A Comparative Study of Adversarial Attacks to Malware Detectors Based on Deep Learning

Corrado Aaron Visaggio, Fiammetta Marulli, Sonia Laudanna, Benedetta La Zazzera, and Antonio Pirozzi

Abstract Machine learning is widely used for detecting and classifying malware. Unfortunately, machine learning is vulnerable to adversarial attacks. In this chapter, we investigate how generative adversarial approaches could affect the performance of a detection system based on machine learning. In our evaluation, we trained several neural networks for malware detection on the EMBER [3] dataset and then we built ten parallel GANs based on convolutional layer architecture (CNNs) for the generation of adversarial examples with a gradient-based method. We then evaluated the performance of our GANs, in a *gray-box* scenario, by computing the evasion rate reached by the adversarial generated samples. Our findings suggest that machine- and deep-learning-based malware detectors could be fooled by adversarial malicious samples with an evasion rate of around 99% providing further attack opportunities.

1 Introduction

Several studies have investigated the effectiveness [1, 7, 8, 17, 19, 48] and drawbacks [4, 40] of machine (and recently also deep) learning in detecting and classifying malware. Independently from the inherent limitations of malware detectors based on machine learning, the generative adversarial networks (GANs, in the remainder of the chapter) become a menace to the effectiveness of these tools.

A GAN is a tool that produces adversarial samples by using the adversarial machine learning [26]: this is a technique that leverages machine learning for fooling classifiers trained with a machine learning algorithm, leading them to wrongly classify some samples.

C. A. Visaggio · S. Laudanna · B. La Zazzera · A. Pirozzi
University of Sannio, Benevento, Italy
e-mail: visaggio@unisannio.it

F. Marulli (✉) · S. Laudanna · B. La Zazzera · A. Pirozzi
Department of Maths and Physics, University of Campania "L.Vanvitelli", Caserta, Italy
e-mail: fiammetta.marulli@unicampania.it

M. Stamp et al. (eds.), *Malware Analysis Using Artificial Intelligence and Deep Learning*, https://doi.org/10.1007/978-3-030-62582-5_19

Adversarial machine learning has been applied with a certain success especially to the field of image recognition with some surprising results [21], but also to speech recognition [2] and biometric recognition [11].

For understanding how powerful maybe this technique, we could mention the case of image recognition. The adversarial sample is an image that has been tampered within a way that cannot be distinguished by a bare eye, but that misleads the classifier. The result is that the image is not recognized at all or, even worst, is classified as a completely different image.

An exemplar case is the automatic recognition of street signs: a street sign is decoded as another street sign. Alike the fields where GANs have been experimented, they could be successfully used for generating samples of malware that are recognized by malware detectors based on machine learning as goodware.

The research community is now investigating the application of GANs to malware analysis, and so far the main result consists of some models of GANs for producing adversarial malware samples.

Our purpose is to investigate how and how much GANs are able to degrade the performance of malware detectors based on machine learning. We trained a set of classifiers using different combinations of features, obtaining a wide spectrum of performances. Thus, we built different models of GANs, observing the degradation of each detector.

This work does not help to identify how to make stronger a detector against an adversarial attack but provides data for quantifying the potential effects of a GAN on a malware detector based on machine learning.

In this chapter, we provide a brief overview of the current state of the art and some open issues related to the vulnerabilities of deep learning models adopted in designing malware recognition systems. More precisely, we focused on the weak points of these approaches when attacked by adversarial examples that are proving to be increasingly sophisticated and effective in misleading defense systems.

We provide further evidence by discussing a case study that shows how adversarial examples and generative adversarial approaches, by the means of generative adversarial neural networks (*GANs*), can degrade the detection performance of a deep learning feature-based malware detector, finally highlighting that certain features may prove to be more sensitive than others.

The chapter is organized as follows: the next section provides the background of adversarial machine learning and the most significant applications, while Sect. 3 compares the related literature. Section 4 shows the research questions we posed and the design of the case study. Section 5 discusses the obtained results, and, finally, conclusions are drawn in the last section.

2 The Deep Learning Models Adopted in Malware Detection

Machine learning (ML) and deep learning (DL) have been successfully employed for detecting malicious objects, e.g., executable files.

New malware programs appear each year in increasing amounts and hence malware detection based on signature matching is increasingly becoming an impractical approach. Machine and deep learning promise to provide valid countermeasures against modern malware because of their capability to potentially detect malware applications without specific signatures of their behavior or data.

Generally, ML-based malware detectors work on the extraction of the malware (and benign programs) features and static and/or dynamic analysis can be performed. Such systems learn from examples for creating models by which they will be able to discriminate whether a given program is a malware or not. These models are then used to estimate the likelihood that a given program is malware.

One of the bottlenecks exhibited by ML-based malware detection is represented by the high time required to learn when the number or size of features is wide or the number of sample programs is large. Although reducing the number of features could shorten the learning time, the accuracy in the detection task likely decreases. So, finding an acceptable trade-off among the detection accuracy, short learning times, and limiting the size of data, obtainable by selecting a convenient combination of sensitive features, is far from being a trivial problem.

A very accurate review of recent findings of adversarial examples in deep neural networks and a deep investigation of existing methods for generating adversarial examples is provided in [50].

2.1 *The Deep Learning Models in a Nutshell*

The essential background about techniques and enabling architectures of deep learning is provided in the following.

Deep learning is a kind of machine learning that makes computers to learn from experience and knowledge without explicit programming and extracts useful patterns from raw data.

Conventional machine learning algorithms exhibit some limitations since it is difficult to extract well-represented features because of the curse of dimensionality, computational bottleneck, and strong requirements of the domain and expert knowledge. Deep neural networks represent a particular kind of machine learning algorithm, leveraging several "deep" layers of networks. Furthermore, deep learning solves the problem of the representation by building multiple simple features to model a complex concept. The more the number of available training data grows, the more powerful the deep learning classifier becomes. Deep learning models solve complicated problems by complex and large models, with the help of hardware acceleration in computational time.

Traditionally, researchers build a single deep learning model using the entire dataset. However, the single deep learning model may not handle the increasing complex malware data distributions effectively since different sample subspaces representing a group of similar malware may have unique data distribution [52].

Since the performance of deep learning models keeps improving with the increasing number of samples [49], researchers build a single deep learning model using an entire data to understand the relationship between data features extracted from malware and the target [9, 27, 39, 45].

These deep learning models mainly use three types of neural network architectures:

- Convolutional neural network (CNN);
- Recurrent neural network (RNN); and
- Fully connected feedforward neural network (FC).

There are two major disadvantages in building a single deep learning model that uses a blended dataset:

- Complex data distribution;
- Scalability.

Each type of malware has unique and different characteristics, proliferation methods, and data distributions [35, 49].

Consequently, merging different types of malware into one dataset results in a very complex overall data distribution. Furthermore, the diversity and sophistication of the merged dataset continue to grow rapidly due to the large number of new malware variants that are created each year [49]. As a result, it is very challenging for a single deep learning model to understand this complex data distribution.

