Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection

Deqiang Li and Qianmu Li

***Abstract*—Malware remains a big threat to cyber security, calling for machine learning based malware detection. While promising, such detectors are known to be vulnerable to evasion attacks. Ensemble learning typically facilitates countermeasures, while attackers can leverage this technique to improve attack effectiveness as well. This motivates us to investigate which kind of robustness the ensemble defense or effectiveness the ensemble attack can achieve, particularly when they combat with each other. We thus propose a new attack approach, named mixture of attacks, by rendering attackers capable of multiple generative methods and multiple manipulation sets, to perturb a malware example without ruining its malicious functionality. This naturally leads to a new instantiation of adversarial training, which is further geared to enhancing the ensemble of deep neural networks. We evaluate defenses using Android malware detectors against 26 different attacks upon two practical datasets. Experimental results show that the new adversarial training significantly enhances the robustness of deep neural networks against a wide range of attacks, ensemble methods promote the robustness when base classifiers are robust enough, and yet ensemble attacks can evade the enhanced malware detectors effectively, even notably downgrading the VirusTotal service.**

***Index Terms*—Adversarial Machine Learning, Deep Neural Networks, Ensemble, Adversarial Malware Detection.**

I. Introduction

MALWARE gains due attention from communities while still being a big threat to cyber security. For example, Symantec reported that 246,002,762 new malware variants emerged in 2018 [1]. Kaspersky detected 5,321,142 malicious Android packages in 2018 [2]. Worse yet, there is an increasing number of malware variants that attempted to undermine anti-virus tools and indeed evaded many malware detection systems [3].

In order to relieve the severe situation, communities have restored to machine learning techniques [4], [5]. While obtaining impressive performance, the learning-based models can be evaded by adversarial examples (see, for example, [6], [7], [8], [9]). Interestingly, a few manipulations are enough to perturb a malware example into an adversarial one, by which the perturbed malware example is detected as benign rather than malicious [10], [11]. This facilitates the research in the context of Adversarial Malware Detection (AMD), catering robust malware detectors against attacks.

Researchers have proposed enhancing the robustness of classifiers using ensemble methods such as adversarial training of ensemble [12] or ensemble adversarial training [13]. For the counterpart, attackers can leverage ensemble methods to promotes attack effectiveness as well such as by evading several classifiers [14], [15] or by waging multiple attacks simultaneously [16], [17]. This naturally raises the question – how is the effectiveness of ensemble attack or the robustness of ensemble defense when they combat with each other.

Our contributions. In this paper, we make the following contributions. First, we propose a new attack approach, named *mixture of attacks*, which enables attackers to leverage multiple generative methods and multiple manipulation sets to produce adversarial malware examples. To realize it, we adapt "max" attack [16], into the AMD context, and further propose iterating "max" attack in a greedy manner, so as to boost attack effectiveness. In addition, we adapt *salt and pepper noises attack* and *pointwise attack* [18], both of which are gradient-free, aiming to wage effective attacks when gradients of loss function suffer from certain issues [19].

Second, we instantiate the *adversarial training* [7] using a mixture of attacks and propose applying a manipulation set with the cardinality as large as possible. Further, we utilize this instantiation to harden the ensemble of deep neural networks, along with a theoretical analysis.

Third, we validate the robustness of malware detectors against 26 evasion attacks, which are categorized into five approaches: gradient-based, gradient-free, obfuscation, mixture of attacks, and transfer attack. Specifically, there are 10 gradient-based attacks: Projected Gradient Descent (PGD)-ℓ_1 [20], PGD-ℓ_2 [20], PGD-ℓ_∞ [21], Grosse [22], bit gradient ascent [7], bit coordinate ascent [7], PGD-Adam [23], Gradient Descent with Kernel Density Estimation (GDKDE) [6], Fast Gradient Sign Method (FGSM) [24], and jacobian-based saliency map attack [25]; 4 gradient-free attacks: 2 Mimicry attacks, salt and pepper noises, and pointwise; 5 obfuscation attacks: *Java* reflection, *string* encryption, *variable* renaming, junk code injection, and all four techniques above combined; 3 mixtures of attacks: "max" PGDs, iterative "max" PGDs, and iterative "max" PGDs+GDKDE, where PGDs means the mixture contains PGD-ℓ_1, PGD-ℓ_2, PGD-ℓ_∞, and PGD-Adam; 4 transfer attacks. We implement 6 Android malware detectors, including Basic DNN with no efforts to harden the model, 3 hardened DNNs incorporating adversarial

This work was supported by the Fundamental Research Funds for the Central Universities under Grant 30916015104; the National key research and development program: key projects of international scientific and technological innovation cooperation between governments under Grant 2016YFE0108000; CERNET Next Generation Internet Technology Innovation Project under Grant NGII20160122; the Project of ZTE Cooperation Research under Grant 2016ZTE04-11; Jiangsu province key research and development plan under Grant BE2016904; Jiangsu province key research and development programs: social development project under Grant BE2017739; industry outlook and common key technology projects under Grant BE2017100; and China Scholarship Council under Grant 201706840123.

D. Li and Q. Li are with Nanjing University of Science and Technology. e-mail: qianmu@njust.edu.cn

training with attack rFGSM (dubbed AT-rFGSM)[7], PGD-Adam (dubbed AT-Adam) [23], and "max" PGDs, and 2 adversarial deep ensembles (i.e., the hardened ensemble of deep neural networks incorporating adversarial training). We conduct systematical experiments on Drebin [26] and Androzoo [27] datasets, centering at the aforementioned question. Our findings are that:

- The hardened models incorporating "max" PGDs can detect more malware examples than the Basic DNN, AT-rFGSM, and AT-Adam at the cost of lowering detection accuracy on benign examples, and thus F1 score in the absence of attacks. Adversarial deep ensemble cannot serve as a remedy, and even reduces the detection accuracy on both malicious and benign examples.
- The hardened models incorporating "max" PGDs significantly outperform the Basic DNN, AT-rFGSM, and AT-Adam against evasion attacks. However, all models cannot defeat two types of attacks: attack mimicking benign examples (e.g., GDKDE and Mimicry), and mixture of attacks (e.g., iterative "max" PGDs).
- The ensemble promotes robustness against attacks when base classifiers are robust enough. The usefulness of ensemble, however, is un-deterministic when base models are vulnerable to attacks. Interestingly, the experimental results confirm our theoretical analysis.
- VirusTotal service [28] notably suffers from the iterative "max" PGDs+GDKDE attack. This is important because it shows adversarial evasion attacks may be a practical threat to cyber security.
- When attributing the important features for adversarial deep ensemble, we observe that sub-effective features (e.g., `com.google.ads.AdActivity`) are emphasized, thus leading to its robustness against attacks while trading off accuracy in non-adversarial settings. This implies the necessity of robust feature extraction.

Last but not the least, we make our codes publicly available at https://github.com/deqangss/adv-dnn-ens-malware.

Paper outline. The rest of the paper is organized as follows. Section II reviews the related work. Section III presents the ensemble of deep neural networks, evasion attacks (including two attacks adapted into AMD first time), and adversarial training. Section IV describes the mixture of attacks and adversarial deep ensemble. Section V evaluates attacks and defenses. Section VI discusses certain issues we concern. Section VII concludes the paper and shows future research.

II. Related Work

We review ensemble methods from the attacker's and the defender's perspectives, respectively.

A. Ensemble Attacks

There are two ensemble-based approaches improving the effectiveness of adversarial examples: (i) by attacking multiple classifiers and (ii) by using multiple attack methods.

For type (i), Liu et al. [14] suggest improving the transferability of adversarial examples by attacking an ensemble of deep learning models rather than a single one. This is because adversarial examples that can fool multiple models tend to have strong transferability [14], [15]. Dong et al. [29] further investigate three manners of organizing the base models and demonstrate that the ensemble of averaging *logits* outperforms the others for boosting the attack effectiveness.

For type (ii), Araujo et al. [17] show the difference between ℓ_2 and ℓ_∞ norm-based adversarial examples geometrically. Tramèr et al. [16] further propose attacking a classifier using multiple types of manipulations (e.g., constrained by ℓ_1 or ℓ_∞ norm). The empirical results show that the "max" attack lets the attacker evade the victim effectively.

In the context of AMD, one classical means is to perturb the discriminative features derived by *random forest* [30]. Recently, Al-Dujaili et al. [7] demonstrate the difference between four attacks in a sense that deep neural network based malware detectors enhanced by training with one attack cannot resist the other attacks.

We apply the aforementioned two approaches together with accommodating: (i) the discrete input domain and (ii) the constraint of retaining malicious functionality. We exploit the "max" strategy by permitting attackers to have multiple generative methods and multiple manipulation sets, which is different from the study [16] that focuses on the types of manipulations.

B. Ensemble Defenses

Biggio et al. [31], [32] propose defending against evasion attacks using *bagging* and *random subspace* techniques, which can produce *evenly-distributed* weights. One limitation is that the base classifier is restricted to the linear algorithm. For deep learning models, Abbasi et al. [33] propose "specialists + 1" ensemble, and Xu et al. [34] combine multiple *feature squeezing* techniques, so as to detect adversarial examples effectively. Both defenses, however, are defeated by a following study [35], which demonstrates that the ensemble of weak defenses cannot mitigate evasion attacks. Tramèr et al. [13] further introduce *ensemble adversarial training*, which learns one robust model by augmenting training data with adversarial examples transferred from multiple models. Grefenstette et al. [12] empirically demonstrate that adversarial training of ensemble achieves better robustness than the ensemble of adversarially trained multiple classifiers. Pang et al. [36] suggest enhancing the ensemble by diversifying base classifiers.

In the context of malware detection, Smutz and Stavrou [37] propose leveraging ensemble classifier to detect adversarial examples that are treated as outliers. Stokes et al. [38] show the resilience of ensemble to evasion attacks. Moreover, *stacking ensemble* is utilized to hinder adversarial malware examples [39]. All these defenses neglect the adversarial training that is an effective means to resist evasion attacks.

As a comparison, we aim to circumvent a wide range of evasion attacks in the AMD context. We also enhance the ensemble model by adversarial training (i.e., adversarial training of ensemble) [12], while incorporating a mixture of attacks, along with a theoretical analysis of ensemble.

III. PRELIMINARIES

We present the ensemble of deep neural networks, the evasion attacks, and the framework of adversarial training.

