Macro F1 score of Adversarial Regularization in the absence of adversarial attacks drops slightly when compared with the Basic DNN ($\approx 1\%$). (ii) By comparing Binarization (row 2) and the Basic DNN, Binarization can improve the robustness of DNN against adversarial attacks a little bit (a 0.47% increase in the Macro F1 score). (iii) Ensemble Binarization+AR achieves a higher classification accuracy than Binarization+AR, in the presence or absence of adversarial attacks.

Hyper-Parameters Sensitivity. In Adversarial Regularization, ϵ^r_{max} is crucial and is set manually. Intuitively, a greater ϵ^r_{max} lets the defense perceive a larger space, but inhibiting the convergence of training. In addition, we want to know whether the oversampling is useful or not for Adversarial Regularization. We thus conduct a group of experiments to justify these settings. Table VIII shows the experimental results. We observe that the

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

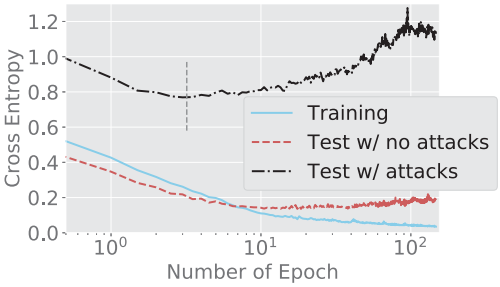


Fig. 3. Cross entropy loss of the classifier hardened by Adversarial Regularization over the training set, the test set with no adversarial evasion attacks, and the test set with adversarial evasion attacks.

Macro F1 score in the presence of adversarial evasion attacks increase with the increase of ϵ^r_{max} from 0% to 10%. Meanwhile, Accuracy and Macro F1 score do not decrease in the absence of adversarial evasion attacks, and actually slightly increase at $\epsilon^r_{max} = 1\%$. Furthermore, when the oversampling technique is leveraged at $\epsilon^r_{max} = 10\%$ (the last row), both Accuracy and Macro F1 score in the absence of adversarial evasion attacks decrease slightly ($< 1\%$). Nevertheless, the Macro F1 score in the presence of adversarial evasion attacks increases from 56.22% to 58.93%. This leads us to draw:

Insight 5. Oversampling is not necessary when there are no adversarial evasion attacks, but improves the effectiveness of Adversarial Regularization against adversarial evasion attacks in terms of macro F1 score.

C. Retrospective Analysis of the AICS'2019 Challenge

Figure 3 demonstrates that adversarial regularization overfits the perturbations searched by the inner maximizer unexpectedly. We observe that the cross-entropy loss induced by the perturbations increases significantly after about 10 epochs. Meanwhile, the cross-entropy loss on the test set with no adversarial evasion attacks changes slightly, until the number of epochs approaches 100. This means that the DNN will memorize the perturbations produced in the training phase, leading to poor generalization. Therefore, new defense strategies are needed in order to achieve a much higher accuracy against the Challenge instances. This suggests:

Insight 6. Adversarial regularization triggers the over-fitting issue; Without knowing the manipulation set, unsupervised learning may play an important role because unsupervised defenses are devised without using label information about the perturbed examples.

VII. Conclusion

We have presented six principles for enhancing the robustness of neural network classifiers against adversarial evasion attacks in the setting of malware classification. These principles guided us to design a framework, which is validated via a real-world dataset and the AICS'2019 Challenge. We drew a number of insights that are useful for real-world defenders.

We hope this paper will inspire more research into this important problem. Future research problems are abundant, such as the following. First, the adversarial training in our study is applied to feature representations satisfying box-constraints (in a discrete space). How should we accommodate other kinds of feature extractions such as graph-based or sequential-like ones [67], [68], [69], [70]? One possible approach is to instantiate the minmax adversarial training using a generic method, which does not need to know the special knowledge of the hardened model. Second, it is imperative to generate adversarial malware examples in an end-to-end fashion, assuring that a perturbed malware example indeed preserves the functionality of the original, unperturbed malware example. Third, it is an open problem to adapt the provable or certified defense [71] into the context of adversarial malware detection because it is not clear how one should define convex manipulation sets for perturbing malware examples. Unlike the image data where ℓ_p-norm may quantify visual semantics, ℓ_p-norm cannot characterize the functionalities of malware examples. Fourth, what are the other principles that can be leveraged to defend against adversarial malware examples?

Acknowledgment

We thank the anonymous reviewers for their constructive comments that guided us in improving the paper.

