PAD: Towards Principled Adversarial Malware Detection Against Evasion Attacks

Deqiang Li, Shicheng Cui, Yun Li, Jia Xu, Fu Xiao and Shouhuai Xu

Abstract—Machine Learning (ML) techniques facilitate automating malicious software (malware for short) detection, but suffer from evasion attacks. Many researchers counter such attacks in heuristic manners short of both theoretical guarantees and defense effectiveness. We hence propose a new adversarial training framework, termed Principled Adversarial Malware Detection (PAD), which encourages convergence guarantees for robust optimization methods. PAD lays on a learnable convex measurement that quantifies distribution-wise discrete perturbations and protects the malware detector from adversaries, by which for smooth detectors, adversarial training can be performed heuristically with theoretical treatments. To promote defense effectiveness, we propose a new mixture of attacks to instantiate PAD for enhancing the deep neural network-based measurement and malware detector. Experimental results on two Android malware datasets demonstrate: (i) the proposed method significantly outperforms the state-of-the-art defenses; (ii) it can harden the ML-based malware detection against 27 evasion attacks with detection accuracies greater than 83.45%, while suffering an accuracy decrease smaller than 2.16% in the absence of attacks; (iii) it matches or outperforms many anti-malware scanners in VirusTotal service against realistic adversarial malware.

Index Terms—Malware Detection, Evasion Attack, Adversarial Example, Provable Defense, Deep Neural Network.

1 Introduction

INTERNET is widely used for connecting various modern devices, which facilitates the communications of our daily life, but spreads cyber attacks as well. For example, Kaspersky [1] reported detecting 33,412,568 malware samples in the year of 2020, 64,559,357 in 2021, and 109,183,489 in 2022. The scale of this threat motivates the use of Machine Learning (ML) techniques, including Deep Learning (DL), to automate the detection. Promisingly, empirical evidence demonstrates the advanced performance of ML-based malware detection (see, e.g., [2], [3], [4], [5], [6]).

Unfortunately, ML-based malware detectors are vulnerable to *adversarial examples*. These examples, a type of malware variants, are generated by modifying non-functional instructions in the existing executable programs (rather than writing a new one from scratch) [7], [8], [9], [10], [11], [12]. Adversarial examples can equip with *poisoning attack* [13], [14], *evasion attack* [12], [15], [16], or both [17], while we narrow down the scope and focus on the evasion attack solely, which allows the attacker to mislead a model in the test phase. To combat evasive attacks, pioneers propose several approaches, such as input transformation [18], weight regularization [19], and classifier randomization [20], most of which, however, have been broken by sophisticated attacks (e.g., [10], [21], [22], [23]). Nevertheless, recent studies empirically demonstrate that *adversarial training* can harden ML models to certain extent [24], [25], which endows a model with robustness by learning from adversarial examples, akin to "vaccines".

Figure 1 illustrates the schema of adversarial training. Owing to the efficiency issue of mapping representation perturbations back to the problem space, researchers conduct adversarial training in the feature space [10], [15], [24], [25], [26]. Therefore, the attained robustness should propagate to the problem space, even though there are "side-effect" features impeding the inverse-feature mapping [10]. In the feature space, adversarial training typically involves inner maximization (searching perturbations) and outer minimization (optimizing model parameters). Both are handled by heuristic methods short of theoretical guarantees [24], [25], leading to the limitation of rigorously analyzing which types of attacks the resultant model can thwart, particularly in the context of discrete domains (e.g., malware detection). Wherein, the fundamental concern is the optimization convergence: the inner maximization shall converge to a stationary point, and the resultant perturbation approaches the optimal one; the outer maximization has gradients of loss w.r.t. parameters proceed toward zero regarding certain metrics (e.g., ℓ_2 norm) in gradient-based optimization. Intuitively, as long as convergence requirements are met, the defense model mitigates attacks incomparable to the one used for adversarial training.

Existing methods handle the limitations above with mild assumptions held [27], [28], [29]. For instance, Qi et al. search text perturbations with theoretical guarantees on attackability by assuming the non-negative model [28], which further motivates new attacks counting on the submodular optimization [30]. Indeed, the non-negative ML model leads to binary monotonic classification (without outer minimization used), which intrinsically circumvents any attack that utilizes either feature addition or removal perturbations, but not both [27], [31]. However, this type of classifier tends to

- *D. Li is with the School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, 210023, China*
- *S. Cui is with the School of Computer Engineering, Nanjing Institute of Technology, Nanjing, 211167, China*
- *Y. Li, J. Xu, and F. Xiao are with the School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, 210023, China*
- *S. Xu is with the University of Colorado Colorado Springs, 1420 Austin Bluffs Pkwy, Colorado Springs, Colorado, 80918 USA.*
  Email: sxu@uccs.edu

1545-5971/XX$XX.XX © 2023 IEEE

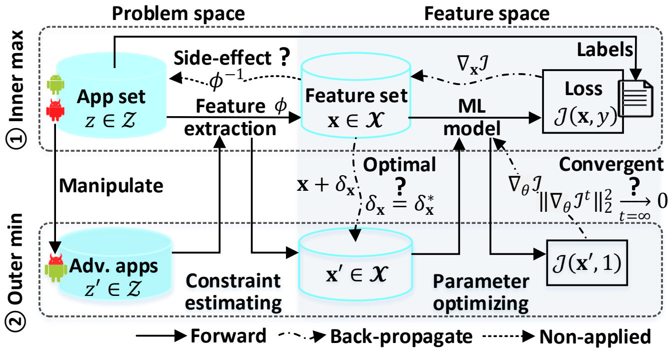

Problem space
Feature space
① Inner max
② Outer min
Side-effect ?
ϕ^{-1}
App set
$z \in \mathcal{Z}$
Feature ϕ extraction
Feature set
$\mathbf{x} \in \mathcal{X}$
$\nabla_{\mathbf{x}}\mathcal{J}$
Labels
Loss
$\mathcal{J}(\mathbf{x}, y)$
ML model
Manipulate
$\mathbf{x} + \delta_{\mathbf{x}}$
Optimal
$\delta_{\mathbf{x}} \overset{?}{=} \delta^*_{\mathbf{x}}$
$\nabla_\theta \mathcal{J}$
Convergent
$\|\nabla_\theta \mathcal{J}^t\|_2^2 \xrightarrow[t=\infty]{?} 0$
Adv. apps
$z' \in \mathcal{Z}$
Constraint estimating
$\mathbf{x}' \in \mathcal{X}$
Parameter optimizing
$\mathcal{J}(\mathbf{x}', 1)$
Forward
Back-propagate
Non-applied

Fig. 1: Schema of feature space adversarial training and its limitations on whether (i) the attained robustness can back-propagate to the problem space (upper left), (ii) the inner maximization searches perturbations optimally (middle), and (iii) the outer minimization optimizes model parameters convergently (right).

sacrifice the detection accuracy notably [10]. In order to relax the over-restrictive assumption, a recent study [29] resorts to theories of weakly submodular optimization, which necessitates a concave and smooth model. However, modern ML architectures (e.g., deep neural networks) may be not built in concavity. Moreover, these proposed attacks are not contextualized within malware detection or are used to implement adversarial training. From a broader domain of image processing, pioneers [32], [33], [34] propose utilizing smooth ML models, because specific distance metrics (e.g., ℓ_2 norm) can be incorporated to shape loss landscape, leading to the local concavity w.r.t. the input, and thus easing the inner maximization. Furthermore, the smoothness benefits the convergence of outer minimization [32]. Because the specific metrics are utilized for continuous input, they may be unsuitable for the software samples inherent in discrete space. Worst yet, semantic-preserved adversarial malware is not necessarily measured by small perturbations [10], [24].

Our Contributions. In this paper, we decorate adversarial training methods for malware detection, which tackles three limitations as follows: (i) Robustness gap: we relax the dependencies between "side-effect" features in training, and demonstrate that the resultant feature-space model can defend against practical attacks. (ii) Adversarial training without convergence guaranteed: we learn convex measurements from data for quantifying distribution-wise perturbations, which regard examples falling outside the underlying distribution as adversaries. In this way, the inner maximizer has to bypass the malware detector and the adversary detector, leading to a constrained optimization problem, whose Lagrangian relaxation, for the smooth malware detector, can be concave. Consequently, the smoothness encourages the convergence of gradient-based outer minimization [32]. (iii) Incapable of rigorously resisting a range of attacks: we organize multiple types of gradient-based attack methods to approximate the optimal attack, which is used to implement adversarial training benefiting from the optimization convergence. Our contributions are summarized as follows:

- **Adversarial training with formal treatment**. We propose a new adversarial training framework, dubbed Principled Adversarial Malware Detection (PAD), which for smooth models, encourages convergence guarantees for adversarial training, resulting in provable robustness. PAD extends the malware detector with a customized adversary detector, wherein the customization is the convex distribution-wise measurement.
- **Robustness improvement**. We establish a PAD model by combining a Deep Neural Network (DNN)-based malware detector and an input convex neural network based adversary detector. Furthermore, we enhance the model by adversarial training incorporating a new mixture of attacks, dubbed Stepwise Mixture of Attacks, leading to the defense PAD-SMA. Theoretical analysis shows the robustness of PAD-SMA, including attackability of inner maximization and convergence of outer minimization.
- **Experimental validation**. We compare PAD-SMA with seven defenses in the literature via widely-used Drebin [35] and Malscan [36] malware datasets while considering a spectrum of attack methods, ranging from no attacks, 13 oblivious attacks, and 18 adaptive attacks. Experimental results show that PAD-SMA significantly outperforms other defenses against attacks with trading-off detection accuracy on the test dataset slightly. Specifically, PAD-SMA thwarts a broad range of attacks effectively, exhibiting accuracy $\geq 81.18\%$ under 30 attacks on Drebin and accuracy $\geq 83.45\%$ under 27 attacks on Malscan, except for the Mimicry attack guided by multiple (e.g., 30 on Drebin or 10 on Malscan) benign software samples [9], [26]; it outperforms some anti-malware scanners (e.g., Symantec, Comodo), matches with some others (e.g., Microsoft), but falls behind Avira and ESET-NOD32 in terms of defending against adversarial malware examples (while noting the attacker knows our features but not scanners).

To the best of our knowledge, this is the first principled adversarial training framework for malware detection. We feel the responsibility to make the codes publicly available at https://github.com/deqangss/pad4amd.

Paper outline. Section 2 reviews some background knowledge. Section 3 elaborates the framework of principled adversarial malware detection, and an instantiated defense method is described in Section 4. Section 5 analyzes the proposed method in a theoretical manner. Section 6 presents the experiments and results. Section 7 discusses related prior studies. Section 8 concludes the paper.

2 Background Knowledge

Notations. The main notations are summarized as follows:

- **Input space**: Let $\mathcal{Z}$ be the software space (i.e., problem space), and $z \in \mathcal{Z}$ be an example.
- **Malware detector**: Let $f : \mathcal{Z} \rightarrow \mathcal{Y}$ map $z \in \mathcal{Z}$ to the label space $\mathcal{Y} = \{0, 1\}$, where "0" ("1") stands for the benign (malicious) example.
- **Adversary detector**: Let $g : \mathcal{Z} \rightarrow \mathbb{R}$ map $z \in \mathcal{Z}$ to a real-value confidence score such that $g(z) > \tau$ indicates z is adversarial and is non-adversarial otherwise, where τ is a pre-determined threshold.
- **Feature extraction**: Let $\phi : \mathcal{Z} \rightarrow \mathcal{X}$ be a hand-crafted feature extraction function, where $\mathcal{X} \subset \mathbb{R}^d$ is a discrete space and d is the number of dimensions.

- **Learning model**: We extend the malware detector f with a secondary detector g for identifying adversarial examples. Let f use an ML model $\varphi_\theta : \mathcal{X} \to \mathcal{Y}$ with $f(\cdot) = \varphi_\theta(\phi(\cdot))$ and g use an ML model ψ_ϑ with $g(\cdot) = \psi_\vartheta(\phi(\cdot))$, where θ, ϑ are learnable parameter sets.
- **Loss function for model**: $\mathcal{F}(\theta, \mathbf{x}, y)$ and $\mathcal{G}(\vartheta, \mathbf{x})$ are the loss functions for learning models φ_θ and ψ_ϑ, respectively.
- **Criterion for attack**: Let $\mathcal{J}(\mathbf{x})$ justify an adversarial example (only malware will be perturbed), which is based on $\mathcal{F}$ or a combination of $\mathcal{F}$ and ψ_ϑ according to the context.
- **Training dataset**: Let D_z denote the training dataset that contains example-label pairs. Furthermore, we have $D_\mathbf{x} = \{(\mathbf{x}, y) : \mathbf{x} = \phi(z), (z, y) \in D_z\}$ in the feature space, which is sampled from a unknown distribution $\mathbb{P}$.
- **Adversarial example**: Adversarial malware example $z' = z + \delta_z$ misleads f and g simultaneously (if g exists), where δ_z is a set of manipulations (e.g., string injection and encryption). Correspondingly, let $\mathbf{x}' = \phi(z')$ denote the adversarial example in the feature space with $\delta_\mathbf{x} = \mathbf{x}' - \mathbf{x}$.

2.1 ML-based Malware & Adversary Detection

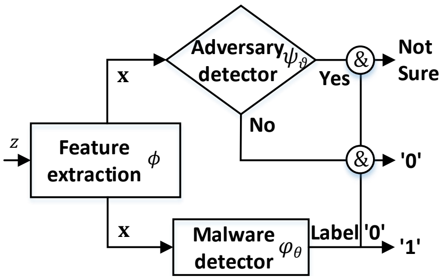

Adversary detector ψ_ϑ
x
Yes
&
Not Sure
No
z
Feature extraction ϕ
&
'0'
x
Malware detector φ_θ
Label '0'
'1'

Fig. 2: Illustration of the workflow of integrated malware detection and adversary detection.

