Robust Android Malware Detection against Adversarial Example Attacks

Heng Li[*]
Huazhong University of Science and Technology
WuHan, China
liheng@hust.edu.cn

Shiyao Zhou[*]
The Hong Kong Polytechnic University
Hong Kong, China
shiyao.zhou@connect.polyu.hk

Wei Yuan[†]
Huazhong University of Science and Technology
WuHan, China
yuanwei@mail.hust.edu.cn

Xiaopu Luo
The Hong Kong Polytechnic University
Hong Kong, China
csxluo@comp.polyu.edu.hk

Cuiying Gao
Huazhong University of Science and Technology
WuHan, China
gaoc@hust.edu.cn

Shuiyan Chen
Huazhong University of Science and Technology
WuHan, China
chenshuiyan@hust.edu.cn

ABSTRACT

Adversarial examples pose severe threats to Android malware detection because they can render the machine learning based detection systems useless. How to effectively detect Android malware under various adversarial example attacks becomes an essential but very challenging issue. Existing adversarial example defense mechanisms usually rely heavily on the instances or the knowledge of adversarial examples, and thus their usability and effectiveness are significantly limited because they often cannot resist the unseen-type adversarial examples. In this paper, we propose a novel robust Android malware detection approach that can resist adversarial examples without requiring their instances or knowledge by jointly investigating malware detection and adversarial example defenses. More precisely, our approach employs a new VAE (variational autoencoder) and an MLP (multi-layer perceptron) to detect malware, and combines their detection outcomes to make the final decision. In particular, we share a feature extraction network between the VAE and the MLP to reduce model complexity and design a new loss function to disentangle the features of different classes, hence improving detection performance. Extensive experiments confirm our model's advantage in accuracy and robustness. Our method outperforms 11 state-of-the-art robust Android malware detection models when resisting 7 kinds of adversarial example attacks.

CCS CONCEPTS

• **Security and privacy** → **Malware and its mitigation**.

KEYWORDS

Android Malware Detection, Adversarial Example, Mobile Security

[*]Both authors contributed equally to this research.
[†]Corresponding author

ACM Reference Format:
Heng Li, Shiyao Zhou, Wei Yuan, Xiaopu Luo, Cuiying Gao, and Shuiyan Chen. 2021. Robust Android Malware Detection against Adversarial Example Attacks. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021 ,Ljubljana, Slovenia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3442381.3450044

1 INTRODUCTION

Android is the most popular mobile operating system, capturing roughly 85% of the global market share [20]. Unfortunately, Android has become the primary target of 97% mobile malware [27]. Many Android malware detection approaches have been proposed [22, 24, 29, 31, 32], and most of them leverage machine learning algorithms to train a binary classifier for determining whether an App is malicious or not according to its features [17, 23, 26, 35]. These classifiers are, however, vulnerable to adversarial examples, which are generated by adding purposeful perturbations to natural examples [10, 16]. For instance, the attack method proposed by Chen *et al.* [6] generates adversarial malware examples that can deceive two famous Android malware detection Drebin [1] and MaMaDroid [25], by injecting adversarial perturbations into APKs' Dexcode and AndroidManifest.xml. Adversarial perturbations can make the malware conceal themselves better when confronting machine learning based malware detectors.

Unfortunately, there are few defense mechanisms to defend against adversarial example attacks on Android malware detectors. Although there exist defense mechanisms developed for image adversarial examples [9, 30], they cannot be directly applied to Android malware detection because of the significant differences between images and Android Apps. First, the similarity criterion used in image domain[1] is not suitable for the modification of Apps because they do not take into account how to preserve the malicious functionality in malware [33]. Second, the input space of image domain is continuous whereas that of Android malware is often discrete [37]. The input of Android malware detector (i.e., the features extracted from Android App) will be described detailedly in §2.

[1]In the domain of images, the perturbations imposed on natural examples should be small so that the adversarial examples are similar to the natural ones.

In this paper, to hedge the gap, we don't directly migrate the adversarial example defense mechanisms in image domain to malware detection, but design a new defense mechanism for Android malware detection. To be specific, we design an adversarial example defense model that can detect the similarity between nature examples (including benign and malicious examples) and adversarial examples. Further more, we jointly investigate malware detection and adversarial example defense, and then develop a novel Android App classifier that is inherently resistant to adversarial examples. The key idea of our approach is to achieve an ideal decision boundary separating benign examples from malicious/adversarial ones. It is challenging to achieve this goal because in practice the realistic decision boundary is often non-ideal due to the absence of some crucial examples close to the class boundary. In particular, because of the inconformity between the ideal and the realistic decision boundary, there exist some intersected regions in the input space that result in misclassification, which leaves room for adversarial example attacks. That is, to mislead a classifier, an adversary just needs to move a correctly classified malicious example into one intersected region by adding some adversarial perturbations.

To get the desired decision boundary and decrease the likelihood of adversarial examples being successfully generated, we employ a similarity constraint to squeeze the room for adversarial examples. More specifically, we introduce a variational autoencoder (VAE) [4, 18] to distinguish benign examples from others according to reconstruction errors. Only when the adversarial examples are very similar to the benign ones can they evade the detection. Different from the existing adversarial example defenses, our method does not require the instances or the knowledge of adversarial examples, making it practical to deploy. To further enhance discrimination capability, we develop a novel loss function for the VAE to disentangle the features of different classes. For convenience, our VAE is termed FD-VAE in the remainder of this paper. Accordingly, the FD-VAE can learn a compact representation for its inputs, which has a small inner-class distance and a large inter-class distance. Moreover, our model uses an MLP (multi-layer perceptron) for classification. Our model makes a final decision on malware detection by combining the decisions of the FD-VAE and the MLP.

Since our model simultaneously fulfills two tasks (i.e., malware detection and adversarial example defense), its complexity may be higher than those targeting one task. In this paper, we propose a network sharing based technique to reduce model complexity. Recall that our model consists of two deep neural networks, i.e., a FD-VAE and an MLP, both of which need to extract features from the raw inputs for classification. This motivates us to share the feature extraction network between two modules for parameter amount reduction. This technique reduces model complexity and also facilitates model training. More details can be found in §3.

The major contributions of this work include:

- To our best knowledge, this is the first work that develops a classification model for both malware detection and adversarial malware defense, without requiring the instances or knowledge about adversarial examples.
- We design a novel loss function for feature disentangle to enhance detection performance, and propose to share the feature extraction network among our model's components to reduce model complexity.
- Extensive experiments demonstrate that our method can successfully resist 7 adversarial example attack methods, and significantly outperform 11 robust detection models.

The remainder of this paper is organized as follows. Section 2 describes the problems of Android malware detection and adversarial example defense. Section 3 proposes our method, including model, loss function design and algorithm. Experimental results and analyses are provided in Section 4. Related work is briefly introduced in Section 5, followed by the concluding remarks in Section 6.