References

[1] D. Li, Q. Li, Y. Ye, and S. Xu, "Enhancing robustness of deep neural networks against adversarial malware samples: Principles, Framework, and aics'2019 challenge," CoRR, vol. abs/1812.08108, 2018. [Online]. Available: http://arxiv.org/abs/1812.08108
[2] Symantec. Symantec, ONLINE. 2018. [Online]. Available: https://www.symantec.com/security-center/threat-report
[3] CISCO. Cisio, ONLINE. 2018. [Online]. Available: https://www.cisco.com
[4] Y. Ye, T. Li, D. A. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 8, no. 3, pp. 41:1–41:40, 2017.
[5] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2017, pp. 62–79.
[6] L. Chen, Y. Ye, and T. Bourlai, "Adversarial machine learning in malware detection: Arms race between evasion attack and defense," in *Proc. EISIC 2017*, 2017, pp. 99–106.
[7] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. O'Reilly, "Adversarial deep learning for robust detection of binary encoded malware," in *Proc. IEEE Secur. Privacy Workshops*, 2018, pp. 76–82.
[8] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Make evasion harder: An intelligent android malware detection system," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 5279–5283.
[9] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *Proc. IEEE Symp. Secur. Privacy.*, 2020, pp. 1332–1349.
[10] D. Li and Q. Li, "Adversarial deep ensemble: Evasion attacks and defenses for malware detection," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3886–3900, 2020.
[11] Y. Kucuk and G. Yan, "Deceiving portable executable malware classifiers into targeted misclassification with practical adversarial examples," in *Proc. 10th ACM Conf. Data Appl. Secur. Privacy*, 2020, pp. 341–352.
[12] C. Szegedy *et al.*, "Intriguing properties of neural networks," in *Proc. 2nd Int. Conf. Learn. Representations*, 2014.
[13] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015.
[14] A. C. Serban and E. Poll, "Adversarial examples-a complete characterisation of the phenomenon," 2018, arXiv:1810.01185.
[15] D. Li, Q. Li, Y. Ye, and S. Xu, "SOK: Arms race in adversarial malware detection," 2020, arXiv:2005.11671.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

[16] A. Demontis *et al.*, "Yes, machine learning can be more secure! A case study on android malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 4, pp. 711–724, Jul. 2019.

[17] M. Nazir. Utsa Wins Global Cyber Security Challenge, ONLINE. 2019. [Online]. Available: https://www.eurekalert.org/pub_releases/2019-01/uota-uwg011819.php

[18] B. Biggio, G. Fumera, and F. Roli, "Multiple classifier systems for robust classifier design in adversarial environments," *Int. J. Mach. Learn. Cybern.*, vol. 1, no. 1-4, pp. 27–41, 2010.

[19] B. Biggio, G. Fumera, and F. Roli, "Multiple Classifier Systems Under Attack," in *Proc. Int. Workshop Multiple Classifier Syst.*, 2010, pp. 74–83.

[20] C. Smutz and A. Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors," *23rd Annu. Netw. Distrib. Syst. Secur. Symp.*, 2016.

[21] J. W. Stokes, D. Wang, M. Marinescu, M. Marino, and B. Bussone, "Attack and defense of dynamic analysis-based, adversarial neural malware detection models," in *Proc. MILCOM IEEE Mil. Commun. Conf.*, 2018, pp. 1–8.

[22] Q. Wang, and *et al.* "Adversary resistant deep neural networks with an application to malware detection," in *Proc. 23rd KDD*. ACM, 2017, pp. 1145–1153.

[23] D. Li, R. Baral, T. Li, H. Wang, Q. Li, and S. Xu, "Hashtran-dnn: A. framework for enhancing robustness of deep neural networks against adversarial malware samples," 2018, arXiv:1809.06498.

[24] L. Chen, S. Hou, Y. Ye, and S. Xu, "Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*, 2018, pp. 782–789.

[25] W. Xu, D. Evans, and Y. Qi, "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks," *25th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2018.

[26] L. Xu, Z. Zhan, S. Xu, and K. Ye, "An evasion and counter-evasion study in malicious websites detection," in *Proc. CNS, IEEE Conf.*, 2014, pp. 265–273.

[27] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proc. 5th Int. Conf. Learn. Representations*, 2017.

[28] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.

[29] H. Drucker and Y. Le Cun, "Improving generalization performance using double backpropagation," *IEEE Trans. Neural Netw.*, vol. 3, no. 6, pp. 991–997, Nov. 1992.

[30] C. Lyu, K. Huang, and H.-N. Liang, "A unified gradient regularization family for adversarial examples," in *Proc. IEEE Int. Conf. Data Mining (ICDM), ser. ICDM '15*. Washington, DC, USA: IEEE Comput. Soc., 2015, pp. 301–309.

