MalPurifier: Enhancing Android Malware Detection with Adversarial Purification against Evasion Attacks

Yuyang Zhou, *Member, IEEE*, Guang Cheng, *Member, IEEE*, Zongyao Chen, *Student Member, IEEE*, and Shui Yu, *Fellow, IEEE*

Abstract—Machine learning (ML) has gained significant adoption in Android malware detection to address the escalating threats posed by the rapid proliferation of malware attacks. However, recent studies have revealed the inherent vulnerabilities of ML-based detection systems to evasion attacks. While efforts have been made to address this critical issue, many of the existing defensive methods encounter challenges such as lower effectiveness or reduced generalization capabilities. In this paper, we introduce a novel Android malware detection method, MalPurifier, which exploits adversarial purification to eliminate perturbations independently, resulting in attack mitigation in a light and flexible way. Specifically, MalPurifier employs a Denoising AutoEncoder (DAE)-based purification model to preprocess input samples, removing potential perturbations from them and then leading to correct classification. To enhance defense effectiveness, we propose a diversified adversarial perturbation mechanism that strengthens the purification model against different manipulations from various evasion attacks. We also incorporate randomized "protective noises" onto benign samples to prevent excessive purification. Furthermore, we customize a loss function for improving the DAE model, combining reconstruction loss and prediction loss, to enhance feature representation learning, resulting in accurate reconstruction and classification. Experimental results on two Android malware datasets demonstrate that MalPurifier outperforms the state-of-the-art defenses, and it significantly strengthens the vulnerable malware detector against 37 evasion attacks, achieving accuracies over 90.91%. Notably, MalPurifier demonstrates easy scalability to other detectors, offering flexibility and robustness in its implementation.

Index Terms—Android Malware Detection, Machine Learning, Evasion Attacks, Adversarial Purification, Denoising Autoencoder.

1 Introduction

Due to the popularity of the Android operating system, it has become the primary victim of malware attacks. In 2021, Zimperium reported that 2 billion new malware emerged in the wild [1], and Kaspersky detected 1,661,743 mobile malware or unwanted software installers in 2022 [2]. As a result, the prevalence of Android malware has grown exponentially in recent years, posing a significant threat to the security and privacy of mobile users worldwide.

The magnitude of this threat has spurred the use of Machine Learning (ML) techniques, particularly Deep Learning (DL), to automate *Android malware detection*. Empirical evidence has shown that these approaches offer advanced performance in detecting malware (see, e.g., [3], [4], [5], [6], [7]), making them a promising avenue for mitigating this security concern.

Unfortunately, ML-based detectors are vulnerable to adversarial examples, which are created by modifying non-functional instructions in executable programs of existing malware [8], [9], [10]. Adversarial examples can enable a range of attacks, including *evasion attacks* [11], [12], [13], *poisoning attacks* [14], [15], [16], or a combination of both [17]. In this study, we specifically narrow our focus to evasion attacks, which are designed to deceive ML-based detection during the testing phase.

So far, *adversarial training*-based methods have shown great potential to safeguard ML models from evasion attacks [18], [19], [20]. By augmenting the training dataset with generated adversarial samples, adversarial training can increase the robustness of the trained model in future use. However, these methods still have certain disadvantages, including high computational costs [21] and a significant sacrifice in accuracy on clean data [22]. Furthermore, its effectiveness is strongly influenced by the similarity between the adversarial examples employed during the training and testing phases [23]. This may lead to overfitting of the model to specific perturbations, thereby negatively impacting its ability to generalize and detect unseen attacks.

Another defense technique, known as *adversarial purification* [24], [25], [26], aims to remove potential perturbations from input samples, resulting in *purified* samples that can be correctly classified by the target classifier. The purification model is usually trained independently of the classification model and does not necessarily require class labels [27]. As a result, it can mitigate unseen threats in a *plug-and-play* manner without re-training the target classifier [28], leading to less training overhead and more flexible employment. Nevertheless, existing purification solutions for image classification are not easily applicable to Android malware detection due to significant differences. The input space of the image classifier is continuous, while malware features

- *Yuyang Zhou, Guang Cheng, and Zongyao Chen are with the School of Cyber Science and Engineering, Southeast University, Purple Mountain Laboratories, and Jiangsu Province Engineering Research Center of Security for Ubiquitous Network, Nanjing 211189, China. E-mail: {yyzhou, chengguang, solar1s}@seu.edu.cn.*
- *Shui Yu is with the School of Computer Science, University of Technology Sydney, Ultimo, NSW 2007, Australia. E-mail: Shui.Yu@uts.edu.au.*
- *Guang Cheng is the corresponding author.*

are often discrete [29]. Hence, it is more challenging to develop a robust Android malware detector.

Inspired by the complementary strengths of existing approaches in image classification, we propose a customized adversarial purification-based method to defeat evasion attacks in Android malware detection, as illustrated in Fig. 1. This novel mechanism has the following highlighted features that enable it to eliminate perturbations in adversarial malware and restore it closer to its original form: (i) We first generate diversified adversarial malware samples by adding perturbations in a progressively increasing manner to represent varying evasion attacks. This approach enables the purification model to fully acquire the capability to effectively purify these perturbed samples. (ii) We further injected "*protective noises*" into the benign samples during the training of the purification model. This step enhances the distinction between benign and adversarial samples, preventing excessive purification of benign samples. (iii) Based on these generated samples with their original versions, we then train a *Denoising AutoEncoder* (DAE) based purification model with a customized loss function in an *unsupervised* manner, allowing us to accurately pre-process the samples for classification purposes. (iv) More importantly, the proposed method is *non-intrusive*, *computationally efficient*, and *easily scalable* to different detection models. The main contributions of this paper can be summarized as follows:

- **Attack mitigation with adversarial purification.** We propose a robust Android malware detection method, termed *MalPurifier*, that leverages a plug-and-play DAE model to purify adversarial malware. To the best of our knowledge, this is the *first* attempt to use adversarial purification to mitigate evasion attacks in the Android ecosystem.
- **Trade-off between robustness and detection accuracy.** Our proposed mechanism injects a diversified range of perturbations into malware samples, ranging from no injection to worst-case perturbations, to enhance robustness against different types of attacks. Moreover, we also implement safeguards to prevent false positives on benign samples during the purification process.
- **Accurate sample recovery via denoising autoencoder.** We establish a DAE-based purification model for purifying adversarially perturbed samples, which is independent of the label space and the detection model. Especially, we incorporate reconstruction loss and prediction loss to help the model handle complex and noisy data, leading to better feature representation and sample recovery.
- **Experimental validation on public datasets.** We compare MalPurifier with the state-of-the-art (SOTA) methods via *Drebin* [30] and *Androzoo* [31] datasets. Experimental results show that MalPurifier significantly outperforms other defenses against attacks with less overhead required. Finally, we release the source code at https://github.com/SEU-ProactiveSecurity-Group/MalPurifier.

The remainder of this paper is organized as follows. Section 2 reviews some preliminaries and Section 3 presents the problem formulation. Section 4 elaborates the methodology, and the experimental results are presented in Section 5. We explore the limitations of our work and open challenges in Section 6, and discuss related work in Section 7. Finally, conclusion and future work are summarized in Section 8.

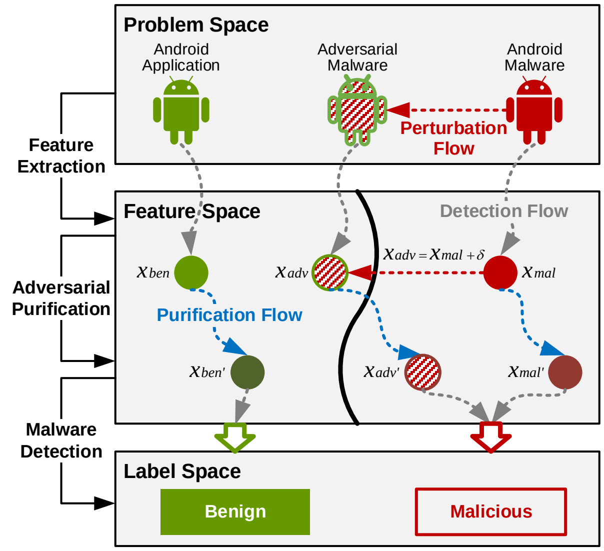

Fig. 1. Illustration of MalPurifier pre-processing samples via adversarial purification, before feeding them into the malware detector. This method projects the various perturbed samples back to their original forms, while reserving the feature representation of clean data, successfully striking a balance between robustness and accuracy without necessitating any changes to the architecture or parameters of the detection model.

2 Preliminaries

2.1 ML-based Malware Detection

The ML-based Android malware detection can be briefly described as follows. Formally, let $\mathcal{Z}$ be the problem space, and $z \in \mathcal{Z}$ be an Android application sample. In the context of machine learning, there will be a feature extraction function $\phi : \mathcal{Z} \to \mathcal{X}$ which maps the problem space into the feature space, where $\mathcal{X} \subset \mathbb{R}^d$ is a d-dimensional discrete space.

The Android malware detection can be usually treated as a binary classification, thus, let $f : \mathcal{Z} \to \mathcal{Y}$ be the malware detector that maps the problem space to the label space $\mathcal{Y} = \{0, 1\}$, where "0" (or "1") means that corresponding example is benign (or malicious), respectively. Additionally, let the malware detector use an ML model $\varphi_\theta : \mathcal{X} \to \mathcal{Y}$, where θ represents the model's parameters. Therefore, we can conclude that $f(\cdot) = \varphi_\theta(\phi(\cdot))$.

Given a sample-label pair (z, y) and ML-based malware detector f, we then have $x = \phi(z)$. We can easily achieve the prediction $f(z)$ and compare it with the ground-truth label y to analyze the accuracy. To improve the detection accuracy, the main task of Android malware detection is to achieve the optimal parameters as follows.

$$\theta^* \in \arg\min_{\theta} \mathbb{E}_{(z,y)\in\mathcal{D}}[\mathcal{L}(\theta, x, y)], \tag{1}$$

where $\mathcal{L}(\theta, x, y)$ is the loss function for the ML model φ_θ, and $\mathcal{D}$ represents the underlying data distribution of training examples.