Additionally, the single CNN model treats malware as the image while the single RNN model considers the behavior as the sequence of events. Both models only analyze the data distribution from only one perspective. In the case of malware, the analysis of data distribution in different sample subspaces from multiple angles is preferred in order to combine the knowledge and strength of these single models effectively.

Second, building a single deep learning model for malware detection lacks scalability to train on increasingly large malware datasets. Training deep learning models on very large datasets is a computationally expensive process [20]. Since the number of new malware samples has exponentially increased through time [9, 31], building a single deep learning model requires longer computation time. This slow training process makes difficult to search and rebuild the learning model rapidly in order to adapt to the fast changing malware landscape and respond to the new techniques adopted by the malware writers. An alternative to using a single deep learning model to build malware detection systems (MDSs) is the development of ensemble-based deep learning models. Multiple deep learning models in the ensemble can work together to enhance the performance of MDSs. Researchers have developed the ensemble-based deep learning models, where each model is constructed on the whole blended dataset.

2.1.1 The Most Popular Deep Neural Network Architectures

A neural network layer includes a set of perceptrons (artificial neurons), and each one is able to map a set of inputs to output values by evaluating a simple activation function. The function of a neural network is formed in a chain $f(x) = f^{(k)}(\ldots f^{(2)}(f^{(1)}(x)))$, where $f^{(i)}$ is the function of the ith layer of the network, with $i = [1; 2; \ldots; k]$.

Convolutional neural networks (CNNs) and **recurrent neural networks** (RNNs) are the two most popular and adopted neural network architectures in recent times.

CNNs deploy convolution operations on hidden layers for weight sharing and parameter reduction. CNNs can extract local information from grid-like input data. CNNs have shown incredible successes in computer vision tasks, such as image classification [24], object detection [44] and semantic segmentation [14].

RNNs are neural networks adopted for processing sequential input data with variable length. RNNs produce an output at each step. The hidden neuron at each step is calculated based on input data and hidden neurons at a previous step. To avoid vanishing/exploding gradients of RNNs in long-term dependency, long short-term memory (LSTM) and gated recurrent unit (GRU) with controllable gates are widely used in practical applications.

Generative adversarial networks (GANs) are a type of generative model introduced by [22], where adversarial examples can be exploited to improve the representation of deep learning and perform unsupervised learning. A generative network (generator) creates artificial samples while a discriminative network (discriminator) acts as an adversary to determine if the generated samples are genuine or fake. This kind of network architectures are typically referred as generative adversarial network (GAN) and solve an optimization function described by

$$\min_G \max_D V(D,\ G) = \mathbb{E}_{x\sim P_r}[\log D(x)] + \mathbb{E}_{z\sim P_z}[\log D(G(z))],$$

where D and G denote the discriminator and generator, and P_r and P_z are, respectively, the distribution of input data and noise. In this competition, GAN is able to generate raw data samples that look close to the real data.

Due to the wide use and breakthrough successes, ML- and DL-based detection systems have become a major target for attacks, where adversaries are usually applied to evaluate the attack methods. Unfortunately, both ML and DL approaches to malware detection can be fooled by adversarial examples that consist of small changes to the input data causing misclassification at testing time.

3 Adversarial Attacks Against Deep Learning-Based Malware Detection System

In this section, we explore the adversarial attack techniques on ML models that have been applied to intrusion and malware attack scenarios.

Several techniques have been proposed to create adversarial examples. Most approaches suggest minimizing the distance between the adversarial example and the instance to be manipulated in order to cause the ML classifier to misclassify the testing dataset with high confidence.

Some methods require access to the gradients of the model, which typically introduce perturbations optimized for certain distance metrics between the original and perturbed samples: this kind of attack is called *white-box attack*. Other methods only require access to the prediction function, which makes these methods model-agnostic: this kind of attacks are called *black-box attack*.

A simple indiscriminate approach is gradient ascent during the training of ML model. Szegedy et al. [47] proposed a first gradient method to generate adversarial examples applied to the imaging field, using *box-constrained limited-memory Broyden-Fletc.her-Goldfarb-Shanno* (L-BFGS) optimization, an optimization algorithm that works with gradients. The adversarial examples were generated by minimizing the following function:

$$\begin{aligned}&Minimize\ \|r\|_2\ subject\ to:\\&1.\ f(x+r)=l\\&2.\ x+r\in[0,1]^m\ ,\end{aligned}$$

where x is an image represented as a vector of pixels, r represents the perturbations to be made on the pixels to create an adversarial image, l is the target label (the desired outcome class), and the parameter c is used to balance the distance between images and the distance between predictions.

Goodfellow et al. [22] proposed a simple and fast gradient-based method called *fast gradient sign method* (FGSM), using the gradient of the underlying model to find adversarial examples and the original image x is manipulated by adding or subtracting a small error ϵ to each pixel:

$$\eta\ = \epsilon * \mathrm{sign}\left(\nabla_x\ J\left(x, y\right)\right).$$

Here, η is the perturbed sample, ϵ is a hyperparameter controlling the amount of perturbation added to each feature (pixel), $\nabla_x\ J$ is the gradient of the models loss function with respect to the original input pixel vector x and y the target label (the true label vector for x). The sign of the gradient is positive if an increase in pixel intensity increases the error the model makes and negative if a decrease in pixel intensity increases the error. This approach requires many pixels to be changed, for this reason, Su et al. [46] demonstrated that it is actually possible to deceive image classifiers by changing a single pixel (the RGB value). The one-pixel attack uses differential evolution to find out which pixel is to be changed and how.

Brown at. al [12] proposed how to create image patches that can be added to a scene, and force a classifier into reporting a class of the attacker's choosing. This method differs from the methods aforementioned since the adversarial image isn't close to the original image but it is removed and a part of the image is replaced with a patch that can take on any shape.

Carlini and Wagner [13] modified the objective function and used a different optimizer compared with the L-BFGS attack described in [47]. Instead of using the same loss function as in L-BFGS, they solved the following box-constraint optimization problem to find an adversarial perturbation δ, making the problem more efficient to solve. CW finds the adversarial instance by finding the smallest noise $\delta \in R^{nxn}$ added to an image x that will change the classification to a class t and uses the L_2 norm (i.e., Euclidean distance) to quantify the difference between the adversarial and the original examples. Formally:

$$\text{minimize } \|\delta\|_p \text{ subject to } C(x + \delta) = t,\ x + \delta \in [0, 1]^n,$$

where $C(x)$ is the class label returned with an image x.