[31] T. Miyato, A. M. Dai, and I. Goodfellow, "Adversarial training methods for semi-supervised text classification," in *Proc. 5th Int. Conf. Learn. Representations*, 2017.

[32] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, 2010.

[33] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 135–147, doi: 10.1145/3133956.3134057.

[34] M. Nazir. Aics 2019 Workshop Challenge Problem. 2019. [Online]. Available: http://www-personal.umich.edu/ arunesh/AICS2019/challenge.html

[35] I. C. B. Biggio *et al.* "Evasion attacks against machine learning at test time," in *Mach. Learn. Knowl. Discov. Databases: Eur. Conf.*, 2013, pp. 387–402.

[36] P. L. Nedimrndic, "Practical evasion of a learning-based classifier: A. case study," in security and privacy (SP)," in *Proc. IEEE Symp.*, 2014, pp. 197–211.

[37] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Generic black-box end-to-end attack against rnns and other calls based malware classifiers," *Res. Attacks Intrusions Defenses*, M. Bailey, and T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds., pp. 490–510, 2018.

[38] L. Chen, S. Hou, and Y. Ye, "Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks," in *Proc. 33rd Annu. Comput. Secur. Appl. Conf.*, 2017, pp. 362–372, doi: 10.1145/3134600.3134636.

[39] H. Dang, Y. Huang, and E.-C. Chang, "Evading classifiers by morphing in the dark," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 119–133, doi: 10.1145/3133956.3133978.

[40] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading Machine Learning Malware Detection," Black Hat, 2017.

[41] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers: A case study on pdf malware classifiers," *23rd Annu. Netw. Distrib. Syst. Secur. Symp.*, 2016.

[42] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," *CoRR*, vol. abs/1702.05983, 2017. [Online]. Available: http://arxiv.org/abs/1702.05983

[43] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. 38th IEEE Symp. Secur. Privacy*., 2017, pp. 39–57.

[44] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Secur. Privacy*,IEEE Eur. Symp., 2016, pp. 372–387.

[45] O. Suciu, R. Marginean, Y. Kaya, H. D. III, and T. Dumitras, "When does machine learning FAIL? generalized transferability for evasion and poisoning attacks," in *Proc. 27th USENIX Secur. Symp. (USENIX Security 18)*. Baltimore, MD: USENIX Assoc., Aug. 2018, pp. 1299–1316.

[46] F. Tramèr and D. Boneh, "Adversarial training and robustness for multiple perturbations," in *Adv. 32nd Annu. Conf. Neural Inform. Process. Syst.*, 2019, pp. 5858–5868.

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015.

[48] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole exe," *UMBC Faculty Collection*, 2019.

[49] Z.-H. Zhou, Ensemble Methods: Foundations and Algorithms. Boca Raton, FL, USA: CRC press, 2012.

[50] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. PAMI*, vol. 20, no. 8, pp. 832–844, Aug. 1998.

[51] A. Demontis *et al.*, "Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks," in *Proc. 28th {USENIX} Secur. Symp.* ({USENIX} Secur. 19), 2019, pp. 321–338.

[52] E. Grefenstette, R. Stanforth, B. O'Donoghue, J. Uesato, G. Swirszcz, and P. Kohli, "Strength in numbers: Trading-off robustness and computation via adversarially-trained ensembles," CoRR, vol. abs/1811.09300, 2018. [Online]. Available: http://arxiv.org/abs/1811.09300

[53] L. Schott, J. Rauber, W. Brendel, and M. Bethge, "Towards the first adversarially robust neural network model on MNIST," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://arxiv.org/pdf/1805.09190.pdf

[54] G. Alain and Y. Bengio, "What regularized auto-encoders learn from the data-generating distribution," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3563–3593, 2014.

[55] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 1412–1421.

[56] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket, *21st Annu. Netw. Distrib. Syst. Secur. Symp.*, 2014.

[57] A. Demontis *et al.*, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *Proc. 28th USENIX Secur. Symp. (USENIX Secur. 19)*. Santa Clara, CA: USENIX Assoc., Aug. 2019, pp. 321–338.

[58] (2018, May) Virustotal. 2018. [Online]. Available: https://www.virustotal.com

[59] M. Sebastiàn, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A. tool for massive malware labeling," in *Proc. Res. Attacks, Intrusions, Defenses*.2016, pp. 230–253.