2.2 Evasion Attacks

2.2.1 Attack Principle

According to Ref. [8], evasion attack can be categories into two types: *problem-space* attacks and *feature-space* attacks. In the problem-space attack, the adversary perturbs a malware sample from z to z' to evade the detector f. Accordingly, they can be mapped into the feature space with $x = \phi(z)$ and $x' = \phi(z')$. Formally, given a feature-label pair (x, y) of a malware sample and an adversarial manipulation δ, the evasion attack can be written as

$$\varphi_\theta(x') = \varphi_\theta(x + \delta) = 0, \quad \text{s.t.}(x' \in \mathcal{X}) \wedge (x' \in [\check{u}, \hat{u}]), \quad (2)$$

where x' is the perturbed feature representation. Recent studies have suggested that it obeys a box constraint [17], such that $x' \in [\check{u}, \hat{u}]$, where $\check{u}$ and $\hat{u}$ denote the lower and upper boundaries in the feature space, respectively.

In addition, to establish the inverse mapping from the feature space to the problem space, we adopt the methodology proposed in Ref. [32]. This approach facilitates the design of attack tactics while maintaining the effectiveness of the attack. By utilizing an approximate inverse function $\tilde{\phi}^{-1}$, we can directly map the perturbation vector δ to the problem space.

2.2.2 Attack Methods

The attack method defines how the attacker implements malicious actions. In this section, we consider four distinct attack methods as follows.

Obfuscation Attacks. This kind of attacks suggests malware authors leveraging obfuscation technology to camouflage malicious functionality [11], [33]. Typically, adversaries exploit certain techniques (e.g., encryption, renaming, etc.) to produce malware variants that can deceive detection. Note that this attack does not require knowledge of the target classifier, making it a zero-query black-box attack that can be directly performed on the problem space.

Gradient-based Attacks. These attacks apply small perturbations in the direction of gradients to produce adversarial malware samples. For example, Projected Gradient Descent (PGD) attack [34] initializes the perturbation with a zero vector and perturbs it via an iterative process, such that

$$\delta^{t+1} = \Pi_{[\check{u}-x, \hat{u}-x]}\Big(\delta^t + \lambda \nabla_\delta \mathcal{L}(\theta, x + \delta^t, y)\Big), \quad (3)$$

where t is the iteration, $\lambda > 0$ is the step size, $\Pi_{[\check{u}-x, \hat{u}-x]}$ is the projection operator that keeps δ^{t+1} within a set of range $[\check{u} - x, \hat{u} - x]$, and ∇_δ indicates the gradient of the loss function $\mathcal{L}$ with respect to δ. Due to the small magnitudes of gradients in practical scenarios, researchers have been motivated to normalize the gradients in a direction of interest, such as the ℓ_1, ℓ_2, or ℓ_∞ norm [35].

Furthermore, this study incorporates several other algorithms to perform gradient-based attacks, including Bit Coordinate Ascent (BCA) [36], Fast Gradient Sign Method (FGSM) [19], and Grosse [37].

Gradient-free Attacks. These attacks are permitted to get access to a surrogate dataset and wage evasion attacks via perturbations. The salt and pepper noises attack [13] involves manipulating malware samples by randomly replacing feature values with either the maximum or minimum intensity values, resembling the spread of salt and pepper particles. This study also investigates the use of pointwise attacks [38], in which the adversary first adds noise perturbation and then modifies features to generate an adversarial sample with the least perturbation.

Ensemble Attacks. These attacks provide attackers with the capability to compromise the victim via a combination of multiple attack methods and manipulations. For instance, Li et al. [20] proposed a series of ensemble-based attacks, including the "Max" strategy enabled Mixture of Attacks (MaxMA), iterative MaxMA (iMaxMA), and Stepwise Mixture of Attacks (StepwiseMA), which effectively enhance the attack performance. Additionally, Croce and Hein [39] combined powerful attacks to create an ensemble attack namely AutoAttack, which demonstrates strong generalization across different models.

2.3 Adversarial Purification

The fundamental concept behind adversarial purification is to preprocess the input data directly, preventing any embedded adversarial components from feeding into the target model, so that the influence of attacks can be mitigated. These methods are widely regarded as model-agnostic and highly efficient, making them easy to train and utilize while demonstrating strong generalization capabilities.

Let g be the adversarial purifier that uses a generative model ψ_ϑ with $g(\cdot) = \psi_\vartheta(\phi(\cdot))$ to learn the data distribution closer to the training distribution and restore an adversarial example to its corresponding clean example, where ϑ represents its parameters. Given $x = \phi(z)$ and $x' = \phi(z')$, thus, the training objective of purification is then

$$\begin{aligned} \vartheta^* \in \arg\min_{\vartheta} \mathbb{E}_{(z,y)\in\mathcal{D}}[\mathcal{J}(\vartheta, x', x)], \\ \text{s.t.}(x' \in \mathcal{X}) \wedge (x' \in [\check{u}, \hat{u}]) \wedge (\psi_\vartheta(x') \in \mathcal{X}), \end{aligned} \quad (4)$$

where $\mathcal{J}(\vartheta, x', x)$ represents the loss function for the learning model ψ_ϑ, x denotes the original feature, and $x' = x + \delta$ is the perturbed feature representation. This training procedure only focuses on the differences of the representation between the sample after purification and its original version, thus, we can clearly conclude that the purification model is trained independently of the class label.

Unlike adversarial example attacks in the image domain that perturb images with inconspicuous noises, adversaries in this field specifically employ discrete manipulations on malware samples to evade detection. These adversarial examples closely resemble benign data in their feature representation, posing a significant challenge for the purification model. For example, clean data might be mistaken for adversarial examples, resulting in incorrect restoration into samples with malicious feature representation. As a consequence, the accuracy of clean data may experience a substantial decrease. Thereby, it remains a question of how to effectively enhance the trade-off between robustness and accuracy of adversarial purification.

3 Problem Formulation

Here we first introduce the threats considered in our work, and then propose a defense formulation to guide the design of the robust Android malware detection method.

3.1 Threat Model

To perform analysis on potential attacks and develop countermeasures, we consider the threat model in terms of explicit assumptions regarding the attacker's capabilities and knowledge of the target system. Especially, we discuss prevailing attack scenarios specified by three different threat models in this section.

3.1.1 Black-Box Attacks

The attacker has no knowledge about the malware detector f or the purifier g, but can only obtain the prediction results from the target system. Given a malware example z, we can have $x = \phi(z)$, and the attacker aims to modify it for producing wrong prediction such that

$$\varphi_\theta(\psi_\vartheta(x')) = 0, \quad \text{s.t.}(x' \in \mathcal{X}) \wedge (x' \in [\breve{u}, \hat{u}]), \tag{5}$$

3.1.2 Grey-Box Attacks

The adversary has knowledge of the original malware classifier f but is not aware of the purifier g (i.e., oblivious attack). Hence, the adversary's goal is only to fool the unsecured model f using adversarial examples generated from malicious examples. Formally, given a malware example z with its feature representation $x = \phi(z)$ and label $y = 1$, the attacker will modify it to obtain the adversarial feature representation x' that can evade detection, by solving

$$\max_{x' \in [\breve{u}, \hat{u}]} \mathcal{L}(\theta, x', 1), \quad \text{s.t.} x' \in \mathcal{X}, \tag{6}$$

where we substitute $\varphi_\theta(x') = 0$ with maximizing $\mathcal{L}(\theta, x', 1)$ owing to the non-differentiability of $\phi(\cdot)$.

3.1.3 White-Box Attacks

In this attack scenario, the adversary has full knowledge of the model f and is aware of the structure of the purifier g (i.e., adaptive attack). In other words, the adversary needs to mislead both the malware detector and adversarial purifier simultaneously by perturbing x into x'. It is worth noting that the sample will be first feed into the purifier and its output will be then classified by the malware detector.

Thereby, given a malware feature-label pair (x, y) in this study, the attacker needs to solve

$$\max_{x' \in [\breve{u}, \hat{u}]} \mathcal{L}(\theta, \psi_\vartheta(x'), 1), \quad \text{s.t.}(x' \in \mathcal{X}) \wedge (\psi_\vartheta(x') \in \mathcal{X}), \tag{7}$$

where $\psi_\vartheta(x')$ denotes the purified sample obtained by applying the adversarial purifier g to the input x'.

3.2 Defense Formulation

As aforementioned, MalPurifier is rooted in adversarial purification, aiming to eliminate potential adversarial manipulations before detection. Thereby, we propose incorporating the detector f with the adversarial purifier $g(\cdot) = \psi_\vartheta(\phi(\cdot))$. To develop the framework of MalPurifier, we need to train the detection model φ_θ according to Eq. (1) and build the purification model ψ_ϑ based on Eq. (4), respectively. To this end, given a feature-label pair (z, y) with possible adversarial perturbations δ in the feature space, the desired parameters θ^* and ϑ^* of MalPurifier can be derived by solving the following problem

$$\begin{gathered}\varphi_{\theta^*}(\psi_{\vartheta^*}(x+\delta)) = y, \\ \text{s.t.}(x \in \mathcal{X}) \wedge (y \in \mathcal{Y}) \wedge (x + \delta \in [\breve{u}, \hat{u}]),\end{gathered} \tag{8}$$

when $\delta = 0$, it indicates that no adversarial manipulations have been applied to this sample, rendering it clean. The above formulation points out two tasks as follows.

- **High-accuracy Android malware detection.** Developing a malware detection model that can classify clean samples into benign or malicious with high accuracy, that is, $\varphi_\theta(x) = y$, in which x can be the feature representation of a normal Android application or a malware sample, and thus $y = 0$ or 1. This model can be easily obtained by utilizing some ML algorithms (i.e., Deep Neural Network (DNN)), which have shown promising results with 99% accuracy in their laboratory settings [6], [12], [40].
- **Effective adversarial manipulation elimination.** This formulation highlights the importance of effectively integrating adversarial purification with the pre-trained detector, as summarized into two aspects. (i) Given the feature representation of a malware sample x with its adversarial version x', it is crucial to minimize the impact of evasion attacks by achieving $\psi_\vartheta(x') \approx x$. This enables us to accurately identify it through the malware detector, indicated by $\varphi_\theta(\psi_\vartheta(x')) = 1$. (ii) When dealing with a clean feature-label pair (x, y), it is essential to preserve the accuracy on it. This entails ensuring that the prediction results of clean data remain unaffected, represented by $\varphi_\theta(\psi_\vartheta(x)) = y$.