While successful, gradient-based methods work only under "white-box" settings. Papernot et al. [38] showed a type of zero-knowledge attack (black-box attack) to create adversarial examples without internal model information and without access to the training data. This technique, called *Jacobian-based saliency map attack* (JSMA), unlike the previous method, proposed to use the gradient of loss with each class label with respect to every component of the input, i.e., Jacobian matrix to extract the sensitivity direction. Then a saliency map is used to select the dimension which produces the maximum error using the following equation:

$$s_t = \frac{\partial t}{\partial x_i};\ s_o = \sum_{j \neq t} \frac{\partial j}{\partial x_i};\ s(x_i) = s_t\,|s_o| \cdot (s_t < 0) \cdot (s_o > 0).$$

In the previous formula , s_t represents the Jacobian of target class t and s_o represents the sum of Jacobian values of all non-target class. Changing the selected pixel will significantly increase the probability of the model labeling the image as the target class. The purpose of JSMA attack is to optimize the L_o distance metric (the amount of perturbed features).

Moosavi-Dezfooli et al. [33] proposed an algorithm, *DeepFool*, to compute adversarial examples using an iterative linearization of the classifier to generate minimal perturbations that are sufficient to change the classification labels. Starting with a binary classification problem, this method creates an adversarial example computing the Euclidean distance between perturbed samples and original samples in an iterative manner until $sign(f(x)) \neq sign(f(x + r))$ where r is the minimum perturbation required.

Zeroth-order optimization attack (ZOO) was proposed by Chen et al. [15] and consists of approximating the full gradient via a random gradient estimate using the difference between the predicted probability of the target model and the desired class label. Precisely, the method uses zeroth-order stochastic coordinate descent to optimize the malicious sample by adding perturbations to each feature and querying the classifier to estimate the gradient and Hessian of the different features. In this scenario, solving the optimization problem is computationally expensive and the authors proposed a ZOO-Adam algorithm to find the optimal perturbations for the target sample.

Most of the attacks presented have been initially tested on image domains by introducing perturbations to existing images but they can equally be applied to other types of data, such as datasets with a limited number of features since these attacks are not data-type dependent. In a cybersecurity scenario, a malicious user could access any type of data used by a classifier and produce adversarial examples.

4 Generative Adversarial Attacks Against Malware Detection Systems

In this section, we examine existing generative adversarial algorithms used to attack malware detectors.

Generative adversarial algorithms have been mainly applied to image recognition, where generative adversarial networks (GANs) were used to generate images that were indistinguishable from real ones. In the process of image generation, for example, the GAN network modifies some features like pixels, while a human eye does not perceive the difference from an original one. Using GAN to create a binary file poses more difficulties than an image, because changing a bit in a binary may corrupt the file. For this reason, generating executable files with a GAN could be challenging.

The main difference between image and malware is that images are continuous while malware features are binary. Changing byte arbitrarily could break semantics and syntax of portable executable (PE) so we are limited in the types of modification that can be done without breaking the malware functionality. For this reason, different approaches have been proposed in the literature such as adding padding bytes (adversarial noise) at the end of a file beyond PE boundaries [30]. Another approach consists of injecting the adversarial noise in an unused PE region that is not mapped in memory [32]. Most works in literature simply ignore this problem. In order to overcome this limitation, attackers must have a white-box model in which the type of the ML algorithm used and the features to be used are known. One of the first demonstrations of an adversarial creation of a PE is the work [25]; in this paper, authors adopt a gray-box model in which they only know the set of used features based on API calls but do not know the ML model used by the classifier. In MalGAN, authors generate adversarial examples by adding some irrelevant features to the binary files because removing features may crack the executable or its intended behavior. The adversarial generated example is expressed by the following formula:

$$m' = m|o',$$

where m is the initial feature vector, each element of m corresponds to the presence/absence of a particular feature in a malware, and then this input vector is fed into a multi-layer feedforward neural network with weights

$$\theta_g.$$

The output layer of this network has M neurons and has a sigmoid as an activation function which is continuous in the range (0,1) as the last layer. The output of this network is denoted as o.

Since malware features are binary, the output in the continuous space must be transformed into the binary space with a transformation called *binarization*. This procedure generates a new binary vector *o'*. Then the resulting *m'* is a binary vector obtained by the initial m vector through an OR bit-wise operation with the o' binary vector.

The non-zero element of the binary vector *o'* acts as an irrelevant feature to be added to the original malware. While MalGAN and the detector use the same API as features quantity and this could affect the performance of avoidance, in [28] authors add some noise to malware, extracting features (API list) from clean malware and input them to the generator.

In another work [30], the authors present a gradient-based attack to generate adversarial malware binaries but their limit is the manipulation to the padding bytes appended at the end of the PE to guarantee that the malware integrity is preserved. With this approach, they reach an evasion rate of 60% against *raff2017malware* used as a classifier. GANs are also used to generate a malicious document. In [51], the authors propose a method based on *Wasserstein generative adversarial network* (WGAN) to generate a malicious PDF with an evasion rate of 100% as stated. A malicious PDF is a document that embeds and executes malicious code. In this work, the authors generate adversarial examples by modifying 68 features extracted from various attributes: size, metadata, and structural attributes.

5 Case Study

We carried out a case study to examine the effectiveness of adversarial models against malware detectors based on deep learning. To this aim, we considered a cooperative system of generative adversarial networks, where multiple GANs (couples of generators and discriminators) run in parallel for supporting a multistage black-box attack.

Under the realistic hypothesis that an attacker knows very little about the system he wants to attack (the case of a black-box attack), the attacker could set up some sort of brute-force attack by deploying a pool of specific generators built for interacting with a corresponding pool of specific targets (discriminators).

The attack strategy envisages multiple stages (steps). As first, the only knowledge owned by the attacker consists of knowing that the target victim could behave according to a ML or DL model and the kind of inputs it could accept, so the attacker trains several generators working over different groups of features (possibly, he could try over all the sensible combinations) for refining the generation of artificial adversarial samples.

This training stage is performed without effectively interacting yet with the real target. Discriminators play the role of the potential victims, as substitutes of real victim systems. In the middle of attacking time, the attacker will start a smooth interaction with its victim, this time represented by a black box. By carefully analyzing responses from the black box, it is able to figure out what features are used by the malware detector black box. Adversarial test cases are produced by exploiting all the trained generators in the attacker's wallet.

Most of these samples will be harmless since they will not act on the right set of features but we can suppose that almost one of these generators will be able to generate samples that will produce some effects. By this way, the attacker will gain knowledge about its adversary and can implement a gray-box attack, basing on the features set its victim works over.

At the real attack time, the attacker will exploit only the right generator and proceed to attack and refine its generation model until its target is reached out. The case study we propose should not be regarded as exhaustive but it can be regarded as proof of the concept that adversary attacks pointing ML and DL systems can be implemented in many alternative and successful ways, for tampering with real existing defense systems.

5.1 Case Study Design

As first, we trained ten parallel GANs simultaneously, where both the detectors and the generators were realized by adopting deep neural network models. In particular, the models used for the discriminators implement a fully connected feedforward network architecture (FFNNs) while, for models of the generators, we adopted a convolutional layer architecture (CNNs).