A. Ensemble of Deep Neural Networks

Given a malware example z in the file space $\mathcal{Z}$, i.e., $z \in \mathcal{Z}$, mapped as a d dimensional feature vector (i.e., feature representation) $\mathbf{x} \in \mathcal{X}$ by feature extraction method $\phi : \mathcal{Z} \to \mathcal{X}$, an ensemble classifier $f : \mathcal{X} \to \mathcal{Y}$ takes $\mathbf{x}$ as input and returns the predicted label 0 or 1, i.e., $\mathcal{Y} = \{0, 1\}$, where '0' means benign file and '1' means malicious. We consider the deep ensemble that is a linear combination of l Deep Neural Networks (DNNs) $\{\mathbf{F}_i\}_{i=1}^{l}$. Let $\mathbf{w}$ denote a weight vector $\mathbf{w} = (w_1, w_2, \cdots, w_l)$, satisfying $\mathbf{w} \in \mathcal{W}$ with $\mathcal{W} = \{\mathbf{w} : \mathbf{1}^\top \cdot \mathbf{w} = 1, w_i \geq 0\}$ for $i = 1, \ldots, l$. For waging effective attacks, we apply *logit ensemble* [29], which is:

$$\overline{\mathbf{F}}(\mathbf{x}) = \mathrm{softmax}(\sum_{i=1}^{l} w_i \mathbf{Z}_i(\mathbf{x})), \quad \text{s.t.} \quad \mathbf{w} \in \mathcal{W} \tag{1}$$

where $\mathbf{Z}_i$ denotes the logits of $\mathbf{F}_i$. Further, a stable version would be implemented to transform $\mathbf{Z}_i$ to $\mathbf{Z}_i - \max(\mathbf{Z}_i)$, so as to avoid numerical overflow, where $\max$ returns the maximum element in a vector. We obtain the predicted label by $f(\mathbf{x}) = \arg\max_j \overline{\mathbf{F}}(\mathbf{x})$, where $\arg\max$ returns the index of maximum element in a vector.

The parameters of ensemble, collectively denoted by $\overline{\theta}$, are optimized by minimizing a cost over the joint feature vector and label space, namely:

$$\min_{\overline{\theta}} \mathbb{E}_{(\mathbf{x},y)\in\mathcal{X}\times\mathcal{Y}} \left[L(\overline{\mathbf{F}}(\mathbf{x}), y) \right], \tag{2}$$

where $L : \mathbb{R}^{|\mathcal{Y}|} \times \mathcal{Y}$ is a loss function (e.g., cross entropy [40]). For simplifying notations, we use $\overline{\mathbf{F}}(\cdot)$ (rather than $\overline{\mathbf{F}}(\overline{\theta}; \cdot)$) to denote a parameterized ensemble of deep neural networks.

B. Evasion Attacks

1) Definition: We focus on evasion attacks in the context of AMD, which generally confines attackers by: perturbing malware examples in the test phase (rather than training phase) and retaining malicious functionality [7], [11], [41]. The terminology of *adversarial malware example* and *adversarial example* are used interchangeably, referring to the perturbed example that can evade the victim successfully.

Two versions of evasion attacks are used, corresponding to the *file space* $\mathcal{Z}$ and the *feature space* $\mathcal{X}$, respectively. In the file space, the attacker perturbs the malware example z into z' in order to mislead the classifier f [42]. Formally, given malware example-label pair (z, y) and manipulation set $\mathcal{M}_z$, the attacker intents to achieve:

$$\begin{aligned} & z' = z \oplus \delta, \\ \text{s.t.} \quad & (\delta \in \mathcal{M}_z) \wedge (f(\phi(z')) \neq y) \wedge (f(\phi(z)) = y) \end{aligned} \tag{3}$$

where $\oplus$ refers to the operation of applying manipulations, and manipulations in $\mathcal{M}_z$ retain the functionality of z (e.g, junk codes injection).

Instead of perturbing executable malware examples directly, an attacker could derive manipulations in the feature space $\mathcal{X}$, guiding the generation of adversarial examples in the file space [43], [44]. Let $\mathbf{M_x}$ denote *representation* manipulations for $\mathbf{x} = \phi(z)$, which is derived by

$$\mathbf{M_x} = \{(\phi(z') - \phi(z)) : z' = z \oplus \delta\} \text{ for } \forall\ \delta \in \mathcal{M}_z. \tag{4}$$

Formally, given a tuple $(\mathbf{x}, y)$, manipulations in the feature space $\mathbf{M_x}$, and the inverse feature extraction ϕ^{-1}, the attacker intents to achieve:

$$\begin{aligned} & z' = \phi^{-1}(\mathbf{x}'), \\ \text{s.t.} \ & (\mathbf{x}' = \mathbf{x} + \delta_{\mathbf{x}}) \wedge (\delta_{\mathbf{x}} \in \mathbf{M_x}) \wedge (f(\mathbf{x}') \neq y) \wedge (f(\mathbf{x}) = y). \end{aligned} \tag{5}$$

Notably, *feature space evasion attack* necessitates following two assumptions. Assumption 1 below says the inverse feature extraction ϕ^{-1} is solvable, and otherwise the corresponding adversarial example only exists in theory.

Assumption 1 (solvability assumption [43])**.** *Given $z \in \mathcal{Z}$ and feature extraction ϕ, the attacker can obtain $z' = \phi^{-1}(\mathbf{x}')$ when the representation $\mathbf{x}'$ is perturbed from $\mathbf{x}$ for $\mathbf{x} = \phi(z)$.*

Eq.(5) shows ϕ^{-1} works upon the manipulation set $\mathbf{M_x}$ which is derived by Eq.(4). It is a brute-force solution by calculating all perturbed files in advance, incurring efficiency issues. Researchers thus suggest alternatives [43], [44], [45], [46]. One example is predetermining $\mathbf{M_x}$ empirically based on the file space manipulation set $\mathcal{M}_z$. This would rely on the below assumption that the perturbed representation is bounded by a box constraint [44]. Let $\check{\mathbf{u}}$ and $\hat{\mathbf{u}}$ respectively denote the lower and upper boundaries in the feature space (i.e., elements in $\check{\mathbf{u}}$ are not greater than corresponding elements in $\hat{\mathbf{u}}$).

Assumption 2 (manipulation assumption [44])**.** *$\mathbf{x}'$ perturbed from $\mathbf{x}$ satisfies $\mathbf{x}' \in [\check{\mathbf{u}}, \hat{\mathbf{u}}]$ for $\forall\ \mathbf{x} \in \mathcal{X}$.*

With regard to Assumption 2 (i.e., $\mathbf{M_x} \subset [\check{\mathbf{u}} - \mathbf{x}, \hat{\mathbf{u}} - \mathbf{x}]$), we cannot modify the file z by following $\mathbf{x}' - \mathbf{x}$ particularly. One reason is features may be interdependent (i.e., modifying a representation value triggers other correlated ones changed), resulting in dependent manipulations [43], [46]. However, Assumption 2 misses to capture this dependence. In this work, we implement $\phi^{-1}(\mathbf{x}')$ subject to retaining malicious functionality, which may break the intuition of $\phi(\phi^{-1}(\mathbf{x}' - \mathbf{x})) = \mathbf{x}' - \mathbf{x}$. Our preliminary experiments show that the attack effectiveness barely suffers from this side-effect for most of the used features are independent[1] (see Section V-A3).

2) Threat Models: We consider a threat model specified by attacker's capability of perturbing examples and attacker's knowledge of the target system [6], [43], [45], [41], [47].

Attacker capability. As aforementioned earlier, an attacker is capable of perturbing malware examples in the test phase with malicious functionality preservation. In addition, the attacker is permitted to generate an example that is miss-classified with high cost (e.g., maximizing classifier's loss) [43], [7].

[1]This observation is application-special. To accommodate various types of feature extraction, we need to bridge the gap between the feature space attack and the file space attack, avoiding the use of empirical $\mathbf{M_x}$. We leave this problem in further work.

Attacker knowledge. There are three attack scenarios: *white-box* vs. *black-box* vs. *grey-box*. In the white-box scenario, the attacker knows everything of the targeted system, including defense mechanisms; in the black-box scenario, the attacker does not know internals of the victim, except for the predicted labels; the grey-box scenario resides in between, and we permit the attacker to know the dataset, the feature extraction method, and the learning algorithm, but not defense methods and learned parameters of the targeted malware detector.

3) Attack Strategies in the Literature: We consider a broad range of methods to specify evasion attacks, which are categorized into four approaches: gradient-based attacks, gradient-free attacks, obfuscations, and transfer attacks.

Gradient-based attacks. This approach permits attackers to wage white-box attacks, deriving adversarial examples using gradients of classifier's loss function. For example, ℓ_p $(p = 1, 2, \infty)$ norm based Projected Gradient Descent (PGD) is an effective means to maximizing the classifier's loss appropriately [21]. Note that the perturbation $\delta_{\mathbf{x}}$ is continuous during optimization process when following the direction of gradients. We map the perturbed representation by looking for the feasible nearest neighbor [20]. In addition, several related methods are adapted or proposed in the context of AMD, including Grosse [22], Fast Gradient Sign Method (FGSM) [24], [7], Jacobian-based Saliency Map Attack (JSMA) [48], [25], Bit Gradient Ascent (BGA) [7], Bit Coordinate Ascent (BCA) [7] and PGD-Adam [23], while noting that Grosse, JSMA, BGA, and BCA permit the feature addition only. Moreover, another method, named Gradient Descent with Kernel Density Estimation (GDKDE) [6], lifts perturbed examples into the populated region of benign ones.

Gradient-free attacks. This approach permits attackers to wage grey-box attacks. The Mimicry is usually applied, known as perturbing a malware example to mimic the benign ones [43], [45]. We further adapt two methods (both are originally proposed in the context of image processing) to modify malware examples: *salt and pepper noises* and *pointwise* [18]. Algorithm 1 shows the former one in the feature space, which perturbs representations using salt and pepper noises repetitively. Algorithm 2 describes the pointwise that, given an adversarial example, reduces its degree of manipulations while keeping its adversarial property.

Obfuscations. This approach enables attackers to wage *file space evasion attacks* with knowing nothing about the victim model (i.e., zero-query black-box attack). Typically, software sample can be manipulated using certain techniques (e.g., variable renaming). Indeed, researchers have reported the obfuscated malware examples can bypass detection [49], [45]. This inspires us to investigate this attack approach.

Transfer attacks. This approach suggests attackers waging *transfer attack* when knowing certain knowledge of f, such as a portion of features [43], [44]. The procedure mainly has following steps: perform reverse-engineering to obtain a surrogate model which resembles the targeted model f; perturb malware examples against the surrogate model; target the classifier f using the perturbed examples.

Algorithm 1: Salt and pepper noises attack in the feature space.

Input: The feature representation-label pair $(\mathbf{x}, y)$; the classifier f; the manipulation set $\mathbf{M}_{\mathbf{x}}$; a scalar $0 \leq \epsilon_{max} \leq 1$; the number of scalars N_s; the number of repetitions N_{rept}.
Output: The perturbed point $\mathbf{x}^*$.