We treat malware detection as binary classification. In addition, an auxiliary ML model is utilized to detect adversarial examples [21], [23], [37]. Fig.2 illustrates the workflow when combining malware detection and adversary detection. Formally, given an example-label pair (z, y), malware detector $f = \varphi_\theta \circ \phi$ and adversary detector $g = \psi_\vartheta \circ \phi$, the prediction is

$$\text{predict}(z) = \begin{cases} f(z), & \text{if } g(z) \leq \tau \\ 1, & \text{if } (g(z) > \tau) \wedge (f(z) = 1) \\ \text{not sure}, & \text{if } (g(z) > \tau) \wedge (f(z) = 0). \end{cases} \quad (1)$$

Where g protects f against z if $g(z) > \tau$ but $f(z) = 1$. The "not sure" option abstains f from classification, calling for further analysis. A small portion of normal (i.e., unperturbed) examples will be flagged by g. Detectors φ_θ and ψ_ϑ are learned from training dataset $D_\mathbf{x}$ by minimizing:

$$\min_{\theta,\vartheta} \mathbb{E}_{(z,y)\in D_\mathbf{x}} \left[\mathcal{F}(\theta, \mathbf{x}, y) + \mathcal{G}(\vartheta, \mathbf{x})\right], \quad (2)$$

where $\mathcal{F}$ is the loss for learning φ_θ (e.g., cross-entropy [38]) and $\mathcal{G}$ for learning ψ_ϑ (which is specified according to the downstream un-supervised task).

2.2 Evasion Attacks

The evasion attack is manifested in both the problem space and the feature space [9], [10]. In the problem space, an attacker perturbs a malware example z to z' to evade both f and g (if g exists). Consequently, we have $\mathbf{x} = \phi(z)$ and its counterpart $\mathbf{x}' = \phi(z')$ in the feature space. Owing to the non-differentiable nature of ϕ, previous studies suggest $\mathbf{x}'$ obeys a "box" constraint $\underline{\mathbf{u}} \preceq \mathbf{x}' \preceq \overline{\mathbf{u}}$ (i.e., $\mathbf{x}' \in [\underline{\mathbf{u}}, \overline{\mathbf{u}}]$) corresponding to file manipulations, where "$\preceq$" is the element-wise "no bigger than" relation between vectors [9], [17], [24]. The evasion attack in the feature space is

$$\mathbf{x}' = \mathbf{x} + \delta_\mathbf{x}, \quad (3)$$
$$\text{s.t.,}(\varphi_\theta(\mathbf{x}') = 0) \wedge (\psi_\vartheta(\mathbf{x}') \leq \tau) \wedge (\mathbf{x}' \in \mathcal{X}) \wedge (\mathbf{x}' \in [\underline{\mathbf{u}}, \overline{\mathbf{u}}]).$$

Considering that ψ_ϑ may not exist, we followingly review former attack methods as they are, then introduce the existing strategies to target both φ_θ and ψ_ϑ, and finally bring in the current inverse-mapping solutions (i.e., mapping feature perturbations into the problem space, as ϕ^{-1} in Figure 1).

2.2.1 Evasion Attack Methods

Mimicry attack. A mimicry attacker [19], [26], [39] perturbs a malware example to mimic a benign application as much as possible. The attacker does not need to know the internal knowledge of models, but can query them. In such case, the attacker uses N_{ben} ($N_{ben} \geq 1$) benign examples separately to guide the manipulations, resulting in N_{ben} perturbed examples, of which the one can bypass the victim is picked.

Grosse attack. This attack [40] perturbs "sensitive" features to evade detection, where the sensitivity is quantified by the gradients of the DNN's *softmax* output with respect to the input. A larger gradient value means higher sensitivity. This attack adds features absent in the original example.

FGSM attack. This attack is proposed in the context of image classification [41] and then adapted to malware detection [18], [24]. It perturbs a feature vector $\mathbf{x}$ in the direction of ℓ_∞ norm of gradients (i.e., sign operation) of the loss function with respect to the input:

$$\mathbf{x}' = \text{round}\left(\text{Proj}_{[\underline{\mathbf{u}},\overline{\mathbf{u}}]}\left(\mathbf{x} + \varepsilon \cdot \text{sign}(\nabla_\mathbf{x} \mathcal{F}(\theta, \mathbf{x}, 1))\right)\right),$$

where $\varepsilon > 0$ is the step size, $\text{Proj}_{[\underline{\mathbf{u}},\overline{\mathbf{u}}]}$ projects an input into $[\underline{\mathbf{u}}, \overline{\mathbf{u}}]$, and round is an element-wise operation which returns an integer-value vector.

Bit Gradient Ascent (BGA) and **Bit Coordinate Ascent (BCA) attacks**. Both attacks [24] iterate multiple times. In each iteration, BGA increases the feature value from '0' to '1' (i.e., adding a feature) 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 $\sqrt{d}$, where d is the input dimension. By contrast, at each iteration, BCA flips the value of the feature from '0' to '1' corresponding to the max gradient of the loss function with respect to the input. Technically, given a malware instance-label pair $(\mathbf{x}, y)$, the attacker needs to solve

$$\max_{\mathbf{x}' \in [\underline{\mathbf{u}},\overline{\mathbf{u}}]} \mathcal{F}(\theta, \mathbf{x}', 1) \text{ s.t., } \mathbf{x}' \in \mathcal{X}.$$

Projected Gradient Descent (PGD) attack. It is proposed in the image classification context [42] and adapted to malware detection by accommodating the discrete input space [25].

The attack permits both feature addition and removal with retaining malicious functionalities, giving more freedom to the attacker. It finds perturbations via an iterative process with the initial perturbation as a zero vector:

$$\delta_{\mathbf{x}}^{(t+1)} = \mathrm{Proj}_{[\underline{\mathbf{u}}-\mathbf{x},\overline{\mathbf{u}}-\mathbf{x}]}\left(\delta_{\mathbf{x}}^{(t)} + \alpha \nabla_{\delta_{\mathbf{x}}} \mathcal{F}(\theta, \mathbf{x}+\delta_{\mathbf{x}}^{(t)}, 1)\right) \quad (4)$$

where t is the iteration, $\alpha > 0$ is the step size, $\mathrm{Proj}_{[\underline{\mathbf{u}}-\mathbf{x},\overline{\mathbf{u}}-\mathbf{x}]}$ projects perturbations into the predetermined space $[\underline{\mathbf{u}} - \mathbf{x}, \overline{\mathbf{u}} - \mathbf{x}]$, and $\nabla_{\delta_{\mathbf{x}}}$ denote the derivative of loss function $\mathcal{F}$ with respect to $\delta_{\mathbf{x}}^{(t)}$. Since the derivative values may be too small to make the attack progress, researchers normalize $\nabla_{\delta_{\mathbf{x}}} \mathcal{F}$ in the direction of ℓ_1, ℓ_2, or ℓ_∞ norm [42], [43]:

$$\mathbf{e}_p = \arg\max_{\|\mathbf{e}\|_p=1} \langle \nabla_{\delta_{\mathbf{x}}} \mathcal{F}(\theta, \mathbf{x}+\delta_{\mathbf{x}}^{(t)}, 1), \mathbf{e}\rangle,$$

where $\mathbf{e}_p$ is the direction of interest, $\langle\cdot,\cdot\rangle$ denotes the inner product, and $p = 1, 2, \infty$. Adjusting p leads to the PGD-ℓ_1, PGD-ℓ_2, and PGD-ℓ_∞ attack, respectively. After the loop, an extra operation is conducted to discretize the real-valued vector. For example, $\mathrm{round}(\mathbf{a})$ returns the vector closest to $\mathbf{a}$ in terms of the ℓ_1 norm distance.

Mixture of Attacks (MA). This attack [9] organizes a mixture of attack methods upon a set of manipulations as large as possible. Two MA strategies are used: the "max" strategy selects the adversarial example generated by several attacks via maximizing a criterion (e.g., classifier's loss function $\mathcal{F}$); the iterating "max" strategy puts the resulting example from the last iteration as the new starting point, where the initial point is $\mathbf{x}$. Herein the iteration can promote the attack effectiveness because of the non-concave ML model.

2.2.2 Oblivious vs. Adaptive Attacks

The attacks mentioned above do not consider the adversary detector g, meaning that they degrade to oblivious attacks when g is present and would be less effective. By contrast, an adaptive attacker is conscious of the presence of $g(\cdot) = \psi_\vartheta(\phi_\theta(\cdot))$, leading to an additional constraint $\psi_\vartheta(\mathbf{x}') \leq \tau$ for a given feature representation vector $\mathbf{x}$:

$$\max_{\mathbf{x}'\in[\underline{\mathbf{u}},\overline{\mathbf{u}}]} \mathcal{F}(\theta, \mathbf{x}', 1) \ \text{ s.t., } (\psi_\vartheta(\mathbf{x}') \leq \tau) \wedge (\mathbf{x}' \in \mathcal{X}), \quad (5)$$

where we substitute $\varphi(\mathbf{x}') = 0$ with maximizing $\mathcal{F}(\theta, \mathbf{x}', 1)$ owing to the aforementioned issue of non-differentiability.

However, ψ_ϑ may not be affine (e.g., linear transformation), meaning that the effective projection strategies used in PGD are not applicable anymore. In order to deal with $\psi_\vartheta(\mathbf{x}') \leq \tau$, researchers suggest three approaches: (i) Use gradient-based methods to cope with

$$\max_{\mathbf{x}'\in[\underline{\mathbf{u}},\overline{\mathbf{u}}]} [\mathcal{F}(\theta, \mathbf{x}', 1) - \lambda \psi_\vartheta(\mathbf{x}')], \quad (6)$$

where $\lambda \geq 0$ is a penalty factor for modulating the importance between the two items [23]. (ii) Maximize $\mathcal{F}(\theta, \mathbf{x}', 1)$ and $-\psi_\vartheta(\mathbf{x}')$ alternatively as it is notorious for setting λ properly [21]. (iii) Maximize $\mathcal{F}(\theta, \mathbf{x}', 1)$ and $-\psi_\vartheta(\mathbf{x}')$ in an "orthogonal" manner [23], where "orthogonal" means eliminating the mutual interaction between $\mathcal{F}$ and ψ from the geometrical perspective. For example, the attack perturbs $\mathbf{x}$ in the direction orthogonal to the direction of the gradients of $-\psi_\vartheta$, which is in the direction of the gradients of $\mathcal{F}$, to make it evade φ_θ but not react ψ_ϑ. Likewise, the attack alters the orthogonal direction to evade ψ_ϑ but not φ_θ.

2.2.3 The Inverse Feature-Mapping Problem

There is a gap between the feature space and the problem or software space. Since feature extraction ϕ is non-differentiable, gradient-based methods cannot produce end-to-end adversarial examples. Moreover, ϕ^{-1} cannot be derived analytically due to interdependent features (or "side-effect" features) [10].

To fill the gap, Srndic and Laskov [26] propose directly mapping the perturbation vector $\delta_{\mathbf{x}}$ into the problem space, leading to $\phi(\tilde{\phi}^{-1}(\mathbf{x}')) \neq \mathbf{x}'$, where $\tilde{\phi}^{-1}$ is approximate to ϕ^{-1}. Nevertheless, experiments demonstrate that the attacks can evade anti-malware scanners. Li and Li [9] leverage this strategy to produce adversarial Android examples. Researchers also attempt to align δ_z with $\delta_{\mathbf{x}}$ as much as possible. For example, Pierazzi et al. [10] collect a set of manipulations from gadgets of benign applications and implement ones that mostly align with the gradients of the loss function with respect to the input. Zhao et al. [11] propose incorporating gradient-based methods with the Reinforcement Learning (RL) strategy, of which the RL-based model assists in obtaining manipulations in the problem space under the guidance of gradient information. In addition, black-box attack methods (without knowing the internals of the detector) directly manipulate malware examples, which avoids the inverse feature-mapping procedure [15].

In this paper, we use an approximate $\tilde{\phi}^{-1}$ (implementation details are in supplementary material) because it relatively eases the attack tactics and besides, the side-effect features cannot decline the attack effectiveness notably in our refined Drebin feature space [35].

2.3 Adversarial Training

Adversarial training augments training dataset with adversarial examples by solving a min-max optimization problem [24], [40], [42], [44], [45], [46], as shown in Figure 1. The inner maximization looks for adversarial examples, while the outer minimization optimizes the model parameters upon the updated training dataset. Formally, given the training dataset $D_{\mathbf{x}}$, we have

$$\min_\theta \mathbb{E}_{(\mathbf{x},y)\in D_{\mathbf{x}}} \left[\mathcal{F}(\theta, \mathbf{x}, y) + \beta \max_{\mathbf{x}'\in[\underline{\mathbf{u}},\overline{\mathbf{u}}]} \mathcal{F}(\theta, \mathbf{x}', 1)\right], \quad (7)$$
$$\text{s.t., } (\mathbf{x}' = \mathbf{x} + \delta_{\mathbf{x}}) \wedge (\mathbf{x}' \in \mathcal{X})$$

where $\beta \geq 0$ is used to balance detection accuracy and robustness, and only the malware representations take part in the inner maximization.

However, Owing to the NP-hard nature of searching discrete perturbations [32], the adversarial training methods incorporate the (approximate) optimal attack without convergence guaranteed [24], [25], triggering the questionable robustness. For example, Al-Dujaili et al. [24] approximate the inner maximization via four types of attack algorithms, while showing that a hardened model cannot mitigate the attacks absent in the training phase. Furthermore, a mixture of attacks is used to instantiate the framework of adversarial training [9]. Though the enhanced model can resist a range of attacks, it is still vulnerable to a mixture of attacks with iterative "max" strategy (more iterations are used, see Section 2.2.1). Thereby, it remains a question of rigorously uncovering the robustness of adversarial training.

3 The PAD Framework

PAD, reshaping adversarial training, aims to render the inner maximization solvable analytically by establishing a concave loss w.r.t. the input. The core part of the improvement is a learnable convex distance metric, with which the distribution-wise perturbations can be measured, leading to a constraint attack problem, whose Lagrange relaxation is concave at reasonable circumstances.

3.1 Threat model and Design Objective

Threat Model. Given a malware example z, malware detector f, and adversary detector g (if g exists), an attacker modifies z by searching for a set of non-functional instructions δ_z upon knowledge of detectors. With following Kerckhoff's principle that defense should not count on "security by obscurity" [10], we consider *white-box* attacks, meaning that the attacker has full knowledge of f and g. For assessing the robustness of defense models, we also utilize *grey-box* attacks where the attacker knows f but not g (i.e., oblivious attack [47]), or knows features used by f and g.