2 PROBLEM

The machine learning based Android malware detection can be briefly described as follow. Let $L(\boldsymbol{\theta}, \boldsymbol{x}, y)$ denote the loss of the classifier (i.e., Android malware detector), where $\boldsymbol{\theta}$ represents the classifier's parameters, $\boldsymbol{x}$ represents the natural example, and $y \in \{0, 1\}$ denotes the corresponding label. $y = 0$ or $y = 1$ means that the corresponding example is benign or malicious, respectively. The main task of Android malware detection is to achieve the optimal parameters of the classifier, i.e.,

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

where $\mathcal{D}$ is the underlying data distribution of training examples.

An adversary uses adversarial examples generated from malicious examples to mislead an Android malware detector. The adversarial examples are usually generated by adding some perturbations into malicious examples.

Let $\bar{\boldsymbol{x}}$ be a malicious example, and $\mathcal{S}(\bar{\boldsymbol{x}})$ denote the set of the candidate examples that preserve the malicious functionality of the App corresponding to $\bar{\boldsymbol{x}}$. Then the adversarial examples $\boldsymbol{x}^*$ can be obtained through solving

$$\max_{\boldsymbol{x}^* \in \mathcal{S}(\bar{\boldsymbol{x}})} L(\boldsymbol{\theta}, \boldsymbol{x}^*, y). \tag{2}$$

How to generate the adversarial examples $\boldsymbol{x}^*$ depends on the features chosen for malware detection. In the existing literature, the features fall into two categories: dynamic and static. Dynamic features are exacted by monitoring the running of Apps, including network traffic, battery usage, IP address, etc. Static features are obtained before the execution of Apps, including permissions, intent-actions and sensitive API calls. Extracting dynamic features usually requires monitoring Apps' behavior at run-time, resulting in significant overhead. Therefore, static features are more common in practical applications [36].This motivates us to study the Android malware adversarial examples derived from static features.

More specifically, as shown in Fig. 1, we consider the static features of permission requirements, intent action declarations and sensitive API calls.

- Permissions must be required by an App from its user to access sensitive user data and certain system resources. For example, *CAMERA* permission allows an application to access the camera to take pictures.
- Intent actions are a way of telling Android what standard operation activities can perform. For example, Android knows that all activities registered for a *send* action are capable of sending messages.
- As for sensitive API calls, they can describe App behaviors as they clearly state what an App has done. For instance,

calling *sendTextMessage*() means an App is sending a text message.

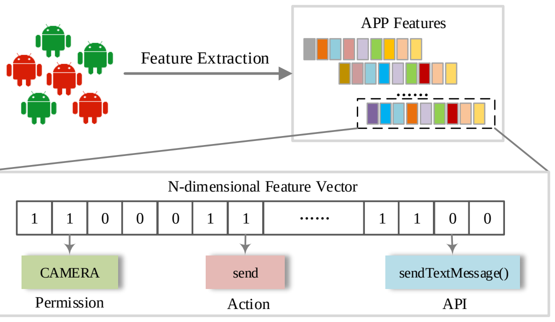


Figure 1: Feature extraction.

In summary, permission requirements, intent action declarations and sensitive API calls provide the clues for identifying malicious Apps. Hence they are chosen as the features for our malware detection.

For convenience, the values of these features are vectorized as a binary vector $\boldsymbol{x} = [x_1, \cdots, x_m] \in \{0,1\}^m$, where x_j indicates whether the jth feature is present ($x_j = 1$) or not ($x_j = 0$). Then we have $\mathcal{S}(\bar{\boldsymbol{x}}) = \{\boldsymbol{x}^* | \boldsymbol{x}^* \in \{0,1\}^m, \boldsymbol{x}^* \vee \bar{\boldsymbol{x}} = \boldsymbol{x}^*\}$, where $\vee$ means bit OR operation and $\boldsymbol{x}^* \vee \bar{\boldsymbol{x}} = \boldsymbol{x}^*$ corresponds to the requirement of malicious functionality preservation. To mislead the classifier $\boldsymbol{\theta}$ with adversarial examples, the adversary attempts to solve the following integer optimization problem

$$\max_{\boldsymbol{x}^* \in \mathcal{S}(\bar{\boldsymbol{x}})} L(\boldsymbol{\theta}, \boldsymbol{x}^*, y), \tag{3}$$

s.t.

$$\begin{cases} \boldsymbol{x}^* \in \{0,1\}^m \\ \exists \bar{\boldsymbol{x}} \in \Omega, \boldsymbol{x}^* \vee \bar{\boldsymbol{x}} = \boldsymbol{x}^* \end{cases} \tag{4}$$

where Ω denotes the set of malicious examples.

The existing adversarial example attacks are divided into two categories: white box and black box. The white-box methods assume that the knowledge including training data, classifier type and classifier parameters (i.e., $\boldsymbol{\theta}$) are available for the adversary. Contrarily, the black-box methods assume that the adversary has no access to the classifier's parameters.

In general, resisting adversarial examples is more challenging than generating them. At present, most of defenses are dedicatedly designed for image adversarial examples. In this paper, we aim to develop a neural network model which is able to detect Android malware under adversarial example attacks. The key idea is to optimize model parameters for loss minimization, while taking the prediction of the adversaries' response into account. Therefore, this task can be formulated as the minmax optimization problem

$$\min_{\boldsymbol{\theta}} (\max_{\boldsymbol{x}^* \in \mathcal{S}(\bar{\boldsymbol{x}})} y \cdot L(\boldsymbol{\theta}, \boldsymbol{x}^*, y)) \tag{5}$$

s.t.

$$\begin{cases} \boldsymbol{x}^* \in \{0,1\}^m \\ \exists \bar{\boldsymbol{x}} \in \Omega, \boldsymbol{x}^* \vee \bar{\boldsymbol{x}} = \boldsymbol{x}^* \end{cases} \tag{6}$$

Here, the outer optimization represents designing an optimal classifier to minimize performance loss under the anticipation for the adversary's response. The inner optimization reflects the response of the adversarial when facing a classifier with parameters $\boldsymbol{\theta}$. In practice, the adversary usually obtains adversarial examples by obtaining or approximating the inner optimization problem's solution. Furthermore, the inner objective function is set to $y \cdot L(\boldsymbol{\theta}, \boldsymbol{x}^*, y)$ instead of $L(\boldsymbol{\theta}, \boldsymbol{x}^*, y)$, because adversarial examples are generated only based on malicious Apps[2].

3 METHODOLOGY

In some existing literature, malware detection and adversarial example defense are studied independently. The methods proposed for detection and defense are then combined to form a closed-loop solution for robust malware detection[3]. In this paper, however, we attempt to simultaneously solve these two problems through tactfully designing a classification model.