4 The MalPurifier Approach

In this section, we provide an in-depth exploration of the MalPurifier approach. We begin by offering an overview that encompasses the processes of model training and inference. Subsequently, we delve into the mechanisms of diversified adversarial perturbation and protective noise injection, which are employed to generate training data for the purification model. Finally, we elaborate on the establishment of a label-independent DAE-based purification model, which plays a crucial role in producing purified samples and thereby contributing to accurate predictions.

4.1 Defense Architecture

The MalPurifier plays two important roles, i.e., adversarial purification and malware detection. First, it performs input preprocessing in a label-independent manner and generates reconstructed representations in new forms. Especially, adversarial samples are restored to their original versions, while the characteristics of clean data remain unchanged. Second, it determines the benignity of the inputs by analyzing the prediction results of the DNN model on the processed samples. Fig. 2 depicts an overview of the MalPurifier architecture and explains how the functionally different components are jointly optimized.

As we can see, the proposed method consists of three main modules: (i) a feature extractor $\phi(\cdot)$ that maps an Android application into a feature vector, (ii) an adversarial purifier g that processes samples via a DAE model ψ_ϑ, and (iii) a malware detector f that uses a DNN model φ_θ for detection. In the training phase, we initiate the process by extracting features from multiple batches of clean (or natural) examples, which include both benign and malicious examples without manipulation. These features are then fed into the DNN model, which iteratively updates its parameters to minimize the loss function. Once the DNN model is effectively trained, it exhibits exceptional classification accuracy when presented with clean inputs. At the same time, as a detection model, it often requires a substantial amount of training samples. To reduce costs, we will not retrain this model in this study, in contrast to adversarial training methods.

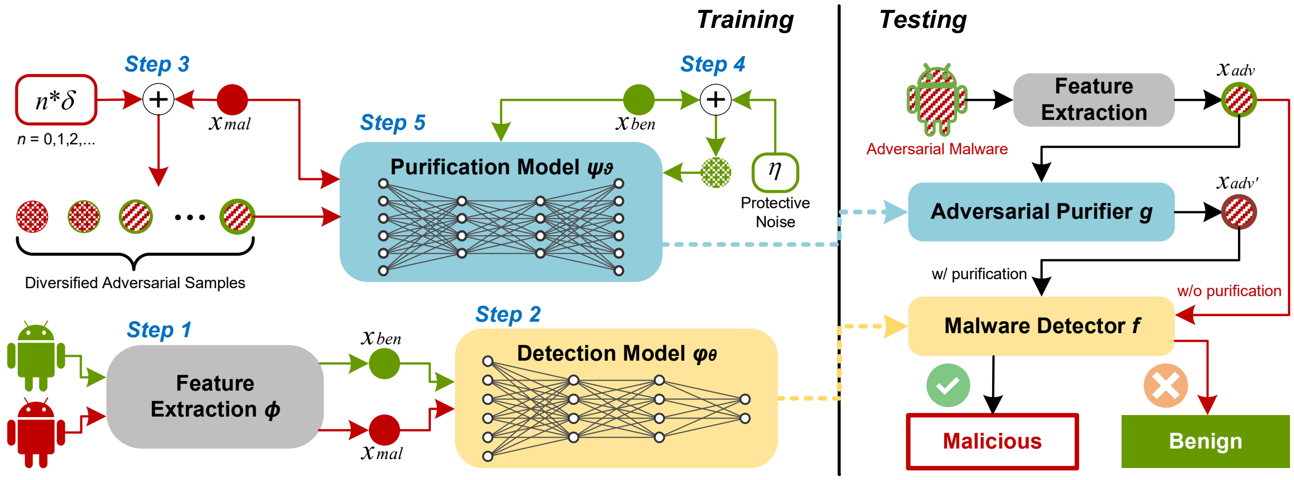

Fig. 2. Overview of MalPurifier architecture. In the training phase, feature vectors are extracted from Android apps in Step 1. Then, a detection model is constructed using features from both benign and malicious apps in Step 2. In Step 3, diversified adversarial perturbations are applied to malware samples in the feature space, while projective noises are introduced into benign samples in Step 4. Finally, in Step 5, the purification model is built using these variant samples along with their corresponding original versions. In the testing phase, a sample undergoes sequential processing by the purifier and detector, ensuring that adversarial malware cannot escape detection.

Different from the DNN, the DAE learns compressed representations of input data and reconstructs the original (clean) data from noisy versions. The performance of a DAE model is significantly influenced by the training data, especially the level and type of noise introduced during training. To enhance the DAE's ability to handle various adversarial perturbations in input data, we propose a diversified adversarial perturbation mechanism, as demonstrated in Section 4.2. This mechanism generates a range of adversarial malware samples with varying attack strengths, thereby improving the model's effectiveness in mitigating evasion attacks. Furthermore, in this study, we introduce a noise injection mechanism (see Section 4.3) to safeguard benign samples from excessive or erroneous purification by the DAE, hence preserving their original feature patterns. Finally, we incorporate reconstruction loss and prediction loss into the DAE model, and feed both generated adversarial samples and noisy benign samples, along with their corresponding clean forms into it. Note that we train the DAE model independent of the label space. More details of the DAE training can be found in Section 4.4.

In the phase of testing, the MalPurifier can be fed with benign, malicious and adversarial examples. By inputting an Android App into our method, one can obtain the prediction result (benign or malicious) as it passes through these modules. As depicted in the right part of Fig. 2, the MalPurifier demonstrates accurate identification of adversarial samples that may have evaded detection without purification.

4.2 Diversified Adversarial Perturbation

To train an adversarial purification model, we need to create perturbed malware samples first. Traditional adversarial training methods usually exploit strong white-box attacks to consider the target parameters θ and perturb the malware sample to create x', such that it will be misclassified by the detection model, i.e., $\varphi_\theta(x') = 0$. Since this method calculates perturbations using gradient directions specific to θ and stops manipulations when the classifier achieves misclassification, these perturbed malware samples cannot generalize well across different attack intensities.

To solve the aforementioned challenges, we design a diversified adversarial perturbation mechanism that can generalize across multiple kinds of attacks, thus enabling a robust defense approach. Specifically, this mechanism aims at increasing the feature space difference between a malware sample x and its perturbed version x' instead of specific label information, which can be represented as

$$\Delta(x, x') = d(\mathcal{F}_\theta(x)|_n, \mathcal{F}_\theta(x')|_n), \quad \text{s.t.} x' \in [\check{u}, \hat{u}], \tag{9}$$

where $\mathcal{F}_\theta(x)|_n$ denotes the internal representation of input data x from the nth layer of the ML-based malware detector f, and $d(\cdot)$ quantifies the difference between the features of the original and perturbed samples. In this study, we employ *Mean Square Error* (MSE) as the distance metric. Furthermore, instead of directly maximizing this gap under constrained conditions, we add perturbations in a gradually increasing manner during the processing of input samples. This approach allows the generated samples to encompass a wider range of attack intensities, ensuring diversified and highly generalizable adversarial examples.

Algorithm 1 demonstrates the generation of diversified adversarial examples as the following steps. (i) We first sample a malware batch $(x_i, y_i = 1)_{i=1}^N$ in Line 2. (ii) Then, Line 3 sets the adversarial depth to a linear relationship with the batch index. (iii) In line 5, we do not inject any perturbations for the first batch of malware samples. (iv) For the other batch, we randomly transform x_i as the initial point in Line 7. We derive the gradients g_t in Line 10 according to the difference calculated in Line 9, thereby creating adversarial samples according to the preset depth in Line 11. To keep the generated adversarial samples practical, we also project them into the binary space in Line 12. After T rounds of iterations, we craft adversarial malware samples for this batch in Line 13. (v) We repeat the above steps on each batch of the dataset to obtain enough diversified adversarial examples as candidates for adversarial purification.

Algorithm 1: Diversified Adversarial Perturbation

Input: Training dataset $(\mathcal{X}, \mathcal{Y})$, number of batches N, number of iterations T, step size s, and random transformation function $\mathcal{R}$;
Output: Generated adversarial subset $\mathcal{X}_{adv}$;
for $i = 1$ *to* N **do**
  Sample a batch of $(x_i, y_i = 1)$ from $(\mathcal{X}, \mathcal{Y})$;
  Adversarial depth $k_i = s * (i - 1)$;
  if $k_i = 0$ **then**
    $x'_i \leftarrow x_i$; ▷ First batch without perturbation
  else
    $x'_{i,1} = \mathcal{R}(x_i)$; ▷ Create a random initial point
    for $t = 1$ *to* T **do**
      Compute Δ between x_i and $x'_{i,t}$ via Eq.(9);
      Compute gradients $g_t = \nabla_{x_i} \Delta(x_i, x'_{i,t})$;
      Generate adversarial samples by $x'_{i,t+1} = x'_{i,t} + k_i * g_t$;
      Project $x'_{i,t+1}$ into the binary space;
    $x'_i \leftarrow x'_{i,T}$; ▷ Other batches with perturbation
Return $\mathcal{X}_{adv} = \{x'_1, x'_2, ..., x'_N\}$

4.3 Protective Noise Injection

The proposed diversified adversarial perturbation mechanism successfully enhances the diversity of adversarial malware samples within the training dataset, however, if benign samples are left unprocessed, the purification model may have a tendency to misclassify them as malicious samples, resulting in erroneous classification outcomes. It is also important to note that adversaries typically do not alter benign samples to make them appear as malware. As a result, the direct application of adversarial perturbation methods to manipulate benign samples is not viable.

To solve this problem and protect clean samples from excessive purification, we here propose a projective noise injection mechanism that preprocesses clean data before the training phase of the purification model. Specifically, we introduce a threshold $\eta \in [0, 1]$ to control the range of noise injection. When η is set to 0, no noise is added, while a value of 1 indicates that all feature values will be flipped under this condition. By employing this approach, the purification model is encouraged to better discriminate between benign and malicious samples within the feature space, thereby ensuring the detection performance of subsequent classifiers on clean data.