For implementing each GAN comprised in our cooperative system, we took our cue from the general system architecture and the generators neural network architecture, implemented as a CNN, suggested in MalGAN [25]. Unlike MalGAN, we don't use a black-box detector and a substitute detector, since we designed our case study from the perspective of a "patient attacker" deploying a multistage attack. Our approach differs from MalGAN also in the kind of features considered both for training detectors and generators and in the generation strategy of the adversarial samples, as it will be detailed in the following sections.

Then, in the first stage, we assume that the attacker knows at least the features adopted by its victim for distinguishing between goodware and malware and has access also to its gradients. In this way, we could directly exploit the gradient's information provided by the detectors for training the generators and refining the capability to artificially generate samples that look like genuine ones.

With regard to the type of samples we analyzed and the kind of analysis performed by the victim detectors, specifically, we considered the surface features extracted from binary files applications, so the detectors answering to adversarial attacks are trained to perform a *static analysis* over the inputs they are fed in.

Correspondingly, adversarial examples will be generated by crafting these surface features. Since the surface feature space is discrete, we will apply a transformation to continuous space, in order to apply a gradient-based method for improving the probability that the generated adversarial examples will go undisturbed through the detection system. The approach suggested in [23] allows us to work in discrete and binary input domains, differently from most of the other proposed approaches [30] that operate only in continuous and differentiable domains.

Furthermore, static analysis has the advantage that does not require the execution of samples in a sandbox or safe environment for studying their behaviors, and the features for training the detector and/or classifiers can be extracted over specific subsets of features. Conversely, the dynamic analysis could reveal more information about malicious behaviors by the applications (e.g.., actions relationships and patterns) but the operative conditions are more difficult to achieve. Challenging results obtained by adopting static analysis in training machine and deep learning algorithms for malware detection are described in [3], where the authors provided, as first, an open-source dataset, namely, "EMBER," consisting in a collection of surface features extracted from a little under a million of malicious applications targeting Windows O.S. environment; furthermore, they provide experiments that compare a baseline gradient boosted decision tree model trained using LightGBM [29] with default settings to MalConv [43], an end-to-end but featureless deep learning model for malware detection, which recently became a very popular benchmark in this kind of experiments.

In the case of malware detection, unlike other application domains, like image and speech recognition, manipulating bytes can severely compromise application functionalities and validity; therefore, generating adversarial examples is not straightforward. An unavoidable requirement that should lay down every manipulation strategy consists in adopting generation techniques that are able to guarantee the preservation of malware functionality in the adversarially manipulated samples.

In our evaluation we trained, validated, and tested discriminator models for malware detection, by adopting for all the same samples, randomly extracted from the EMBER [3] dataset. Finally, we selected the first ten ones that obtained the best accuracy in the detection task. Then, we build ten parallel GANs, and we trained ten generators for the corresponding trained detectors (discriminators). For training the generators, we adopted a descendent gradient-based strategy and we adopted the maximum mean discrepancy (MMD) [5] as distance function for evaluating sample distributions similarity during the training process.

We examined the results obtained in the different stages of our experiment for measuring the effectiveness of the adversarial strategy and the robustness of the malware detectors to these kinds of attacks.

We adopted, for evaluating the reached performances, the following metrics: accuracy, sensitivity, specificity, and evasion rate. The evasion rate represents a measure of the success rate obtained by generator networks in fooling their opponent discriminators; it can be computed as the ratio between the number of adversarial examples that were misclassified as benign samples (also referred in the following as "*goodware*") by each detector, over the total amount of adversarial samples submitted to the discriminators.

By adopting the EMBER dataset, we were able to fit the detection performance obtained from the state of the art. Then, by using the adversarial crafting algorithm, we were able to mislead, on average, the ten detectors by decreasing the average accuracy over all the models ranging from a minimum of 20.63% (best case) to a maximum of 40.8% (worst case), by mixing genuine samples with adversarial one's samples and acting over the surface features.

Our preliminary experiments revealed, at a first sight analysis, that the byte distribution (byte histogram) is among the most sensitive features. This finding could suggest that machine- and deep-learning-based malware detectors, which work on static and surface features, could be fooled by adversarial malicious samples that are able to reach a bytes distribution with a high level of likelihood with the goodware bytes distribution.

5.2 General Architecture

The overall architecture of the system we propose corresponds to the general schema of a GAN (Fig. 1), where each couple made of a generator (G) and a discriminator (D) acts independently from each other.

5.2.1 The Discriminator Network Model

Following the approaches suggested in [36, 42, 43], we adopted a fully connected multilayered feedforward neural network as the base architecture for our discriminator's models. All the models we trained for obtaining the detection systems, as detailed in the following sections, share the same number of dense hidden layers, their size and the size of the output layer, set to 1, since the detection acts as a binary classification task (e.g., malware or goodware). All the trained models differ in the input layer size, since we performed several experiments by changing the size and the values of the input vectors, according to the combinations of features that we