3.1 Motivation

In general, the problem of malware detection can be solved using a binary classifier. In this paper, we introduce an MLP to detect malware. Adversarial example defense is a more challenging problem. Let's reconsider the optimization problem (5)-(6) . The challenge stems from the fact that adversarial examples $\boldsymbol{x}^*(\boldsymbol{\theta})$ are actually unknown in practice. A natural countermeasure is to obtain the knowledge on $\boldsymbol{x}^*(\boldsymbol{\theta})$ from some collected instances of adversarial examples. And this is the intuition behind the popular defense method of adversarial retraining. However, this approach has two main shortcomings. First, one may have either no adversarial examples, or they are not statistically-representative. Second, the reliance on $\boldsymbol{x}^*(\boldsymbol{\theta})$ may weaken the generalization ability of the defense method. As a result, when facing with unseen-type adversarial examples, the defense method may lose efficacy.

In practice, the one-class classification based detection methods can solve the above problem, however, they often suffer from high false alarm rates. In response, we propose to use the disentangled features to clearly separate benign examples from others in a latent space. To this end, we improve the traditional VAEs by introducing a new loss function, which leads the VAEs to enlarge inter-class distance and shorten inner-class distance in the latent space. In this way, we can achieve a better boundary between benign examples and others, and hence reduce false alarms.

Since our model simultaneously fulfills two tasks (i.e., malware detection and adversarial example defense), its complexity may be higher than the models targeting one single task. In this paper, we propose a network sharing technique to reduce model complexity. Recall that our model consists of two deep neural networks, i.e., an improved VAE and an MLP, both of which need to extract features from the raw inputs for classification. This motivates us to share the feature extraction network between two modules. This technique reduces model complexity and facilitates model training.

[2]For an adversary, it is usually meaningless to create an adversarial example that is harmless to users.
[3]More discussions on adversarial example defense can be found in Section §4.

3.2 Preliminaries

VAE is often used to design complex generative models of data and fits them to large datasets [12]. Similar to Autoencoder (AE), VAE is composed of two neural networks [2], i.e., an encoder and a decoder. The encoder encodes the input into a Gaussian distribution, which can be characterized by a mean ($\boldsymbol{\mu}$) and a variance ($\boldsymbol{\sigma}$), so that the decoder can reconstruct the input from a latent vector sampled from the Gaussian distribution. Different from AE, however, VAE not only learns to use a latent vector to reconstruct the input, but also learns the distribution of the input in latent space.

For a VAE, there usually exists a tradeoff between reconstruction accuracy and unit Gaussian distribution match. Therefore, the loss function of a VAE consists of two items, i.e., reconstruction loss and Kullback-Leibler (KL) divergence. The former evaluates how accurately the VAE reconstructs the inputs, and the latter measures how closely the distribution generated by encoder matches the unit Gaussian distribution.

In our model, we use FD-VAE to classify the inputs based on their reconstruction errors. Different from the traditional VAE, our FD-VAE is equipped with a new loss function that disentangles the features of benign data and others. The detail loss design of the FD-VAE will be described in subsection §3.4.

3.3 Model

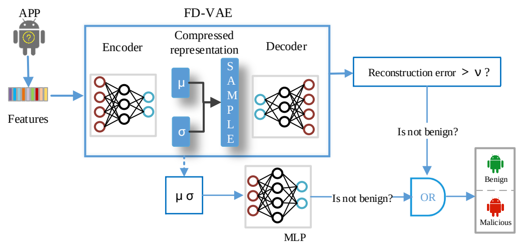


Figure 2: Our proposed model.

As shown in Fig. 2, our model consists of two modules: 1) an FD-VAE, and 2) an MLP. The FD-VAE plays two important roles, i.e., a feature extractor and a classifier. First, it decides whether its inputs are benign or not according to its reconstruction error. An input declares to be benign only when its reconstruction error does not exceed a predetermined threshold ν. Second, its encoder extracts the discriminative features from its inputs, which are then used by its decoder and the MLP.

In the training phase, the FD-VAE is fed with multiple batches of natural examples (i.e., benign and malicious examples) and iteratively updates its parameters for loss function minimization. Once the FD-VAE has been well trained, its encoder will be able to learn good representation for inputs. In the phase of testing, the FD-VAE can be fed with benign, malicious and adversarial examples. It permits through benign examples and rejects the malicious/adversarial ones.

Different from the FD-VAE, the MLP is a binary classifier, whose output gives the probabilities that an input is benign or not. In general, the performance of a classifier depends heavily on the features extracted from the raw inputs. To exploit the feature extraction capability of the FD-VAE, we connect the MLP with the encoder of the FD-VAE. Accordingly, the encoder acts as the pre-trained module of the MLP. In this way, we share the feature extraction network between the FD-VAE and the MLP, hence reducing model complexity. To train the MLP, we feed both benign and malicious examples into the encoder that has been well trained. The outputs of the encoder are then forwarded to the MLP for classification. During training, the parameters of MLP are iteratively updated, but the encoder of the FD-VAE is fixed.

Finally, to test an Android App, one just needs to input its feature vector to our model and then combine the decisions of FD-VAE and MLP to make a final decision. More details can be found in §3.5.

3.4 Loss Functions

We design two loss functions to lead the FD-VAE and the MLP to optimize their parameters during training, respectively.

The loss function of the FD-VAE is composed of three items, i.e.,

$$L_{FD-VAE} = \lambda_1 L_1 + \lambda_2 L_2 + \lambda_3 L_3, \tag{7}$$

where λ_1, λ_2 and λ_3 are positive weights. Here, L_1 corresponds to reconstruction loss which evaluates how accurately the FD-VAE reconstructs the inputs, L_2 denotes the Kullback-Leibler (KL) divergence resulting from the mismatch between the compressed representation and the unit Gaussian distribution, and L_3 is the new loss for feature disentangle. Let $e(\cdot)$ and $d(\cdot)$ denote the encoder and the decoder, respectively. We use $\mathcal{D}$ and $\mathcal{D}_{\mathcal{B}}$ to represent the distribution of natural examples and benign examples, respectively. Then we have

$$L_1 = -E_{\boldsymbol{x}\sim\mathcal{D}_B}[\boldsymbol{x}\log(d(e(\boldsymbol{x}))) + (1-\boldsymbol{x}) * \log(1 - d(e(\boldsymbol{x})))] \tag{8}$$

and

$$\begin{aligned} L_2 &= KL(\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\sigma})||\mathcal{N}(0,1)) \\ &= \frac{1}{2}E_{\boldsymbol{x}\sim\mathcal{D}_B}(\boldsymbol{\mu}^2 + \boldsymbol{\sigma}^2 - \log\boldsymbol{\sigma}^2 - 1) \end{aligned} \tag{9}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and the variance of the Gaussian distribution, as shown in Fig. 2.