The detailed algorithm of the protective noise injection is described in Algorithm 2. (i) For each batch of training dataset, we first select benign samples $(x_i, y_i = 0)$ in Line 2, where $i \in [1, N]$. (ii) Then, we calculate the batch size b_i and length l_i of this batch of benign data for the following noise injection. (iii) We randomly generate a mask and determine the injection position according to the preset noise level η. (iv) We modify binary feature values on these positions (flipping from "0" to "1" and from "1" to "0") based on these generated masks. (v) Finally, by applying the aforementioned steps to all batches of the training dataset, we obtain a subset of benign samples that have been augmented with protective noise.

Algorithm 2: Protective Noise Injection

Input: Training dataset $(\mathcal{X}, \mathcal{Y})$, number of batches N, number of iterations T, and noise level η;
Output: Processed benign subset $\mathcal{X}_{ben}$;
1 **for** $i = 1$ *to* N **do**
2   Sample a batch of $(x_i, y_i = 0)$ from $(\mathcal{X}, \mathcal{Y})$;
3   Obtain its batch size b_i and length l_i;
4   Generate random mask $m = rand(b_i, l_i) < \eta$;
5   Flip feature values via $x_i[m] = 1 - x_i[m]$;
6 Return $\mathcal{X}_{ben} = \{x_1, x_2, ..., x_N\}$

4.4 Accurate Sample Recovery

In this section, we introduce the technical details of the proposed purification mechanism and explain how to effectively recover perturbed adversarial malware samples while not affecting the detection of clean data through this mechanism. Our purification model is built on a denoising autoencoder, which has excellent capability to remove noise and improve the quality of the data for downstream tasks.

As illustrated in Fig. 2, the input of the purification model ψ_ϑ is the perturbed sample generated from diversified adversarial perturbation in Algorithm 1 (see Step 3), the modified benign samples created from protective noise injection in Algorithm 2 (see Step 4), as well as their original forms. During the training process of the purification model, these data are passed through and the network parameters ϑ can be optimized to minimize the customized loss that is described below.

Reconstruction Loss. A commonly used loss function for DAEs is the MSE loss, which measures the average squared difference between two groups of data. To help train a more robust purification model, we aim to minimize the discrepancy of the feature representation between the reconstructed output and the target data (i.e., reconstruction loss) as follows.

$$\mathcal{L}_{rec} = d(x, \psi_\vartheta(x')), \quad \text{s.t.} x' \in \mathcal{X}_{adv} \cup \mathcal{X}_{ben}, \tag{10}$$

where x and $\psi_\vartheta(x')$ denote the original data and the purified data, respectively.

Prediction Loss. Given that the ultimate objective of adversarial purification is to enhance the classification performance of the subsequent malware detector, we propose using prediction loss to further optimize the performance of the purification model as follows.

$$\mathcal{L}_{pre} = \Delta(x, \psi_\vartheta(x')) = d(\mathcal{F}_\theta(x)|_n, \mathcal{F}_\theta(\psi_\vartheta(x'))|_n), \tag{11}$$

where Δ is formally defined in Eq. (9), and the distance measure used to compute Δ is MSE as well. In this way, the purification model will be strengthened in which the purified data will be closer to the original version in the internal representation of the prediction model (i.e., the malware detector), leading to more accurate prediction results.

Therefore, the overall loss function for ψ_ϑ is the combination of reconstruction loss and prediction loss as follows.

$$\mathcal{L}_{\psi_\vartheta} = \alpha \mathcal{L}_{rec} + \beta \mathcal{L}_{pre}, \quad \text{s.t.} \alpha, \beta \in [0, 1], \tag{12}$$

where α and β are the weights of two loss terms, and we have $\alpha + \beta = 1$. The parameters ϑ will be optimized by minimizing the customized loss of the purification model.

5 Experiments and Evaluation

In this section, we conduct extensive experiments to validate the soundness of MalPurifier by answering the following Research Questions (RQs):

- **RQ1: Effectiveness and cost without attacks.** How is the effectiveness and overhead of MalPurifier when there is no attack?
- **RQ2: Robustness against obfuscation attacks.** How is the robustness of MalPurifier against obfuscation attacks?
- **RQ3: Robustness against oblivious attacks.** How robust is MalPurifier against oblivious attacks where the attacker is unaware of the additional defense mechanism (e.g., the adversarial purifier g)?
- **RQ4: Robustness against adaptive attacks.** How robust is MalPurifier against adaptive attacks in which the adversary has full knowledge of all defense mechanisms?
- **RQ5: Enhancement brought to other detectors.** Can the purification model act as a plug to enhance other detectors against evasion attacks?

Datasets. Our experiments utilize two popular Android malware datasets: *Drebin* [30] and *Androzoo* [31]. The Drebin dataset[1] consists of 5,560 malicious samples and SHA256 values of 123,453 benign applications, which were collected before 2013. For evaluation purposes, we downloaded 47,770 benign APKs from various markets (e.g., Google Play Store, AppChina, Anzhi). To obtain more recent files, we collected 170,851 APKs from the AndroZoo dataset[2], specifically those attached with dates falling between January 1st and December 31st, 2021. We submitted these APKs to the *VirusTotal*[3] service, labeling a sample as malicious if at least five anti-virus scanners raised alarms, and considering it benign if no scanner detected it. We randomly selected 10,987 benign examples and 10,998 malicious examples from Androzoo for our experiments. Note that each dataset was randomly split into three distinct sets for training (60%), validation (20%), and testing (20%).

Feature extraction. Drebin [30] analyzes a set of APKs and constructs a suitable feature space. Thus, we here utilize the *Androguard*[4] tool to perform a static analysis and extract the Drebin features, which can be organized in 8 different feature sets, including 4 subsets extracted from the `manifest` (e.g., hardware components), and the other 4 subsets extracted from the disassembled `dexcode` (e.g., API calls). The APK is mapped into the feature space as a binary feature vector, in which we can have 0 or 1 along each dimension, indicating the presence or absence of the corresponding feature. Following prior work [20], we exclude certain features that can be easily renamed or modified (e.g., package name) and retain the most frequent 10,000 ones in this study.

Defenses considered for comparative analysis. In this paper, we compare the SOTA defense mechanisms as follows:

- **DNN** [41]. It employs a DNN model for malware detection without any countermeasures against evasion attacks.
- **DNN$^+$** [42]. It enhances the robustness of the detector by another detector trained with an additional outlier class for detecting adversarial examples.
- **KDE** [43]. It introduces a secondary detector that utilizes a *Kernel Density Estimate* (KDE) method. This detector identifies adversarial examples in the final layer of the DNN that deviate significantly from normal data.
- **FD-VAE** [23]. It improves the DNN model by introducing an additional *Variational AutoEncoder* (VAE) for *Feature Disentangle* (FD) in different classes and combining their detection outcomes to make the final decision (FD-VAE).
- **AT-rFGSM^k** [36]. It strengthens the detector by *Adversarial Training* (AT) with randomized rounding projection enabled *FGSM^k* attack (AT-rFGSM^k).
- **AT-Adam** [44]. It enhances the robustness of the DNN model via incorporating *Adversarial Training* with the PGD attack optimized by Adam (AT-Adam).
- **PAD-SMA** [20]. It achieves *Principled Adversarial Detection* by a DNN-based malware detector and an Input Convexity Neural Network (ICNN) based adversary detector, both of which are strengthened by adversarial training incorporating the *Stepwise Mixture of Attacks* (PAD-SMA).

All above defenses consider DNN as the baseline classifier, they either improve the malware detector via adversarial training, or introduce another detector to identify adversarial examples, or both of them. Unlike these methods, MalPurifier takes a different approach. Firstly, it avoids the use of adversarial training methods to retrain the malicious detector, opting instead to fix the detector to minimize costs. Secondly, to guide the detector towards accurate classification, MalPurifier focuses on removing potential perturbations from the samples rather than attempting to detect adversarial samples as a new class.

Metrics. The effectiveness of defenses is assessed using five standard metrics: False Positive Rate (FPR), False-Negative Rate (FNR), Accuracy (Acc), balanced Accuracy (bAcc), and F1 score. The inclusion of balanced Accuracy and F1 score accounts for the imbalanced nature of the dataset. In addition, we include training time to evaluate the overhead of these methods.

5.1 RQ1: Effectiveness and Cost without Attacks

Experimental Setup. We compare MalPurifier with the aforementioned approaches on the two datasets. We use a DNN model with 2 fully-connected hidden layers (each having 200 neurons) with the ELU activation, and the other methods also use this architecture for malware detection.

In detail, DNN$^+$ [42] leverages another detector hardened by adversarial training with the MaxMA attack against the DNN model to identify adversarial examples, and KDE [43] relies on the close distance between activations to reject large manipulations without retraining. FD-VAE [23] incorporates a VAE-based indicator to classify clean data and adversarial examples, in which both the encoder and decoder consist of two layers (each layer having 600 neurons) with the Softplus activation. Moreover, AT-rFGSM^k [36] uses the PGD-ℓ_∞ attack, which has 50 iterations with step size 0.02, and AT-Adam [44] exploits the Adam optimizer with iterations 50, step size 0.02, and random starting point. PAD-SMA [10] uses three attacks,

1. https://www.sec.cs.tu-bs.de/ danarp/drebin
2. https://androzoo.uni.lu
3. https://www.virustotal.com/gui/home/upload
4. https://github.com/androguard/androguard

TABLE 1
Effectiveness and overhead of detectors in the absence of attacks.

	Defense	Effectiveness (%)					Overhead (s)
		FPR	FNR	Acc	bAcc	F1	Training time
Drebin	DNN	0.51	8.07	98.72	95.71	93.58	592
	DNN^+	0.54	7.79	98.72	95.83	93.60	1559
	KDE	0.53	8.10	98.67	95.68	93.53	592
	FD-VAE	1.14	23.3	96.62	87.79	82.12	1021
	AT-rFGSM^k	2.35	5.47	97.33	96.09	87.77	616
	AT-Adam	4.01	5.47	95.84	95.26	82.14	1341
	PAD-SMA	1.70	5.94	97.87	96.18	89.93	21627
	MalPurifier	2.55	7.05	97.00	95.20	86.23	750
Androzoo	DNN	0.32	1.22	99.23	99.23	99.23	527
	DNN^+	0.05	0.83	99.54	99.56	99.56	1148
	KDE	0.15	1.22	99.31	99.32	99.32	527
	FD-VAE	11.26	4.37	92.22	92.19	92.55	2182
	AT-rFGSM^k	1.42	0.72	98.93	98.93	98.95	582
	AT-Adam	3.54	0.59	97.95	97.94	98.00	1018
	PAD-SMA	0.90	1.62	98.73	98.74	98.77	29837
	MalPurifier	2.30	0.59	98.57	98.56	98.59	696

including PGD-ℓ_1 attack iterates 50 times, PGD-ℓ_2 attack iterates 50 times with step size 0.5, and PGD-ℓ_∞ iterates 50 times with step size 0.02.