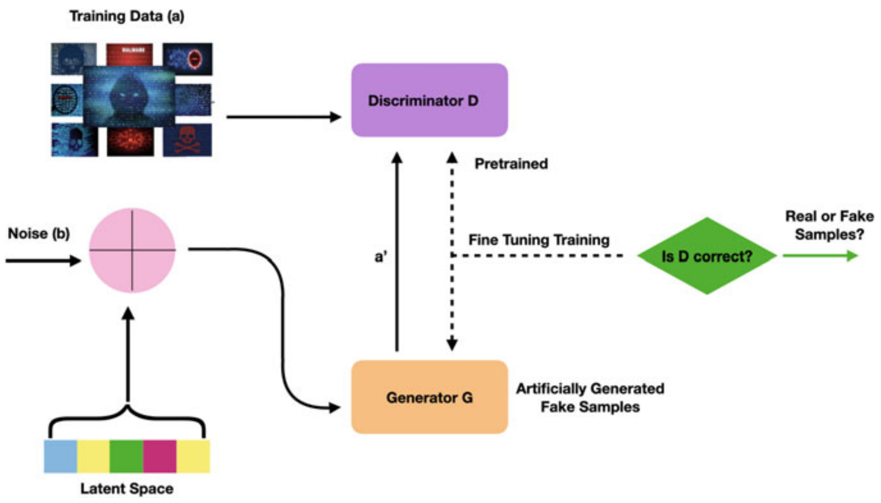


Fig. 1 GANs logic and building blocks of the proposed GANs-based architecture

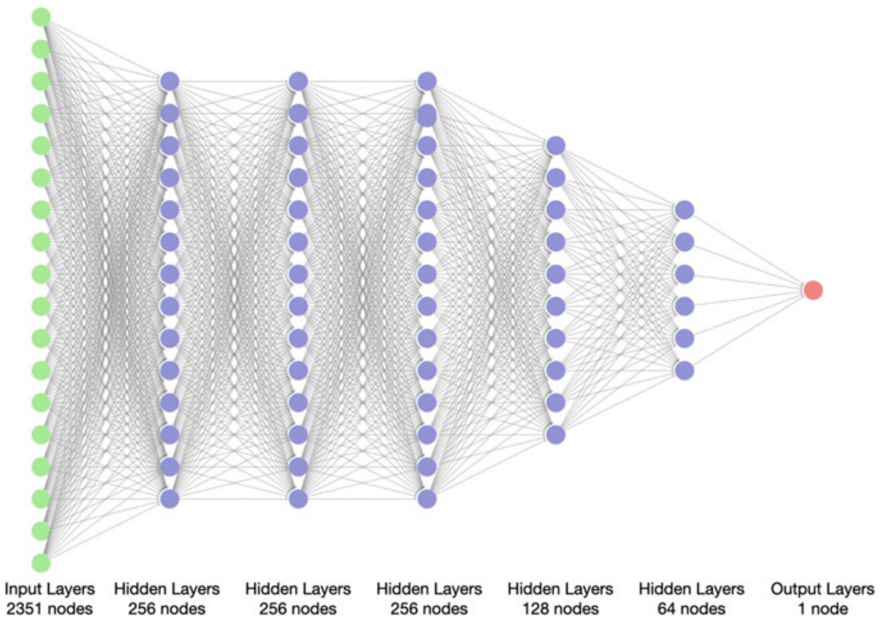


Fig. 2 The discriminator network: malware detector architectural schema

aimed to test. Figure 2 shows the general architecture of the discriminator network we adopted in our study.

The basic model adopted for each discriminator of our pool includes five hidden dense and fully connected layers characterized by decreasing size *(256-256-256-128-*

64); for each discriminator, the input size was variable, according to the subset of features we considered for each model. The maximum size of the input layer was set to *2,351*, when we consider all the available features provided in the EMBER dataset for implementing a static malware analysis over the input samples. In addition, we adopted the *Adam* algorithm as optimization function and the *binary cross entropy* as loss function. Finally, as an activation function, we adopted the *ReLU* that allowed to alleviate the vanishing gradient issues and is faster when compared with other non-linear activation functions.

We performed all the training cycles for *250 epochs* with *batch size* set to *64*. We adopted different learning rates l_r varying in the range [0.01;0.5]; finally, all the discriminators models were able to converge to an accuracy rate $a \geq 80\%$ and a *false positive rate* $(FPR) \leq 1\%$ (where *FPR* is computed as the number of benign samples misclassified as malicious over the total number of malicious samples detected), by adopting a $l_r = 0.05$ and a number of iterations = *250 epochs*. The performance metrics values were cross validated over the validation and the test sets. All the experiments were repeated five times and the average values obtained in these experiments were considered as the values of the final hyperparameters for tuning the networks.

The reason underlying the strategy of training several models was dictated also by the need to apply a reduction to the whole set of the features provided by PEs files; even though the best accuracy is performed when a detection model is trained over the whole features set, we need also to limit the performance decay, in terms of data size and training time, in order to make this approach feasible for real-world scenarios. So, we applied a strategy for reducing features and we were able to obtain a trade-off among accuracy, data size, and learning time. Anyway, we didn't investigate more space and time complexity on this occasion, but it will be the object of further and necessary investigations.

5.2.2 The Generator Network Model

For the generation network model, we followed the general setting adopted in [25].

The model we adopted for the generators is represented by a convolutional neural network (CNN) trained on a sample fraction extracted by the EMBER dataset. We split the Ember dataset in order to save a fraction of samples, made of benign and malicious samples that were not included in the training set of our detectors. We trained the generators until all of them reached at least an accuracy rate $a \geq 98\%$, when artificially reproducing the original samples, as it will be detailed in the following of this section. For the generation of adversarial examples (AEs), we set two main constraints:

- ***Functionality preserving***: Adding noise for generating adversarial examples should not break the sample's behavior.
- ***Features probability distributions invariance***: Since we worked only on surface features, we don't manipulate the content of binaries but we try to change the

surface information as their probability distribution looks like more close to the distribution of good samples.

The CNNs we adopted for generators are characterized by layer sizes set to X-256-X, where X represents the variable size for both the input and the output, according to the input dimension that has to be transformed and the adversarial sample size that has to be produced. Noise vectors adopted for manipulating genuine inputs have the same dimensions of the input, according to the number of features that are considered in each couple of generator-discriminator. The Adam optimizer was selected as an optimization function. Each generator was trained for 500 epochs with a learning rate set to 0.05. These training parameters were obtained after several experiments until the best tuning that guarantees convergence for all the generators with an accuracy rate over the original ground truth stabilized to 98%. This accuracy was cross validated also over the validation and test set. All the experiments were repeated five times and the average values obtained in these experiments were considered as the values of the final hyperparameters for tuning the networks.

The generation of AEs is usually done by adding small perturbations to the original input in the direction of the gradient. The gradient-based methods work for continuous input sets but they fail in the case of discrete input sets. If we denote the set of the features as $X \subseteq [0, N - 1]$, where $N = 2351$, the features comprised in the PEs files can be arbitrarily represented as scalars in a set X [0, N - 1], where $N = 2,351$. So, AEs can be generated in a continuous embedding space E and reconstructed them to original X.

5.2.3 Adversarial Example Generation Problem

Given a trained deep learning model f, an original input data sample x, generating an adversarial example x', can generally be described as a box-constrained optimization problem:

$$\begin{aligned} \mathbf{min\, x'}\ \left\|\mathbf{x}' - \mathbf{x}\right\| \\ \mathbf{s.t.\ m}\left(\mathbf{x}\right) = \mathbf{l} \\ \mathbf{m}\left(\mathbf{x}'\right) = \mathbf{l}' \\ \mathbf{l} \neq \mathbf{l}' \\ \mathbf{x}' \in [\mathbf{0},\ \mathbf{1}], \end{aligned}$$

where

- x is the genuine input sample;
- x' is the artificial input sample;
- m $(\cdot)$ represents the trained deep learning model;
- l and l' represent, respectively, the output labels produced by the model m $(\cdot)$ when processing x and x'; and
- ||·|| denotes the distance between two samples.

δ is the difference between x' and x, and represents the perturbation (noise) added to x. This optimization problem minimizes the perturbation while misclassifying the prediction with a constraint of input data. Other variants of this optimization problem can be considered in different scenarios and assumptions. For instance, in the image recognition domain, some adversaries consider that if $\delta < \epsilon$, the perturbation is small enough to be unnoticeable to humans and it is viewed as a constraint. The optimization objective function becomes the distance of the targeted prediction score from the original prediction score.

5.3 Adversary Logic

As described in [50], the adversarial examples can be categorized in a taxonomy along seven axes. In our study, we followed the axes of the *adversarial falsification* and the *iterative attack*. For the first dimension, we were interested in training generators able to lead a decay in the detection accuracy of each detector, as it will be shown in the results subsection. For the second dimension, by exploiting the transferability of adversarial examples [37], we divided our attack into multiple stages until we reach a fine-tuned generator for addressing the victim's vulnerabilities. We considered as:

- *False positive*: The negative examples artificially generated that are misclassified as positive samples.
- *False negative*: The positive examples artificially generated that are misclassified as negative samples.