Now we explain how to design L_3 to lead the encoder to disentangle the features of benign and malicious/adversarial examples. We feed a number of natural example pairs to the VAE's encoder. For convenience, we denote any example pair by $\{(\boldsymbol{x}_i, y_i), (\boldsymbol{x}_j, y_j)\}$ ($i \neq j$), where y_i and y_j represent the labels of natural examples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, respectively. To disentangle the features of different classes, the compressed representations of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are required to be as different as possible, if $y_i \neq y_j$ holds. Otherwise, they are required to be as close as possible. This strategy can be described by a piecewise function

$$f(\boldsymbol{x}_i,\boldsymbol{x}_j) = \begin{cases} (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^2 & y_i = y_j \\ \max(k - (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^2, 0) & y_i \neq y_j \end{cases} \tag{10}$$

In (10), k is a positive number, and $(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^2$ is used to measure the difference between the representations of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. If $y_i = y_j$ holds, the difference is expected to be as small as possible. Otherwise, we consider two cases. First, when $(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^2 < k$ holds, the loss equals to $k - (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^2$. Under this situation, the loss leads the FD-VAE to update its parameters for feature disentangle. Second, once $(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^2$ equals to or exceeds k, the loss reduces to zero. Accordingly, the parameters of the FD-VAE will not be updated anymore, which avoids training collapse due to the endless pursuit of feature disentangle. Now we can define L_3 as

$$L_3 = E_{\boldsymbol{x}_i, \boldsymbol{x}_j \sim \mathcal{D}} f(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{11}$$

Finally, we design a loss function for the MLP. We use $m(\cdot)$ to represent the MLP. Then the loss function of the MLP is defined by

$$\begin{aligned} L_{mlp} = -E_{\boldsymbol{x} \sim \mathcal{D}}[y * \log(m(e(\boldsymbol{x})))+ \\ (1-y) * \log(1 - m(e(\boldsymbol{x})))] \end{aligned} \tag{12}$$

where y is the label of $\boldsymbol{x}$, $e(\boldsymbol{x})$ denotes the output of the encoder, and $m(e(\boldsymbol{x}))$ represents the output of the MLP.

3.5 Algorithm

To implement our method, we develop a robust Android malware detection algorithm (RAMDA). Its main procedure is described in Algorithm 1, and its notations are explained in Table 1. RAMDA has two stages, i.e., training and testing. In the phase of training, the FD-VAE and the MLP are trained with natural examples successively. In the phase of testing, natural or adversarial examples are fed to our model for classification. The FD-VAE and the MLP make their decisions independently. It is noted that the FD-VAE's decision is based on the reconstruction error L_1. Only when both the FD-VAE and the MLP declare that an input is benign (i.e., $d_1^i = 0$ and $d_2^i = 0$) can RAMDA output a benign outcome (i.e., $d^i = 0$).

Notations	Description
$\boldsymbol{\theta}_{vae}$	Parameters of the FD-VAE module
$\boldsymbol{\theta}_{mlp}$	Parameters of the MLP module
η_1	Learning rate of the FD-VAE
η_2	Learning rate of the MLP
$\lambda_1, \lambda_2, \lambda_3$	Weights of the losses of the FD-VAE
Ω_i, Θ_i	Natural example batches for FD-VAE training
ν	Decision threshold on reconstruction error
d_1^i, d_2^i, d^i	Decisions of FD-VAE, MLP and our model

Table 1: Notations used in RAMDA.

Now we explain why our model can reliably reject adversarial examples although it has not seen any adversarial examples. Recall that only the inputs resulting in small reconstruction errors can be accepted by our model. Equivalently, our method actually imposes a similarity constraint upon adversarial example. However, Android malware adversary examples are required to preserve malicious functionalities, and hence they are more similar to malicious examples instead of benign ones. As a consequence, our model will reject them with high probability. On the other hand, our model uses an MLP-based classifier to distinguish benign examples from others, which also helps to reject adversarial examples.

Algorithm 1: Robust Android Malware Detection

```
Stage I: Training
  • Initialize $\boldsymbol{\theta}_{vae}$, $\boldsymbol{\theta}_{mlp}$, $\eta_1$, $\eta_2$;
  • In each epoch:
    – In each batch:
      (1) Sample a batche of benign example
          $\boldsymbol{X}_B = \{x_b^1, x_b^2, \cdots, x_b^n\}$ and two batches of natural
          data, i.e., $\Omega_i$ and $\Theta_i$;
      (2) Update $\boldsymbol{\theta}_{vae}$ using $\boldsymbol{X}_B$, $\Omega_i$ and $\Theta_i$ by
          $\boldsymbol{\theta}_{vae} \leftarrow \boldsymbol{\theta}_{vae} - \eta_1 \nabla_{\theta_{vae}} (\lambda_1 L_1 + \lambda_2 L_2 + \lambda_3 L_3);$    (13)
  • In each epoch:
    – In each batch:
      (1) Sample a benign example $\boldsymbol{X}_B = \{x_b^1, x_b^2, \cdots, x_b^n\}$
          and malicious data $\boldsymbol{X}_M = \{x_m^1, x_m^2, \cdots, x_m^n\}$;
      (2) Update $\boldsymbol{\theta}_{mlp}$ using $\boldsymbol{X_B}$ and $\boldsymbol{X_M}$ by
          $\boldsymbol{\theta}_{mlp} \leftarrow \boldsymbol{\theta}_{mlp} - \eta_2 \nabla_{\theta_{mlp}} L_{mlp};$    (14)
Stage II: Testing
  • Foreach $\boldsymbol{x}^i$ in testing dataset:
  (1) Get the first decision $d_1^i$ from the FD-VAE by
      $d_1^i = \begin{cases} 0 & L_1(\boldsymbol{x}^i) < \nu \\ 1 & L_1(\boldsymbol{x}^i) \geq \nu \end{cases}$    (15)
  (2) Get the second decision $d_2^i$ from the MLP;
  (3) Output the final decision $d^i = d_1^i \vee d_2^i$;
```

4 EVALUATION

In this section, we conduct experiments to evaluate the proposed method in terms of accuracy and robustness[4].

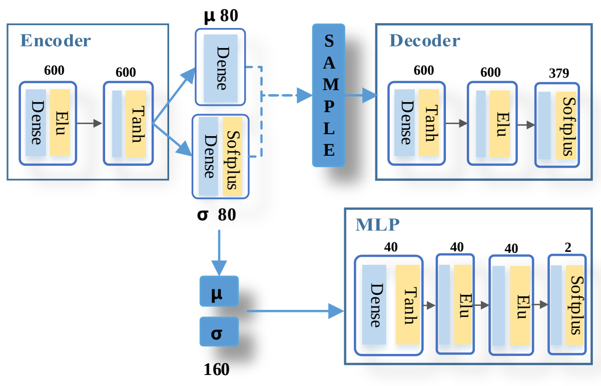


Figure 3: Model settings in our experiments.