The proposed method exploits a DAE-based purification model, which has two layers (each having 600 neurons) for both the encoder and decoder with the Sigmoid activation and introduces attention weights following the encoder. The training data of the DAE model is generated by Algorithm 1 with step size 0.01 and Algorithm 2 with noise level 0.001. We conduct a group of preliminary experiments and finally set $\alpha = \beta = 0.5$ on both datasets. In addition, all detectors are tuned by the Adam optimizer with 100 epochs, batch size 128, and learning rate 0.001.

Results. Table 1 exhibits the effectiveness and overhead of all detectors when there is no attack. We observe that DNN^+ achieves the highest detection accuracy (98.72% on Drebin and 99.23% on Androzoo) and F1 score (93.58% on Drebin and 99.56% on Androzoo), which are a little higher than those of the basic DNN model. The reason may be that adversarial training introduces extra adversarial examples that help DNN^+ identify more malicious samples, resulting in a lower FNR and higher F1 score.

An interesting observation is that KDE takes the same time as DNN whereas the other methods have higher overhead in the training phase. The underlying reason may be that the KDE method builds another KDE-based detector without retraining, while the others train a separative model to detect adversarial examples upon the DNN model. We further observe that MalPurifier's FNR decreases but FPR increases, leading to decreased accuracy (1.72% on Drebin and 0.66% on Androzoo) on clean data. which is similar to that of adversarial training methods (e.g., AT-rFGSM^k, AT-Adam, and PAD-SMA). This is because our focus primarily lies on "purifying" adversarial examples into original forms that can be detected by the following DNN-based detector, although we do process benign samples as well.

Answer to RQ1: MalPurifier exhibits a slight decrease in accuracy on clean data with a slightly increased overhead. In comparison, FD-VAE experiences a significant decrease in accuracy whereas PAD-SMA has an excessively long training time on both datasets.

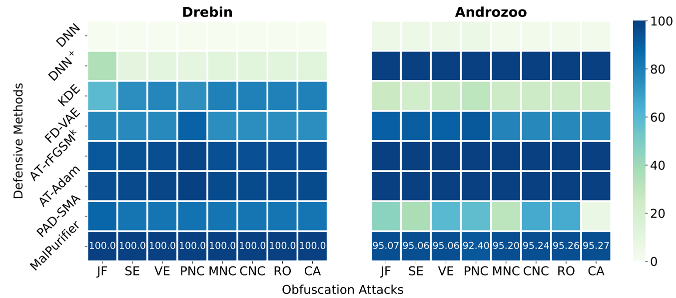

Fig. 3. The accuracy of different detectors against obfuscation attacks on Drebin and Androzoo datasets. The color gradient ranging from light to dark represents the increasing accuracy from low to high, with the effectiveness of MalPurifier against each attack annotated in the square.

5.2 RQ2: Robustness against Obfuscation Attacks

Experimental Setup. We measure the accuracy of all aforementioned methods under obfuscation attacks via the Drebin and Androzoo datasets. Since these attacks belong to black-box attacks and can be conducted in the problem space, therefore, we leverage obfuscation technology to modify malicious APKs and extract features from the modified versions. Specifically, we utilize an obfuscator called AVPASS [33], to wage 8 obfuscation-based attacks to perturb malware examples on the test set.

For Java Reflection (JF), this attack can hide public and static system APIs invoked in Smali using the reflection API. The encryption attacks typically encrypt the const-string and variable names in the decode, i.e., String Encryption (SE) and Variable Encryption (VE). The Package Name Change (PNC), Method Name Change (MNC), and Class Name Change (CNC) attacks change the names of packages, methods, and classes by replacing them with random characters, respectively. For the Resource Obfuscation (RO) attack, it changes pixel or adds one byte to the image files of APKs, along with the modification of related `AndroidManifest.xml`. Finally, we combine the above techniques to produce a Combined Attack (CA).

Results. Fig. 3 illustrates the accuracy of the detectors on Drebin (left panel) and Androzoo (right panel) datasets under 8 obfuscation attacks. We make the first observation that DNN can not defeat all these attacks (accuracy $\leqslant 0.344\%$ on Drebin and $\leqslant 5.871\%$ on Androzoo), demonstrating that such attacks can hide malicious features to evade detection. Nevertheless, obfuscation attackers produce adversarial examples in a black-box manner, they cannot effectively evade these detectors except for the DNN model.

We further observe significant differences between the robustness of some detectors against these attacks on the two datasets. For example, DNN^+ shows poor performance on Drebin whereas achieves high robustness on Androzoo. This can be attributed to the fact that the sample structures used in the two datasets are significantly different (Drebin collected in 2013 whereas Androzoo collected in 2021) and the data imbalance may lead to this phenomenon as well. Another interesting observation is that some adversarial training methods (e.g., AT-rFGSM^k, AT-Adam) show high robustness on obfuscation attacks, which may be attributed to the similarity between adversarial examples generated by PGD attacks and obfuscation technology.

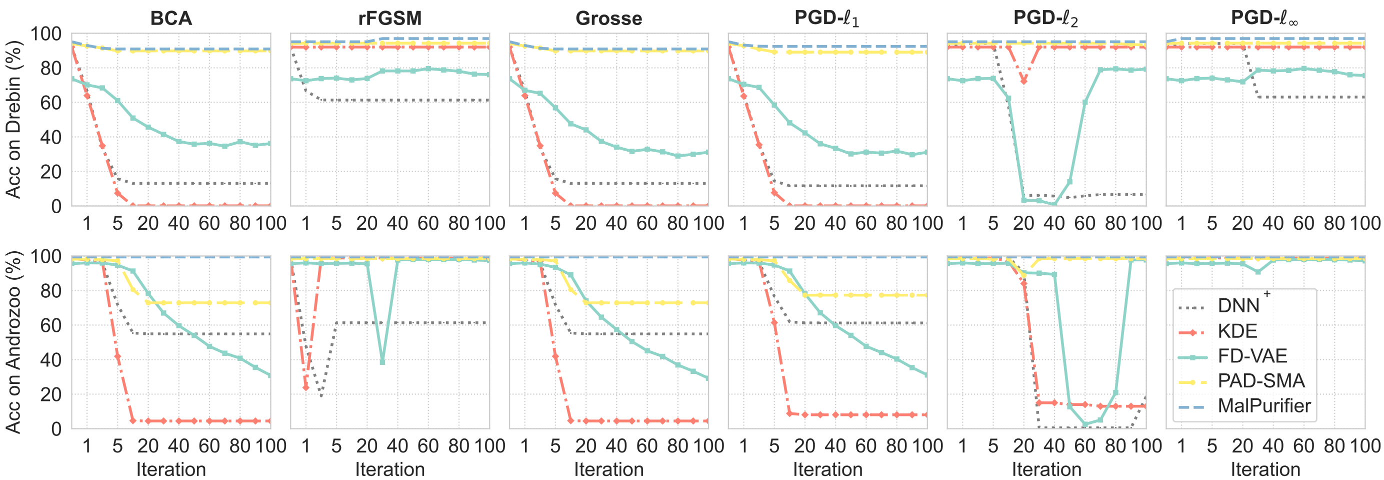

Fig. 4. The accuracy of different detectors against gradient-based oblivious attacks on Drebin (top panel) and Androzoo (bottom panel) datasets, along with the iteration ranging from 0 to 100.

Answer to RQ2: MalPurifier outperforms the other methods against all obfuscation attacks on the Drebin dataset (accuracy of 100%), and achieves accuracy $\geqslant 95\%$ against 7 obfuscation attacks on the Androzoo dataset.

5.3 RQ3: Robustness against Oblivious Attacks

Experimental Setup. Since the DNN, AT-rFGSM^k, and AT-Adam approaches do not have additional detectors or indicators, we only consider the robustness of DNN$^+$, KDE, FD-VAE, PAD-SMA, and MalPurifier against oblivious attacks.

We separately wage 12 oblivious attacks on test malware examples. For BCA [36], Grosse [37], and PGD-ℓ_1 [35] attacks, we perturb one feature per time with a maximum iteration of 100. For rFGSM [19] and PGD-ℓ_∞ [35], we iterate these attack algorithms with the iterations 100 and step size 0.02. The PGD-ℓ_2 [35] attack is set with 100 iterations and step size 0.5. Moreover, we perform Salt & Pepper attack [13] by increasing the noise intensity of 0.001 each time until misclassification and repeating this process 10 times, and the Pointwise [38] attack utilizes Salt & Pepper as the initial attack and minimizes the needed perturbations.

Additionally, we combine PGD-ℓ_1, PGD-ℓ_2, and PGD-ℓ_∞ attacks to perform MaxMA [20] attack and run it 5 times with the random starting point for the iMaxMA [20] attack. We also iterate the SMA [20] attack 100 times with the step size 0.5 for PGD-ℓ_2 and 0.02 for PGD-ℓ_∞, and the AutoAttack [39] consists of APGD-CE and FAB with ℓ_2 norm in our experiments.

Results. Fig. 4 depicts the accuracy curves of these methods on Drebin (top panel) and Androzoo (bottom panel) datasets under 6 gradient-based oblivious attacks with the iteration from 0 to 100. We first observe an important observation that none of these attacks can evade MalPurifier (accuracy $\leqslant 90.91\%$ on Drebin and $\leqslant 99.41\%$ on Androzoo), demonstrating the high robustness of the proposed approach.

We further observe that there is a decreasing trend on the curves of DNN$^+$, KDE, and PAD-SMA against BCA, Grosse, and PGD-ℓ_1 attacks until 20 iterations. This is because such attacks in an oblivious manner will stop manipulations when the adversarial example can evade the malware detector. Moreover, there exists a dip in some accuracy curves of DNN$^+$, KDE, and FD-VAE against rFGSM and PGD-ℓ_2 attacks with the iteration ranging from 0 to 100. This is because such attacks create small perturbations that can successfully escape on a specific iteration, and as the number of iterations increases, they will be recognized due to the larger perturbations.