Initialize $\mathbf{x}^* \leftarrow \mathbf{x}$;
repeat
  Produce evenly spaced scalars $\epsilon_1, \ldots, \epsilon_{N_s}$ over the range of $[0, \epsilon_{max}]$;
  for $j = 1$ *to* N_s **do**
    Generate salt and pepper noises $\delta_{\mathbf{x}}$ with the maximum degree of manipulation $\epsilon_j * d$ and $\delta_{\mathbf{x}} \in \mathbf{M}_{\mathbf{x}}$;
    Set $\mathbf{x}' \leftarrow \mathbf{x} + \delta_{\mathbf{x}}$;
    if $f(\mathbf{x}') \neq y$ **then**
      Set $\mathbf{x}^* \leftarrow \mathbf{x}'$ and $\epsilon_{max} \leftarrow \epsilon_j$;
      break;
  end
until N_{rept} *is reached*;
return $\mathbf{x}^*$.

Algorithm 2: Pointwise attack in the feature space.

Input: The feature representation-label pair $(\mathbf{x}, y)$; the classifier f; the adversarial representation $\mathbf{x}^* = (x_1^*, \ldots, x_d^*)$; the perturbed representation set $\mathcal{X}_{\mathbf{M}}$.
Output: The perturbed point $\check{\mathbf{x}}^*$.

repeat
  Set $\check{\mathbf{x}}^* \leftarrow \mathbf{x}^*$;
  Shuffle the list of indices $[d] = \langle 1, \cdots, d \rangle$ to another list $[(d)] = \langle (1), \cdots, (d) \rangle$ randomly;
  for $i = 0$ *to* d **do**
    if $x^*_{(i)} = x_{(i)}$ **then**
      continue;
    Modify $\mathbf{x}^*$ locally by setting $x^*_{(i)} \leftarrow x_{(i)}$;
    if $(\mathbf{x}^* \notin \mathcal{X}_{\mathbf{M}}) \vee (f(\mathbf{x}^*) = y)$ **then**
      reset $x^*_{(i)} \leftarrow \check{x}^*_{(i)}$;
  end
until $\check{\mathbf{x}}^* = \mathbf{x}^*$;
return $\check{\mathbf{x}}^*$.

C. Minmax Adversarial Training

Adversarial training lets classifiers know certain attacks proactively by augmenting the training data with adversarial examples [50], [24], [51], [52]. In particular, it has been proposed to consider adversarial training with the optimal attack, which in a sense corresponds to the worst-case scenario and therefore leads to classifiers that are robust against the non-optimal ones, namely the *minmax adversarial training* [7]:

$$\min_{\theta} \mathbb{E}_{(\mathbf{x},y)\in\mathcal{X}\times\mathcal{Y}} \left(L(\mathbf{F}(\theta;\mathbf{x}), y) + \max_{\mathbf{x}'\in\mathcal{X}_{\mathbf{M}}} L(\mathbf{F}(\theta;\mathbf{x}'), y) \right), \tag{6}$$

where θ denotes the parameters of a DNN $\mathbf{F}$ and $\mathcal{X}_{\mathbf{M}} \subseteq [\check{\mathbf{u}}, \hat{\mathbf{u}}]$ denotes a predetermined set of perturbed representations. It is worthy reminding that the inner maximization is intractable because of the non-convex DNN, resulting in local maxima.

IV. Methodology

We elaborate the mixture of attacks and adversarial deep ensemble, of which the adversarial deep ensemble relies on a mixture of attacks.

A. Mixture of Attacks

We propose mixture of attacks, an ensemble based approach, permitting attackers to perturb a malware example via multiple attack methods and multiple manipulation sets. Though this setting gives attackers more freedom, it is practical in the context of AMD. For instance, researchers suggest perturbing Android Packages by adding instructions into the file of *AndroidManifest.xml* [11], while addition operation can be applied to the file of *classes.dex* as well [25] and moreover, certain objects (e.g., *string*) can be hidden [45].

1) Overall Idea: With regard to Assumption 1 and 2 (which are empirically handled in Section V), we consider feature space evasion attacks. Let H denote the space of generative method and $\Delta_{\mathbf{x}}$ denote the space of manipulation set such that

Definition 1 (generative method)**.** *A generative method $h \in H$ takes as input the representation $\mathbf{x}$ with a constraint $\mathbf{M}_{\mathbf{x}} \in \Delta_{\mathbf{x}}$, and returns a perturbed representation $\mathbf{x}' = h(\mathbf{M}_{\mathbf{x}}; \mathbf{x})$.*

We characterize the "strength" of an attack upon h and $\mathbf{M}_{\mathbf{x}}$ via some scoring measurements, wherein the classifier loss $L(\overline{\mathbf{F}}(h(\mathbf{M}_{\mathbf{x}}; \mathbf{x})), y)$ is leveraged for a given $(\mathbf{x}, y)$ tuple. The higher loss value indicates a stronger attack.

The mixture of attacks has the same objective as the aforementioned approaches (e.g., gradient-based), aiming to maximize a score when manipulating malware examples. In contrast to attack strategies that design a generative method upon a manipulation set, a mixture of attacks attempts to construct n generative methods $\{h^i\}_{i=1}^n$ and m manipulation sets $\{\mathbf{M}_{\mathbf{x}}^j\}_{j=1}^m$, and then combine them to wage an attack, where $n \geq 1$ and $m \geq 1$.

2) Two Attack Strategies: In order to realize the mixture of attacks, we apply two straightforward strategies as follows:

- "Max" strategy: Given a representation-label pair $(\mathbf{x}, y)$, n generative methods $\{h^i\}_{i=1}^n$, and m manipulation sets $\{\mathbf{M}_{\mathbf{x}}^j\}_{j=1}^m$, the attacker attempts to choose a generative method, say $\widetilde{h}$, and a manipulation set, say $\widetilde{\mathbf{M}}_{\mathbf{x}}$, both of which joint to produce the optimal attack, namely that

$$\widetilde{h}; \widetilde{\mathbf{M}}_{\mathbf{x}} = \arg\max_{h; \mathbf{M}_{\mathbf{x}}} L(\overline{\mathbf{F}}(h(\mathbf{M}_{\mathbf{x}}; \mathbf{x})), y) \quad (7)$$
$$\text{s.t.,} \quad h \in \{h^i\}_{i=1}^n \wedge \mathbf{M}_{\mathbf{x}} \in \{\mathbf{M}_{\mathbf{x}}^j\}_{j=1}^m$$

- Iterative "max" strategy: The attacker performs the "max" strategy iteratively with each round adding perturbations on the resulting example came from the last iteration (in the first iteration, perturbations are applied on $\mathbf{x}$).

Algorithm 3 unifies and summarizes the two strategies. We calculate perturbed examples using two loops corresponding to generative methods $\{h^i\}_{i=1}^n$ and manipulation sets $\{\mathbf{M}_{\mathbf{x}}^j\}_{j=1}^m$ (line 3 - line 7). For line 8 and line 9, we select the optimal combination of attack method and manipulation set to perturb an example. Herein, parts of the algorithm (from line 3 to line 9) belongs to the "max" attack. The iterative case continues to proceed with taking as input the resulting point calculated by "max" attack, as shown in line 10. The procedure halts until the predetermined number of iteration is reached (line 2) or the convergence criterion is met (line 11, the change of score is less than a small scalar such as $\varepsilon = 10^{-9}$). The iterative case improves the attack effectiveness greedily, resulting in more directions being explored. In the future, one may consider more strategies to improve the "max" attack.

Algorithm 3: Iterative "max" attack in the feature space.

Input: The feature representation-label pair $(\mathbf{x}, y)$; n generation methods $\{h^i\}_{i=1}^n$; m manipulation sets $\{\mathbf{M}_{\mathbf{x}}^j\}_{j=1}^m$; the score measurement L; the number of iterations N; a small constant $\varepsilon > 0$.
Output: The perturbed point $\mathbf{x}'$.

```
1  Initialize x'_0 ← x;
2  for k = 1 to N do
3      for i = 1 to n do
4          for j = 1 to m do
5              Calculate h^i(M^j_x; x'_{k-1});
6          end
7      end
8      Select h̃ and M̃_x via Eq.(7);
9      Set x' ← h̃(M̃_x; x);
10     Set x'_k ← x';
11     if |L(x'_k, y) − L(x'_{k-1}, y)| < ε then
12         return x';
13 end
14 return x'.
```

B. Adversarial Deep Ensemble

We enhance the robustness of deep ensemble by adversarial training technique incorporating the "max" attack. In the end, we instantiate the minmax training (see Eq.6) as:

$$\min_{\overline{\theta}} \mathbb{E}_{(\mathbf{x},y)\in\mathcal{X}\times\mathcal{Y}} \left(L(\overline{\mathbf{F}}(\mathbf{x}), y) + \max_{h; \mathbf{M}_{\mathbf{x}}} L(\overline{\mathbf{F}}(h(\mathbf{M}_{\mathbf{x}}; \mathbf{x})), y) \right), \quad (8)$$
$$\text{s.t.} \quad h \in H \ \wedge \ \mathbf{M}_{\mathbf{x}} \in \Delta_{\mathbf{x}}.$$

Because of the information barrier between the defender and the attacker, we shall use the notations H and $\Delta_{\mathbf{x}}$ to replace attacker's empirical set $\{h^i\}_{i=1}^n$ and $\{\mathbf{M}_{\mathbf{x}}^j\}_{j=1}^m$, respectively. In contrast to the former study [7], there are three differences:

- H contains multiple approximate maximizers.
- $\Delta_{\mathbf{x}}$ contains a huge number of manipulation sets.
- $\overline{\mathbf{F}}$ is a deep ensemble (rather than a single DNN).

First, we may apply all possibly approximate maximizers to enhance the malware detectors, for the aim of defending

against a wide range of attacks. Owing to the efficiency issue, we choose and use the gradient-based maximizers.

Second, let $\check{\mathbf{M}}_{\mathbf{x}}$ and $\hat{\mathbf{M}}_{\mathbf{x}}$ denote two manipulation sets. The theorem below suggests producing adversarial examples upon the union of all manipulation sets, namely $\mathbf{M}_{\mathbf{x}} = \bigcup \Delta_{\mathbf{x}}$.