4.1 Experiment Settings

Dataset: All the experiments are conducted over a dataset with 58,447 natural examples. The dataset contains 36,862 benign examples and 21,621 malicious examples. All these Apps come from

[4]More details can be found in https://github.com/YangXiuxuan/Robust-Android-Malware-Detection-Against-Adversarial-Example-Attacks

AndroZoo [21]. We use 1,500 benign examples and 1,500 malicious examples as the testing set and others as the training set.

For every App, the values of these features are vectorized as a 379-dimensions vector before being fed to our model. We choose 147 permissions, 126 intent actions, and 106 sensitive API calls as the features for malware detection.

Parameter Settings: The settings of our model are depicted in Fig. 3. This figure shows neuron number and the activate function of every layer in the network. In addition, we set $\lambda_1 = 10$, $\lambda_2 = 1$ and $\lambda_3 = 10$ in our experiments.

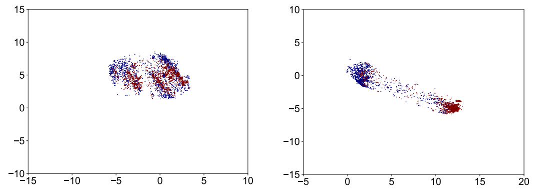
Baselines: For comparison, we introduce some adversarial example defenses in our experiments. As mentioned earlier, these defense methods cannot work independently in practice. To evaluate a defense method, one needs to combine it with a malware detector. Following this idea, we first implement an MLP as a malware detector. The MLP has 4 dense layers. Each of the first three layers has 128 neurons, and the last layer contains two neurons. Then we consider three state-of-the-art adversarial example defense strategies, i.e., adversarial retraining (AR) [10], robust training (RT) [15], and defense distillation (DD) [13, 28]. These defense strategies are popular in recent years [7, 8, 38]. We apply AR, RT and DD to the MLP, and obtain some robust malware detection models, which are comparable to our model. We apply DD directly to the MLP and obtain a robust malware detector, denoted by MLP_{DD}. However, both AR and RT cannot directly work with the MLP due to their reliance on the instances of adversarial examples. To get the instances of adversarial examples, we introduce five white-box attacks [11, 15]: dFGSM^k, rFGSM^k, BGA^k, BCA^k, APDNN. Accordingly, we obtain 10 robust malware detection models: $dFGSM^k_{RT}$, $rFGSM^k_{RT}$, BGA^k_{RT}, BCA^k_{RT}, $APDNN_{RT}$, $dFGSM^k_{AR}$, $rFGSM^k_{AR}$, BGA^k_{AR}, BCA^k_{AR}, and $APDNN_{AR}$. In our experiments, we will compare them with our method.

To be specific, $dFGSM^k$ and $rFGSM^k$ are the variants of $FGSM$. $dFGSM^k$ applies k rounds of FGSM to the original vector. Then it sets the elements of the vector greater than a threshold as 1 and the rest as 0. Different from $dFGSM^k$, $rFGSM^k$ replaces the fixed threshold with a random number. Finally, to keep the malicious functionalities, $dFGSM^k$ and $rFGSM^k$ will conduct an OR operation for the modified vector and the original input. Multi-step Bit Gradient Ascent (BGA^k) iteratively sets the bit of the input vector as 1 if its corresponding partial derivative of the loss is greater than or equal to the loss gradient's l2-norm divided by $\sqrt{m}$ (m is the length of the input vector). Similar to BGA^k, Bit Coordinate Ascent (BCA^k) only operates the bit which has the maximum partial derivative. What's more, BGA^k and BCA^k should do the OR operation used in $dFGSM^k$ and $rFGSM^k$ in every round of modification. APDNN iteratively updates the bit which has the maximal positive gradient until the algorithm either reaches the maximum number of allowed changes or successfully causes a misclassification.

4.2 Feature Disentangle

One main contribution of our method is to design a new loss function L_3 for the FD-VAE. In this subsection, we aim to verify that the new loss function can really disentangle the features of different classes. For this purpose, we compare our FD-VAE with the traditional VAE that does not introduce the new loss function. We use them to obtain the compressed representations of some natural examples. For visualization, the dimensions of the latent spaces of FD-VAE and VAE are set to 2. Therefore, the compressed representations learned by FD-VAE and VAE can be shown in a $2D$ coordinate system.

The experimental results are given in Fig. 4. In both figures, the blue points correspond to the compressed representations of benign examples, and the red ones represent the compressed representations of malicious examples of different classes. Obviously, our FD-VAE succeeds in disentangling the features, but the traditional VAE fails to do it. The comparisons validate that the new loss function L_3 helps to enlarge the inter-class distance and shorten the inner-class distance for the compressed representations. And this is one of the main reasons why our method is able to achieve high detection performance, as will be shown in the next subsection.



(a) VAE. (b) FD-VAE.

Figure 4: Compressed representations learned by VAE and FD-VAE.

4.3 Malware Detection

In this section, we evaluate our method in the scenarios where no adversarial examples exist. Considering that using AE defenses usually decreases the accuracy on natural examples, we decide to evaluate our method through comparing it with other robust malware detectors in the detection performance on natural examples. In our experiments, we choose a balanced dataset consisting of natural examples, and set the threshold v to 30. Our experimental results are given in Table 2. In this table, column 2 represents the false alarm rate (i.e., false positive rate), column 3 denotes the miss detection rate (i.e., false negative rate), and column 4 corresponds to the accuracy. It can be seen from this table that our model outperforms all the other models in terms of miss detection rate and accuracy. Besides, our model achieves an accuracy of 91.30%, which poses advantages over a majority of its competitors.

4.4 White-Box Adversarial Example Defense

In this section, we evaluate the robustness of our method under white-box adversarial example attacks. We consider five white-box attacks, including dFGSM^k, rFGSM^k, BGA^k, BCA^k, and APDNN. Noted that due to OR operation , our model cannot provide the gradient information from the final output. Following the method in [34], we use the gradient information of MLP instead.

The experimental results are provided in Table 3. This table compares our method with 11 robust malware detection models under five white-box attacks. Every element in this table indicates the ratio of adversarial examples being captured in every scenario.

	False Alarm	Miss Detection	Accuracy
$dFGSM^k_{RT}$	0.1665	0.0754	0.8699
$rFGSM^k_{RT}$	0.1064	0.2780	0.8180
BGA^k_{RT}	0.1573	0.0680	0.8790
BCA^k_{RT}	0.1244	0.0847	0.8927
$APDNN_{RT}$	0.2375	0.0860	0.8147
$dFGSM^k_{AR}$	0.1071	0.0774	0.9060
$rFGSM^k_{AR}$	0.1371	0.0894	0.8830
BGA^k_{AR}	**0.1034**	0.1034	0.8966
BCA^k_{AR}	0.2524	0.1134	0.7936
$APDNN_{AR}$	0.1742	0.0674	0.8700
MLP_{DD}	0.1058	0.1094	0.8926
Ours	0.1041	**0.0654**	**0.913**

Table 2: Detection Performance Evaluation.