In the case of the malware detection task, a benign software being classified as malware is a false positive. Conversely, a false negative is a malware (usually considered as positive) that cannot be identified by the trained model. This is also known as *machine/deep learning evasion*.

5.3.1 Threat Model

We define the threat model as follows:

- The adversaries can attack only at the testing/deploying stage. They can tamper with only the input data in the testing stage after the victim deep learning model is trained. Further, we assume that neither the trained model nor the training dataset can be modified. The adversaries may have knowledge of the trained model (architectures and parameters) but not allowed to modify the model, which is a common assumption for many online machine learning services. We are not considering attacks at the training stage (e.g., training data poisoning [16, 34]), even if they are another interesting topic to explore.
- Since we considered adversarial attacks for deep neural networks, the adversaries target only the integrity of their inputs. In general, integrity is essential to a deep

learning model, although other security issues related to confidentiality and privacy have drawn attention in deep learning. Anyway, in the case we considered PE files, the integrity of the input is crucial. So, we focused on the attacks that degrade the performance of deep learning models for malware detection: attacks cause the increase of false positives and false negatives.

5.3.2 Adversarial Examples Generation

AEs are artificial inputs that are generated by modifying legitimate inputs so as to fool the classification models. In the fields of image and speech recognition, modified inputs are considered adversarial when they are indistinguishable by humans from the legitimate inputs, and yet they fool the model. Conversely, discrete sequences are inherently different than speech and images, as changing one element in the sequence may completely alter its meaning. For example, changing one word in a sentence may hinder its gradient in a binary file, where the input is a discrete sequence of bytes, changing one byte may result in invalid bytecode or different runtime functionality. In malware detection, an AE is a binary file that is generated by modifying an existing malicious binary. While the original file is correctly classified as malicious, its modified version is misclassified as benign. Recent works as [23] have shown that AEs cause catastrophic failures in malware detection systems, trained on a set of handcrafted features such as file headers and API calls. Our experiment (contribution) is focused on changing surface features by keeping the same original distribution of benign samples.

5.4 Dataset

We chose EMBER released by Endgame [3] as the dataset for our case study. EMBER is a collection of features extracted from a large corpus of Windows portable executables.

The first version of the dataset is a collection of 1.1 million PEs that were all scanned by VirusTotal in 2017. The second EMBER dataset release consisted of features extracted from samples collected in or before 2018.

The set of binary files is divided as follows:

- 900,000 training samples grouped in:
  - 300,000 malicious;
  - 300,000 benign; and
  - 300,000 unlabeled.
- 200,000 test samples grouped in:
  - 100,000 malicious;
  - 100,000 benign.

```
{
  "sha256": "04637...", "appeared": "2017-01",
  "label": 1,
  "histogram": [ 3818, 155, 135, ... ],
  "byteentropy": [ 0, 0, 0, ... ],
  "strings": { "numstrings": 170,
               "avlength": 8.170588235294117,
               ... },
  "general": { "size": 33334, "vsize": 45056,
               "has_debug": 0, ... },
  ...
}
```

Fig. 3 Code snippet from Ember dataset JSON files describing PEs features

The dataset is made up of JSON files. Each sample includes

- the sha256 hash of the original file as a unique identifier;
- the month the files was first seen;
- a label, which may be 0 for benign, 1 for malicious, or -1 for unlabeled; and
- eight groups of raw features that include both parsed values and format-agnostic histograms.

A code snippet from the JSON file is shown in Fig. 3.

5.4.1 Raw Features

The raw features include both parsed features and format-agnostic histograms and counts of strings. Parsed features, extracted from the PE file, are

- *General file*: Information including the file size and basic information obtained from the PE header.
- *Header information*: Reporting the timestamp, the target machine (string), and a list of image characteristics (list of strings). From the optional header, the target subsystem (string); DLL characteristics (a list of strings); the file magic as a string (e.g., “PE32”); major and minor image versions; linker versions; system versions and subsystem versions; and the code, headers, and commit sizes are provided.
- *Imported functions*: After having parsed the import address table, the imported functions by the library are reported.
- *Exported functions*: The raw features include a list of the exported functions.
- *Section information*: Properties of each section are provided, including the name, the size, the entropy, the virtual size, and a list of strings representing section characteristics.

The EMBER dataset also includes three groups of features that are format-agnostic, as they do not require parsing the PE file:

- Byte histogram contains 256 integer values, representing the count of each byte value within the file. The byte histogram is normalized to a distribution, since the file size is represented as a feature in the general file information.
- Byte-entropy histogram approximates the joint distribution p(H,X) of entropy H and byte value X.
- String information reported is the number of strings, their average length, a histogram of the printable characters within those strings, and the entropy of characters across all the printable strings.

5.5 Performance Metrics

Four metrics were used to evaluate the detectors (*accuracy, sensitivity, specificity, and evasion rate*) obtained under different testing conditions. We provide the general standard definitions for these metrics while reserving us to improve the explanation about how they were specifically computed in the specific sections. As first we provide definitions for true positives, true negatives, false positives, and false negatives.

True positive: (TP) = the number of malicious samples correctly identified as malicious;

False positive: (FP) = the number of benign (goodware) samples incorrectly identified as malicious;

True negative: (TN) = the number of benign samples correctly identified as benign; and

False negative: (FN) = the number of malicious samples incorrectly identified as benign.

By combining these observations it is possible to compute further indicators, whose general meaning is provided as follows:

Accuracy: The accuracy of a test is defined as its ability to differentiate the benign and malicious samples correctly. To estimate the accuracy of a test, we compute the proportion of true positive (TP) and true negative (TN) in all the evaluated cases. Mathematically, this can be stated as follows:

$$\text{Accuracy} = \frac{TP + TN}{(TP + FP + TN + FN)}.$$

Sensitivity: The sensitivity of a test is its ability to determine the malicious cases correctly. To estimate it, we should calculate the proportion of true positive (TP) in malicious cases. Mathematically, this can be stated as follows:

$$\text{Sensitivity} = \frac{TP}{(TP + FN)}.$$

Specificity: The specificity of a test is its ability to determine the good cases correctly. To estimate it, we compute the proportion of true negative among good cases. Mathematically, this can be stated as follows:

$$\text{Specificity} = \frac{TN}{(TN + FP)}.$$

Finally, we consider another indicator of sensitiveness known as the ***evasion rate***. When a dataset for testing the ability exhibited by a system in detection and/or classification task is poisoned with carefully designed adversarial examples, there are two adversary perspectives: the victim (detector) and the attacker (generator) ones. So, if we are interested to estimate the robustness of a detection system by computing its performance decay under an adversarial attack (e.g., an accuracy decay), we are interested in the estimation of the ability of the generator to produce adversarial examples that are misclassified. In this perspective, the evasion rate can be adopted as an indicator for measuring the generation ability and is defined, according to the definition provided in [10], as follows:

$$\text{Evasion Rate (EV)} = \frac{FN_{\text{AEs}}}{N_{\text{AEs}}},$$

where N_{AEs} represents the number of artificially generated adversarial samples of malware submitted to the detector and FN_{AEs} is the fraction of the overall counted false negatives (malware incorrectly classified as goodware) represented by adversarial samples set (that is to say, artificially generated malware incorrectly classified as goodware).