Theorem 1. *Upon two manipulation sets $\check{\mathbf{M}}_{\mathbf{x}}$ and $\hat{\mathbf{M}}_{\mathbf{x}}$, given a representation-label pair $(\mathbf{x}, y)$, the generative method h perturbs $\mathbf{x}$ by maximizing the classifier loss L, which leads to $L(\overline{\mathbf{F}}(h(\check{\mathbf{M}}_{\mathbf{x}}; \mathbf{x})), y) \leq L(\overline{\mathbf{F}}(h(\hat{\mathbf{M}}_{\mathbf{x}}; \mathbf{x})), y)$ when $\check{\mathbf{M}}_{\mathbf{x}} \subseteq \hat{\mathbf{M}}_{\mathbf{x}}$.*

Theorem 1 can be easily proved. Given $\check{\mathbf{M}}_{\mathbf{x}} \subseteq \hat{\mathbf{M}}_{\mathbf{x}}$, we derive $\mathcal{X}_{\check{\mathbf{M}}} = \{\mathbf{x} + \check{\delta}_{\mathbf{x}} : \check{\delta}_{\mathbf{x}} \in \check{\mathbf{M}}_{\mathbf{x}}\}$ and $\mathcal{X}_{\hat{\mathbf{M}}} = \{\mathbf{x} + \hat{\delta}_{\mathbf{x}} : \hat{\delta}_{\mathbf{x}} \in \hat{\mathbf{M}}_{\mathbf{x}}\}$, and further get $\mathcal{X}_{\check{\mathbf{M}}} \subseteq \mathcal{X}_{\hat{\mathbf{M}}}$. Due to $h(\check{\mathbf{M}}_{\mathbf{x}}; \mathbf{x}) \in \mathcal{X}_{\check{\mathbf{M}}}$, we reason $h(\check{\mathbf{M}}_{\mathbf{x}}; \mathbf{x}) \in \mathcal{X}_{\hat{\mathbf{M}}}$ and obtain $L(\overline{\mathbf{F}}(h(\check{\mathbf{M}}_{\mathbf{x}}; \mathbf{x})), y) \leq L(\overline{\mathbf{F}}(h(\hat{\mathbf{M}}_{\mathbf{x}}; \mathbf{x})), y)$. Recalling the Assumption 2 (i.e., box constraint), we apply $\mathbf{M}_{\mathbf{x}} = \bigcup \Delta_{\mathbf{x}}$. Rigorously, the theorem works when the generative method is the exact solution to maximizing the loss L, which is an intricate problem (see discussion in Section III-C). Nevertheless, our preliminary experiments show that the projected gradient descent based maximizers follow this theorem well.

Third, we design a robust ensemble as below.

1) Base Classifiers: The generalization error of ensemble drops significantly when base classifiers are effective and independent [53]. We consider diversifying base classifiers using the approach of data sample manipulation [54]. Specifically, we have each base classifier perceive adversarial examples that are produced by an approximate maximizer. This means we regularize the base classifiers using adversarial training but incorporating distinct attacks. Formally, the objective, denoted by J_{ens}, is

$$J_{ens} = J + \gamma \sum_{i=1}^{l} J_i, \tag{9}$$

where J denotes the Eq.(8), γ is a factor to balance the two items, and J_i is the regularization for ith base classifier:

$$J_i = \min_{\theta_i} \mathbb{E}_{(\mathbf{x},y)\in\mathcal{X}\times\mathcal{Y}} L(\mathbf{F}_i(\theta_i; h^i(\mathbf{M}_{\mathbf{x}}; \mathbf{x})), y). \tag{10}$$

Here θ_i $(i = 1, \ldots, l-1)$ denotes the parameters of base DNN $\mathbf{F}_i$, $\mathbf{M}_{\mathbf{x}} = \bigcup \Delta_{\mathbf{x}}$, and h^i is an approximate maximizer. In order to accommodate the unperturbed examples, the lth base classifier is learned from the pristine training set.

2) Combination: We optimize weights $\mathbf{w}$ by the Eq.(8), when given the DNNs $\mathbf{F}_1, \mathbf{F}_2, \cdots, \mathbf{F}_l$ with frozen parameters. The optimal weights can be solved by *Lagrange multiplier* [55], [56]. In order to make the optimization compatible to that of DNN, we leverage projected gradient descent:

$$\mathbf{w}_{i+1} = \mathrm{Proj}_{\mathcal{W}} \underbrace{\left(\mathbf{w}_i - \beta \cdot \nabla_{\mathbf{w}} J(\overline{\mathbf{F}})\right)}_{\mathbf{V}}, \tag{11}$$

where $\beta > 0$ is the learning rate and $\mathrm{Proj}_{\mathcal{W}}$ projects $\mathbf{V}$ into the space $\mathcal{W}$. We use the algorithm of Duchi et al. [56] to conduct the projection.

3) Analysis: We present a theoretical analysis of the deep ensemble against evasion attacks. Specifically, we quantify the robustness, by comparing to an ideal DNN, using a relaxation of averaging mean square error over logits. Formally, given an adversarial example set $\mathcal{X}^*_{\mathbf{M}} \subseteq \mathcal{X}_{\mathbf{M}}$, an ideal DNN $\mathbf{F}^*$ and a learned DNN $\mathbf{F}$, the error is defined as

$$Error_{\mathbf{F}}(\mathcal{X}^*_{\mathbf{M}}) = \mathbb{E}_{\mathbf{x}^*\in\mathcal{X}^*_{\mathbf{M}}} \left(\mathbf{Z}^*(\mathbf{x}^*_i) - \mathbf{Z}(\mathbf{x}^*_i)\right)^2,$$

where $\mathbf{Z}^*$ and $\mathbf{Z}$ denote the logits of DNN $\mathbf{F}^*$ and $\mathbf{F}$, respectively. Let $\eta_{\mathbf{Z},\mathbf{x}^*} = \mathbf{Z}^*(\mathbf{x}^*) - \mathbf{Z}(\mathbf{x}^*)$ denote the offset between $\mathbf{F}^*$ and $\mathbf{F}$. Without confusion, we drop $\mathbf{x}^*$ for $\eta_{\mathbf{Z},\mathbf{x}^*}$ and $\mathcal{X}^*_{\mathbf{M}}$ for $Error_{\mathbf{F}}(\mathcal{X}^*_{\mathbf{M}})$, leading to the compactness $\eta_{\mathbf{Z}}$ and $Error_{\mathbf{F}}$. Instead of the logits error, one may consider others (e.g., error upon softmax). Without loss of generalization, we herein aim to accommodate the logit ensemble, but the result obtained below can be extended to other ensembles.

We additionally make a hypothesis of $\overline{\mathbf{F}}$'s base classifiers being non-negatively correlated, i.e., $\mathbb{E}(\eta_{\mathbf{Z}_i})\mathbb{E}(\eta_{\mathbf{Z}_j}) \geq 0$ for $i, j \in \{1, \ldots, l\}$ and $i \neq j$, in a sense that all DNNs are learned from the same dataset and to solve similar tasks (which is validated in Section VI). Let $\mathbb{E}(\check{\eta}_{\mathbf{Z}})^2$ denote the smallest error achieved by one of base DNNs. We present the following theorem:

Theorem 2. *Given the deep ensemble $\overline{\mathbf{F}}$ with $\mathbb{E}(\eta_{\mathbf{Z}_i})\mathbb{E}(\eta_{\mathbf{Z}_j}) \geq 0$ for $i, j \in \{1, \ldots, l\}$ and $i \neq j$, the error of the ensemble satisfies $Error_{\overline{\mathbf{F}}} \geq (\mathbb{E}(\check{\eta}_{\mathbf{Z}})^2/l)$.*

Proof. We derive:

$$Error_{\overline{\mathbf{F}}} = \mathbb{E}(\eta_{\overline{\mathbf{Z}}})^2 = \mathbb{E}\left(\sum_{i=1}^{l} (w_i \eta_{\mathbf{Z}_i})^2\right)$$
$$= \sum_{i=1}^{l} w_i^2 \mathbb{E}(\eta_{\mathbf{Z}_i})^2 + \sum_{i=1}^{l}\sum_{j\neq i}^{l} \left(w_i w_j \mathbb{E}(\eta_{\mathbf{Z}_i})\mathbb{E}(\eta_{\mathbf{Z}_j})\right). \tag{12}$$

The optimal weights for the problem

$$\min_{\mathbf{w}\in\mathcal{W}} \sum_{i=1}^{l} w_i^2 \mathbb{E}(\eta_{\mathbf{Z}_i})^2$$

is

$$w_i = \left(\sum_{i=1}^{l} \frac{1}{\mathbb{E}(\eta_{\mathbf{Z}_i})^2}\right)^{-1} \frac{1}{\mathbb{E}(\eta_{\mathbf{Z}_i})^2} \ (i = 1 \cdots, l). \tag{13}$$

Substituting Eq. (13) into Eq. (12), we obtain:

$$Error_{\overline{\mathbf{F}}} \geq \sum_{i=1}^{l} w_i^2 \mathbb{E}(\eta_{\mathbf{Z}_i})^2 \geq \frac{1}{\sum_{i=1}^{l} \frac{1}{\mathbb{E}(\eta_{\mathbf{Z}_i})^2}} \geq \frac{\mathbb{E}(\check{\eta}_{\mathbf{Z}})^2}{l}.$$

This leads the theorem follows. □

From Theorem 2, we draw:

Insight 1. *The error of ensemble $Error_{\overline{\mathbf{F}}}$ could be smaller than the best base DNN, thus showing more robust than any individual classifier that is enclosed into the ensemble; The error of ensemble could be arbitrarily large when base classifiers cannot resist the evasion attacks.*

V. Experiments and Evaluation

A. Experimental Setup

In this section, we describe data pre-processing, classifiers (i.e., defenses) training, and manipulations in file and feature spaces.

1) Data Pre-Processing: We validate the effectiveness of attacks and defenses using Android malware detectors on Drebin [26] and Androzoo [27] datasets. The Drebin dataset [26] contains 5,615 malicious Android packages and SHA256 values of 123,453 benign examples. Based on the given SHA256 values, we downloaded 42,333 benign APKs from the APK markets, including 29,252 applications from GooglePlay store, 7,552 applications from AppChina, and 5,529 from other resources such as Anzhi. Androzoo [27] is an APK repository. To obtain the recent files, we downloaded 134,976 APKs that have the attached date from July 1st to December 31st in 2017. We sent all APKs to the VirusTotal service, which is an ensemble of over 70 anti-virus scanners (e.g., Kaspersky, McAfee, FireEye, Comodo, etc.). Based on the feedback, we obtain 15,467 malware examples and 91,295 benign examples. An APK is labeled as malicious if there are at least five anti-virus scanners say it is malicious, and is labeled as benign if no scanners detect it. We split both datasets respectively into three disjoint sets for training (60%), validation (20%), and testing (20%).

Feature extraction. APK is a zip file which contains *AndroidManifest.xml*, *classes.dex*, and others (e.g., *res*). The *AndroidManifest.xml* describes an APK's information, such as the package name and permission announcement. The functionality is built into *classes.dex* which is understandable by Java Virtual Machine (JVM).