The last row of the table shows the average performance on these five white-box attacks.

Obviously, our model achieves the average defense success rate of 90.94% when faced with different white-box attacks, which has an apparent advantage over its 11 competitors. It is noted that the defenses based on AR and RT often get worsen when faced the unseen-type adversarial examples. For illustration, consider an extreme example BGA^k_{RT}, who employs the adversarial examples generated by BGA^k during training. It can be seen from Table 3 that BGA^k_{RT} can capture about 90% of the adversarial examples generated by BGA^k. However, its performance almost reduces to zero when faced with the adversarial examples generated by dFGSM^k or rFGSM^k. We attribute this problem to the strong reliance of the defense on adversarial examples. Since our method does not require the instances or the knowledge of adversarial examples, it performs stably under various adversarial example attacks. Finally, although MLP_{DD} has relatively high performance under white-box attacks, this method can not resist the black-box attacks, which will be discussed in the next subsection.

4.5 Black-Box Adversarial Example Defense

Now we introduce two state-of-the-art black-box adversarial example attacks called MalGAN [14] and E-MalGAN [19]. These two methods attempt to generate adversarial examples through interacting with the target classifier, without relying on any white-box attacks. MalGAN first uses Generative Adversarial Networks (i.e., GAN) to generate adversarial examples in Android application. E-MalGAN is an improvement of MalGAN and can generate adversarial examples that are more difficult to detect by narrowing the difference between adversarial examples and benign examples.

To show that our model can well resist the black-box attacks, we evaluate it under the attack of MalGAN and E-MalGAN. These two methods need to interact with the target classifier for multiple rounds before generating effective adversarial examples. Hence, we use the metric, i.e., the ratio of adversarial examples being identified during interaction (R-AEI), in our evaluation.

We first consider the attack method MalGAN. The experimental results are given in Fig. 5 and Fig. 6, whose vertical axes denote the values of R-AEI. We can draw some conclusions from these two figures. First, when no defense method is adopted, the R-AEI gradually reduces to zero, as reflected by the orange dotted line in Fig. 5. Second, MLP_{DD} cannot resist the attacks of MalGAN, but our model and the others are robust to MalGAN. Furthermore, it should be pointed out that our method is slightly better than its competitors. This can be verified by the plot corresponding to our model, which remains flat and approaches the upper bound 100% after some iterations.

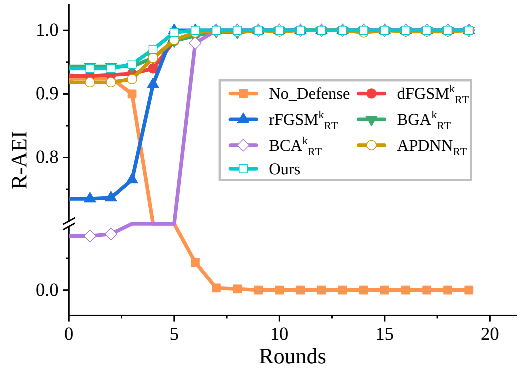


Figure 5: Resisting MalGAN with No_Defense and etc.

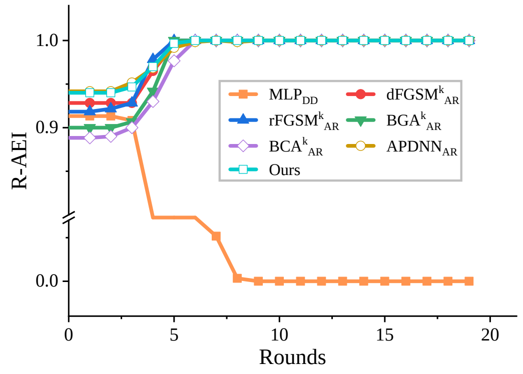


Figure 6: Resisting MalGAN with and MLP_{DD} etc.

Finally, we consider a more powerful attack method, i.e., E-MalGAN. Different from MalGAN, E-MalGAN is able to evade the detection of adversarial examples. Therefore, it is challenging to resist E-MalGAN. As shown in Fig. 7 and Fig. 8, all the competitors of our model are defeated by E-MalGAN, since their R-AEI values get close to zero after sufficient iterations. Contrarily, our model captures almost all the adversarial examples generated by E-MalGAN, as depicted in Fig. 7 and Fig. 8. The reasons for the above phenomena are explained as follows. The main competitors of our model rely heavily on the instances of adversarial examples [5]. Hence they can identify the adversarial examples similar to those

[5]It is noted that the model MLP_{DD} does not require the instances of adversarial examples. This model is vulnerable to black-box attacks due to the limitation of defense distillation.

	$dFGSM^k_{RT}$	$rFGSM^k_{RT}$	BGA^k_{RT}	BCA^k_{RT}	$APDNN_{RT}$	$dFGSM^k_{AR}$	$rFGSM^k_{AR}$	BGA^k_{AR}	BCA^k_{AR}	$APDNN_{AR}$	MLP_{DD}	Ours
dFGSM^k	0.9073	0.7046	0.0000	0.0793	0.4253	0.8773	0.7760	0.4113	0.5740	0.0466	0.8906	**0.9280**
rFGSM^k	0.5713	0.9640	0.0000	0.0806	0.1026	0.7306	0.9286	0.3520	0.6286	0.0200	0.8906	**1.0000**
BGA^k	0.2373	0.8786	0.8933	0.8540	0.8566	0.7580	0.9946	0.9820	0.9766	0.6140	0.8906	**1.0000**
BCA^k	0.5960	0.7240	0.9320	0.8813	0.9106	0.6580	0.8633	0.5020	0.9746	0.9293	0.8640	**0.9806**
APDNN	0.0700	0.2540	0.5326	0.0820	0.7960	0.0220	0.5646	0.0640	0.5706	0.7993	**0.8320**	0.6386
AVG	0.4763	0.7050	0.4715	0.3954	0.6182	0.6091	0.8254	0.4622	0.7449	0.4818	0.8735	**0.9094**

Table 3: Robustness Evaluation (v = 30).

that have been seen. Unfortunately, once E-MalGAN successfully generates the adversarial examples different from those seen by the targeted classifier, it will mislead the classifier. However, our model distinguishes benign examples from others based on the difference between them. Since the difference cannot be completely erased, our model is able to capture the adversarial examples. What's more, our method performs better on E-MalGAN than on MalGAN. In fact, E-MalGAN generates much smaller perturbations than MalGAN does to guarantee high similarity. However, the pursuit of smaller perturbations is a two-edged sword. On one hand, E-MalGAN can generate the adversarial exampls similar to natural examples. On the other hand, the training instability risk of E-MalGAN is also raised due to its complicated architecture, especially when faced with a challenging adversarial exampls generation task. Different from the existing defense mechanisms, our method pushes the whole (instead of a part of) decision boundary to the ideal boundary using feature disentangle. Therefore, it becomes more challenging for E-MalGAN to generate adversarial exampls, and thus E-MalGAN becomes very unstable and collapses soon.