Design Objective. As aforementioned, PAD is rooted in adversarial training. In contrast, we propose incorporating f with an adversary detector $g(\cdot) = \psi_\vartheta(\phi(\cdot))$, where ψ_ϑ is the convex measurement. To this end, given a malware instance-label pair $(\mathbf{x}, y)$ for $\mathbf{x} = \phi(z)$ and $y = 1$, we mislead both ϕ_θ and ψ_ϑ by perturbing $\mathbf{x}$ into $\mathbf{x}'$, upon which we optimize model parameters. Formally, PAD uses the objective:

$$\begin{aligned}\min_{\theta,\vartheta} \mathbb{E}_{(\mathbf{x},y)\in D_\mathbf{x}} \Big[& \mathcal{F}(\theta, \mathbf{x}, y) + \mathcal{G}(\vartheta, \mathbf{x}) \\ & + \beta_1 \mathcal{F}(\theta, \mathbf{x}', 1) + \beta_2 \mathcal{G}(\vartheta, \mathbf{x}') \Big],\end{aligned} \tag{8a}$$

where

$$\begin{aligned}\mathbf{x}' := & \max_{\mathbf{x}'\in[\underline{\mathbf{u}},\overline{\mathbf{u}}]} \left[\mathcal{F}(\theta, \mathbf{x}', 1) - \lambda\psi_\vartheta(\mathbf{x}')\right], \\ & \text{s.t.}, (\mathbf{x} + \delta_\mathbf{x} = \mathbf{x}') \wedge (\mathbf{x}' \in \mathcal{X})\end{aligned} \tag{8b}$$

β_1, β_2 weight the robustness against $\mathbf{x}'$, and $\lambda \geq 0$ is a penalty factor. We present three merits for this formulation:

(i) **Manipulations in the feature space**: Eq.(8b) tells us we can search feature perturbations without inverse-feature mapping applied and hence saving training time. Though it may trigger concern about whether the attained robustness can propagate to the problem space, we later show it can (Section 3.2).

(ii) **Box-constraint manipulation**: Eq.(8b) has the attackers search $\mathbf{x}'$ in $[\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ without any norm-type constraints. It means the defense aims to resist semantics-based attacks rather than small perturbations.

(iii) **Continuous perturbation may be enough**: It is NP-hard to search optimal discrete perturbations even for attacking a linear model [32]. Eq.(8b) contains an auxiliary detector ψ_ϑ, which can treat continuous perturbations in the range of $[\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ as anomalies with relaxing the discrete space $\mathcal{X}$ constraint in training.

Finally, the above formulation facilitates that we can use the efficient gradient-based optimization methods to solve both Eq.8a and Eq.8b. We next explain the intuition behind this objective and moreover, a smooth $\mathcal{F}$ is necessary (e.g., relieving the pain of setting a proper λ, see Section 2.2.2).

3.2 Design Rationale

Bridge robustness gap. Adversarial training is performed in the feature space while the adversarial malware is in the problem space, which seems to leave a seam for attackers on the alignment with feature extraction ϕ. Recall that we represent the manipulations by $[\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ and discrete space $\mathcal{X}$, where any instance $\mathbf{x}'$ cannot be mapped back to $\mathcal{Z}$ due to "side-effect" features [10]. The "side-effect" means the interdependent relationship between features (e.g., modifying a feature requires changing others such that the functionality is preserved), reminiscent of the structural graph representation. From this view, we can leverage the directed graph to denote the relation: node represents the modifiable feature and edge represents the "side-effect" dependency. In this way, an adjacent matrix can be used to represent the "side-effect" features, which however shrinks the space of $[\underline{\mathbf{u}}, \overline{\mathbf{u}}]$.

Postulating that, for a given malware representation $\mathbf{x}$, we obtain the optimal $\tilde{\mathbf{x}}^*, \mathbf{x}^* \in [\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ with or without the "side-effect" constraint. The criterion results meet $\mathcal{J}(\tilde{\mathbf{x}}^*) = \mathcal{F}(\theta, \tilde{\mathbf{x}}^*, 1) - \lambda\psi_\vartheta(\tilde{\mathbf{x}}^*) \leq \mathcal{J}(\mathbf{x}^*)$, which in turn demonstrates that if an adversarial training model can resist $\mathbf{x}^*$, so can $\tilde{\mathbf{x}}^*$, and otherwise it contradicts the optimality. Noting that $\mathbf{x}^*$ is typically hard to calculate and the suboptimal one $\mathbf{x}'$ is got, potentially leading to the phenomenon $\mathcal{J}(\mathbf{x}') \leq \mathcal{J}(\phi(\tilde{\phi}^{-1}(\mathbf{x}')))$ where $\tilde{\phi}^{-1} \approx \phi^{-1}$.

Therefore, we relax the attack constraint of "side-effect" and conduct the adversarial training in the feature space. More importantly, the robustness can propagate to the problem space, though it would decline the detection accuracy because more perturbations are considered.

Defending against distribution-wise perturbation. We explain Eq.(8b) via distributionally robust optimization [32]. We establish a point-wise distance $C(\cdot, \mathbf{x}) = \max\{0, \psi_\vartheta(\cdot) - \tau\}$ to measure how far a point, say $\mathbf{x}'$, to the population. A large portion of (e.g., 95%) training examples will have $\psi_\vartheta(\mathbf{x}) \leq \tau$. Other measures are suitable as long as they are convex and continuous. Based on C, we have a Wasserstein distance [48]:

$$W(\mathbb{P}', \mathbb{P}) := \inf_\Gamma \left\{ \int C(\mathbf{x}', \mathbf{x}) d\Gamma(\mathbf{x}', \mathbf{x}) : \Gamma \in \prod(\mathbb{P}', \mathbb{P}) \right\}$$

where $\prod(\mathbb{P}', \mathbb{P})$ is the joint distribution of $\mathbb{P}'$ and $\mathbb{P}$ with marginal as $\mathbb{P}'$ and $\mathbb{P}$ w.r.t. to the first and second argument, respectively. That is, the Wasserstein distance gets the infimum from a set of expectations. Because the points $\mathbf{x}, \mathbf{x}'$ are on discrete space $\mathcal{X}$, the integral form in the definition is a linear summation. We aim to build a malware detector f classify $\mathbf{x}'$ correctly with $\mathbf{x}' \sim \mathbb{P}'$ and $W(\mathbb{P}', \mathbb{P}) \leq 0$. Formally, the corresponding inner maximization is

$$\max_{\mathbb{P}':W(\mathbb{P}',\mathbb{P})\leq 0} \mathbb{E}_{\mathbf{x}'\sim\mathbb{P}'} \mathcal{F}(\theta, \mathbf{x}', 1). \tag{9}$$

It is non-trivial to tackle $W(\mathbb{P}', \mathbb{P})$ directly owing to massive vectors on $\mathcal{X} \times \mathcal{X}$ and typically, the dual problem is used:

Proposition 1. *Given the continuous function $\mathcal{F}$, and continuous and convex distance $C(\cdot, \mathbf{x}) = \max\{0, \psi_\vartheta(\cdot) - \tau\}$ with $\mathbf{x} \sim \mathbb{P}$, the dual problem of Eq.*(9) *is*

$$\inf_\lambda \Big\{ \mathbb{E}_{\mathbf{x}\sim\mathbb{P}} \max_{\mathbf{x}'} (\mathcal{F}(\theta, \mathbf{x}', 1) - \lambda\psi_\vartheta(\mathbf{x}') + \lambda\tau) : \lambda \geq 0 \Big\},$$

with $\mathbf{x} + \delta_\mathbf{x} = \mathbf{x}' \in \mathcal{X}$, $\mathbf{x}' \sim \mathbb{P}'$ and $\psi_\vartheta(\mathbf{x}') \geq \tau$.

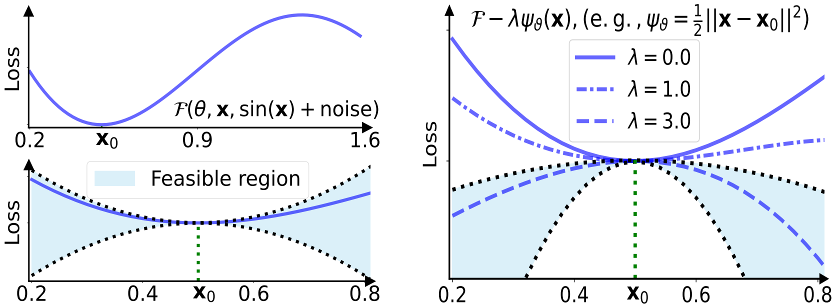


Fig. 3: An example of loss changes under perturbations with a reasonable assumption that $\mathcal{F}$ is smooth (feasible region in the bottom-left figure), leading to $\mathcal{F} - \lambda\psi_\vartheta$ strongly convex (feasible region in the rightmost figure) at $\mathbf{x}_0$ when $\lambda = 3.0$.

Its empirical version is Eq.(8b) for fixed λ and τ, except for the constraint $[\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ handled by clip operation. The proposition tells us PAD defends against distributional perturbations. The proof is available in the supplementary material.

Concave inner maximization. Given an instance-label pair $(\mathbf{x}, y)$, let Taylor expansion approximate $\mathcal{F} - \lambda\psi_\vartheta$:

$$\begin{aligned}\mathcal{F}(\theta, \mathbf{x}{+}\delta_{\mathbf{x}}, y) - \lambda\psi_\vartheta(\mathbf{x} + \delta_{\mathbf{x}}) &\cong \mathcal{F} - \lambda\psi_\vartheta \\ &+ \langle\nabla_{\mathbf{x}}(\mathcal{F} - \lambda\psi_\vartheta), \delta_{\mathbf{x}}\rangle + \frac{1}{2}\delta_{\mathbf{x}}^\top \nabla_{\mathbf{x}}^2(\mathcal{F} - \lambda\psi_\vartheta)\delta_{\mathbf{x}},\end{aligned}$$

where $\mathcal{F} - \lambda\psi_\vartheta$ is a short denotation of $\mathcal{F}(\theta, \mathbf{x}, y) - \lambda\psi_\vartheta(\mathbf{x})$. The insight is that if (i) the values of the entities in $\nabla_{\mathbf{x}}\mathcal{F}$ are finite (i.e., smoothness [32]) when $\mathbf{x} \in [\underline{\mathbf{u}}, \overline{\mathbf{u}}]$, and (ii) $\nabla_{\mathbf{x}}\psi_\vartheta > 0$ (i.e., strongly convex), then we can make $\mathcal{F} - \lambda\psi_\vartheta$ concave by tweaking λ; this eases the inner maximization.

Figure 3 illustrates the idea behind the design, by using a smoothed DNN model to fit the noising sin function (top-left figure). Owing to the smoothness of φ_θ and $\mathcal{F}$ (bottom-left figure), we transform the loss function to be a concave function by incorporating a convex ψ_ϑ. The concavity is achieved gradually by raising λ, along with the feasible region changed, as shown in the right-hand figure.

In the course of adjusting λ [23], there are three possible scenarios: (i) λ is large enough, leading to a concave inner maximization. (ii) A proper λ may result in a linear model, which would be rare because of the difference between φ_θ and ψ_ϑ. (iii) When λ is so small that the inner maximization is still a non-concave and nonlinear problem, as former heuristic adversarial training. In summary, we propose enhancing the robustness of f and g for reducing the smoothness factor of f [49], [50], which intuitively forces the attacker to increase λ when generating adversarial examples.

Since the interval $\mathbf{x} + \delta_{\mathbf{x}} \in [\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ relaxes the constraint of discrete input, we can address this issue by treating continuous perturbations as anomalies, as stated earlier. Therefore, instead of heuristically searching for discrete perturbations, we directly use ψ_θ to detect continuous perturbations without the discretization tricks used.

4 Instantiating the PAD Framework

We instantiate PAD framework with the model establishment and an adversarial training algorithm. Though PAD may be applicable to any differentiable ML algorithms, we consider deep neural network based malware detection because its security has been intensively investigated [9], [51], [52], [53].

4.1 Adjusting Malware Detector

PAD requires the composition of $\mathcal{F}$ and φ_θ to be smooth. DNN consists of hierarchical layers, each of which typically has a linear mapping followed by a non-linear activation function. Most of these ingredients meet the smoothness condition, except for some activation functions (e.g., Rectified Linear Unit or ReLU [38]) owing to non-differentiability at the point zero. To handle non-smooth activation functions, researchers suggest using over-parameterized DNNs, which yield semi-smooth loss landscapes [54]. Instead of increasing learnable parameters, we replace ReLU with smooth activation functions (e.g., Exponential Linear Unit or ELU [55]). The strategy is simple in the sense that the model architecture is changed slightly and a fine-tuning trick suffices to recover the detection accuracy. Despite this, our preliminary experiments show it slightly reduces the detection accuracy.

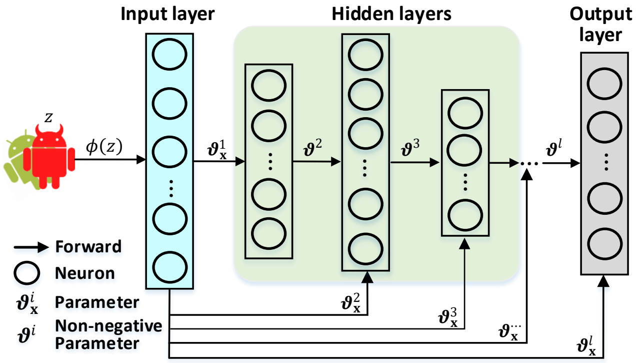

Input layer
Hidden layers
Output layer
z
$\phi(z)$
$\vartheta^1_{\mathbf{x}}$
ϑ^2
ϑ^3
ϑ^l
Forward
Neuron
$\vartheta^i_{\mathbf{x}}$ Parameter
ϑ^i Non-negative Parameter
$\vartheta^2_{\mathbf{x}}$
$\vartheta^3_{\mathbf{x}}$
$\vartheta^{\cdots}_{\mathbf{x}}$
$\vartheta^l_{\mathbf{x}}$

Fig. 4: Architecture of an input convex neural network.

4.2 Adversary Detector

We propose a DNN-based g that is also learned from the features extracted by ϕ. Figure 4 shows the architecture of ψ_ϑ, which is a l-layer Input Convex Neural Network (ICNN) [56]. ICNN maps the input $\mathbf{x}$ recursively via non-negative transformations, along with adding a normal transformation on $\mathbf{x}$:

$$\mathbf{x}^{i+1} = \sigma(\boldsymbol{\vartheta}^i\mathbf{x}^i + \boldsymbol{\vartheta}^i_{\mathbf{x}}\mathbf{x} + \mathbf{b}^i),$$

where $\vartheta = \{\boldsymbol{\vartheta}^i, \boldsymbol{\vartheta}_\mathbf{x}^i, \mathbf{b}^i : i = 1, \ldots, l\}$, $\boldsymbol{\vartheta}^i$ is non-negative, $\boldsymbol{\vartheta}_\mathbf{x}^i$ has no such constraint, $\mathbf{x}^1 = \mathbf{x}$, $\boldsymbol{\vartheta}^1$ is identity matrix, and σ is a smooth activation function (e.g., ELU or Sigmoid [55]).