Following prior adversarial learning studies [26], [45], [22], [25], we use the Drebin features, which consist of 8 subsets of features, including 4 subsets of features extracted from *AndroidManifest.xml* (denoted by S_1, S_2, S_3, S_4, respectively) and 4 subsets of features extracted from the disassembled dexcode (denoted by S_5, S_6, S_7, S_8, respectively). More specifically, (**i**) S_1 contains the features that are related to the access of an APK to the hardware of a smartphone (e.g., camera, touchscreen, or GPS module); (**ii**) S_2 contains the features that are related to the permissions requested by the APK in question; (**iii**) S_3 contains the features that are related to the application components (e.g., *activities*, *service*, *receivers*, etc.); (**iv**) S_4 contains the features that are related to the APK's communications with the operating system; (**v**) S_5 contains the features that are related to the critical system API calls, which cannot run without appropriate permissions or the *root* privilege; (**vi**) S_6 contains the features that correspond to the used permissions; (**vii**) S_7 contains the features that are related to the API calls that can access sensitive data or resources on a smartphone; (**viii**) S_8 contains the features that are related to IP addresses, hostnames and URLs found in the disassembled code.

In order to extract features of the applications, we utilize the Androgurad 3.3.5, which is a static APK analysis toolkit[57]. Note that 141 APKs in Drebin dataset cannot be analyzed. Moreover, a feature selection is conducted to remove those low-frequency features for the sake of computational efficiency [45]. As a result, we keep 10,000 features at top frequencies. The APK is mapped into the feature space as a binary feature vector, where '1' ('0') corresponds to a feature represent the presence (absence) in an APK.

2) Training Classifiers: We train six classifiers: (**i**) the basic DNN with no effort to enhance the model (dubbed Basic DNN); (**ii**) enhanced DNN incorporating *Adversarial Training* with the inner maximizer solved by iterative *FGSM* using *randomized* "rounding" (dubbed AT-rFGSM) [7]; (**iii**) enhanced DNN incorporating *Adversarial Training* with the inner maximizer solved by *Adam* optimizer (dubbed AT-Adam) [23]; (**iv**) enhanced DNN incorporating *Adversarial Training* with the inner maximizer solved by a *Mixture of Attacks* (dubbed AT-MA); (**v**) *Adversarial Deep Ensemble* with the inner maximizer solved by the *Mixture of Attacks (see Eq.*(8)*, dubbed ADE-MA)*; (**vi**) the ADE-MA with *diversity* promoted (see Eq.(9), dubbed dADE-MA).

Hyper-parameter settings. We use DNNs with two fully-connected hidden layers (each layer having neurons 160) and the ReLU activation function. For outer minimization (see Eq.(8)), all classifiers are optimized by using Adam with epochs 150, mini-batch size 128, and learning rate 0.001. For the inner maximization, the iterative FGSM is implemented as ℓ_∞ norm based PGD attack with step size 0.01 and iterations 100. The Adam-based maximizer is set up with the step size 0.02, iterations 100, and *random starting points* [23]. The mixture of attacks is realized as the "max" attack, which has four maximizers: ℓ_1 norm based PGD attack with step size 1.0 and iterations 50, ℓ_2 norm based PGD attack with step size 1.0 and iterations 100, ℓ_∞ norm based PGD attack with aforementioned settings, and Adam based PGD attack with aforementioned settings. Notably, dADE-MA has an extra hyper-parameter γ. We experimentally justify this hyper-parameter and choose $\gamma = 1$ on Drebin dataset and $\gamma = 0.1$ on Androzoo dataset. All attacks perturb the feature representations upon a manipulation set $\mathbf{M_x}$ (will be elaborated later).

Model selection. We learn the classifiers using Drebin and Androzoo training datasets, respectively. A selected model is the one that obtains the best "accuracy" on the validation set. The "accuracy" is the percentage of examples being classified correctly. For adversarial training, an additional term is considered, which is the accuracy of correctly classifying adversarial examples produced by the corresponding inner maximizers. The selected model is used for evaluation.

3) Specifying Manipulations: We specify manipulations applied to Android applications and estimate the perturbations for Drebin [26] feature representations accordingly.

Manipulation set $\mathcal{M}_z$ in the file space. Given an APK, we consider both *incremental* and *decremental* manipulations. For incremental manipulations, the attacker can insert some manifest features (e.g., request extra permissions and hardware, state additional *activities*, *services*, *Intent-filter*, etc.). However, some objects are hard to insert, such as *content-provider*, because the absence of Uniform Resource Identifier (URI) will corrupt an APK. With respect to the *.dex* file, junk codes (e.g., null *OpCode*, debugging information, dead functions or classes) can be injected without destroying the APK example. The similar means can be performed for the *string* (e.g., IP address) injection, as well.

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

```
TelephonyManager telecom = // default ;
String str = telecom.getDeviceId();
String encrypt_str = "EAQMVmZdGUV/VxdAVV9T";
// plain text: sendTextMessage
String mtd_name =
    DecryptString.convertToString(encrypt_str);
SmsManager smgr = SmsManager.getDefault();
Method send_sms = null;
send_sms =
    smgr.getClass().getMethod(mtd_name,
    String.class, String.class, String.class,
    PendingIntent.class, PendingIntent.class);
send_sms.invoke(smgr, "+50 1234567", null,
    str, null, null);
```

Listing 1: `Java` code snippet to hide the API method "sendTextMessage".

One challenge is that the attacker needs to perform fine-grained manipulations on compiled files automatically at scale, while preserving the functionalities of malware samples. This is important because a small mistake in a malware example can render the file un-executable. The preservation of malicious functionalities may be estimated by using a dynamic malware analysis tool, (e.g., Sandbox).

Manipulation set $\mathbf{M_x}$ in the feature space. The aforementioned manipulations modify static Android features such as API calls. We observe that two kinds of perturbations can be applied to the feature representations as follows:

- *Flipping '0' to '1'*: The attacker can increase the representation values of appropriate objects, such as components (e.g., *activity*), system APIs, and IP address.
- *Flipping '1' to '0'*: The attacker can flip '1' to '0' by removing or hiding objects (e.g., *activity* name, *public* or *static* APIs.)

Table I summarizes the operations in the Drebin feature space. We observe that the operations are not applicable to S_6 because this subset of features rely upon S_2 and S_5, meaning that modifications on S_2 or S_5 may cause changes of representation on S_6. In addition, we highlight that the manipulation set is larger than two recent studies [25], [11] that only consider flipping '0' to '1' in the feature space.

TABLE I: Overview of manipulations in the feature space, where ✓(✗) indicates that the operation of *flipping '0' to '1'* or *flipping '1' to '0'* can (cannot) be performed on features in the corresponding subset.

Feature sets (# of features)		$0 \rightarrow 1$	$1 \rightarrow 0$
`manifest`	S_1 Hardware (17)	✓	✗
	S_2 Requested permissions (247)	✓	✗
	S_3 Application components (8,619)	✓	✓
	S_4 Intents (866)	✓	✗
`dexcode`	S_5 Restricted API calls (118)	✓	✓
	S_6 Used permission (20)	✗	✗
	S_7 Suspicious API calls (19)	✓	✓
	S_8 Network addresses (94)	✓	✓

B. Evaluating the Effectiveness of Attacks and Defenses

We evaluate the attacks and defenses with centering at the earlier aforementioned question that is disintegrated as four sub-questions in the following:

- **RQ1**: How is the accuracy of adversarial deep ensemble for detecting malware examples in the absence of attacks?
- **RQ2**: How is the robustness of adversarial deep ensemble against a broad range of attacks, and how is the usefulness of ensemble against attacks?
- **RQ3**: How is the accuracy of anti-virus scanners under the mixture of attacks?
- **RQ4**: Why the enhanced classifiers can (cannot) defend against certain attacks?

Metrics. The effectiveness of classifiers is measured by five standard metrics: False-Positive Rate (FNR), False-Negative Rate (FNR), Accuracy (Acc), balanced Accuracy (bAcc) [58] and F1 score [59]. The balanced Accuracy and F1 score are considered because of the imbalanced dataset.

RQ1: *How is the accuracy of adversarial deep ensemble for detecting malware examples in the absence of attacks?* For answering RQ1, we evaluate the above mentioned six classifiers (i.e., Basic DNN, AT-rFGSM, AT-Adam, AT-MA, ADE-MA, and dADE-MA) on Drebin and Androzoo datasets, respectively.

Table II summarizes the results. We observe that when compared with the Basic DNN, the adversarial training based defenses achieve lower FNRs (at most a 2.13% decrease on Drebin dataset and 1.42% on Androzoo) but higher FPR (at most a 4.64% increase on Drebin dataset and 3.84% on Androzoo). For these defenses, AT-MA achieves the lowest FNR (1.59% on Drebin dataset and 1.32% on Androzoo) and ADE-MA encounters the highest FPR (4.96% on Drebin dataset and 4.32% on Androzoo). This can be explained as follows: by injecting adversarial malware examples into the training set, the learning process makes the model search for malware examples in a bigger space, resulting in the drop in FNR and increase in FPR. AT-MA, ADE-MA, and dADE-MA attain the similar FNR and FPR in the absence of attacks, due to the fact that the same generative methods are leveraged.

TABLE II: Effectiveness of the classifiers when there are no adversarial attacks.

Dataset	Classifier	Effectivenss (%)				
		FNR	FPR	Acc	bAcc	F1
Drebin	Basic DNN	3.72	0.32	99.28	97.98	96.93
	AT-rFGSM [7]	2.74	2.45	97.51	97.40	90.23
	AT-Adam [23]	3.27	1.45	98.34	97.64	93.22
	AT-MA	1.59	3.66	96.58	97.37	87.18
	ADE-MA	1.59	4.96	95.44	96.72	83.61
	dADE-MA	1.95	3.69	96.52	97.18	86.94
Andro-zoo	Basic DNN	2.74	0.48	99.18	98.39	97.26
	AT-rFGSM [7]	2.05	0.66	99.13	98.65	97.11
	AT-Adam [23]	1.61	0.96	98.94	98.72	96.51
	AT-MA	1.32	2.13	97.99	98.27	93.60
	ADE-MA	1.45	4.32	96.11	97.11	88.28
	dADE-MA	1.57	3.65	96.66	97.39	89.77

Moreover, these three defenses increase the balanced accuracy to their achieved accuracy as more malware examples are detected. Interestingly, both ensemble based defenses tend to have higher FNR and FPR than AT-MA. The underlying reason might be that adversarial deep ensemble focuses on more perturbed examples that are outliers. In summary, we draw:

Insight 2. *In the absence of attacks, the hardened models using mixture of attacks can detect more malware examples than the Basic DNN and the other hardened models (because of their smaller FNR), at the price of small side-effect in the FPR, classification accuracy and balanced accuracy, but notable side-effect in F1 score; adversarial ensemble exacerbates this situation, further lowering the effectiveness a little in terms of all measurements.*

RQ2: *How is the robustness of adversarial deep ensemble against a broad range of attacks, and how is the usefulness of ensemble against attacks?* For answering RQ2, we randomly select 800 malware examples from the test set to wage evasion attacks, attempting to fool the aforementioned six classifiers, namely Basic DNN, AT-rFGSM, AT-Adam, AT-MA, ADE-MA, and dADE-MA.