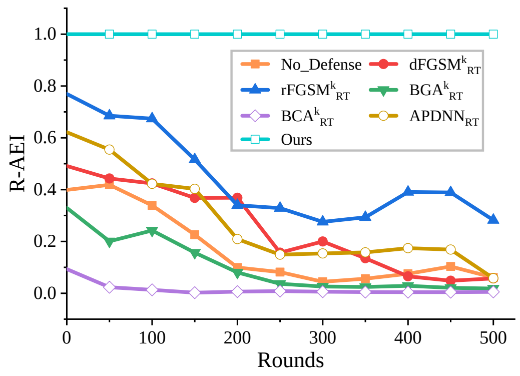


Figure 7: Resisting E-MalGAN with No_Defense and etc.

4.6 Impacts of Threshold v

In practice, the threshold v has a great influence on the result of the experiment. So how to choose a appropriate v is important in our method. We conduct a mass of experiments to analyze the impacts of v. In these experiments, we continue adjusting v and fix the other parameters.

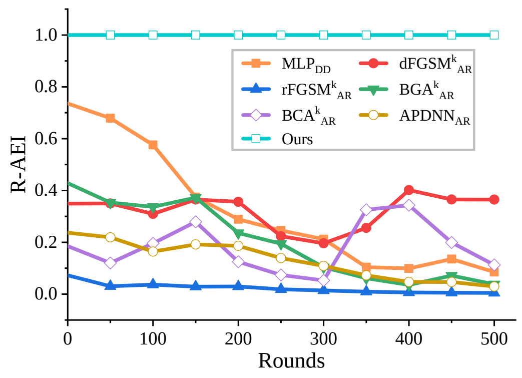


Figure 8: Resisting E-MalGAN with MLP_{DD} and etc.

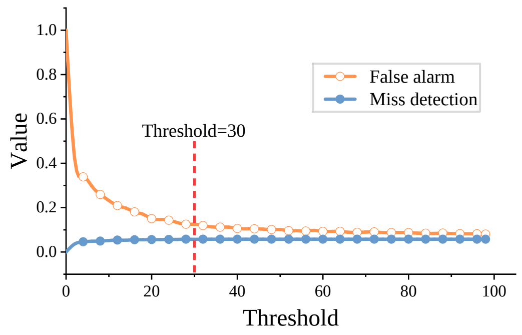


Figure 9: Impacts of threshold v on detection performance.

We first consider how this threshold impacts the distinguishment between malicious Apps and begin Apps. The results are given in Fig. 9, in which the horizontal axis reflects the values of v. The orange line represents the false alarm rate of benign examples, and the blue line reflects the miss detection rate of malicious examples. Not surprisingly, false alarm rate decreases with the increase of v. Intuitively, when v is too large, miss detection of malicious examples will frequently occur. However, the miss detection rate remains stable when v keeps rising, as shown in Fig. 9. We explain this phenomenon as follows. Our method fuses the decisions of the MLP and the FD-VAE using the OR rule. When v becomes too large, the FD-VAE will have lots of miss detections, but the MLP will not generate more miss detections. Due to decision fusion, our model

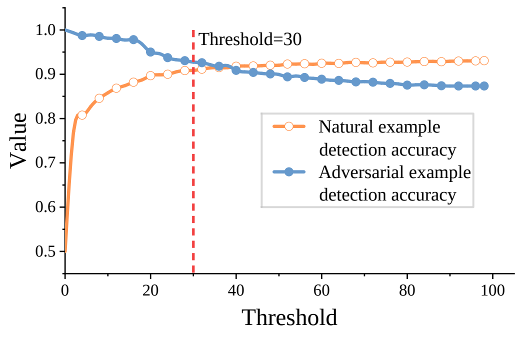


Figure 10: Impacts of threshold v on normal data vs. adversarial data.

can depresses miss detections and keep its accuracy stable. This also indicates that our method is not very sensitive to v, hence easy to deploy in practice.

Then we consider how this threshold impacts the distinguishment between adversarial exampls and natural examples. We employ BGA^k to generate adversarial examples to evaluate our method. The experimental results are given in Fig. 10. The orange line represents the accuracy on natural examples, and the blue one on adversarial examples. Similarly, we keep adjusting v but fix the other parameters in the experiments. Our experiments show the goals of achieving high detection accuracy on natural and adversarial examples are conflicting in practice.

Finally, we select a desired threshold through evaluating all candidate thresholds in both malware detection and adversarial example detection. The threshold 30 is chosen by us, since it achieves satisfactory miss detection, false alarm, and adversarial example detection acccuracy.

5 RELATED WORK

In this section we intruduce the methods of adversarial example attack and defense in Android Malware Detection.

5.1 Adversarial Example Attack

Chen *et al.* [3] propose a method to inject adversarial examples into the training set of the victim model, in order to decrease the dection accuracy. However, it's difficult to gain access to the training set. Huang *et al.* [15] propose to use saddle-point optimization formulations for discrete (e.g., binary) domain to generate adversarial examples for malware detection. Considering Android malware's semantic feature is graph data, Chen *et al.* [5] modify two famous attack methods (i.e., C&W, JSAM) developed for image classification to generate adversarial examples of Android malware and evade being detected by the current models. Grosse *et al.* [11] also expand existing adversarial example crafting algorithms to construct a highly-effective attack against malware detection models. In [14], Hu *et al.* utilize generative adversarial network to generate adversarial examples in black-box mode for malware detection. Li *et al.* [19] improve the work of [14], and design a multi-GAN to generate the adversarial examples more similar to natural examples.

5.2 Adversarial Example Defense

Papernot *et al.* [28] propose a generic framework and utilize defense distillation to reduce the sensitivity of deep neural networks (DNNs) to small perturbation, hence alleviating the negative effect of adversarial examples. However, defense distillation achieves limited success when facing adversarial examples generated by black-box attack methods due to the transferability of adversarial examples. In [10], Goodfellow, *et al.* propose adversarial retraining to defend adversarial example attack. It retrains a classifier with both the original dataset and the labeled adversarial malware examples. However, the effect of adversarial retraining depends heavily on the similarity between the adversarial examples used in training and testing. Huang *et al.* [15] propose robust training and aim at developing a training procedure that helps improving model robustness. They introduce a certain method to generate adversarial examples and use the latter in training. However, robust training may be helpless for resisting unseen-type adversarial examples. Xu *et al.* [34] utilize feature squeezing to transform the input examples to mitigate the negative effects of adversarial examples on the classifier. They are difficult to be applied to Android malware due to its discrete input domain.

6 CONCLUSIONS

In this paper, we design a robust Android malware detection method inherently resistant to adversarial examples. Our method does not require the instances or the knowledge of adversarial examples for model training, and hence it is practical to deploy and more robust to unseen-type adversarial examples. Extensive experiments show that our model is able to resist 5 white-box and 2 black-box attacks, and significantly outperforms 11 robust Android malware detection models.