We cast the adversary detection as a one-class classification task [57]. In the training phase, we perturb examples in $D_\mathbf{x}$ to obtain a set of new examples $\{\mathbf{x} + \delta_\mathbf{x} : (\mathbf{x}, y) \in D_\mathbf{x}\}$, wherein $\delta_\mathbf{x}$ is a vector of salt-and-pepper noises, meaning that at least half of elements in $\mathbf{x}$ are randomly selected and their values are set as their respective maximum. Formally, given an example $\mathbf{x}^1 \in \{\mathbf{x} : (\mathbf{x}, y) \in D_\mathbf{x}\} \cup \{\mathbf{x}+\delta_\mathbf{x} : (\mathbf{x}, y) \in D_\mathbf{x}\}$, the loss function $\mathcal{G}$ is

$$\mathcal{G}(\vartheta, \mathbf{x}^1) = \mathsf{pert}\log(\psi_\vartheta(\mathbf{x}^1)) + (1 - \mathsf{pert})\log(1 - \psi_\vartheta(\mathbf{x}^1)),$$

where $\mathsf{pert} = 0$ indicates $\mathbf{x}^1$ is from $D_\mathbf{x}$, and the otherwise $\mathsf{pert} = 1$. In the test phase, we let the input pass through ψ_ϑ to perform the prediction as shown in Eq.(1).

4.3 Adversarial Training Algorithm

For the *inner maximization* (Eq.8b), we propose a mixture of PGD-ℓ_1, PGD-ℓ_2 and PGD-ℓ_∞ attacks (see Section 2.2.1). The attacks proceed iteratively via "normalized" gradients

$$\mathbf{e}_p = \arg\max_{\|\mathbf{e}\|_p=1} \langle \nabla_{\delta_\mathbf{x}} (\mathcal{F}(\theta, \mathbf{x} + \delta_\mathbf{x}^{(t)}, 1) - \lambda\psi_\vartheta(\mathbf{x} + \delta_\mathbf{x}^{(t)})), \mathbf{e}\rangle, \tag{10}$$

and perturbations

$$\left\{\delta_{\mathbf{x},p}^{(t+1)} = \mathrm{Proj}_{[\underline{\mathbf{u}}-\mathbf{x}, \overline{\mathbf{u}}-\mathbf{x}]}\left(\delta_{\mathbf{x},p}^{(t)} + \alpha_p \mathbf{e}_p\right) : p \in \{1, 2, \infty\}\right\}, \tag{11}$$

in which the one is chosen by the scoring rule

$$\begin{aligned}\delta_\mathbf{x}^{(t+1)} = \arg\max_{\delta_{\mathbf{x},p}^{(t+1)}} \Big[& \mathcal{F}(\theta, \mathrm{round}(\mathbf{x} + \delta_{\mathbf{x},p}^{(t+1)}), 1) \\ & - \lambda\psi_\vartheta(\mathrm{round}(\mathbf{x} + \delta_{\mathbf{x},p}^{(t+1)}))\Big]\end{aligned} \tag{12}$$

at the t^{th} iteration. The round operation is used since our initial experiments show that it leads to better robustness against the attack itself. This attack selects the best attack in a stepwise fashion and thus is termed Stepwise Mixture of Attacks (SMA).

Notably, for the attacker, there are three differences: (i) We treat the dependencies between features as graphical edges. Since the summation of gradients can measure the importance of a group in the graph [58], we accumulate the gradients of the loss function with respect to the side-effect features and use the resulting gradient to decide whether to modify these features together. (ii) The round is used to discretize perturbations when the loop is terminated [25]. (iii) Map the perturbations back into the problem space.

For the *outer minimization* (Eq.8a), we leverage a Stochastic Gradient Descent (SGD) optimizer, which proceeds iteratively to find the model parameters. Basically, SGD samples a batch of B (a positive integer) pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^B$ from $D_\mathbf{x}$ and updates the parameters with

$$\begin{aligned}\theta^{(j+1)} = & \quad \theta^{(j)} - \gamma\nabla_\theta \frac{1}{B}\sum_{i=1}^{B} \mathcal{F}(\theta^{(j)}, \mathbf{x}_i + \delta_{\mathbf{x}_i}^{(T)}, y_i) \text{ and} \\ \vartheta^{(j+1)} = & \quad \vartheta^{(j)} - \gamma\nabla_\vartheta \frac{1}{B}\sum_{i=1}^{B} \mathcal{G}(\vartheta^{(j)}, \mathbf{x}_i + \delta_{\mathbf{x}_i}^{(T)}),\end{aligned}$$

where j is the iteration, γ is the learning rate, and $\delta_{\mathbf{x}_i}^{(T)}$ is obtained from Eq.(12) with T loops for perturbing $\mathbf{x}_i$. We optimize the model parameters by Eq.(8a).

Algorithm 1: Adversarial training

```
   Input: Training set $D_z$, epoch $N$, batch size $B$,
          factors $\beta_1$, $\beta_2$ and $\lambda$, iteration $T$, and step size
          $\alpha_p$ for norm $p \in \{1, 2, \infty\}$.
1  Get $D_\mathbf{x} = \{(\phi(z), y) : (z, y) \in D_z\}$ for the given $D_z$;
2  for $j = 1$ to $N$ do
3      Sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^B$ from $D_\mathbf{x}$;
4      Apply salt-and-pepper noises to $\{\mathbf{x}_i\}_{i=1}^B$;
5      for $t = 0$ to $T - 1$ do
6          for $p \in \{1, 2, \infty\}$ do
7              Calculate perturbation $\delta_{\mathbf{x},p}^{(t+1)}$ by Eq.(10)
                 and Eq.(11) for $\mathbf{x} \in \{\mathbf{x}_i\}_{i=1}^B$ with $y_i = 1$;
8          end
9          Select $\delta_\mathbf{x}^{(t+1)}$ by Eq.(12);
10     end
11     Calculate the adversarial training loss via Eq.(8a);
12     Backpropagate the errors for updating $\theta$ and $\vartheta$;
13 end
```

Algorithm 1 summarizes a PAD-based adversarial training by incorporating the stepwise mixture of attacks. Given the training set, we preprocess software examples and obtain their feature representations (line 1). At each epoch, we first perturb the feature representations via salt-and-pepper noises (line 4) and then generate adversarial examples with the mixture of attacks (lines 5-10). Using the union of the original examples and their perturbed variants, we learn malware detector f and adversary detector g (lines 11-13).

5 Theoretical Analysis

We analyze effectiveness of the inner maximization and optimization convergence of the outer minimization. The joint of both supports the robustness of the proposed method. As aforementioned, we make an assumption in PAD which requires smooth learning algorithms (Section 4.1).

Assumption 1 (Smoothness assumption [32])**.** *The composition of $\mathcal{F}$ and φ_θ meets the smoothness condition:*

$$\begin{aligned}\|\nabla_\mathbf{x}\mathcal{F}(\theta, \mathbf{x}, y) - \nabla_\mathbf{x}\mathcal{F}(\theta, \mathbf{x}', y)\|_2 &\leq \mathsf{L}_{\mathbf{xx}}^f\|\mathbf{x} - \mathbf{x}'\|_2, \\ \|\nabla_\mathbf{x}\mathcal{F}(\theta, \mathbf{x}, y) - \nabla_\mathbf{x}\mathcal{F}(\theta', \mathbf{x}, y)\|_2 &\leq \mathsf{L}_{\mathbf{x}\theta}^f\|\theta - \theta'\|_2, \\ \|\nabla_\theta\mathcal{F}(\theta, \mathbf{x}, y) - \nabla_\theta\mathcal{F}(\theta, \mathbf{x}', y)\|_2 &\leq \mathsf{L}_{\theta\mathbf{x}}^f\|\mathbf{x} - \mathbf{x}'\|_2,\end{aligned}$$

and ψ_ϑ meets the smoothness condition:

$$\begin{aligned}\|\nabla_\mathbf{x}\psi_\vartheta(\mathbf{x}) - \nabla_\mathbf{x}\psi_\vartheta(\mathbf{x}')\|_2 &\leq \mathsf{L}_{\mathbf{xx}}^g\|\mathbf{x} - \mathbf{x}'\|_2, \\ \|\nabla_\mathbf{x}\psi_\vartheta(\mathbf{x}) - \nabla_\mathbf{x}\psi_{\vartheta'}(\mathbf{x})\|_2 &\leq \mathsf{L}_{\mathbf{x}\vartheta}^g\|\vartheta - \vartheta'\|_2,\end{aligned}$$

where $\mathbf{x}' \in [\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ is changed from $\mathbf{x} = \phi(z)$ for a given example z and $\mathsf{L}_{**}^* > 0$ denotes the smoothness factor ($*$ is the wildcard).

Recall that the ψ_ϑ meets the strongly-convex condition:

$$\|\nabla_\mathbf{x}\psi_\vartheta(\mathbf{x}) - \nabla_\mathbf{x}\psi_\vartheta(\mathbf{x}')\|_2 \geq \mathsf{M}_{\mathbf{xx}}^g\|\mathbf{x} - \mathbf{x}'\|_2,$$

where $\mathsf{M}_{\mathbf{xx}}^g > 0$ is the convexity factor.

Proposition 2. *Assume the smoothness assumption holds. The loss of* $\mathcal{F} - \lambda\psi_\vartheta$ *is* $(\lambda\mathsf{M}^{g}_{\mathbf{xx}} - \mathsf{L}^{f}_{\mathbf{xx}})$*-strongly concave and* $(\lambda\mathsf{L}^{g}_{\mathbf{xx}} + \mathsf{L}^{f}_{\mathbf{xx}})$*-smoothness when* $\mathsf{L}^{f}_{\mathbf{xx}} < \lambda\mathsf{M}^{g}_{\mathbf{xx}}$*. That is*

$$-\frac{\lambda\mathsf{L}^{g}_{\mathbf{xx}} + \mathsf{L}^{f}_{\mathbf{xx}}}{2}\|\mathbf{x}' - \mathbf{x}\|_2^2 \leq \mathcal{L} \leq -\frac{\lambda\mathsf{M}^{g}_{\mathbf{xx}} - \mathsf{L}^{f}_{\mathbf{xx}}}{2}\|\mathbf{x}' - \mathbf{x}\|_2^2,$$

where $\mathcal{L} = \mathcal{F}(\theta, \mathbf{x}', y) - \lambda\psi_\vartheta(\mathbf{x}') - \mathcal{F}(\theta, \mathbf{x}, y) + \lambda\psi_\vartheta(\mathbf{x}) - \langle\nabla_{\mathbf{x}}(\mathcal{F} - \lambda\psi_\vartheta), \delta_{\mathbf{x}}\rangle = \mathcal{J}(\mathbf{x}') - \mathcal{J}(\mathbf{x}) - \langle\nabla_{\mathbf{x}}\mathcal{J}(\mathbf{x}), \delta_{\mathbf{x}}\rangle$.

Proof. By quadratic bounds derived from the smoothness, we have $-\frac{\mathsf{L}^{f}_{\mathbf{xx}}}{2}\|\mathbf{x}' - \mathbf{x}\|_2^2 \leq \mathcal{F}(\theta, \mathbf{x}', y) - \mathcal{F}(\theta, \mathbf{x}, y) - \langle\nabla_{\mathbf{x}}\mathcal{F}, \mathbf{x}' - \mathbf{x}\rangle \leq \frac{\mathsf{L}^{f}_{\mathbf{xx}}}{2}\|\mathbf{x}' - \mathbf{x}\|_2^2$. Since ψ_ϑ is convex, we get $\psi_\vartheta(\mathbf{x}') - \psi_\vartheta(\mathbf{x}) - \langle\nabla_{\mathbf{x}}\psi_\vartheta, \mathbf{x}' - \mathbf{x}\rangle \geq \frac{\mathsf{M}^{g}_{\mathbf{xx}}}{2}\|\mathbf{x}' - \mathbf{x}\|_2^2$. Since ψ_ϑ is smooth, we get $\psi_\vartheta(\mathbf{x}') - \psi_\vartheta(\mathbf{x}) - \langle\nabla_{\mathbf{x}}\psi_\vartheta, \mathbf{x}' - \mathbf{x}\rangle \leq \frac{\mathsf{L}^{g}_{\mathbf{xx}}}{2}\|\mathbf{x}' - \mathbf{x}\|_2^2$. Combining the above two-side inequalities with the format of $\mathcal{F} - \lambda\psi_\vartheta$ leads to the proposition. □

Theorem 1 below quantifies the gap between the approximate adversarial example $\mathbf{x}' = \mathbf{x} + \delta_{\mathbf{x}}^{(T)}$ and the optimal one, denoted by $\mathbf{x}^* = \mathbf{x} + \delta_{\mathbf{x}}^*$. The proof is lengthy and deferred to the supplementary material.

Theorem 1. *Suppose the smoothness assumption holds. If* $\mathsf{L}^{f}_{\mathbf{xx}} < \lambda\mathsf{M}^{g}_{\mathbf{xx}}$*, the perturbed example* $\mathbf{x}' = \mathbf{x} + \delta_{\mathbf{x}}^{(T)}$ *from Algorithm 1 satisfies:*

$$\frac{\mathcal{J}(\mathbf{x}^*) - \mathcal{J}(\mathbf{x}')}{\mathcal{J}(\mathbf{x}^*) - \mathcal{J}(\mathbf{x})} \leq \exp\left(-\frac{T}{d} \cdot \frac{\lambda\mathsf{M}^{g}_{\mathbf{xx}} - \mathsf{L}^{f}_{\mathbf{xx}}}{\lambda\mathsf{L}^{g}_{\mathbf{xx}} + \mathsf{L}^{f}_{\mathbf{xx}}}\right),$$

where d is the input dimension.

We now focus on the convergence of SGD when applied to the outer minimization. Without loss of generality, the following theorem is customized to the composition of φ_θ and $\mathcal{F}$, which can be extended to the composition of ψ_ϑ and $\mathcal{G}$. Let $\mathcal{H}(\theta) = \mathbb{E}_{(\mathbf{x},y)\in D_{\mathbf{x}}}\mathcal{F}(\theta, \mathbf{x}^*(\theta), y)$ denote the optimal adversarial loss on the entire training dataset $D_{\mathbf{x}}$.