For gradient-based methods, in the settings of Grosse [22], BGA [7], BCA [7], JSMA [25], and ℓ_1 norm-based PGD (dubbed PGD-ℓ_1), we perturb one feature per time with the maximum iterations 100. For GDKDE [6], PGD-Adam [23], and ℓ_∞ norm-based PGD attack (dubbed PGD-ℓ_∞), we iterate the algorithm with the maximum iterations 1,000 and step size 0.01. The ℓ_2 norm-based PGD attack (dubbed PGD-ℓ_2) is set with the maximum iterations 1,000 and step size 1.0.

For gradient-free attacks, we wage salt and pepper noises attack (dubbed Salt+Pepper) with $N_{rept} = 10$, $\epsilon_{max} = 1$, and $N_s = 1,000$. Moreover, let Mimicry$\times N_{ben}$ denote a mimicry attack, in which we use N_{ben} benign examples to guide the perturbation of a malware example, leading to N_{ben} perturbed examples; then, we select the one from these N_{ben} perturbed example that causes the miss-classification with the smallest perturbations. Pointwise takes the resultant examples of Mimicry$\times N_{ben}$ as input (dubbed Pointwise$\times N_{ben}$).

The obfuscations are implemented via the AVPASS which is a tool to obfuscate Android applications [49]. We apply five attacks: *Java* reflection, *string* encryption, *variable* renaming, junk code injection, and the four techniques above combined.

In order to wage mixture of attacks, we leverage four PGD attacks (PGD-adam, PGD-ℓ_1, PGD-ℓ_2, and PGD-ℓ_∞), thus denoted "max" attack as "max" PGDs. The iterative version is denoted as I-"max" PGDs with iteration $N = 5$ and $\varepsilon = 10^{-9}$. Furthermore, by jointing the PGDs and GDKDE, we wage I-"max" PGDs+GDKDE attack.

When performing transfer attacks for the targeted model, we treat the other five classifiers as surrogate models individually. This means that, given a malware example, we perturb it upon the other five models, respectively. All the perturbed examples will be sent to query the targeted model.

Table III reports the accuracy of classifiers against attacks on Drebin and Androzoo datasets. From the upper half of Table III, we make the following observations. First, three defenses (AT-MA, ADE-MA, and dADE-MA) significantly enhance the robustness of DNNs, achieving the accuracy of $\geq 90.13\%$ under 19 attacks, $\geq 88.25\%$ under 18 attacks, and $\geq 89.75\%$ under 19 attacks, respectively. These achievements considerably outperform the Basic DNN, AT-rFGSM, and AT-Adam except for a lower accuracy (smaller than 8%) than AT-Adam under the GDKDE attack and a small lower ($< 1.99\%$) than AT-rFGSM under an obfuscation attack.

Second, ADE-MA and dADE-MA improve the robustness against gradient-free attacks, obfuscation attacks, and transfer attacks. When came to the gradient-based attacks and mixture of attacks, however, both defense models are not useful, and even hinder the robustness in some cases. For instance, compared to the AT-MA, neither of the two ensembles mitigate the I-"max" PGDs effectively (a 26.88% decrease for ADE-MA and 12.38% decrease for dADE-MA).

Third, all defenses suffer from the GDKDE ($\leq 78.13\%$), PGD-ℓ_∞ ($\leq 84.5\%$), Mimicry$\times$30 ($\leq 83\%$), Pointwise ($\leq 81.75\%$) and the three mixtures of attacks ($\leq 79.63\%$). For GDKDE and Mimicry, the reason may be that these generative methods produce adversarial representations that have similar data distribution as benign examples, leading that no learning-based classifiers can detect these attacks effectively. Pointwise further promotes the attack effectiveness when using Mimicry$\times$30 as the initial attack. For PGD-ℓ_∞, the reason may be that the corresponding maximizer does not suffice to obtain adversarial examples in the training phase. This further leads to the effectiveness of "max" PGDs.

From the lower half of Table III, we additionally make the following observations. Fourth, the effectiveness of gradient-based attacks is not comparative to the gradient-free attacks (e.g., Mimicry), which counters the results in Drebin dataset. This may be that the feature representations of malicious examples are closer to benign ones in the Androzoo dataset than that in Drebin.

Fifth, though AT-MA achieves higher accuracy of detecting most of gradient-based attacks, its robustness is lower than AT-Adam under the PGD-ℓ_2 attack and also lower than AT-rFGSM under the PGD-ℓ_∞ attack. This can be explained that AT-MA induces a loss over the four attacks, which in turn leads to lower robustness than hardened models that focus on an attack solely.

TABLE III: Effectiveness of the six classifiers, including Basic DNN, Adversarial Training (AT)-rFGSM, AT-Adam, AT-Mixture of Attacks (MA), Adversarial Deep Ensemble (ADE)-MA, and diversified ADE-MA (dADE-MA), against adversarial evasion attacks.

Dataset	Attack Type	Attack Name	Accuracy (%)					
			Basic DNN	AT-rFGSM	AT-Adam	AT-MA	ADE-MA	dADE-MA
Drebin	No Attack	–	96.63	97.25	96.63	**98.38**	**98.38**	98.25
	Gradient-based attacks	Grosse [22]	0.00	58.50	63.63	**92.00**	88.25	89.88
		BGA [7]	0.00	97.25	96.63	**98.38**	98.25	98.25
		BCA [7]	0.00	60.75	63.38	92.00	89.63	**93.13**
		JSMA [25]	0.00	65.25	63.63	**92.00**	89.25	91.38
		FGSM [7]	0.00	97.25	96.63	**98.38**	**98.38**	98.25
		GDKDE [6]	0.00	66.75	**78.13**	70.13	76.13	71.13
		PGD-Adam [23]	0.38	93.50	89.25	**96.63**	94.75	**96.63**
		PGD-ℓ_1 [20]	0.00	41.50	49.00	**90.13**	85.25	89.75
		PGD-ℓ_2 [20]	0.63	96.00	89.38	96.00	94.50	**96.50**
		PGD-ℓ_∞ [20]	0.00	48.75	74.13	72.38	81.75	**84.50**
	Gradient-free attacks	Salt+Pepper	78.25	96.25	94.00	95.00	**97.63**	95.75
		Mimicry $\times 1$	53.88	90.63	87.00	95.75	**98.13**	94.75
		Mimicry $\times 30$	10.63	62.13	60.75	80.38	**83.00**	77.75
		Pointwise $\times 30$	9.50	59.38	59.00	79.25	**81.75**	76.50
	Obfuscation	*Java* reflection	96.63	97.25	96.63	**98.38**	**98.38**	98.25
		String encryption	96.42	96.93	96.55	98.21	**98.34**	98.08
		Variable renaming	96.84	97.47	96.58	**98.35**	**98.35**	98.23
		Junk code injection	95.70	99.26	98.81	99.11	**99.70**	99.56
		All techniques combined	90.66	**99.60**	99.40	98.21	99.40	97.61
	Mixture of attacks (MA)	"max" PGDs	0.00	30.13	45.00	71.13	71.00	**79.63**
		I-"max" PGDs	0.00	22.75	15.00	**65.13**	38.25	52.75
		I-"max" PGDs+GDKDE	0.00	4.50	14.88	**63.13**	36.13	51.00
	Transfer attacks	GDKDE	36.75	88.83	87.60	92.25	**95.30**	92.58
		PGD-ℓ_1	17.03	96.08	92.68	97.35	**98.20**	97.05
		PGD-ℓ_∞	34.35	96.65	95.25	96.75	**97.33**	97.18
		I-"max" PGDs+GDKDE	23.10	95.53	94.38	94.23	**97.33**	96.93
Androzoo	No Attack	–	97.75	98.13	98.63	**98.75**	98.25	98.13
	Gradient-based attacks	Grosse [22]	22.63	39.00	26.00	**92.25**	89.25	89.38
		BGA [7]	23.00	98.13	98.63	**98.75**	98.13	98.13
		BCA [7]	22.63	39.00	26.00	**92.13**	89.63	91.63
		JSMA [25]	41.25	47.63	43.50	**92.50**	89.38	91.38
		FGSM [7]	37.75	98.13	98.63	**98.75**	98.25	98.13
		GDKDE [6]	1.13	10.38	2.63	30.63	**54.50**	50.88
		PGD-Adam [23]	50.25	85.13	55.13	93.50	95.75	**97.25**
		PGD-ℓ_1 [20]	22.63	37.88	26.00	**89.88**	88.38	88.50
		PGD-ℓ_2 [20]	51.13	74.00	82.25	76.50	95.63	**96.38**
		PGD-ℓ_∞ [20]	22.63	**97.75**	78.25	52.38	92.13	88.13
	Gradient-free attacks	Salt+Pepper	10.00	70.38	87.88	69.75	81.13	**97.00**
		Mimicry $\times 1$	0.13	21.75	13.25	58.13	69.38	**71.00**
		Mimicry $\times 30$	0.00	2.75	0.75	22.88	**48.88**	46.25
		Pointwise $\times 30$	0.00	1.75	0.38	19.88	**46.75**	40.63
	Obfuscation	*Java* reflection	97.98	97.98	98.85	**99.14**	96.54	96.25
		String encryption	95.15	95.59	95.59	**96.92**	94.71	94.71
		Variable renaming	96.82	97.27	97.50	**97.73**	96.82	96.59
		Junk code injection	31.43	69.52	68.57	86.67	98.10	**99.05**
		All techniques combined	13.79	34.48	37.93	48.28	**100.0**	96.55
	Mixture of attacks	"max" PGDs	22.63	37.25	26.00	42.25	**83.88**	82.75
		I-"max" PGDs	22.63	36.00	25.75	29.75	59.63	**72.75**
		I-"max" PGDs+GDKDE	0.63	3.13	0.25	19.13	**36.50**	30.13
	Transfer attacks	GDKDE	6.95	48.63	42.13	**77.35**	74.70	76.73
		PGD-ℓ_1	20.63	81.20	70.03	93.53	**96.95**	96.88
		PGD-ℓ_∞	48.20	96.15	91.70	97.53	**98.83**	98.80
		I-"max" PGDs+GDKDE	3.45	89.08	75.55	85.55	93.93	**97.15**