5.6 Case Study Treatments

The case study was conducted on an Ubuntu 18.04 platform, running on a cluster composed of five machines, with the same hardware configuration, equipped with an Intel Xeon E5-2620 processor and 128 GB RAM. We exploited the GPU functionalities of 5 NVIDIA GeForce RTX 2080 boards, by using the CUDA toolkit 9.0 and cuDNN with a TensorFlow-GPU v.1.13.1 version, running with Python 3.7. We further adopted the Ember script tools version 0.1.0, LightGBM 2.1.0, scikitlearn 0.19.1, NumPy 1.14.2, and SciPy 1.0.0, Matplotlib 3.2.2 for plotting results.

Table 1 PEs surface feature groups in the ember dataset

Feature group ID	Description and original name	Number of features
FG00	All	2351
FG01	General file info (General)	10
FG02	Header info (Header)	62
FG03	Imported functions (Imports)	1280
FG04	Exported functions (Exports)	128
FG05	Section info (Section)	255
FG06	Byte histogram (Histogram)	256
FG07	Byte-entropy histogram (Byte entropy)	256
FG08	String info (Strings)	104

5.6.1 Malware Detector Training: The Method

The total number of features comprised in each PE is equal to 2,351, grouped in eight families, according to the PE specifications[41]. Families' names and their quantity are provided in Table 1. We added, for convenience of comparison, the 9th group (FG00) representing the group including all the feature families, that is to say, 2,351 features.

For our case study, we started from considering all the features belonging to a group as a unit, so we always selected all the features in a feature group or we selected none.

Each combination was evaluated according to the following information:

- selected feature groups;
- accuracy and false positive rates (FPR) computed by varying the threshold of malware-likelihood scores by 0.01.

As described in the performance metric subsection, we define the accuracy as the ratio of the number of correct answers to the number of all answers, and FPR as the ratio of the number of malware-determination answers to the number of good samples.

For each feature combination, we associated a set of feature vectors with a ground-truth label and trained a different model; finally, we performed testing (malware-likelihood computation) operations. After performing training, validation, and tests, we selected the ten best detectors, according to the best accuracy values in the detection and a FP rate less than the limit threshold of 0.01.

5.6.2 Adversarial Examples Generator: The Method

As for the generation strategy for adversarial examples, we worked on the surface features of binary files and we focused on producing small changes on the most sensitive features groups, in order to reproduce, for the artificially generated samples, the distributions of the same features exhibited by goodware samples.

We want to remark that, for the purposes of the case study, we only considered applications metadata, extracted by the original binary files and conveniently provided in the EMBER dataset. We were interested to provide further evidence that feature-based models for malware detection, even if realized by the means of deep neural networks, may be broken by adversarial samples properly designed. We haven't considered the whole binary files, because manipulating the content of a binary file, even also changing a small number of byte, can severely compromise the behavior and the functionalities of the application. This aspect, also investigated in the works of [30, 32], will be a matter of further investigations, possibly joining both surface features and payload of binary files.

With previous works, we share the common approach of generating AEs by adding small perturbations to the original malicious inputs, in order to follow the probability distributions of the selected features groups, in the direction of the descent gradient, for reducing the distance between probabilities distribution.

Since we considered surface features, we observed that some features are more sensitive than others. So, our generation strategy consisted in following the probability distribution trend of these sensitive features in genuine goodware samples, thus producing a noise able to make closer the surface features probability distributions of the followed model (the genuine benign sample) with the probability distributions of the following model (the malware sample that has to be manipulated).

5.6.3 Training, Validation, and Test Sets Composition

In this section, details about the size of the training, validation, and test sets employed for performing the case study are provided. The samples composing these sets have been randomly extracted as a subset of the EMBER files collection, only excluding the adversarial samples that were artificially generated.

- **Training set for discriminators**: 300,000 genuine samples, divided into 150,000 goodware and 150,000 malware (X_{trainD}).
- **Training set for generators**: 300,000 genuine samples, divided into 150,000 goodware and 150,000 malware (X_{trainG}); this training set is intersectionless with the set adopted for training discriminators:

$$X_{trainD} \cap X_{trainG} = \emptyset.$$

- **Validation set for discriminators**: 50,000 genuine samples (GEs), divided into 25,000 goodware and 25,000 malware ($X_{validationD}$).

- **Test set for discriminators (excluding the adversarial samples)**: 45,000 genuine samples, divided into 15,000 goodware and 30,000 malware ($X_{testGEs}$).
- **Test set for GANs (discriminators including the adversarial scenario):** 45,000 samples, divided into 15,000 genuine benign samples, 15,000 genuine malicious samples (the same of discriminators without attack), 15,000 artificially generated adversarial examples of malware ($X_{testAEs}$).

5.7 *Case Study Results and Performance Evaluation*

In order to provide a clear and convenient explanation of our case study and its results, we decide to present the results splitting them into two scenarios, in order to compare how the malware detector performances degrade when attacked with adversarial samples. Results of our tests will be summarized in terms of *accuracy*, *sensitivity*, and *specificity* metrics.

Regarding the first scenario, sensitivity corresponds to the *true positive rate (TPR)*, where we considered as true positives all the malicious samples that were correctly identified as malware in the detection task. Finally, we considered the *false positive rate (FPR)* obtained in the malware detection task, computed according to the following equation:

$$FPR = \frac{FP}{FP + TP}.$$

Regarding the second scenario, instead, accuracy is computed as the success rate, i.e., the *evasion rate* (ER) obtained from the generator against his opponent (the detector), and measures the number of adversarial examples that pass undisturbed. In this scenario, the ER (coinciding with the TPR) is computed as the number of adversarial malicious examples that are misclassified as "good guys" (goodware); it corresponds to the ratio between the number of adversarial examples that successfully pass as "good guys" and the total number of adversarial examples submitted to the detector (discriminator).