Theorem 2. *Suppose the smoothness assumption holds. Let* $\Delta = \mathcal{H}(\theta^{(0)}) - \min_\theta \mathcal{H}(\theta)$*. If we set the learning rate to* $\gamma^{(j)} = \gamma = \min\{1/\mathsf{L}, \sqrt{\Delta/(\mathsf{L}\zeta^2 N)}\}$*, the adversarial training satisfies*

$$\frac{1}{N}\sum_{j=0}^{N}\mathbb{E}\left\|\nabla\mathcal{H}(\theta^{(j)})\right\| \leq \zeta\sqrt{8\frac{\Delta\mathsf{L}}{N}} + 2\hat{c}, \tag{13}$$

where N is the number of epochs, $\mathsf{L} = \frac{\mathsf{L}^{f}_{\theta\mathbf{x}}(\lambda\mathsf{L}^{g}_{\mathbf{x}\theta} + \mathsf{L}^{f}_{\mathbf{x}\theta})}{\lambda\mathsf{M}^{g}_{\mathbf{xx}} - \mathsf{L}^{f}_{\mathbf{xx}}} + \mathsf{L}^{f}_{\theta\theta}$*,* $\hat{c} = (\mathcal{J}(\mathbf{x}^*) - \mathcal{J}(\mathbf{x}))\frac{2\mathsf{L}^{f}_{\theta\mathbf{x}}}{\lambda\mathsf{M}^{g}_{\mathbf{xx}} - \mathsf{L}^{f}_{\mathbf{xx}}}\exp(-\frac{T}{d} \cdot \frac{\lambda\mathsf{M}^{g}_{\mathbf{xx}} - \mathsf{L}^{f}_{\mathbf{xx}}}{\lambda\mathsf{L}^{g}_{\mathbf{xx}} + \mathsf{L}^{f}_{\mathbf{xx}}})$*, and* ζ *is the variance of stochastic gradients.*

The proof is also deferred to the supplementary material. Theorem 2 says that the convergence rate of the adversarial training is $\mathcal{O}(1/\sqrt{N})$. Moreover, the approximation of the inner maximization has a constant effect on the convergence because of $\hat{c}$. More importantly, attacks achieving a lower attack effectiveness than this approximation possibly enlarge the effect and can be mitigated by this defense.

6 Experiments

We conduct experiments to validate the soundness of the proposed defense in the absence and presence of evasion attacks, while answering 4 Research Questions (RQs):

- **RQ1: Effectiveness of defenses in the absence of attacks**: How effective is PAD-SMA when there is no attack? This is important because the defender does not know for certain whether there is an attack or not.
- **RQ2: Robustness against oblivious attacks**: How robust is PAD-SMA against oblivious attacks where "oblivious" means the attacker is unaware of adversary detector g?
- **RQ3: Robustness against adaptive attacks**: How robust is PAD-SMA against adaptive attacks?
- **RQ4: Robustness against practical attacks**: How robust is PAD-SMA against attacks in the problem space?

Datasets. Our experiments utilize two Android malware datasets: Drebin [35] and Malscan [36]; both are widely used in the literature. The Drebin dataset initially contains 5,560 malicious apps and features extracted from 123,453 benign apps, while noting that the apps were collected before the year 2013. In order to obtain self-defined features, [9] re-collects the benign apps from the Androzoo repository [59] and re-scans the collections by VirusTotal service, resulting in 42,333 benign examples. This leads to the Drebin dataset used in this paper containing 5,560 malicious apps and 42,333 benign apps. Malscan [36] contains 11,583 malicious apps and 11,613 benign apps, spanning from 2011 to 2018. The software examples from both datasets are labeled using the VirusTotal service [60] (which contains tens of malware scanners), such that an app is flagged as malicious if at least five malware scanners say the example is malicious, and as benign if no malware-scanners flag it. We randomly split a dataset into three disjoint sets: 60% for training, 20% for validation, and 20% for testing.

Feature extraction and manipulation. We use two families of features. (**i**) **Manifest** features, including: *hardware* statements (e.g., camera and GPS module) because they may incur security concerns; *permissions* because they may be abused to steal user's privacy; implicit *Intents* because they are related to communications between app components (e.g., services). These features can be perturbed by injecting operations but may not be removed without undermining a program's functionality [9], [19]. (**ii**) **Classes.dex** features, including: "restricted" and "dangerous" Application Programming Interfaces (APIs), where "restricted" means invoking these APIs necessitates declaring the corresponding permissions and "dangerous" APIs include Java reflection usage (e.g., `getClass`, `getMethod`, `getField`), encryption usage (e.g., `javax.crypto`, `Crypto.Cipher`), the explicit intent indication (e.g., `setDataAndType`, `setFlags`, `addFlags`), dynamic code loading (e.g., `DexClassLoader`, `System.loadLibrary`), and low-level command execution (e.g., `Runtime.getRuntime.exec`). These APIs can be injected along with dead codes [10]. It is worth mentioning that APIs with the `public` modifier can be hidden by Java reflection [9], which involves reflection-related APIs that are used by our detector, referred to as "side-effect" features. These features may benefit the defense.

We exclude some features. For **manifest** features (e.g., package name, *activities*, *services*, *provider*, and *receiver*), they can be injected or renamed [9], [61]. For **Classes.dex** features, existing manipulations include *string* (e.g., IP address) injection/encryption [9], [19], *public* or *static* API calls hidden by Java reflection [9], [61], Function Call Graph (FCG) addition and rewiring [62], anti-data flow obfuscation [63],

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3265665

and control flow obfuscation (by using arithmetic branches) [61]. For **other types** of features, app signatures can be re-signed [61]; native libraries can be modified by Executable and Linkable Format (ELF) section-wise addition, ELF section appending, and instruction substitution [64].

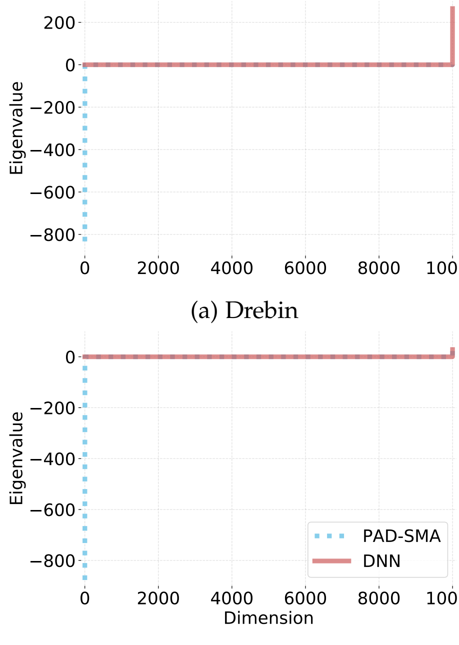
We use Androguard, a reverse engineering toolkit [65], to extract features. We apply a binary feature vector to denote an app, where "1" means a feature is present and "0" otherwise. The 10,000 top-frequency features are used.

Defenses we consider for comparison purposes. We consider 8 representative defenses:

- **DNN** [40]: _D_eep _N_eural _N_etwork (DNN) based malware detector with no effort at hardening it;
- **AT-rFGSM^k** [24]: DNN-based malware detector hardened by _A_dversarial _T_raining with _r_andomized round operation enabled _FGSM_^{k} attack (AT-rFGSM^k);
- **AT-MaxMA** [9]: DNN-based malware detector hardened by _A_dversarial _T_raining with the "_Max_" strategy enabled _M_ixture of _A_ttacks (AT-MaxMA);
- **KDE** [47]: Combining DNN model with a secondary detector for quarantining adversarial examples. The detector is a _K_ernel _D_ensity _E_stimator (KDE) built upon activations from the penultimate layer of DNN on normal examples;
- **DLA** [37]: The secondary detector aims to capture differences in DNN activations from the normal and adversarial examples. The adversarial examples are generated upon DNN. The activations from all dense layers are utilized, referred to as Dense Layer Analysis (DLA);
- **DNN$^+$** [21], [66]: The secondary detector plugs an extra class into the DNN model for detecting adversarial examples that are generated from DNN (DNN$^+$);
- **ICNN**: The secondary detector is the _I_nput _C_onvexity _N_eural _N_etwork (ICNN), which does not change the DNN but extends it from the feature space (Section 4.2);
- **PAD-SMA**: _P_rincipled _A_dversarial _D_etection is realized by a DNN-based malware detector and an ICNN-based adversary detector, both of which are hardened by adversarial training incorporating the _S_tepwise _M_ixture of _A_ttacks (PAD-SMA, Algorithm 1).

These defenses either harden the malware detector or introduce an adversary detector. DNN serves as the baseline. AT-rFGSM^k can achieve better robustness than adversarial training methods with the BGA, BCA, or Grosse attack [24]; AT-MaxMA with three PGD attacks can thwart a broad range of attacks but not iMaxMA, which is the iterative version of MaxMA [9]; KDE, DLA, DNN$^+$ and ICNN aim to identify the adversarial examples by leveraging the underlying difference inherent in ML models between a pristine example and its variant; PAD-SMA hardens the combination of DNN and ICNN by adversarial training.

Metrics. We report classification results on the test set via five standard metrics of False Negative Rate (FNR), False Positive Rate (FPR), F1 score, Accuracy (Acc for short, percentage of the test examples that are correctly classified) and balanced Accuracy (bAcc) [67]. Since we introduce g, a threshold τ is calculated on the validation set for rejecting examples. Let "@#" denote the percentage of the examples in the validation set being outliers (e.g., @5 means 5% of the examples are rejected by g).



(a) Drebin

(b) Malscan

Fig. 5: Sorted eigenvalues of Hessian matrix of $\mathcal{F} - \lambda\psi_\vartheta$ w.r.t. input when $\lambda = 1$.

6.1 RQ1: Effectiveness of Defenses in the Absence of Attacks

Experimental Setup. We learn the aforementioned 8 detectors on the two datasets, respectively. In terms of model architecture of the malware detector, the DNN detector has 2 fully-connected hidden layers (each layer having 200 neurons) with the ELU activation. The other 7 models also use this architecture. Besides, the adversary detector of DLA has the same settings as in [37]; ICNN has 2 convex hidden layers with 200 neurons each. For adversarial training, feature representations can be flipped from "0" to "1" if injection operation is permitted and from "1" to "0" if removal operation is permitted. Moreover, AT-rFGSM^k uses the PGD-ℓ_∞ attack, which additionally allows feature removals. It has 50 iterations with step size 0.02. AT-MaxMA uses three attacks, including PGD-ℓ_∞ iterates 50 times with step size 0.02, PGD-ℓ_2 iterates 50 times with step size 0.5, and PGD-ℓ_1 attack iterates 50 times, to conduct the training with penalty factor $\beta = 0.01$ because a large β incurs a low detection accuracy on the test sets. DLA and DNN$^+$ are learned from the adversarial examples generated by the MaxMA attack against the DNN model (i.e., adversarial training with an oblivious attack). PAD-SMA has three PGD attacks with the same step size as AT-MaxMA's except for g, which is learned from continuous perturbations. We set the penalty factors $\beta_1 = 0.1$ and $\beta_2 = 1.0$ on the Drebin dataset and $\beta_1 = 0.01$ and $\beta_2 = 1.0$ on the Malscan dataset. In addition, we conduct a group of preliminary experiments to choose λ from $\{10^{-3}, 10^{-2}, \dots, 10^3\}$ and finally set $\lambda = 1$ on both datasets. All detectors are tuned by the Adam optimizer with 50 epochs, mini-batch size 128, and learning rate 0.001, except for 80 epochs on the Malscan Dataset.

Preliminary results. Figure 5 illustrates sorted eigenvalues of Hessian matrix of the loss function $\mathcal{F} - \psi_\vartheta$ w.r.t. input. We

TABLE 1: Effectiveness (%) of detectors without adversary detection capability in the absence of attacks.

	Defense	Effectivenss (%)				
		FNR	FPR	Acc	bAcc	F1
Drebin	DNN [40]	3.64	0.45	99.18	97.96	96.45
	AT-rFGSM^k [24]	2.36	3.43	96.69	97.10	87.18
	AT-MaxMA [9]	1.73	3.11	97.05	97.58	88.46
	KDE [47]	3.64	0.45	99.18	97.96	96.45
	DLA [37]	3.18	0.58	99.12	98.12	96.21
	DNN$^+$ [21], [66]	3.36	0.50	99.17	98.07	96.42
	ICNN	3.64	0.45	99.18	97.96	96.45
	PAD-SMA	2.45	2.36	97.63	97.59	90.43
Malscan	DNN [40]	1.87	2.73	97.70	97.70	97.73
	AT-rFGSM^k [24]	0.84	5.49	96.86	96.84	96.96
	AT-MaxMA [9]	0.39	8.84	95.43	95.39	95.65
	KDE [47]	1.87	2.73	97.70	97.70	97.73
	DLA [37]	1.45	3.35	97.61	97.60	97.65
	DNN$^+$ [21], [66]	2.81	1.84	97.67	97.68	97.68
	ICNN	1.87	2.73	97.70	97.70	97.73
	PAD-SMA	0.42	8.58	95.54	95.50	95.75

TABLE 2: Accuracy (%) and F1 score (%) of detectors with adversary detection capability in the absence of attacks.

	Defense	@1 (%)		@5 (%)		@10 (%)	
		Acc	F1	Acc	F1	Acc	F1
Drebin	KDE	99.19	96.45	99.15	96.33	99.17	96.43
	DLA	99.14	96.27	99.13	96.27	99.14	96.53
	DNN$^+$	99.37	97.20	99.43	97.44	99.54	97.93
	ICNN	99.21	96.58	99.21	96.58	99.14	96.58
	PAD-SMA	97.79	90.82	97.99	88.61	98.14	79.54
Malscan	KDE	97.68	97.71	97.61	97.61	97.82	97.80
	DLA	97.65	97.67	97.69	97.63	97.80	97.64
	DNN$^+$	97.81	97.81	98.37	98.38	98.58	98.56
	ICNN	97.68	97.73	97.64	97.74	97.70	97.83
	PAD-SMA	95.66	95.89	95.72	95.83	95.59	95.47

randomly choose 100 instance-label pairs from test datasets of Drebin and Malscan, respectively. Let instances separately pass-through PAD-SMA or DNN (which has $\psi_\vartheta = 0$) for calculating eigenvalues, and then average the eigenvalues element-wisely corresponding to the input dimension. We observe that most eigenvalues are near 0, PAD-SMA produces large negative eigenvalues, and DNN has relatively small positive eigenvalues. This demonstrates that our PAD-SMA method can yield a concave inner maximization, and render the theoretical results applicable. Notably, PAD-SMA still has positive eigenvalues on the Malcan dataset, whereas certain robustness may be obtained as former adversarial training methods, yet no formal guarantees.