An interesting observation is that KDE can mitigate PGD-ℓ_∞ on both datasets whereas fails to defeat the other attacks. The underlying reason for this observation may be that KDE relies on the close distance between activations to reject large manipulations that are used by the PGD-ℓ_∞ attack, while the basic DNN model is very sensitive to small perturbations, leading to the failure against the other attacks. Another interesting observation is that the robust accuracies of FD-VAE against all these gradient-based attacks are very different on the two datasets. The reason may be that the threshold of reconstruction error in FD-VAE relies on the distribution of training datasets, which may be significantly different from each other.

TABLE 2
Accuracy of different defenses against oblivious attacks in which attackers only know the original malware detector.

	Attack Name	Accuracy (%)				
		DNN$^+$	KDE	FD-VAE	PAD-SMA	MalPurifier
Drebin	No Attack	92.21	91.93	73.19	94.06	**95.08**
	Salt & Pepper	0.000	0.000	100.0	100.0	**100.0**
	Pointwise	0.000	0.000	69.46	89.70	**100.0**
	MaxMA	24.58	91.93	58.63	94.25	**96.66**
	iMaxMA	24.58	91.93	58.63	94.25	**96.66**
	StepwiseMA	12.71	0.649	13.82	89.05	**96.66**
	AutoAttack	81.35	82.93	42.30	93.69	**96.94**
Androzoo	No Attack	98.74	98.83	95.59	98.20	**99.41**
	Salt & Pepper	87.98	89.46	91.90	**99.96**	99.24
	Pointwise	77.36	71.56	90.68	97.97	**99.24**
	MaxMA	19.71	98.83	94.78	98.42	**99.41**
	iMaxMA	19.71	98.83	94.78	98.42	**99.41**
	StepwiseMA	61.21	8.236	18.23	77.32	**99.41**
	AutoAttack	1.800	27.41	38.25	98.42	**99.41**

Table 2 reports the results of Salt & Pepper, Pointwise, MaxMA, iMaxMA, StepwiseMA, and Autoattack, which are not suitable for a large number of iterations. We first observe

TABLE 3
Accuracy of different defenses under adaptive attacks where adversaries know all defensive mechanisms (if applicable).

	Attack Name	Accuracy (%)							
		DNN	AT-rFGSM^k	AT-Adam	DNN$^+$	KDE	FD-VAE	PAD-SMA	MalPurifier
Drebin	BCA	0.000	6.122	42.21	0.000	0.000	5.102	80.15	**90.91**
	rFGSM	0.000	14.94	85.81	59.46	91.93	74.40	94.25	**95.08**
	Grosse	0.000	6.122	41.93	0.000	0.000	2.690	78.76	**90.91**
	PGD-ℓ_1	0.000	0.186	41.84	0.000	0.000	0.835	77.83	**99.72**
	PGD-ℓ_2	9.833	30.61	86.74	0.093	62.06	2.319	92.30	**95.08**
	PGD-ℓ_∞	0.000	15.21	92.30	51.67	91.93	75.97	94.25	**95.08**
	Salt & Pepper	0.000	97.96	99.07	0.000	0.000	80.43	95.55	**100.0**
	Pointwise	0.000	94.43	94.53	0.000	0.000	67.97	87.65	**99.72**
	MaxMA	0.000	0.371	45.83	0.000	0.000	1.299	77.37	**95.08**
	iMaxMA	0.000	0.371	45.83	0.000	0.000	1.299	77.83	**95.08**
	StepwiseMA	0.000	0.371	45.55	0.371	25.05	1.206	88.22	**95.08**
	AutoAttack	0.000	78.39	71.71	81.35	82.93	42.30	93.69	**96.94**
	Orth PGD-ℓ_1	–	–	–	0.000	6.401	12.43	94.25	**95.08**
	Orth PGD-ℓ_2	–	–	–	3.711	0.000	17.90	94.25	**95.08**
	Orth PGD-ℓ_∞	–	–	–	46.01	89.98	55.29	94.25	**95.08**
	Orth MaxMA	–	–	–	0.000	0.000	13.08	94.25	**95.08**
	Orth iMaxMA	–	–	–	0.000	0.000	13.08	94.25	**95.08**
Androzoo	BCA	0.000	0.090	35.01	0.000	10.62	30.56	86.72	**99.37**
	rFGSM	0.000	8.776	98.16	6.841	98.83	96.00	98.43	**99.41**
	Grosse	0.000	0.090	35.01	0.000	0.045	28.04	56.30	**99.37**
	PGD-ℓ_1	0.000	0.000	15.26	0.000	0.585	3.465	40.41	**99.41**
	PGD-ℓ_2	62.96	89.96	93.70	89.83	73.58	90.14	**98.34**	97.75
	PGD-ℓ_∞	0.000	90.19	93.34	18.59	98.83	97.71	98.42	**99.41**
	Salt & Pepper	89.47	99.73	**99.82**	87.17	89.06	91.85	98.38	99.41
	Pointwise	71.56	99.25	99.33	76.50	68.36	90.68	94.27	**99.37**
	MaxMA	0.000	0.000	14.94	0.000	0.585	7.111	40.41	**97.44**
	iMaxMA	0.000	0.000	14.94	0.000	0.585	6.391	40.41	**97.44**
	StepwiseMA	0.000	0.000	16.11	0.000	79.16	3.555	76.73	**99.41**
	AutoAttack	0.540	98.25	97.80	1.800	27.41	38.25	98.42	**99.41**
	Orth PGD-ℓ_1	–	–	–	0.000	0.000	8.731	98.42	**99.41**
	Orth PGD-ℓ_2	–	–	–	0.045	78.22	43.70	98.42	**99.41**
	Orth PGD-ℓ_∞	–	–	–	19.13	93.65	97.21	98.42	**99.41**
	Orth MaxMA	–	–	–	0.000	0.000	7.111	98.42	**99.41**
	Orth iMaxMA	–	–	–	0.000	0.000	7.111	98.42	**99.41**

that MalPurifier can effectively mitigate these attacks and achieves the highest accuracy against them, except for the Salt & Pepper attack with 99.24% accuracy on Androzoo. The results indicate that our method significantly improves the robustness and outperforms the state-of-the-art methods in terms of gradient-free, gradient-based, and ensemble-based attacks.

We further observe that there exist significant differences in the accuracy values of all defensive methods when dealing with various attacks, except for MalPurifier. For example, KDE can effectively mitigate MaxMA and iMaxMA with an accuracy of 91.93% but cannot defeat the StepwiseMA attack with an accuracy of only 0.649% on Drebin. This is because the other methods are vulnerable to large or small manipulations, while the diversified adversarial perturbation mechanism in MalPurifier can help defend against a range of perturbations from different attacks. Note that the purification model is not dependent on any specific attacks, so all attacks in our experiments are unknown to MalPurifier, highlighting the advantages of this method.

Answer to RQ3: MalPurifier is significantly more robust than DNN$^+$, KDE, FD-VAE, and PAD-SMA. It achieves an accuracy of $\geqslant$ 90.91% and $\geqslant$ 99.24% on both datasets against oblivious attacks, wherein all of these adversarial examples are previously unseen by the model.

5.4 RQ4: Robustness against Adaptive Attacks

Experimental Setup. To evaluate the robustness of these methods against strong attacks, we also wage adaptive attacks in which adversaries know both the malware detector f and adversarial indicator or purifier g. First, we transform the 12 oblivious attacks into adaptive attacks by solving the problem in Eq. 7. Since DNN, AT-rFGSM^k, and AT-Adam do not have any adversarial indicator, the oblivious attacks can trivially meet the adaptive requirement.

Furthermore, we adapt the other 5 attacks in an "orthogonal" (dubbed Orth) manner [45], including Orth PGD-ℓ_1, PGD-ℓ_2, PGD-ℓ_∞, MaxMA, and iMaxMA. Since DNN, AT-rFGSM^k, and AT-Adam do not have another detector, orthogonal attacks are not applicable for these methods. Note that we utilize the same hyper-parameters as those in Section 5.3, except for PGD-ℓ_1 with 500 iterations, PGD-ℓ_2 with 200 iterations and step size 0.05, and PGD-ℓ_∞ with 500 iterations and step size 0.002 in all PGD-based attacks.

Results. Table 3 reports the experimental results against adaptive attacks. First, we can observe that DNN is very vulnerable to all attacks, with 0% accuracy against 11 attacks on Drebin and 8% accuracy against 8 attacks on Androzoo. However, the Salt & Pepper and Pointwise attacks achieve the lowest attack effectiveness in evading DNN on Androzoo, because both of them modify malware examples without using the internal information of target detectors.

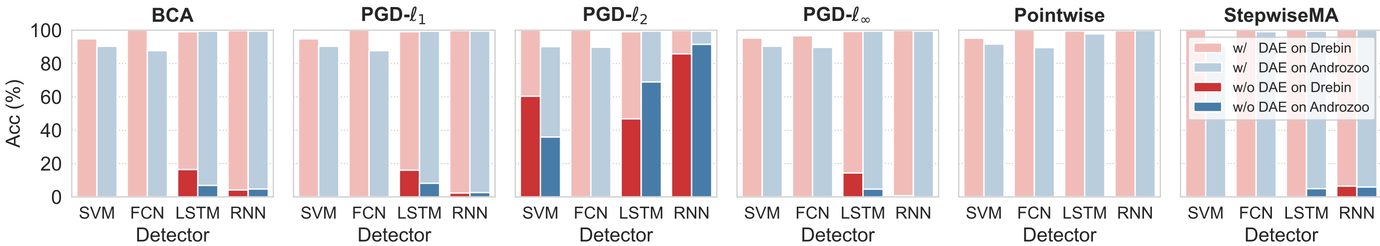

Fig. 5. The accuracy of different detectors against various evasion attacks when equipped with (w/) or without (w/o) the DAE-based purification model on Drebin (drawn in red) and Androzoo (drawn in blue) datasets.

Second, adversarial training methods (e.g., AT-rFGSM^k, and AT-Adam) can harden the robustness of DNN to some extent. For example, AT-rFGSM^k can mitigate the Salt & Pepper, Pointwise, and AutoAttack attacks, and AT-Adam is also effective in defeating rFGSM, PGD-ℓ_2 and PGD-ℓ_∞ attacks. Nevertheless, they are still sensitive to BCA, Grosse, PGD-ℓ_1, MaxMA, iMaxMA, and StepwiseMA attacks (with an accuracy $\leqslant$ 45.83% on both datasets) that are unseen previously. These results indicate the limitations of adversarial training methods in terms of generalization.