ACKNOWLEDGMENTS

This work was supported by HK RGC Project (PolyU 152239/16E), HKPolyU Research Grant (ZVQ8) and National Natural Science Foundation of China (Grant No. 61571205).

REFERENCES

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proc. Network & Distributed System Security Symposium.*

[2] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating Sentences from a Continuous Space. In *Proc. Conference on Computational Natural Language Learning.* 10–21.

[3] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. 2018. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *computers & security* 73 (2018), 326–344.

[4] Wenxiao Chen, Haowen Xu, Zeyan Li, Dan Peiy, Jie Chen, Honglin Qiao, Yang Feng, and Zhaogang Wang. 2019. Unsupervised Anomaly Detection for Intricate KPIs via Adversarial Training of VAE. In *Proc. International Conference on Computer Communications.* 1891–1899.

[5] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. 2019. Android HIV: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security* 15 (2019), 987–1001.

[6] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. 2020. Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection. *IEEE Transactions on Information Forensics and Security* 15 (2020), 987–1001.

[7] Yizheng Chen, Shiqi Wang, Dongdong She, and Suman Jana. 2020. On Training Robust {PDF} Malware Classifiers. In *29th {USENIX} Security Symposium ({USENIX} Security 20).* 2343–2360.

[8] Weiyu Cui, Xiaorui Li, Jiawei Huang, Wenyi Wang, Shuai Wang, and Jianwen Chen. 2020. Substitute Model Generation for Black-Box Adversarial Attack Based on Knowledge Distillation. In *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 648–652.
[9] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. 2013. Large-scale malware classification using random projections and neural networks. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*. 3422–3426.
[10] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *Proc. International Conference on Learning Representations*.
[11] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, M Backes, and Patrick Mcdaniel. 2017. Adversarial Examples for Malware Detection. In *Proc. European Symposium on Research in Computer Security*. 62–79.
[12] Irina Higgins, Loic Matthey, Arka Pal, Christopher P Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. 2017. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *Proc. International Conference on Computer Communications*.
[13] Geoffrey E Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. In *Proc. Neural Information Processing Systems Deep Learning Workshop*.
[14] Weiwei Hu and Ying Tan. 2017. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *arXiv: 1702.05983* (2017).
[15] Alex Huang, Abdullah Aldujaili, Erik Hemberg, and Unamay Oreilly. 2018. Adversarial Deep Learning for Robust Detection of Binary Encoded Malware. In *Proc. IEEE Security and Privacy Workshops*.
[16] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-box Adversarial Attacks with Limited Queries and Information.. In *Proc. International Conference on Machine Learning*. 2137–2146.
[17] Taeguen Kim, Boojoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. 2019. A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. *IEEE Transactions on Information Forensics and Security* 14, 3 (2019), 773–788.
[18] Diederik P Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *Proc. International Conference on Learning Representationse*.
[19] Heng Li, ShiYao Zhou, Wei Yuan, and Henry Leung. 2019. Adversarial-Example Attacks towards Android Malware Detection System. *IEEE Systems Journal* (2019).
[20] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisaan, and Heng Ye. 2018. Significant Permission Identification for Machine-Learning-Based Android Malware Detection. *IEEE Transactions on Industrial Informatics* 14, 7 (2018), 3216–3225.
[21] Li Li, Jun Gao, Médéric Hurier, Pingfan Kong, Tegawendé F Bissyandé, Alexandre Bartel, Jacques Klein, and Yves Le Traon. 2017. AndroZoo++: Collecting Millions of Android Apps and Their Metadata for the Research Community. *arXiv:1709.05281* (2017).
[22] Yongfeng Li, Tong Shen, Xin Sun, Xuerui Pan, and Bing Mao. 2015. Detection, classification and characterization of android malware using API data dependency. In *Proc. International Conference on Security and Privacy in Communication Systems*. 23–40.
[23] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection.. In *Proc. The Network and Distributed System Security Symposium*.
[24] Aravind Machiry, Nilo Redini, Eric Gustafson, Yanick Fratantonio, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. 2018. Using loops for malware classification resilient to feature-unaware perturbations. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 112–123.
[25] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon J Ross, and Gianluca Stringhini. 2019. MAMADROID: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proc. ACM Transactions on Privacy and Security*.
[26] John San Miguel, Megan Kline, Roger A Hallman, Scott M Slayback, Alexis Rogers, and Stefanie S F Chang. 2018. Aggregated Machine Learning on Indicators of Compromise in Android Devices. In *Proc. Conference on Computer and Communications Security*. 2279–2281.
[27] Rene Millman. 2015. Updated: 97% of malicious mobile malware targets Android. *SC Media UK News* (2015).
[28] Nicolas Papernot, Patrick Mcdaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *Proc. IEEE Symposium on Security and Privacy*. 582–597.
[29] Guillermo Suareztangil, Santanu Kumar Dash, Mansour Ahmadi, Johannes Kinder, Giorgio Giacinto, and Lorenzo Cavallaro. 2017. DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware. In *Proc. Conference on Data and Application Security and Privacy*. 309–320.
[30] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proc. International Conference on Learning Representationse*.
[31] Shanshan Wang, Qiben Yan, Zhenxiang Chen, Bo Yang, Chuan Zhao, and Mauro Conti. 2017. Detecting Android Malware Leveraging Text Semantics of Network Flows. *IEEE Transactions on Information Forensics and Security* 13, 5 (2017), 1096–1109.
[32] Shengqu Xi, Shao Yang, Xusheng Xiao, Yuan Yao, Yayuan Xiong, Fengyuan Xu, Haoyu Wang, Peng Gao, Zhuotao Liu, Feng Xu, et al. 2019. DeepIntent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2421–2436.
[33] Ke Xu, Yingjiu Li, Robert H Deng, and Kai Chen. 2018. DeepRefiner: Multi-layer Android Malware Detection System Applying Deep Neural Networks. In *Proc. IEEE European Symposium on Security and Privacy*. 473–487.
[34] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks.. In *Proc. The Network and Distributed System Security Symposium*.
[35] Yanfang Ye, Tao Li, Donald Adjeroh, and S S Iyengar. 2017. A Survey on Malware Detection Using Data Mining Techniques. *Comput. Surveys* 50, 3 (2017), 41.
[36] W. Yuan, Y. Jiang, H. Li, and M. Cai. 2019. A Lightweight On-Device Detection Method for Android Malware. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2019), 1–12.
[37] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2015. Droid-Sec: deep learning in android malware detection. *Proc. ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 371–372.
[38] Haizhong Zheng, Ziqi Zhang, Juncheng Gu, Honglak Lee, and Atul Prakash. 2020. Efficient adversarial training with transferable adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1181–1190.