Results. Table 1 reports the effectiveness of detectors on the two test sets. We observe that DNN achieves the highest detection accuracy (99.18% on Drebin and 97.70% on Malscan) and F1 score (96.45% on Drebin and 97.73% on Malscan). These accuracies are comparative to those reported in [35], [36], [40]. We also observe that KDE and ICNN have the same effectiveness as DNN because both are built upon DNN while introducing a separative model to detect adversarial examples. We further observe that when training with adversarial examples (e.g., AT-rFGSM^k, AT-MaxMA, DLA, DNN$^+$, and PAD-SMA), detectors' FNR decreases while FPR increases, leading to decreased F1 scores. This can be attributed to the fact that only the perturbed malware is used in the adversarial training and that the data imbalance makes things worse.

Table 2 reports the accuracy and F1 score of detectors with adversary detection capability g. In order to observe the behavior of g, we abstain f from the prediction when $g(x) \geq \tau$. We expect to see that the trend of accuracy or F1 score will rise when removing as outliers more examples with high confidence from g on the validation set. However, this phenomenon is not always observed (e.g., DLA and ICNN). This might be caused by the fact that DLA and ICNN distinguish the pristine examples confidently in the training phase, while the rejected examples on the validation set are in the distribution and thus have little impact on the detection accuracy of f. PAD-SMA gets the downtrend of F1 score but not accuracy, particularly on the Drebin dataset. Though this is counter-intuitive, we attribute it to the adversarial training with adaptive attacks, which implicitly pushes g to predict the pristine malware examples with higher confidence than the benign ones. Thus, rejecting more validation examples actually causes more malware examples to be dropped, causing the remaining malware samples to be more similar to the benign ones and f to misclassify remaining malware, leading to lower F1 scores.

In summary, PAD-SMA decreases FNR but increases FPR, leading to decreased accuracies ($\leq$2.16%) and F1 scores ($\leq$6.02%), which aligns with the malware detectors learned from adversarial training. The use of adversary detectors in PAD-SMA does not make the situation better.

Answer to RQ1: There is no "free lunch" in the sense that using detectors trained from adversarial examples may suffer from a slightly lower accuracy when there are no adversarial attacks.

6.2 RQ2: Robustness against Oblivious Attacks

Experimental Setup. We measure the robustness of KDE, DLA, DNN$^+$, ICNN, and PAD-SMA against oblivious attacks via the Drebin and Malscan datasets. Since the other detectors (i.e., DNN, AT-rFGSM^k, and AT-MaxMA) do not have g, we do not consider them in this subsection. We use the detectors learned in the last group of experiments. The threshold is computed by dropping 5% validation examples with top confidence, which is suggested in [21], [37], [47], while noting that the accuracy of PAD-SMA is slightly better than that of AT-MaxMT at this setting.

We separately wage 11 oblivious attacks to perturb malware examples on the test set. For Grosse [40], BCA [24], FGSM [24], BGA [24], PGD-ℓ_1 [25], PGD-ℓ_2 [25], and PGD-ℓ_∞ [25], these attacks proceed iteratively till the 100th loop is reached. Grosse, BCA, FGSM, and BGA are proposed to only permit the feature addition operation (i.e., flipping some '0's to '1's). FGSM has the step size 0.02 with random rounding. Three PGD attacks permit both feature addition and feature removal: PGD-ℓ_2 has a step size 0.5 and PGD-ℓ_∞ has a step size of 0.02 (the settings are the same as adversarial training). For Mimicry [26], we leverage N_{ben} benign examples to guide the attack (dubbed Mimicry$\times N_{ben}$). We select the one that can evade f to wage attacks and use a random one otherwise. MaxMA [9] contains PGD-ℓ_1, PGD-ℓ_2, and

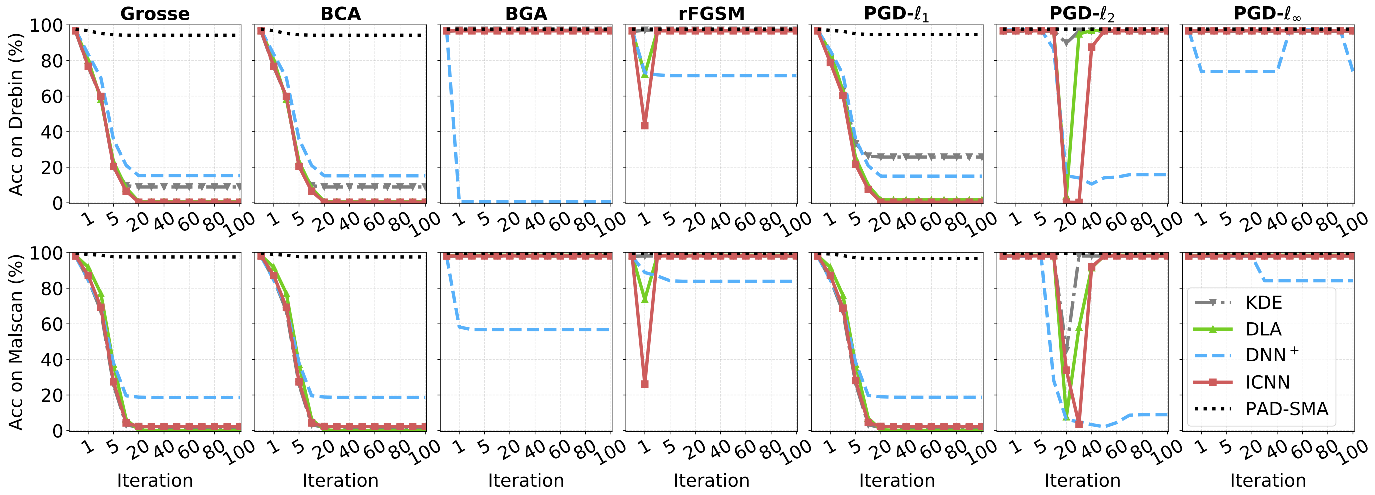


Fig. 6: Accuracy (Acc) of detectors against oblivious attacks with iteration from 0 to 100.

PGD-ℓ_∞ attacks. The iterative MaxMA (dubbed iMaxMA) runs MaxMA 5 times, with the start point updated. SMA has 100 iterations with a step size 0.5 for PGD-ℓ_2 and 0.02 for PGD-ℓ_∞. The three MA attacks use the scoring rule of Eq.(12) without g considered.

Results. Fig.6 depicts the accuracy curves of the detectors on Drebin (top panel) and Malscan (bottom panel) datasets under 7 oblivious attacks, along with the iterations ranging from 0 to 100. We make three observations. First, all these attacks cannot evade PAD-SMA (accuracy $\geq$ 90%), demonstrating the robustness of the proposed model.

Second, the Grosse, BCA, and PGD-ℓ_1 attacks can evade KDE, DLA, DNN$^+$, and ICNN when 20 iterations are used, while recalling that these three attacks stop manipulating malware when the perturbed example can evade malware detector f. It is known that DNN is sensitive to small perturbations; KDE relies on the close distance between activations to reject large manipulations; DLA and DNN$^+$ are learned upon the oblivious MaxMA, which modifies malware examples to a large extent; ICNN is also learned from salt-and-pepper noises which randomly change one half elements of a vector. Therefore, neither malware detector f nor adversary detector g of KDE, DLA, and ICNN can impede small perturbations effectively. This explains why KDE, DLA, and ICNN can mitigate BGA and PGD-ℓ_∞ attacks that use large perturbations.

Third, a dip exists in the accuracy curve of KDE, DLA, or ICNN against rFGSM and PGD-ℓ_2 when the iteration increases from 0 to 100. We find that both attacks can obtain small perturbations: rFGSM uses the random round (the rounding thresholds are randomly sampled from $[0, 1]$) [24] at iteration 1, and PGD-ℓ_2 produces certain discrete perturbations at iteration 20 via round (the threshold is 0.5).

Table 3 reports the attack results of Mimicry, MaxMA, iMaxMA, and SMA, which are not suitable for iterating with a large number of loops. We make three observations. First, PAD-SMA can effectively defend against these attacks, except for Mimicry×30 (with an accuracy of 65.45% on Malscan). Mimicry attempts to modify malware representations to resemble benign ones. As reported in Section 6.1, adversarial training promotes ICNN (g of PAD-SMA) to implicitly distinguish malicious examples from benign ones. Both aspects decrease PAD-SMA's capability of mitigating the oblivious Mimicry attack effectively. Second, all detectors can resist MaxMA and iMaxMA, except for DNN$^+$. Both attacks maximize the classification loss of DNN$^+$, leading DNN$^+$ to misclassify perturbed examples as benign (rather than the newly introduced label). Third, all detectors are vulnerable to the SMA attack (with maximum accuracy of 32.36% on Drebin and 26.68% on Malscan), except for PAD-SMA. This is because SMA stops perturbing malware when a successful adversarial example against f is obtained though the degree of perturbations is small, which cannot be effectively identified by g of KDE, DLA, DNN$^+$, and ICNN.

TABLE 3: Accuracy (%) of detectors under oblivious attacks (i.e., attacker is unaware of adversary detector g).

	Attack name	Accuracy (%)				
		KDE	DLA	DNN$^+$	ICNN	PAD-SMA
Drebin	No Attack	96.28	96.80	97.02	96.62	**97.64**
	Mimicry×1	56.64	55.82	58.18	54.91	**94.18**
	Mimicry×10	20.91	20.91	23.55	21.00	**84.18**
	Mimicry×30	10.64	10.64	12.82	10.00	**81.27**
	MaxMA	96.46	96.82	29.64	96.64	**97.64**
	iMaxMA	96.46	96.82	29.64	96.64	**97.64**
	SMA	32.09	27.82	31.18	32.36	**94.27**
Malscan	No Attack	98.02	98.41	97.86	98.11	**99.65**
	Mimicry×1	49.74	53.65	47.81	49.32	**83.68**
	Mimicry×10	18.13	18.68	21.68	17.06	**69.13**
	Mimicry×30	8.65	6.94	14.23	7.00	**65.45**
	MaxMA	98.13	98.55	84.23	98.16	**99.65**
	iMaxMA	98.13	98.55	84.23	98.16	**99.65**
	SMA	6.00	26.68	19.03	7.32	**96.68**

Answer to RQ2: PAD-SMA is significantly more robust than KDE, DLA, DNN$^+$, and ICNN against oblivious attacks. Still, PAD-SMA cannot effectively resist the Mimicry attacks that are guided by multiple benign samples.

6.3 RQ3: Robustness against Adaptive Attacks

Experimental Setup. We measure the robustness of the detectors against adaptive attacks on the Drebin and Malscan

TABLE 4: Accuracy (%) of detectors under adaptive attacks, where "Orth" stands for "orthogonal", "−" means an attack is not applicable.

	Attack name	Accuracy (%)							
		DNN	AT-rFGSM	AT-MaxMA	KDE	DLA	DNN$^+$	ICNN	PAD-SMA
Drebin	Groose	0.000	48.00	87.64	0.000	0.000	0.000	0.636	**90.91**
	BCA	0.000	47.73	87.64	6.182	0.000	4.727	3.000	**93.00**
	BGA	0.000	95.55	96.64	97.00	2.455	0.000	33.36	**97.64**
	rFGSM	0.000	97.46	**98.18**	97.00	96.82	70.91	96.64	97.64
	PGD-ℓ_1	0.000	44.46	80.91	0.182	0.000	0.000	0.091	**89.72**
	PGD-ℓ_2	3.455	89.73	96.27	87.36	0.000	8.727	0.091	**97.18**
	PGD-ℓ_∞	0.000	96.55	**98.09**	97.00	96.82	63.73	96.64	97.46
	Mimicry×1	54.91	88.91	90.27	56.64	55.82	58.18	54.91	**94.18**
	Mimicry×10	21.00	71.82	74.27	25.73	20.36	19.18	21.00	**81.18**
	Mimicry×30	10.00	66.45	70.64	16.09	10.09	7.909	10.00	**74.27**
	MaxMA	0.000	44.36	80.64	0.182	0.000	0.000	0.091	**89.09**
	iMaxMA	0.000	43.36	69.64	0.000	0.000	0.000	0.000	**88.73**
	SMA	0.000	57.82	84.09	16.36	0.000	8.636	0.000	**94.46**
	Orth PGD-ℓ_1	−	−	−	1.091	0.000	0.000	0.000	**97.64**
	Orth PGD-ℓ_2	−	−	−	17.46	2.455	13.55	3.909	**97.64**
	Orth PGD-ℓ_∞	−	−	−	96.82	31.73	55.18	96.46	**97.64**
	Orth MaxMa	−	−	−	1.091	0.000	0.000	0.000	**97.64**
	Orth iMaxMa	−	−	−	0.182	0.000	0.000	0.000	**97.64**
Malscan	Groose	0.000	9.129	77.26	0.000	0.000	0.000	0.871	**85.26**
	BCA	0.000	8.968	77.03	1.194	0.000	0.097	8.129	**89.32**
	BGA	0.000	10.97	95.68	98.13	0.194	30.19	37.45	**99.45**
	rFGSM	0.000	99.16	99.55	98.13	98.55	83.42	98.16	**99.65**
	PGD-ℓ_1	0.000	6.000	71.68	0.000	0.000	0.000	1.226	**84.87**
	PGD-ℓ_2	34.13	63.94	81.55	38.32	2.097	2.806	2.548	**95.90**
	PGD-ℓ_∞	0.000	99.16	**99.52**	98.13	98.55	41.07	98.10	99.45
	Mimicry×1	49.32	75.39	82.48	49.74	53.65	47.81	49.32	**83.68**
	Mimicry×10	17.06	49.13	60.71	17.52	18.23	11.65	17.06	**59.94**
	Mimicry×30	7.000	39.94	52.48	7.645	6.483	2.452	7.000	**53.68**
	MaxMA	0.000	5.742	61.77	0.645	0.000	0.000	0.935	**85.26**
	iMaxMA	0.000	1.645	47.07	0.097	0.000	0.000	0.935	**83.45**
	SMA	0.000	28.77	78.36	0.323	8.258	1.000	0.903	**97.48**
	Orth PGD-ℓ_1	−	−	−	2.000	0.000	0.032	0.000	**99.65**
	Orth PGD-ℓ_2	−	−	−	38.32	2.097	2.806	2.548	**99.65**
	Orth PGD-ℓ_∞	−	−	−	98.13	87.97	34.23	98.16	**99.65**
	Orth MaxMa	−	−	−	1.806	0.000	0.032	0.000	**99.65**
	Orth iMaxMa	−	−	−	0.484	0.000	0.032	0.000	**99.65**

datasets. We use the 8 detectors in the first group of experiments. The threshold τ is set as the one in the second group of experiments unless explicitly stated otherwise. The attacker knows f and g (if applicable) to manipulate malware examples on the test sets. We change the 11 oblivious attacks to adaptive attacks by using the loss function given in Eq.(6), which contains both $\mathcal{F}$ and ψ_ϑ. When perturbing an example, a linear search is conducted to look for a λ from the set of $\{10^{-5}, \ldots, 10^{5}\}$. In addition, the Mimicry attack can query both f and g and get feedback then. On the other hand, since DNN, AT-rFGSM, and AT-MaxMA contain no adversary detector, the oblivious attacks trivially meet the adaptive requirement. The other 5 attacks are adapted from orthogonal (Orth for short) PGD [23], including Orth PGD-ℓ_1, PGD-ℓ_2, PGD-ℓ_∞, MaxMA, and iMaxMA. We use the scoring rule of Eq.(12) to select the orthogonal manner. The hyper-parameters of attacks are set as same as the second group of experiments, except for PGD-ℓ_1 using 500 iterations, PGD-ℓ_2 using 200 iterations with a step size 0.05, and PGD-ℓ_∞ using 500 iterations with a step size 0.002.