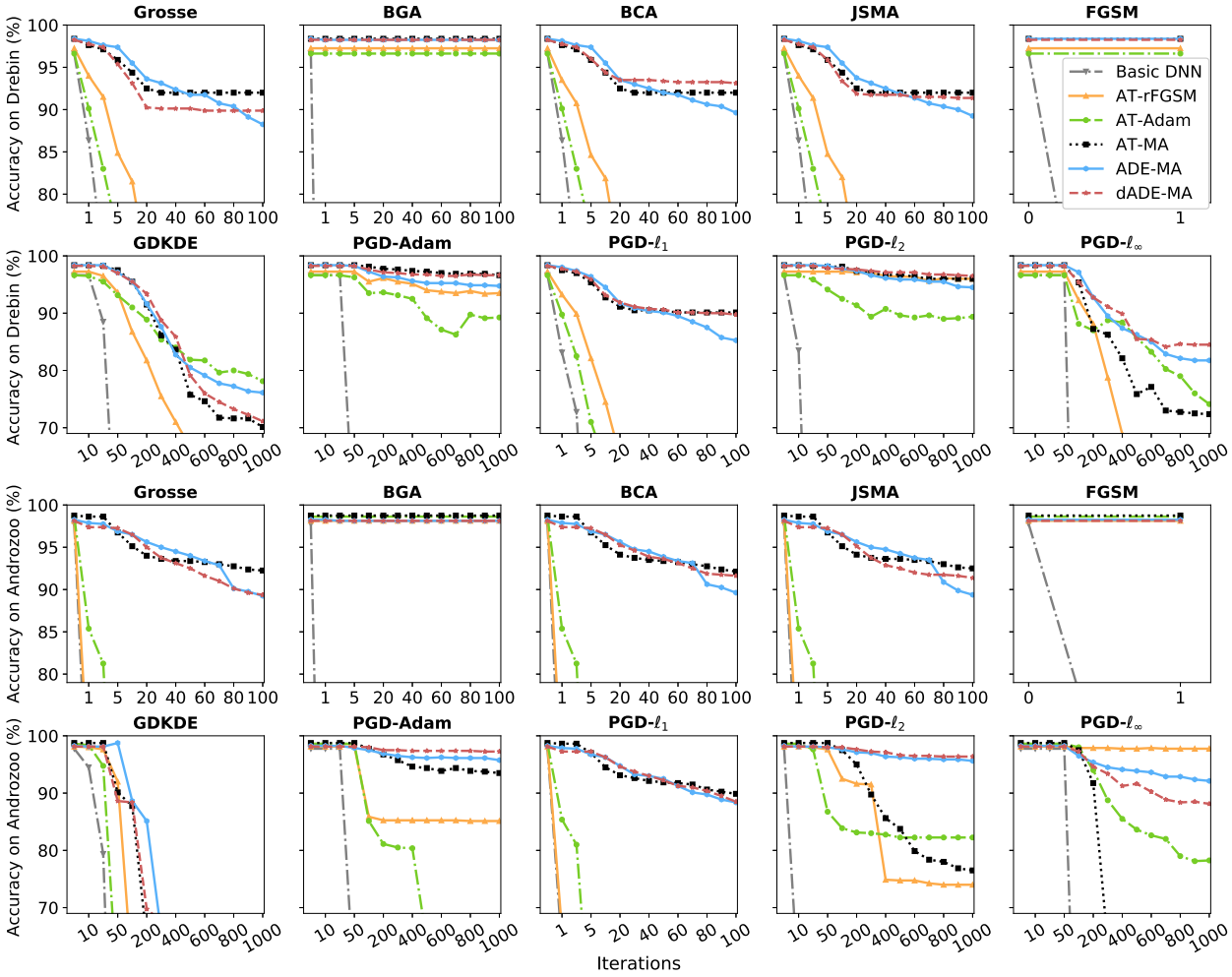


Fig. 1: Accuracy of classifiers against gradient-based attacks with different iterations on Drebin and Androzoo datasets.

Figure 1 depicts the accuracy of classifiers against gradient-based attacks with different iterations. With more details, we further make the observations as follows. Sixth, once the iterations exceed a certain extent, GDKDE can evade all hardened models effectively. Moreover, the hardened models, incorporating adversarial training with "max" PGDs, cannot thwart the PGD-ℓ_∞ attack as effectively as the PGD-ℓ_1 attack.

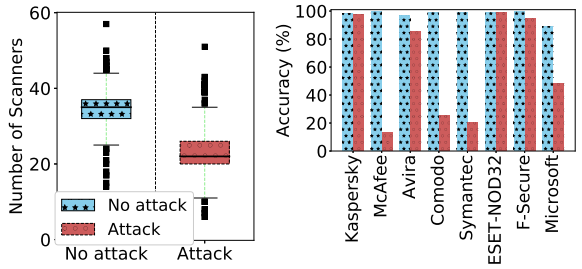
Seventh, ADE-MA tends to achieve better accuracy than the AT-MA when attacks are launched with small iterations, while this situation exchanges when iterations are increased to a large extent. It is worth noting that AT-MA attains the best accuracy on the unperturbed malware examples in the Androzoo dataset, resulting in the following phenomenon that, along with the iteration increased, AT-MA obtains higher accuracy than ADE-ME at the start of curves, while lower accuracy in the middle, and back to higher accuracy in the end. To some extent, this confirms our theoretical analysis of ensemble that promotes the robustness when base classifiers are robust enough.

Eighth, under most of the attacks such as Groose, BCA, JSMA and PGD ℓ_1, the defense of dADE-MA serves as a remedy for ADE-MA against attacks at large iterations (about > 60). This explains why dADE-MA circumvents more attacks than ADE-MA (see Table III). In summary, we draw insights:

Insight 3. *The hardened models incorporating mixture of attacks can defend against a broad range of attacks effectively, but remaining vulnerable to mimicry attacks and mixtures of attacks; Ensemble promotes the robustness against an attack when base classifiers are robust enough.*

RQ3: *How is the accuracy of anti-virus scanners under the mixture of attacks?* For answering RQ3, we wage transfer attacks to target the VirusTotal. The surrogate model is dADE-MA and the generative method is I-"max" PGDs+GDKDE. We perturb the randomly selected 800 malware examples from Drebin test set. Apktool is leveraged to perform reverse engineering [60]. We finally obtain 800 perturbed malware examples, along with their unperturbed versions, which are together queried VirusTotal service.

Figure 2 exhibits the experimental results of attacking VirusTotal. From the box-plot of Figure 2a, we observe that malware examples are predicted as malicious confidently except that about 1% of them ($\sim$8 files in 800 examples) being detected by below 24 scanners, while adversarial attacks notably affect the prediction, by noting that most of the perturbed examples are detected by lower than 25 scanners.



(a) (b)

Fig. 2: Waging transfer attack against VirusTotal service. (a) Box plot of the number of anti-virus scanners that predict a queried example as malicious when applied 800 pristine malware examples (i.e., no attack) and their 800 perturbed ones (i.e., attack). We perturb the malware examples using I-"max" PGDs+GDKDE attack against dADE-MA. (b) Accuracy of 8 scanners when applied the (un)perturbed examples.

Nonetheless, no attacks can evade VirusTotal thoroughly. Figure 2b showcases 8 famous scanners. We observe that the attack barely affects the Kaspersky, ESET-NOD32, and F-Secure. McAfee, Comodo, and Symantec, however, present vulnerability to these adversarial examples. In summary, we draw:

Insight 4. *The transfer version of mixture of attacks can downgrade the VirusTotal service and evade certain anti-virus scanners effectively.*

RQ4: *Why the enhanced classifiers can (cannot) defend against certain attacks?* For answering RQ4, we statistically attribute "important" features for the defender and the attacker, respectively. Here the "important" feature is the one that has a large influence on the classification accuracy. To this purpose, we present a case study on the defense dADE-MA and the attack I-"max" PGDs+GDKDE on the Drebin dataset. For the attacker, we intuitively investigate the features perturbed by the adversary with high-frequencies. For the defender, an intuitive solution is deficient due to the intricate structure of deep ensemble. Therefore, we introduce an alternative using feature selection technique via a surrogate model. The procedure proceeds as follows: (i) train a surrogate model on the training set to mimic dADE-MA; (ii) rank the features based on feature importance extracted from the surrogate model; (iii) mask feature representations based on the ranking result by setting non-important features value as 0; (iv) send the masked feature representations to dADE-MA and calculate the change of accuracy; (v) repeat step (iii)-(iv) and terminate it until a predetermined number of important features is reached or the accuracy is lower than a threshold; (vi) refine the importance upon the retained features using *permutation importance* [61]. We use *random forest* to learn the surrogate model, aiming to obtain feature importance coarsely. In step (iii)-(v), we leverage binary search to speed up filtering out trivial features.

Table IV demonstrates the top 20 important features for the classifier dADE-MA and the attack I-"max" PGDs+GDKDE, respectively. We make the following observations. First, 18 features (90%) benefit the detection of benign examples, which may be induced by the imbalanced dataset. Second, 8 important features (40%) of the classifier are manipulated by I-"max" PGDs+GDKDE frequently, in particular the feature `getNetworkInfo`, which promotes malware detection. Third, features of the classifier that rank at top are neglected by this attack. Two reasons could account for this: the attack is failed to search for some of these features; both malicious and benign applications use these features frequently, such as `android.permission.INTERNET` for accessing internet service. Finally, `com.google.ads.AdActivity` is at the 5th place. Though counter-intuitive, it may be that adversarial deep ensemble enforces the model to focus on this neutral feature. The above observations collaboratively explain why dADE-MA obtains an accuracy of 51% against I-"max" PGDs+GDKDE attack and sacrifices detection accuracy in the absence of attacks.

Insight 5. *The hardened model is prone to use sub-effective features, so as to gain robustness against attacks but a little trading off accuracy in absence of attacks.*

VI. Discussion

Functionality Estimation. We emulate malware behaviors by Cuckoodroid [62] that is an automated static and dynamical analysis toolkit for APKs. Due to the efficiency issue, we randomly select 10 APKs from 800 perturbed examples that are sent to VirusTotal, along with their original versions, to conduct the estimation. We observe that two malware examples and their perturbed versions cannot run on the emulator, and thus exclude them from the testing. For other perturbed applications, 3 of them execute successfully on the emulator, 2 apps can be deployed into Android runtime but failed to run, and the remained 3 apps cannot be installed. This shows more research is needed to solve the problem of retaining malicious functionality.

Small vs. large degree of manipulations. In this study, we let attackers suffice iterations to maximize the classifier's loss when perturbing malware examples as long as the malicious functionality is preserved. In term of ℓ_1 norm, some attacks perturb malware examples slightly, for example the JSMA attack against the Basic DNN with the average perturbations 4.48; some others perturb malware examples to a large extent, for example the PGD-ℓ_∞ attack against the Basic DNN with the average perturbations 2,772.87. In addition, the I-"max" PGDs+GDKDE attack perturbs malware examples with the average perturbations 2,600.02, 53.23, and 45.16 when targeting the Basic DNN, AT-Adam, and dADE-MA, respectively.