Third, although DNN$^+$, KDE, and FD-VAE incorporate another adversary detector, they only show limited effectiveness under a few attacks, and suffer from unseen attacks such as PGD-ℓ_1, MaxMA, iMaxMA, Orth PGD-ℓ_1, Orth MaxMA, and Orth iMaxMA (with an accuracy $\leqslant$ 13.08% on both datasets). Additionally, PAD-SMA not only hardens the DNN with adversarial training, but also combines it with an ICNN-based adversary detector, significantly improving its robustness against different attacks. Especially, PAD-SMA achieves the highest accuracy of 98.34% against the PGD-ℓ_2 attack on Androzoo. However, PAD-SMA is still sensitive to the Grosse, PGD-ℓ_1, MaxMA, and iMaxMA attacks (with an accuracy $\leqslant$ 78.76% on Drebin and $\leqslant$ 56.30% on Androzoo). Considering its high time overhead reported in Table 1, it cannot be called a perfect solution.

In summary, MalPurifier significantly outperforms other defenses, achieving the highest accuracy against all 17 adaptive attacks on the Drebin dataset and 15 attacks on the Androzoo dataset. This indicates that the purification model can accurately recover the original forms of adversarial examples even if the adversary knows its existence.

Answer to RQ4: MalPurifier outperforms other defenses in the condition that adversaries know all defensive mechanisms, and significantly hardens the malware detector against a wide range of adaptive attacks (with an accuracy $\geqslant$ 90.91% on Drebin and $\geqslant$ 97.44% on Androzoo).

5.5 RQ5: Enhancement Brought to Other Detectors

Experimental Setup. To evaluate the flexibility and generalization of the purification model, we also package the DAE model as a plug to protect other detectors, such as *Support Vector Machine* (SVM), *Fully Convolutional Network* (FCN), *Long Short Term Memory* (LSTM), and *Recurrent Neural Network* (RNN). In detail, We use an SVM model with the Sigmoid activation and an FCN model with 3 fully-connected hidden layers (each having 512, 256, and 128 neurons) with the ReLU activation. The LSTM model has a hidden layer with 200 neurons, a sequence length of 1, and uses the Sigmoid function as the activation. Furthermore, we build the RNN model with 3 hidden layers (each having 200 neurons) and the Sigmoid activation. All these models are tuned by the Adam optimizer with 100 epochs, batch size 128, and learning rate 0.001.

Note that the DAE-based purification model works as a plug-and-play preprocessing method, in which we do not retrain the model but directly apply it from the aforementioned experiments. In addition, we wage 6 attacks on test malware examples in an adaptive manner, including BCA, PGD-ℓ_1, PGD-ℓ_2, PGD-ℓ_∞, Pointwise, and StepwiseMA, with the same hyper-parameters in Section 5.4.

Results. Fig. 5 depicts the accuracy improvement of these methods equipped with the DAE-based purification model on Drebin (drawn in red) and Androzoo (drawn in blue) datasets under 6 adaptive attacks. We make three observations as follows.

First, all detectors without enhancement cannot mitigate these adaptive attacks (with accuracy $\leqslant$ 16.42% on Drebin and $\leqslant$ 8.19% on Androzoo), except for the PGD-ℓ_2 attack. This is because ML-based detectors themselves are very vulnerable to these evasion attacks with small perturbations. Especially, the attack effectiveness of the Pointwise attack is 100% when adversaries wage attacks on original detectors. Nevertheless, the PGD-ℓ_2 attack, if running with a large number of iterations, will produce larger perturbations that may not evade detection.

Second, the DAE-based purification model works very well and can significantly improve the robustness of these detectors as a security plug (accuracy increase by $\geqslant$ 39.61% for SVM, $\geqslant$ 87.71% for FCN, $\geqslant$ 30.47% for LSTM, and $\geqslant$ 7.97% for RNN). Significantly, it boosts the accuracy of the RNN model against the Pointwise attack from 0% to 99.72% on Drebin and from 0% to 99.82% on Androzoo, rendering the previously vulnerable RNN model robust against this specific attack. These results strongly demonstrate the versatility of the DAE model, as it can seamlessly transfer to other models without the need for retraining.

Third, the accuracy values of these detectors are similar against different evasion attacks when equipped with the DAE model. The underlying reason for the observation is that the DAE model is trained in an independent and unsupervised way, and can accurately return the adversarial examples to their original forms. Hence, the effectiveness of equipping this security plug relies more on the detector itself, as the DAE model solely preprocesses the input data while the detector is responsible for classification.

Answer to RQ5: The DAE model (the core module of MalPurifier) can be flexibly transferred to other detectors without retraining. Notably, it greatly enhances the robustness against evasion attacks, even when these models are highly vulnerable to such attacks.

6 Limitations and Open Issues

While MalPurifier has yielded highly promising results, it is evident that this approach possesses some limitations. We additionally conduct experiments to assess the robustness of MalPurifier against the Mimicry attack, wherein the attacker introduces perturbations to a malware sample to closely resemble a benign application. Unfortunately, MalPurifier cannot effectively resist Mimicry ($\leqslant 60\%$) and has less effectiveness as malware samples are guided by more benign samples. This is primarily because the DAE model tends to purify these samples as benign due to their similarity to benign samples, leading to their misclassification. To mitigate this issue, we believe that implementing countermeasures, such as generating a more diverse set of adversarial examples or enhancing the adversarial purification model itself, would be beneficial. We plan to explore these possibilities in our future research.

Another limitation of our approach may be the potential performance degradation as new malware samples evolve. Even within this study, the effectiveness on the Androzoo dataset differs from that on the Drebin dataset, partly due to the inclusion of new samples. This vulnerability does not lie within the MalPurifier framework itself. We believe that by incorporating new data and leveraging dynamic updates, we can enhance the detection system's capabilities.

While our approach may not be foolproof, we firmly believe that it substantially enhances the resistance of Android malware detection against diverse evasion attacks in a lightweight and plug-and-play manner.

7 Related Work

This section begins with a review of existing studies on ML-based Android malware detection methods, followed by an introduction to evasion attacks against these approaches and a brief discussion of state-of-the-art solutions.

7.1 ML-based Android Malware Detection

Researchers have developed numerous ML-based Android malware detection methods that typically classify APKs using features extracted from the manifest and bytecode. For instance, *Drebin* [30] identifies Android malware by exploiting binary static features and employing SVM for classification. *MaMaDroid* [46] extracts sequences of API calls and then trains classifiers like K-Nearest Neighbors (KNN) to detect malware.

Additionally, DL-based methods [40], [47] have demonstrated remarkable capabilities. For example, Andre [48] is a hybrid representation learning approach that clusters Android malware from multiple sources and classifies them using a three-layer DNN when they behave like existing families. Qiu *et. al* [5] proposed a framework that extracts heterogeneous features and utilizes DNN to recognize unknown and evolving malware.

Given the widespread use and outstanding performance of DNNs in Android malware detection, this study aims to enhance the robustness of DNN-based detectors using an independently trained purifier to pre-process input samples. This purifier restores the feature representation of adversarial malware to its original version and preserves the features of clean samples as much as possible, enabling the DNN model to correctly classify Android applications.

7.2 Evasion Attacks in Android Malware Detection

In the context of Android malware detection, evasion attacks employ crafted inputs to mislead models such that malicious apps will be classified as benign. As discussed in Section 2, it can be divided into problem-space attacks and feature-space attacks.

Problem-space attacks modify the Android apps directly, such as perturbations onto Android manifest and Dalvik bytecode [12] or insertion of benign components into malicious samples [8], for generating adversarial malware to deceive ML-based detectors. On the contrary, feature-space attacks map the malware example into a feature vector, and then introduce perturbations to the vector values [29] or reconstruct the vector representation [41] to achieve misclassification. Moreover, recent studies demonstrate that the utilization of ensemble attacks [13], [49] intensifies the impact of the attacks, presenting a more formidable challenge for defense mechanisms.

To combat the escalating prevalence of evasion attacks, the method presented in this paper is not tailored to counter any specific attack. Instead, it strives to establish a universally applicable approach that effectively mitigates both problem-space attacks and feature-space attacks. Additionally, the proposed method significantly enhances robustness while maintaining accuracy on clean samples.

7.3 Defenses against Evasion Attacks

Adversarial training [18], [22], [50] is widely recognized as one of the most popular methods for defeating evasion attacks. Recent research [13], [51] has further shown that combining adversarial training with *ensemble learning* can enhance model robustness. However, it is worth noting that adversarial training typically retrains the model by generating and incorporating adversarial examples, which can lead to a significant increase in computational burden. Also, the defenses may not effectively mitigate attacks that differ significantly from the ones encountered during training.

In addition, there are also several countermeasures to identify evasion attacks through an auxiliary model. For example, Li *et. al* [23] introduced a Variational AutoEncoder (VAE) to distinguish benign examples from adversarial malware according to reconstruction errors, and Li *et. al* [20] leverages a convex DNN model-based detector to recognize the evasion attacks. Despite not requiring retraining of the target model, the auxiliary model remains closely coupled with the malware detection model and is still unable to effectively handle sophisticated and adaptive attacks.

Different from existing studies, the proposed method involves diversified perturbations to increase its generalization capabilities against novel or unseen attacks. Furthermore, the independent training of the purification model makes it computationally efficient and easily scalable.

8 Conclusion and Future Work

We devise a plug-and-play defense method against evasion attacks for malware detection. Instead of hardening the malware detector solely, an independently training purification model is used to purify adversarial malware. In the training phase, we generate adversarial examples via injecting diversified perturbations into malware, along with protective noises added onto benign samples, to achieve a trade-off between robustness and detection accuracy. Additionally, we establish a DAE-based purification model with a customized loss function, resulting in better feature representation and robust accuracy. Experiments via two Android datasets demonstrate the soundness of the method against a set of attacks, including unseen ones.