[60] A. Desnos. Androguard, ONLINE. 2019. [Online]. Available: https://github.com/androguard/androguard

[61] I. Revivo and O. Caspi, "Cuckoodroid," in Black Hat USA, Las Vegas, NV, Jul. 2017.

[62] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 1–35, Dec. 2016.

[63] A. Athalye, N. Carlini, and D. A. Wagner, "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, vol. 80, pp. 274–283.

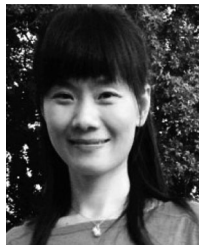
[64] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.

[65] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 506–519.
[66] Y. Sasaki *et al.* "The truth of the f-measure," *Teach Tutor Mater*, vol. 1, no. 5, pp. 1–5, 2007.
[67] S. Cui *et al.*, "Simwalk: Learning Network Latent Representations With Social Relation Similarity," vol. 2018-January, 2017, pp. 1-6. [Online]. Available: http://dx.doi.org/10.1109/ISKE.2017.8258804
[68] Y. Fan, S. Hou, Y. Zhang, Y. Ye, and M. Abdulhayoglu, "Gotcha - sly malware!: Scorpion a metagraph2vec based malware detection system," in *Proc. KDD*2018, 2018, pp. 253–262.
[69] S. Cui, T. Li, S.-C. Chen, M.-L. Shyu, Q. Li, and H. Zhang, "DISL: Deep isomorphic substructure learning for network representations," *Knowl.-Based Syst.*, vol. 189, 2020, Art. no. 105086.
[70] S. Cui, Q. Li, and S.-C. Chen, "An adversarial learning approach for discovering social relations in human-centered information networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2020, no. 1, pp. 1–19, 2020.
[71] M. Balunovic and M. Vechev, "Adversarial training and provable defenses: Bridging the gap," in *Proc. Int. Conf. Learn. Representations*, 2020.

Deqiang Li received the M.E. degree in software engineering from the Nanjing University of Science and Technology, Nanjing, China, where he is currently working toward the Ph.D. degree in computer science and technology. His research interests include adversarial malware detection, adversarial machine learning, and applied data mining in malware detection.

Qianmu Li received the B.E. and Ph.D. degrees in computer application technology from the Nanjing University of Science and Technology, Nanjing, China, in 2001 and 2005, respectively. From 2005 to 2007, he was a Postdoctoral Researcher with Nanjing University, Nanjing, China. He is currently a Full Professor with the Nanjing University of Science and Technology. He has authored or coauthored more than 110 scientific papers. His research interests include big data analysis, cyberspace security, and software systems. He was the recipient of many research grants from China's national and provincial programs.



Yanfang (Fanny) Ye is currently the T. and D. Schroeder Associate Professor with the Department of Computer and Data Sciences, Case Western Reserve University (CWRU), Cleveland, OH, USA. Her research interests include cybersecurity, data mining, machine learning, and health intelligence. Her proposed techniques by advancing AI and data-driven innovations for malware detection have been incorporated into popular commercial cybersecurity products that protect millions of users worldwide. She has expanded her research on health intelligence with focus on combating opioid epidemic and COVID-19 crisis. She was the recipient of the the CSE Research Award in 2019–2020 at CWRU, the NSF Career Award in 2019, the MetroLab Innovation of the Month in May 2020, the IJCAI 2019 Early Career Spotlight, the AICS 2019 Challenge Problem Winner, the SIGKDD 2017 Best Paper Award and Best Student Paper Award (Applied Data Science Track), the IEEE EISIC 2017 Best Paper Award, and the New Researcher of the Year Award in 2016–2017 at WVU.

Shouhuai Xu (Senior Member, IEEE) received the Ph.D. degree in computer science from Fudan University, Shanghai, China. He is currently the Gallogly Chair Professor with the Department of Computer Science, University of Colorado Colorado Springs (UCCS), Colorado Springs, OH, USA. Before joining UCCS, he was with the University of Texas at San Antonio, San Antonio, TX, USA. He pioneered the cybersecurity dynamics approach as foundation for the emerging science of cybersecurity, with three pillars, first-principle cybersecurity modeling and analysis (the x-axis), cybersecurity data analytics (the y-axis, to which the present paper belongs), and cybersecurity metrics (the z-axis). He coinitiated the International Conference on Science of Cyber Security and is the Steering Committee Chair. He is/was an Associate Editor for the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING.

Authorized licensed use limited to: Rochester Institute of Technology. Downloaded on August 31,2022 at 20:03:37 UTC from IEEE Xplore. Restrictions apply.