Validating the hypothesis of theoretical analysis. We empirically justify the hypothesis (see Section IV-B3) that base classifiers of ensemble are non-negatively correlated under adversarial example attacks. In this end, we directly let perturbed examples pass through a deep ensemble and measure the *Pearson correlation coefficient* between any two base models upon the logit belonging to the label '1'. The ideal classifier is treated as a constant and thus neglected

TABLE IV: Top 20 important features for the defense of dADE-MA vs. I-"max" PGDs+GDKDE attack. We further report whether the feature facilitates the classification accuracy for benign ($-$) or malicious ($+$) examples and, whether the feature is frequently manipulated by flipping '1' to '0' ($\downarrow$) or flipping '0' to '1' ($\uparrow$).

Defender	$+/-$	Attacker	$\uparrow$ / $\downarrow$
android/widget/VideoView;→start	$-$	com.airpush.android.PushServiceStart61159	$\uparrow$
android/widget/MediaPlayer;→start	$-$	.httpserver.HttpServerService	$\uparrow$
android.permission.ACCESS_COARSE_LOCATION	$-$	android/telephony/TelephonyManager;→getLine1Numb	$\downarrow$
android.permission.INTERNET	$-$	android/telephony/TelephonyManager;→listen	$\uparrow$
com.google.ads.AdActivity	$-$	android/location/LocationManager;→removeUpdates	$\uparrow$
android.permission.ACCESS_FINE_LOCATION	$-$	android/location/LocationManager;→requestLocationUpdates	$\uparrow$
android/location/LocationManager;→requestLocationUpdates	$-$	android/location/LocationManager;→getLastKnownLocation	$\uparrow$
android/location/LocationManager;→removeUpdates	$-$	android/net/ConnectivityManager;→getNetworkInfo	$\downarrow$
android/net/ConnectivityManager;→getActiveNetworkInfo	$-$	android.permission.ACCESS_FINE_LOCATION	$\uparrow$
getSystemService	$-$	android.permission.READ_CONTACTS	$\uparrow$
android.permission.ACCESS_NETWORK_STATE	$-$	android/app/ActivityManager;→getRunningTasks	$\uparrow$
android/location/LocationManager;→getBestProvider	$-$	android.permission.ACCESS_COARSE_LOCATION	$\uparrow$
android.permission.READ_CONTACTS	$-$	android/telephony/TelephonyManager;→getDeviceId	$\downarrow$
android/net/ConnectivityManager;→getNetworkInfo	$+$	android.permission.MOUNT_UNMOUNT_FILESYSTEMS	$\uparrow$
getPackageInfo	$-$	android/telephony/TelephonyManager;→getCellLocation	$\uparrow$
android/widget/VideoView;→setVideoURI	$-$	getPackageInfo	$\uparrow$
android.permission.WAKE_LOCK	$-$	android/net/ConnectivityManager;→getActiveNetworkInfo	$\uparrow$
android.permission.SEND_SMS	$+$	android/net/wifi/WifiManager;→isWifiEnabled	$\uparrow$
android/widget/VideoView;→setVideoPath	$-$	sendTextMessage	$\downarrow$
printStackTrace	$-$	android.permission.WRITE_SETTINGS	$\uparrow$

by the applied measurement. We perturb the 800 malware examples selected from Drebin datasets using Mimicry$\times$30 and I-"max" PGDs+GDKDE attacks against ADE-MA. There are 5 base models in ADE-MA, each attack resulting in 10 correlation coefficients wherein the mean value is 0.4 ± 0.16 (0.16 is the standard deviation) under Mimicry attack and 0.18 ± 0.22 under I-"max" PGDs+GDKDE. Moreover, we conduct the same estimation for dADE-MA. The results are 0.56 ± 0.15 under Mimicry attack and 0.39 ± 0.17 under I-"max" PGDs+GDKDE. Most of the observations confirm our statement except for several cases in ADE-MA.

VII. Conclusion and Future Work

We have studied the usefulness of ensemble for both the defender and the attacker in the context of adversarial malware detection. We propose the mixture of attacks and adversarial deep ensemble. The adversarial deep ensemble can defend against a broad range of evasion attacks, while cannot thwart mimicry attacks and mixtures of attacks. Ensemble methods promote the robustness against evasion attacks when base classifiers are robust enough. For the attacker, the ensemble methods notably improve the attack effectiveness.

We hope this paper will inspire more research into the context of adversarial malware detection. Future research problems are plentiful, such as: intriguing properties of different adversarial malware examples, seeking effective attacks, robust feature extraction, malicious functionality estimation, defense validation metrics and further designing robust defenses.

References

[1] Symantec, "Internet security threat report 2019 (istr)," Symantec, Tech. Rep., February 2019.

[2] V. Chebyshev. (2019, March 5) Mobile malware evolution. [Online]. Available: https://securelist.com/

[3] CISCO. (2018) Cisio @ONLINE. [Online]. Available: https://www.cisco.com

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

[5] Y. Fan, S. Hou, and et al., "Gotcha - sly malware!: Scorpion A metagraph2vec based malware detection system," in *Proceedings of KDD'2018*, 2018, pp. 253–262.

[6] 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, 01 2013, pp. 387–402.

[7] 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)*. IEEE, 2018, pp. 76–82.

[8] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Make evasion harder: An intelligent android malware detection system," in *Proceedings of the Twenty-Seventh IJCAI*, 2018, pp. 5279–5283.

[9] 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.

[10] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," in *CoRR*, 2018.

[11] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[12] E. Grefenstette, R. Stanforth, B. O'Donoghue, J. Uesato, G. Swirszcz, and P. Kohli, "Strength in numbers: Trading-off robustness and computation via adversarially-trained ensembles," *arXiv preprint arXiv:1811.09300*, 2018.

[13] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

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

[15] H. Kwon, Y. KIM, K.-W. Park, H. Yoon, and D. Choi, "Advanced ensemble adversarial example on unknown deep neural network classifiers," *IEICE Transactions on Information and Systems*, vol. E101.D, pp. 2485–2500, 10 2018.

[16] F. Tramèr and D. Boneh, "Adversarial training and robustness for multiple perturbations," *arXiv preprint arXiv:1904.13000*, 2019.

[17] A. Araujo, R. Pinot, and et al., "Robust neural networks using randomized adversarial training," *arXiv preprint arXiv:1903.10219*, 2019.

[18] L. Schott, J. Rauber, M. Bethge, and W. Brendel, "Towards the first adversarially robust neural network model on mnist," *arXiv preprint arXiv:1805.09190*, 2018.

[19] 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.

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

[21] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[22] K. Grosse, N. Papernot, and et al., "Adversarial examples for malware detection," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 62–79.

[23] D. Li, Q. Li, Y. Ye, and S. Xu, "Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and application to aics'2019 challenge," in *The AAAI-19 Workshop on Artificial Intelligence for Cyber Security (AICS), 2019*, 2019.

[24] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples (2014)," *arXiv preprint arXiv:1412.6572*.

[25] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android hiv: A study of repackaging malware for evading machine-learning detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 987–1001, 2020.

[26] D. Arp, M. Spreitzenbarth, and et al., "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[27] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 468–471. [Online]. Available: http://doi.acm.org/10.1145/2901739.2903508

[28] (2019, May) Virustotal. [Online]. Available: https://www.virustotal.com

[29] Y. Dong, F. Liao, and et al., "Boosting adversarial attacks with momentum," in *Proceedings of the CVPR*, 2018, pp. 9185–9193.

[30] C. Smutz and A. Stavrou, "Malicious pdf detection using metadata and structural features," in *Proceedings of the 28th annual computer security applications conference*. ACM, 2012, pp. 239–248.

[31] B. Biggio, G. Fumera, and F. Roli, "Multiple classifier systems for robust classifier design in adversarial environments," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 27–41, 2010.

[32] ——, "Multiple classifier systems under attack," in *International Workshop on Multiple Classifier Systems*. Springer, 2010, pp. 74–83.

[33] M. Abbasi and C. Gagné, "Robustness to adversarial examples through an ensemble of specialists," in *ICLR 2017 workshop*, 2017.

[34] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint:1704.01155*, 2017.

[35] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, "Adversarial example defense: Ensembles of weak defenses are not strong," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. Vancouver, BC: USENIX Association, Aug. 2017.

[36] T. Pang, K. Xu, C. Du, N. Chen, and J. Zhu, "Improving adversarial robustness via promoting ensemble diversity," in *International Conference on Machine Learning*, 2019, pp. 4970–4979.

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

[38] J. W. Stokes, D. Wang, M. Marinescu, M. Marino, and B. Bussone, "Attack and defense of dynamic analysis-based, adversarial neural malware classification models," 12 2017.

[39] J. Parikh. (2018, August) Protecting the protector:hardening machine learning defenses against adversarial attacks. [Online]. Available: https://www.blackhat.com/us-18/speakers/Jugal-Parikh.html

[40] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *arXiv preprint*, 2016.

[41] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[42] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers: A case study on pdf malware classifiers," in *NDSS*, January 2016.

[43] 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.

[44] A. Demontis, M. Melis, and et al., "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, N. Heninger and P. Traynor, Eds. USENIX Association, 2019, pp. 321–338.

[45] A. Demontis, M. Melis, and et al., "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 711–724, July 2019.

[46] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," *arXiv preprint arXiv:1911.02142*, 2019.

[47] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE TKDE*, vol. 26, pp. 984–996, 04 2014.

[48] N. Papernot, P. McDaniel, and et al., "The limitations of deep learning in adversarial settings," in *2016 EuroS&P*. IEEE, 2016, pp. 372–387.

[49] J. Jung, C. Jeon, and et al., "AVPASS: Leaking and Bypassing Antivirus Detection Model Automatically," in *Black Hat USA Briefings (Black Hat USA)*, Las Vegas, NV, Jul. 2017.

[50] 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.

[51] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[52] L. Xu, Z. Zhan, S. Xu, and K. Ye, "Cross-layer detection of malicious websites," in *Third ACM Conference on Data and Application Security and Privacy (CODASPY'13)*, 2013, pp. 141–152.

[53] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

[54] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.

[55] D. P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[56] J. Duchi, S. Shalev-Shwartz, and et al., "Efficient projections onto the l 1-ball for learning in high dimensions," in *Proceedings of the 25th ICML*. ACM, 2008, pp. 272–279.

[57] A. Desnos. (2019) Androguard @ONLINE. [Online]. Available: https://github.com/androguard/androguard

[58] 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*, 2010, pp. 3121–3124.

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

[60] (2019, May) Apktool. [Online]. Available: https://ibotpeaches.github.io/Apktool

[61] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

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

[63] B. Kolosnjaji, A. Demontis, and et al., "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European Signal Processing Conference (EUSIPCO)*, Sep. 2018, pp. 533–537.