Results. Table 4 summarizes the experimental results. We make three observations. First, DNN is vulnerable to all attacks, especially totally ineffective against 9 attacks (with 0% accuracy). The Mimicry attack achieves the lowest effectiveness in evading DNN because it modifies examples without using the internal information of victim detectors. AT-rFGSM can harden the robustness of DNN to some extent, but is still sensitive to BCA, PGD-ℓ_1, MaxMa, and iMaxMA attacks (with an accuracy $\leq$ 47.73% on both datasets). With an adversary detector, KDE, DLA, DNN$^+$, and ICNN can resist a few attacks (e.g., rFGSM and PGD-ℓ_∞), but the effectiveness is limited. AT-MaxMA impedes a range of attacks except for iMaxMA (with a 69.94% accuracy on Drebin and 47.07% on Malscan) and Mimicry×30 (with a 70.64% accuracy on Drebin and 52.48% on Malscan), which are consistent with previous results [9].

Second, PAD-SMA, significantly outperforming other defenses (e.g., AT-MaxMA), achieves robustness against 16 attacks on the Drebin dataset and 13 attacks on the Malscan dataset (with accuracy $\geq 85\%$). For example, PAD-SMA can mitigate MaxMA and iMaxMA, while AT-MaxMA can resist MaxMA but not iMaxMA (accuracy dropping by 11% on Drebin and 14.7% on Malscan). The reason is that PAD-SMA is optimized with convergence guaranteed, causing more iterations cannot promote attack effectiveness, which echoes our theoretical results. Moreover, PAD-SMA gains high detection accuracy ($\geq$ 97.64%) against orthogonal attacks because the same scoring rule is used and PAD-SMA renders loss function concave.

Third, Mimicry×30 can evade all defenses (with accuracy $\leq$ 74.27% on Drebin and $\leq$ 53.68% on Malscan). We additionally conduct two experiments on Drebin: (i) when we retrain PAD-SMA with penalty factor β_1 increased from $\beta_1 = 0.1$ to $\beta_1 = 1.0$, the detection accuracy increases to 85.27% against Mimicry×30 with the detection accuracy on the test dataset decreasing notably (F1 score decreasing to 78.06%); (ii) when we train PAD-SMA on Mimicry×30 with additional 10 epochs, the robustness increases to 83.64% against Mimicry×30 but also decreases the detection accuracy on the test set. These hint our method, as other adversarial malware training methods, suffers from a trade-off between robustness and accuracy.

Answer to RQ3: PAD-SMA, outperforming other defenses, can significantly harden malware detectors against a wide range of adaptive attacks but not Mimicry×30.

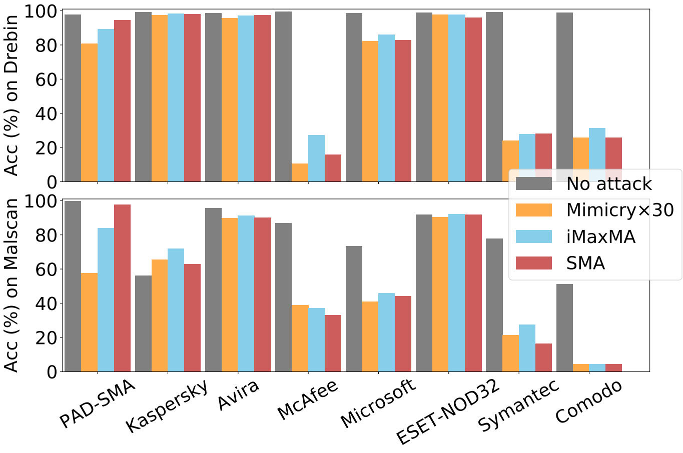


Fig. 7: Effectiveness of PAD-SMA and malware scanners against practical attacks.

6.4 RQ4: Robustness against Practical Attacks

Experimental Setup. We implement a system to produce adversarial malware for all attacks considered. We handle the inverse feature mapping problem (Section 4.3) as in [9], which maps perturbations in the feature space to the problem space. Our manipulation proceeds as follows: (i) obtain feature perturbations; (ii) disassemble an app using Apktool [68]; (iii) perform manipulation and assemble perturbed files using Apktool. We add manifest features and do not remove them so as not to manipulate an app's functionality. We permit all APIs that can be added and the APIs with `public` modifier but no *class inheritance* can be hidden by the reflection technique (see supplementary materials for details). In addition, the functionality estimation is conducted by *Android Monkey*, which is an efficient fuzz testing tool that can randomly generate app activities to execute on Android devices, along with logs. If an app and its modified version have the same activities, we treat them as having the same functionality. However, we manually re-analyze the non-functional ones to cope with the randomness of Monkey. We wage Mimcry×30, iMaxMA, and SMA attacks because they achieve a high evasion capability in the feature space.

TABLE 5: The number of apps with functionalities preserved from 100 randomly selected examples.

Dataset	Functionality	Apps (#)			
		No attack	Mimicry×30	iMaxMA	SMA
Drebin	Installable	89	89	89	89
	Monkey	80	68	66	65
Andro-zoo	Installable	86	84	86	83
	Monkey	76	58	65	64

Results. We respectively modify 1,098, 1,098, and 1,098 apps by waging the Mimcry×30, iMaxMA, and SMA attacks to the Drebin test set (leading to 1,100 malicious apps in total), and 2,790, 2,791, and 2,790 apps to the Malscan test set (leading to 3100 malicious apps in total). Most failed cases are packed apps against ApkTool. Table 5 reports the number of modified apps that retain the malicious functionality. Given 100 randomly chosen apps, 89 apps on Drebin and 86 apps on Malscan can be deployed on an Android emulator (running Android API version 8.0 and ARM library supported). Monkey testing says that the ratio of functionality preservation is at least 73.03% (65 out of 89) on the Drebin dataset and 69.05% (58 out of 84) on the Malscan dataset. Through manual inspection, we find that the injection of `null` constructor cannot pass the verification mechanism of the Android Runtime. Moreover, Java reflection sometimes breaks an app's functionality when the app verifies whether an API name is changed and then chooses to throw an error.

Fig.7 depicts the detection accuracy of detectors against Mimicry×30, iMaxMA, and SMA attacks. We observe that PAD-SMA cannot surpass Avira and ESET-NOD32 on both the Drebin and Malscan datasets. Note that these attacks know the feature space of PAD-SMA but not anti-malware scanners. Nevertheless, PAD-SMA achieves comparable robustness to the three attacks by comparing with Microsoft, and outperforms McAfee, Symantec, and Comodo. In addition, Kaspersky is seemingly adaptive to these attacks because it obtains a slightly better accuracy on the modified apps than the unperturbed ones ($\leq$15.59%) on the Malscan dataset.

Answer to RQ4: PAD-SMA is comparable to anti-malware scanners in the presence of practical attacks. It effectively mitigate iMaxMA and SMA attacks, but has limited success against Mimicry×30, akin to the cases of circumventing feature space attacks.

7 Related Work

We divide related studies into two contexts: Adversarial Malware Detection (AMD) vs. Adversarial ML (AML).

Defenses Against Adversarial Examples in AMD. We review related literature on (i) robust feature extraction, (ii) learning model enhancement, and (iii) adversarial malware detection. For the first aspect, Drebin features, including manifest instructions (e.g., required permissions) and syntax instructions (e.g., sensitive APIs), are usually applied to resist adversarial examples [10], [14], [35], [40]. Furthermore, Demontis et al. [19] demonstrate the robustness of Drebin features using several evasion attacks. However, a following study questions this observation using a mixture of attacks [9]. Moreover, to cope with *obfuscation* attacks, researchers suggest leveraging system API calls [5], and further enrich the representation by incorporating multiple modalities such as structural information (e.g., call graph), API usage (e.g., method argument types, API dependencies), and dynamic behaviors (e.g., network activity, memory dump) [4], [6], [69]. In the paper, we mainly focus on improving the robustness of the learning model, though the feature robustness is also important. Therefore, we refine Drebin features by screening out the ones that can be easily manipulated.

For the second aspect, the defense mechanisms aim to enhance a malware detector itself to classify adversarial examples accurately. Several approaches exist such as classifier randomization, ensemble learning, input transformation, and adversarial training, which are summarized by a recent survey [20]. We focus on adversarial training, which augments the training dataset with adversarial examples [24], [40], [44], [45]. In order to promote the robustness, the min-max adversarial training [42] in the ML context is adapted into the malware detection, endowing detectors with perceiving the optimal attack in a sense to resist non-optimal ones [24], [25]. In practice, the attackers are free enough to generate multiple types of adversarial examples, straightly leading to the instantiation of adversarial training incorporating a mixture of attacks [9]. In addition, combining adversarial training and ensemble learning further promotes the model robustness as long as the base model is robust enough [9]; a recent study also demonstrates that diversified features also promote the robustness of ensemble model [69]. This paper aims to establish principled min-max adversarial training methods with rigorously identifying the model's robustness. Moreover, a new mixture of attacks is used to instantiate our framework.

For the third aspect, the defenses quarantine the adversarial examples for further analysis. There are two categories of studies on recognizing adversarial examples. The first category studies detectors based on traditional ML models such as ensemble learning based (e.g., [70]). Inspired by the observation that grey-box attacks cannot thwart all basic building-block classifiers, Smutz et al. [70] propose identifying evasion attacks via prediction confidences. However, it is not clear how to adapt these ideas to deep learning models because they leverage properties that may not exist in DL models (e.g., neural networks are poorly, rather than well, calibrated [71]). The second class of studies leverages the invariant in malware features or in malware detectors to recognize adversarial examples. For example, Grosse et al. [66] demonstrate the difference between examples and their perturbed versions using the statistic test. Li et al. [72] and Li et al. [73] respectively propose detecting adversarial malware examples via stacked denoising autoencoders. However, these defense models seemingly cannot deal with the adaptive attacks effectively, based on recent reports [23], [66], [72]. Moreover, some defense models are not validated under the adaptive attacks [73]. When compared with these prior studies, our solution leverages a convex DNN model to recognize the evasion attacks, which not only works to detect adversarial examples, but also promotes principled defenses [32], leading to a formal treatment on robustness. Though our model has malware and adversary detectors, it is distinguishable from ensemble learning in the sense that different losses are utilized.

Adversarial training in AML. We review related literature on adversarial training. Adversarial training approach augments the training set with adversarial examples [41], [49]. Multiple heuristic strategies have been proposed to generate adversarial example, one of which is of particular interest to cast the adversarial training as the min-max optimization problem [42]. It minimizes the loss for learning ML models upon the most powerful attack (i.e., considering the worst-case scenario). However, owing to the non-linearity of DNNs, it is NP-hard to solve the inner maximization exactly [42]. Nevertheless, following this spirit, there are mainly two lines of studies to improve the min-max adversarial training: a line aims to select or produce the optimal adversarial examples (e.g., via advanced criterion or new learning strategies [34], [46], [74], [75]); another line aims to analyze statistical properties of resulting models theoretically (e.g.,via specific NN architectures or convexity assumptions [32], [76]). However, because adversarial training is domain-specific, it is non-trivially to straightly exploit these advancements for enhancing ML-based malware detectors.

8 Conclusion

We devised a provable defense framework against adversarial examples for malware detection. Instead of hardening the malware detector solely, an indicator is used to alert adversarial examples. We instantiate the framework via adversarial training with a new mixture of attacks, along with a theoretical analysis on the resulting robustness. Experiments via two Android datasets demonstrate the soundness of the framework against a set of attacks, including 3 practical ones. Future research needs to design other principled or verifiable methods. Learning or devising robust features, particularly the dynamical analysis, may be key to defeating adversarial examples. Other open problems include unifying practical adversarial malware attacks, designing application-agnostic manipulations, and formally verifying functionality preservation and model robustness.

References

[1] V. CHEBYSHEV. (2020, March) Mobile malware evolution 2020 @ONLINE. [Online]. Available: https://securelist.com/

[2] E. Raff, J. Barker, J. Sylvester, and et al., "Malware detection by eating a whole exe," *arXiv preprint arXiv:1710.09435*, 2017.

[3] Y. Ye, T. Li, D. A. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 41:1–41:40, 2017.

[4] X. Zhang, Y. Zhang, M. Zhong, and et al., "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the 2020 CCS*. New York, NY, USA: Association for Computing Machinery, 2020, p. 757–770.

[5] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proceedings of the 23rd KDD*. Halifax, NS, Canada: ACM, 2017, pp. 1507–1515.

[6] L. Onwuzurike, E. Mariconti, P. Andriotis, and et al., "Mamadroid: Detecting android malware by building markov chains of behavioral models," *ACM TOPS*, vol. 22, no. 2, pp. 1–34, 2019.

[7] X. Chen, C. Li, and et al., "Android HIV: A study of repackaging malware for evading machine-learning detection," *IEEE T-IFS*, vol. 15, pp. 987–1001, 2020.

[8] L. Chen, S. Hou, and Y. Ye, "Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks," in *ACSAC*. USA: ACM, 2017, pp. 362–372.