Given the current trend in the use of diffusion models for adversarial purification, the future development of our work, which may further improve classifier security, is to leverage this technology to defend against evolving evasion attacks. Another interesting future extension of our approach may be to investigate robust methods against poisoning attacks, including the purification samples in the both training and test phases. These two parts of the research will substantially improve the security of employing machine learning techniques in Android malware detection.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62202097 and 62072100, in part by the China Postdoctoral Science Foundation under Grant 2022M710677, and in part by the Jiangsu Funding Program for Excellent under Grant 2022ZB137.

References

[1] Zimperium. 2022 global mobile threat report. [Online]. Available: https://www.zimperium.com/global-mobile-threat-report/

[2] T. Shishkova. The mobile malware threat landscape in 2022. [Online]. Available: https://securelist.com/mobile-threat-report-2022/108844/

[3] H. Zhu, Y. Li, R. Li, J. Li, Z. You, and H. Song, "Sedmdroid: An enhanced stacking ensemble framework for android malware detection," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 984–994, 2021.

[4] J. Xu, Y. Li, R. H. Deng, and K. Xu, "Sdac: A slow-aging solution for android malware detection using semantic distance based api clustering," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 1149–1163, 2022.

[5] J. Qiu, Q.-L. Han, W. Luo, L. Pan, S. Nepal, J. Zhang, and Y. Xiang, "Cyber code intelligence for android malware detection," *IEEE Transactions on Cybernetics*, vol. 53, no. 1, pp. 617–627, 2022.

[6] H.-J. Zhu, L.-M. Wang, S. Zhong, Y. Li, and V. S. Sheng, "A hybrid deep network framework for android malware detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5558–5570, 2022.

[7] W. Fang, J. He, W. Li, X. Lan, Y. Chen, T. Li, J. Huang, and L. Zhang, "Comprehensive android malware detection based on federated learning architecture," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3977–3990, 2023.

[8] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 1332–1349.

[9] K. Zhao, H. Zhou, Y. Zhu, X. Zhan, K. Zhou, J. Li, L. Yu, W. Yuan, and X. Luo, "Structural attack against graph based android malware detection," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3218–3235.

[10] H. Li, Z. Cheng, B. Wu, L. Yuan, C. Gao, W. Yuan, and X. Luo, "Black-box adversarial example attack towards fcg based android malware detection under incomplete feature information," in *32rd USENIX Security Symposium (USENIX Security 23)*, 2023.

[11] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "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, 2019.

[12] 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.

[13] D. Li and Q. Li, "Adversarial deep ensemble: Evasion attacks and defenses for malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3886–3900, 2020.

[14] C. Li, X. Chen, D. Wang, S. Wen, M. E. Ahmed, S. Camtepe, and Y. Xiang, "Backdoor attack on machine learning based android malware detectors," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3357–3370, 2022.

[15] G. Severi, J. Meyer, S. Coull, and A. Oprea, "{Explanation-Guided} backdoor poisoning attacks against malware classifiers," in *30th USENIX security symposium (USENIX security 21)*, 2021, pp. 1487–1504.

[16] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras, "When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1299–1316.

[17] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *28th USENIX security symposium (USENIX security 19)*, 2019, pp. 321–338.

[18] D. Li, Q. Li, Y. Ye, and S. Xu, "A framework for enhancing deep neural networks against adversarial malware," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 736–750, 2021.

[19] Y. Qiao, W. Zhang, Z. Tian, L. T. Yang, Y. Liu, and M. Alazab, "Adversarial elf malware detection method using model interpretation," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 605–615, 2023.

[20] D. Li, S. Cui, Y. Li, J. Xu, F. Xiao, and S. Xu, "Pad: Towards principled adversarial malware detection against evasion attacks," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[21] X. Jia, Y. Zhang, B. Wu, J. Wang, and X. Cao, "Boosting fast adversarial training with learnable adversarial initialization," *IEEE Transactions on Image Processing*, vol. 31, pp. 4417–4430, 2022.

[22] C. P. Lau, J. Liu, H. Souri, W.-A. Lin, S. Feizi, and R. Chellappa, "Interpolated joint space adversarial training for robust and generalizable defenses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[23] H. Li, S. Zhou, W. Yuan, X. Luo, C. Gao, and S. Chen, "Robust android malware detection against adversarial example attacks," in *Proceedings of the Web Conference 2021*, 2021, pp. 3603–3612.

[24] M. Naseer, S. Khan, M. Hayat, F. S. Khan, and F. Porikli, "A self-supervised approach for adversarial robustness," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 262–271.

[25] J. Yoon, S. J. Hwang, and J. Lee, "Adversarial purification with score-based generative models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 062–12 072.

[26] F. Croce, S. Gowal, T. Brunner, E. Shelhamer, M. Hein, and T. Cemgil, "Evaluating the adversarial robustness of adaptive test-time defenses," in *International Conference on Machine Learning*. PMLR, 2022, pp. 4421–4435.

[27] R. Theagarajan and B. Bhanu, "Privacy preserving defense for black box classifiers against on-line adversarial attacks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 12, pp. 9503–9520, 2022.

[28] W. Nie, B. Guo, Y. Huang, C. Xiao, A. Vahdat, and A. Anandkumar, "Diffusion models for adversarial purification," in *International Conference on Machine Learning*. PMLR, 2022, pp. 16 805–16 827.

[29] G. Xu, G. Xin, L. Jiao, J. Liu, S. Liu, M. Feng, and X. Zheng, "Ofei: A semi-black-box android adversarial sample attack framework against dlaas," *IEEE Transactions on Computers*, 2023.

[30] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of

android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.
[31] 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*, 2016, pp. 468–471.
[32] N. Šrndić and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 197–211.
[33] C. Jeon, I. Yun, J. Jung, M. Wolotsky, and T. Kim, "Avpass: Leaking and bypassing antivirus detection model automatically," in *Black Hat USA 2017*. Black Hat, 2017.
[34] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.
[35] T. Zhang and Z. Zhu, "Interpreting adversarially trained convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 7502–7511.
[36] 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.
[37] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," in *International Conference on Data Mining and Big Data*. Springer, 2022, pp. 409–423.
[38] V. Vo, E. M. Abbasnejad, and D. Ranasinghe, "Query efficient decision based sparse attacks against black-box deep learning models," in *International Conference on Learning Representations*, 2021.
[39] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in *International conference on machine learning*. PMLR, 2020, pp. 2206–2216.
[40] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2019.
[41] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *European symposium on research in computer security*, 2017, pp. 62–79.
[42] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," *arXiv preprint arXiv:1702.06280*, 2017.
[43] T. Pang, C. Du, Y. Dong, and J. Zhu, "Towards robust detection of adversarial examples," *Advances in neural information processing systems*, vol. 31, 2018.
[44] D. Li and Q. Li, "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.
[45] O. Bryniarski, N. Hingun, P. Pachuca, V. Wang, and N. Carlini, "Evading adversarial example detection defenses with orthogonal projected gradient descent," in *International Conference on Learning Representations*, 2022.
[46] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version)," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–34, 2019.
[47] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Dl-droid: Deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, p. 101663, 2020.
[48] Y. Zhang, Y. Sui, S. Pan, Z. Zheng, B. Ning, I. Tsang, and W. Zhou, "Familial clustering for weakly-labeled android malware using hybrid representation learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3401–3414, 2019.
[49] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9185–9193.
[50] B. G. Doan, S. Yang, P. Montague, O. De Vel, T. Abraham, S. Camtepe, S. S. Kanhere, E. Abbasnejad, and D. C. Ranashinghe, "Feature-space bayesian adversarial learning improved malware detector robustness," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 12, 2023, pp. 14 783–14 791.
[51] M. Ficco, "Malware analysis by combining multiple detectors and observation windows," *IEEE Transactions on Computers*, vol. 71, no. 6, pp. 1276–1290, 2022.

Yuyang Zhou received the B.S. degree in Electronic Information Engineering from Nanjing University of Science and Technology in 2016 and the Ph.D. degree in Cyberspace Security from Southeast University in 2021. He is currently working as a postdoc with the School of Cyber Science and Engineering, Southeast University. His major research interests include moving target defense, Android malware detection, security modeling, and proactive DDoS mitigation. He has published in some of the topmost journals and conferences like IEEE TIFS, IEEE TII, and ACM CCS, and is involved as a reviewer and in technical program committees of several journals and conferences in the field. He is a Member of IEEE and CCF.

Guang Cheng received the B.S. degree in Traffic Engineering from Southeast University in 1994, the M.S. degree in Computer Application from Hefei University of Technology in 2000, and the Ph.D. degree in Computer Network from Southeast University in 2003. He is a Full Professor in the School of Cyber Science and Engineering, Southeast University, Nanjing, China. He has authored or coauthored eight monographs, and produced more than 100 technical papers, including top journals and top conferences like IEEE ToN, IEEE TIFS, IEEE TII, and INFOCOM. His research interests include network security, network measurement, and traffic behavior analysis. He is the director of Jiangsu Cyber Security Association, China, the vice director of China Computer Federation Technical Committee of Internet (CCF TCI), and the vice director of Jiangsu Computer Society, China. He is a Member of IEEE and a Distinguished Member of CCF.

Zongyao Chen received the B.S. degree in Information Security from HaiNan University in 2022. He is currently pursuing the master degree with the School of Cyber Science and Engineering, Southeast University. His major research interests include moving target defense, Android malware detection, and reverse engineering.

Shui Yu obtained his PhD from Deakin University, Australia, in 2004. He currently is a Professor of School of Computer Science, University of Technology Sydney, Australia. Dr Yu's research interest includes Big Data, Security and Privacy, Networking, and Mathematical Modelling. He has published two monographs and edited two books, and produced more than 500 technical papers, published in top journals such as IEEE TPDS, TC, TIFS, TMC, TKDE, TETC, ToN, and INFOCOM. His h-index is 68. Dr Yu initiated the research field of networking for big data in 2013, and his research outputs have been widely adopted by industrial systems, for example, the auto scale strategy of Amazon Cloud against distributed denial-of-service attacks. He is currently serving a number of prestigious editorial boards, including IEEE Communications Surveys and Tutorials (Area Editor), IEEE Communications Magazine, IEEE Internet of Things Journal, among others. He is a member of AAAS and ACM, a Distinguished Visitor of IEEE Computer Society, a voting member of IEEE ComSoc Educational Services board, and an elected member of Board of Governor of IEEE Vehicular Technology Society. He is a Fellow of IEEE.