[9] D. Li and Q. Li, "Adversarial deep ensemble: Evasion attacks and defenses for malware detection," *IEEE T-IFS*, vol. 15, 2020.

[10] F. Pierazzi, F. Pendlebury, and et al., "Intriguing properties of adversarial ML attacks in the problem space," in *IEEE S&P, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 1332–1349.

[11] K. Zhao, H. Zhou, and et al., "Structural attack against graph based android malware detection," in *CCS, Virtual Event, Republic of Korea, November 15 - 19, 2021*. ACM, 2021, pp. 3218–3235.

[12] W. Song, X. Li, S. Afroz, and et al., "MAB-Malware: A reinforcement learning framework for blackbox generation of adversarial malware," in *ASIA CCS, Japan*. ACM, 2022, pp. 990–1003.

[13] S. Chen, M. Xue, L. Fan, and et al., "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *Comput. Secur.*, vol. 73, pp. 326–344, 2018.

[14] O. Suciu, R. Marginean, Y. Kaya, and et al., "When does machine learning FAIL? generalized transferability for evasion and poisoning attacks," in *USENIX Security Symposium*. USENIX Association, 2018, pp. 1299–1316.

[15] L. Demetrio, B. Biggio, G. Lagorio, and et al., "Functionality-preserving black-box optimization of adversarial windows malware," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 3469–3478, 2021.

[16] L. Demetrio, S. E. Coull, B. Biggio, and et al., "Adversarial examples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection," *ACM Trans. Priv. Secur.*, vol. 24, no. 4, pp. 27:1–27:31, 2021.

[17] A. Demontis, M. Melis, M. Pintor, and et al., "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *28th USENIX Security Symposium*. Santa Clara, CA, USA: USENIX Association, 2019, pp. 321–338.

[18] L. Chen, S. Hou, Y. Ye, and S. Xu, "Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks," in *FOSINT-SI'2018*, 2018, pp. 253–262.

[19] A. Demontis, M. Melis, B. Biggio, and et al., "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE TDSC*, vol. 16, no. 4, pp. 711–724, 2019.

[20] D. Li, Q. Li, Y. F. Ye, and S. Xu, "Arms race in adversarial malware detection: A survey," *ACM Comput. Surv.*, vol. 55, no. 1, 2021.

[21] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. Dallas, TX, USA: ACM, 2017, pp. 3–14.

[22] A. Athalye, N. Carlini, and D. A. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *CoRR*, vol. abs/1802.00420, 2018.

[23] O. Bryniarski, N. Hingun, and et al., "Evading adversarial example detection defenses with orthogonal projected gradient descent," in *10th ICLR*. OpenReview.net, 2022.

[24] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. O'Reilly, "Adversarial deep learning for robust detection of binary encoded malware," in *2018 IEEE Security and Privacy Workshops (SPW)*. San Francisco, USA: IEEE Computer Society, 2018, pp. 76–82.

[25] D. Li, Q. Li, Y. Ye, and S. Xu, "A framework for enhancing deep neural networks against adversarial malware," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 1, pp. 736–750, 2021.

[26] P. L. Nedim Rndic, "Practical evasion of a learning-based classifier: A case study," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 197–211.

[27] I. Incer, M. Theodorides, S. Afroz, and et al., "Adversarially robust malware detection using monotonic classification," in *Proceedings of the ACM IWSPA@CODASPY*. AZ, USA: ACM, 2018, pp. 54–63.

[28] Q. Lei, L. Wu, P. Chen, and et al., "Discrete adversarial attacks and submodular optimization with applications to text classification," in *Proceedings of MLSys 2019, CA, USA, 2019*, A. Talwalkar, V. Smith, and M. Zaharia, Eds. mlsys.org, 2019.

[29] H. Bao, Y. Han, Y. Zhou, and et al., "Towards understanding the robustness against evasion attack on categorical data," in *The Tenth ICLR, Virtual Event*. OpenReview.net, 2022.

[30] Y. Wang, Y. Han, H. Bao, and et al., "Attackability characterization of adversarial evasion attack on discrete data," in *The 26th ACM SIGKDD, Virtual Event, USA, 2020*. ACM, 2020, pp. 1415–1425.

[31] Y. Chen, S. Wang, D. She, and S. Jana, "On training robust PDF malware classifiers," in *29th USENIX Security Symposium*. USENIX Association, 2020, pp. 2343–2360.

[32] A. Sinha, H. Namkoong, and J. C. Duchi, "Certifying some distributional robustness with principled adversarial training," in *6th ICLR, Vancouver, Canada, Apr 30 - May 3*. OpenReview.net, 2018.

[33] Y. Wang, X. Ma, J. Bailey, and et al., "On the convergence and robustness of adversarial training," in *Proceedings of the 36th ICML*, vol. 97. PMLR, 09–15 Jun 2019, pp. 6586–6595.

[34] X. Jia, Y. Zhang, B. Wu, and et al., "LAS-AT: adversarial training with learnable attack strategy," in *IEEE/CVF Conference on CVPR, LA, USA, 2022*. IEEE, 2022, pp. 13 388–13 398.

[35] D. Arp, M. Spreitzenbarth, and et al., "Drebin: Effective and explainable detection of android malware in your pocket." in *NDSS*, vol. 14. San Diego, California, USA: The Internet Society, 2014, pp. 23–26.

[36] Y. Wu, X. Li, D. Zou, and et al., "Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis," in *34th IEEE/ACM International Conference on ASE, San Diego, CA, USA, November 11-15*. IEEE, 2019, pp. 139–150.

[37] P. Sperl, C. Kao, P. Chen, X. Lei, and K. Böttinger, "DLA: dense-layer-analysis for adversarial example detection," in *IEEE EuroS&P, Genoa, Italy, September 7-11*. IEEE, 2020, pp. 198–215.

[38] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[39] I. C. B. Biggio and D. M. et al., "Evasion attacks against machine learning at test time," in *Machine Learning and Knowledge Discovery in Databases: European Conference*. Springer, 2013, pp. 387–402.

[40] K. Grosse, N. Papernot, P. Manoharan, and et al., "Adversarial examples for malware detection," in *ESORICS*. Oslo, Norway: Springer, 2017, pp. 62–79.

[41] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd ICLR*. San Diego, CA, USA: OpenReview.net, 2015.

[42] A. Madry, A. Makelov, L. Schmidt, and et al., "Towards deep learning models resistant to adversarial attacks," in *6th ICLR, BC, Canada*. OpenReview.net, 2018.

[43] D. Li, Q. Li, Y. Ye, and S. Xu, "Enhancing deep neural networks against adversarial malware examples," *arXiv preprint arXiv:2004.07919*, 2020.

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

[45] L. Chen, Y. Ye, and T. Bourlai, "Adversarial machine learning in malware detection: Arms race between evasion attack and defense," in *EISIC'2017*, 2017, pp. 99–106.

[46] F. Tramèr, A. Kurakin, N. Papernot, and et al., "Ensemble adversarial training: Attacks and defenses," in *6th ICLR, BC, Canada*. OpenReview.net, 2018.

[47] T. Pang, C. Du, Y. Dong, and et al., "Towards robust detection of adversarial examples," in *Advances in NeurIPS*, 2018, pp. 4579–4589.

[48] C. Villani, *Topics in optimal transportation*. American Mathematical Soc., 2021, vol. 58.

[49] C. Szegedy, W. Zaremba, I. Sutskever, and et al., "Intriguing properties of neural networks," in *2nd ICLR, Banff, AB, Canada, April 14-16*, 2014.

[50] S. Moosavi-Dezfooli, A. Fawzi, J. Uesato, and et al., "Robustness via curvature regularization, and vice versa," in *IEEE Conference on CVPR, CA, USA*. IEEE, 2019, pp. 9078–9086.

[51] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 9, pp. 2805–2824, 2019.

[52] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, "Deep learning for android malware defenses: A systematic literature review," *ACM Comput. Surv.*, 2022.

[53] B. Kolosnjaji, A. Demontis, B. Biggio, and et al., "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th EUSIPCO*, Sep. 2018, pp. 533–537.

[54] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *Proceedings of the 36th ICML*, vol. 97. Long Beach, USA: PMLR, 2019, pp. 242–252.

[55] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *4th ICLR*. San Juan, Puerto Rico: OpenReview.net, 2016.

[56] B. Amos, L. Xu, and J. Z. Kolter, "Input convex neural networks," in *Proceedings of the 34th ICML, Sydney, NSW, Australia, 6-11 August*, vol. 70. PMLR, 2017, pp. 146–155.

[57] P. Oza and V. M. Patel, "One-class convolutional neural network," *IEEE Signal Process. Lett.*, vol. 26, no. 2, pp. 277–281, 2019.

[58] H. Wu, C. Wang, Y. Tyshetskiy, and et al., "Adversarial examples for graph data: Deep insights into attack and defense," in *Proceedings of the 28th IJCAI*. Macao, China: ijcai.org, 2019, pp. 4816–4823.

[59] K. Allix, T. F. Bissyandé, J. Klein, and et al., "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of International Conference on MSR*. NY, USA: ACM, 2016, pp. 468–471.

[60] H. Sistemas. (2021, May) Virustotal. [Online]. Available: https://www.virustotal.com

[61] F. Pellegatta. (2021, May) Aamo: Another android malware obfuscator. [Online]. Available: https://github.com/necst/aamo

[62] S. Aonzo, G. C. Georgiu, L. Verderame, and A. Merlo, "Obfuscapk: An open-source black-box obfuscation tool for android apps," *SoftwareX*, vol. 11, p. 100403, 2020.

[63] J. Jung, C. Jeon, M. Wolotsky, I. Yun, and T. Kim, "AVPASS: Leaking and Bypassing Antivirus Detection Model Automatically," in *Black Hat USA Briefings (Black Hat USA)*, Las Vegas, NV, Jul. 2017.

[64] Quarkslab. (2021, May) Lief: library for instrumenting executable files. [Online]. Available: https://ibotpeaches.github.io/Apktool

[65] A. Desnos. (2020, February) Androguard @ONLINE. [Online]. Available: https://github.com/androguard/androguard

[66] K. Grosse, P. Manoharan, N. Papernot, and et al., "On the (statistical) detection of adversarial examples," *CoRR*, vol. abs/1702.06280, 2017.

[67] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th International Conference on Pattern Recognition*. Istanbul, Turkey: IEEE Computer Society, 2010, pp. 3121–3124.

[68] C. Tumbleson. (2018, May) Apktool. [Online]. Available: https://ibotpeaches.github.io/Apktool

[69] M. Ficco, "Malware analysis by combining multiple detectors and observation windows," *IEEE Trans. Computers*, vol. 71, no. 6, pp. 1276–1290, 2022.

[70] C. Smutz and A. Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors." in *NDSS*, 2016.

[71] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proceedings of the 34th ICML*, vol. 70. Sydney, Australia: PMLR, 2017, pp. 1321–1330.

[72] D. Li, R. Baral, T. Li, and et al., "Hashtran-dnn: A framework for enhancing robustness of deep neural networks against adversarial malware samples," *arXiv preprint arXiv:1809.06498*, 2018.

[73] H. Li, S. Zhou, W. Yuan, and et al., "Robust android malware detection against adversarial example attacks," in *WWW '21: The Web Conference 2021*. Virtual Event: ACM, 2021, pp. 3603–3612.

[74] Y. Wang, D. Zou, J. Yi, and et al., "Improving adversarial robustness requires revisiting misclassified examples," in *8th ICLR, Addis Ababa, Ethiopia, April 26-30*. OpenReview.net, 2020.

[75] T. Bai, J. Luo, J. Zhao, and et al., "Recent advances in adversarial training for adversarial robustness," in *Proceedings of the IJCAI, Virtual Event, 19-27 August*. ijcai.org, 2021, pp. 4312–4321.

[76] Y. Xing, Q. Song, and G. Cheng, "On the generalization properties of adversarial training," in *The 24th AISTATS, Virtual Event*, vol. 130. PMLR, 2021, pp. 505–513.

[77] A. Paszke, S. Gross, F. Massa, and et al., "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*. BC, Canada: Curran Associates, Inc., 2019, pp. 8024–8035.

[78] F. Ceschin, M. Botacin, G. Lüders, and et al., "No need to teach new tricks to old malware: Winning an evasion challenge with xor-based adversarial samples," in *Reversing and Offensive-Oriented Trends Symposium*. NY, USA: ACM, 2021, p. 13–22.

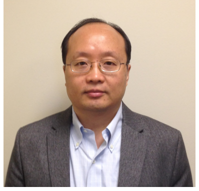
Deqiang Li received the M.E. degree in software engineering and the Ph.D. degree in computer science and technology from Nanjing University of Science and Technology, Jiangsu, China. He is currently a lecturer with Nanjing University of Posts and Telecommunications. His research interests include adversarial malware detection, adversarial machine learning, and applied data mining techniques in malware detection.

Shicheng Cui received the B.E. degree in software engineering and the Ph.D. degree in computer science and technology from Nanjing University of Science and Technology, Jiangsu, China. His research interests include graph mining, network representation learning and deep learning techniques in data science.

Yun Li received the Ph.D. degree in Computer Science from Chongqing University, Chongqing, China, and the postdoctoral fellow in Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. He is currently a professor in the School of Computer Science, Nanjing University of Posts and Telecommunications, China. His research mainly focuses on machine learning, data mining and parallel computing.

Jia Xu (M'15–SM'21) received the M.S. degree in School of Information and Engineering from Yangzhou University, Jiangsu, China, in 2006 and the PhD. Degree in School of Computer Science and Engineering from Nanjing University of Science and Technology, Jiangsu, China, in 2010. He is currently a professor in the School of Computer Science at Nanjing University of Posts and Telecommunications. His main research interests include crowdsourcing, edge computing and wireless sensor networks.

Xiao Fu (M'12) received the Ph.D. degree in computer science and technology from the Nanjing University of Science and Technology, Nanjing, China, in 2007. He is currently a Professor in the Jiangsu Key Laboratory of Big Data Security and Intelligent Processing at Nanjing University of Posts and Telecommunications. His main research interests are wireless sensor networks and mobile computing. He is a member of the IEEE Computer Society and the Association for Computing Machinery.



Shouhuai Xu (M'14–SM'20) is the Gallogly Chair Professor in the Department of Computer Science, University of Colorado Colorado Springs (UCCS). Prior to joining UCCS, he has been with University of Texas at San Antonio. 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 co-initiated the International Conference on Science of Cyber Security and is serving as its Steering Committee Chair. He received his PhD in Computer Science from Fudan University.