5.7.1 Scenario 1: Discriminator Performance Excluding the Adversarial Attack

Results are shown only for the best ten trained models, according to the described criteria for the accuracy and FPR. In addition to these criteria, we performed two different tests for obtaining a further indication of the sensitiveness of the considered feature groups. Defining as $n_{C_{K_i}}$, with $K \in [A, B]$, the maximum number of feature groups considered in the i^{th} combination C_i, Tables 2 and 3 show the accuracy scores of the best ten models (plus the 11^{th} case of selecting all the available different features), respectively, in the case in which we set the additional conditions in the training models to

Table 2 Accuracy scores for the best ten combinations of features by considering the combinations of 4 different feature groups at most(C_A)

Combination ID	Selected feature group combination	Total number of feature	Accuracy rate (%)
C_{A_1}	General, Header, Histogram, Section	583	**92.27**
C_{A_2}	General, Header, Histogram, Strings	432	91.84
C_{A_3}	General, Header, Section, Strings	431	90.66
C_{A_4}	General, Header, Histograms	328	89.45
C_{A_5}	Header, Section, Strings	421	88.23
C_{A_6}	General, Header, Byte entropy	328	87.12
C_{A_7}	General, Section, Strings	369	86.35
C_{A_8}	General, Header, Strings	176	85.73
C_{A_9}	Section, Strings	359	83.07
$C_{A_{10}}$	General, Section	265	80.24
C_{A_0}	**All**	2351	**98.32**

$$\begin{aligned} C_S &: n_{C_i} = 1 \\ C_A &: 1 \leq n_{C_i} \leq 4 \\ C_B &: 5 \leq n_{C_i} \leq 8. \end{aligned}$$

The combination named *All* corresponds to all the eight groups *(Header, Imports, Section, Histogram, General, Exports, Byte entropy, Strings)*, including all the 2,351 features.

The accuracy metric was computed by adopting the test set denoted as ($X_{testGEs}$), comprising 45,000 genuine samples divided into 15,000 benign and 30,000 malicious samples.

By analyzing the results shown in Table 3, we can observe that the highest value for the accuracy is scored by the combination C_{B_0}, including all the features, while the closest score to this combination is obtained with a reduced set of features (combination C_{B_1}), with a difference in accuracy that is at minimum 1.43% (C_{B_0} versus C_{B_1}) and at maximum 2.58% (C_{B_0} versus C_{B_9}).

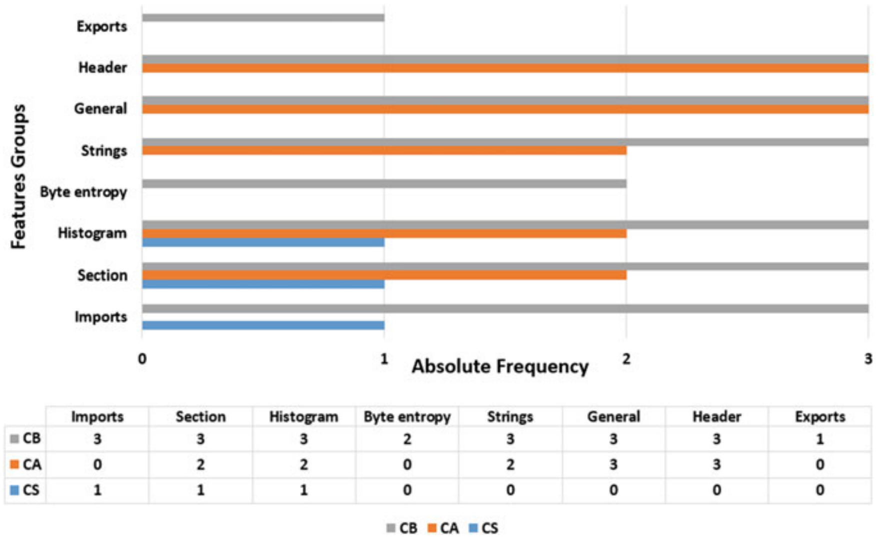
The feature groups *Header, Imports, Section*, and *Histogram* revealed to be particularly sensitive in biasing the accuracy score.

Table 3 Accuracy scores for the best ten combinations of features by considering at least five and at most seven different feature groups (C_B)

Combination ID	Selected feature group combination	Total number of feature	Accuracy rate (%)
C_{B_1}	Header, Imports, Section, Histogram, General, Strings	1967	**96.89**
C_{B_2}	Header, Imports, Section, Histogram, General, Byte entropy, Strings	2223	96.55
C_{B_3}	Header, Imports, Section, Histogram, Byte entropy, String	2213	96.39
C_{B_4}	Header, Imports, Section, Histogram, General, Exports, Byte entropy	2247	96.28
C_{B_5}	Header, Imports, Section, Histogram, String	1957	96.12
C_{B_6}	Header, Imports, Section, Histogram, Exports, Byte entropy	2237	96.07
C_{B_7}	Header, Imports, Section, Histogram, General, Exports, String	2095	95.96
C_{B_8}	Header, Imports, Section, Histogram, Byte entropy	2109	95.89
C_{B_9}	Header, Imports, Section, Histogram, General, Exports	1991	95.74
C_{B_0}	All	2351	**98.32**

Table 4 Accuracy scores for singleton feature group combinations (C_S)

Combination ID	Selected feature group combination	Total number of feature	Accuracy rate (%)
C_{S_1}	Imports	1280	82.79
C_{S_2}	Section	255	73.46
C_{S_3}	Histogram	256	73.14
C_{S_4}	Byte entropy	256	65.81
C_{S_5}	Strings	104	64.73
C_{S_6}	General	10	**61.59**
C_{S_7}	Header	62	54.13
C_{S_8}	Exports	128	20.45



	Imports	Section	Histogram	Byte entropy	Strings	General	Header	Exports
⬜ CB	3	3	3	2	3	3	3	1
🟧 CA	0	2	2	0	2	3	3	0
🟦 CS	1	1	1	0	0	0	0	0

Fig. 4 Feature groups' absolute frequency in the three most accurate models

Particularly, we observed that the feature group *Histogram* appears in the best scores both in the reduced (C_A) and in the extended (C_B) feature group combinations.

Finally, since the information about feature sensitiveness to the accuracy is crucial for designing the generation strategies for adversarial samples that will be effective, we performed the last test considering only single group combinations, as shown in Table 4.

We can observe that the feature groups that scored the best accuracy values were *imports, section*, and *histogram*. These three groups were also included in all the ten best ranking models considered in Table 3, where the best results were generally obtained. Figure 4 summarizes the absolute frequency scored for all the eight features groups over the three best ranked models for each of the three training we performed.

5.7.2 Generator and Discriminator Performance Including Adversarial Attack

For generating adversarial examples and testing the pool comprising the ten most accurate discriminator models described in the previous scenario, we trained correspondingly ten generators.

Given the initial working hypothesis of having knowledge, at this stage, of discriminators gradients generated during the training over the genuine dataset, we had the opportunity to exploit them in combination with the inner gradients of generators, in order to apply a semi-direct training process for the generators. To be clearer, we could adopt both a direct method for training the generators and an indirect one.

The direct method does not require to involve the discriminators during the training of the generators that are trained by simply comparing the difference elapsing between the probability distributions of genuine samples and the artificially generated (adversarial) samples. This method is practicable in this case because we have the true genuine data (a kind of white-box attack at the first stage) available.

In the second and most realistic stage of the attack scenario, we imagined (a black-box attack) genuine data aren't available and the direct method for training generators can't be applied yet. In this situation, the generators can be trained by submitting, at each training iteration, the generated outputs to the victim and collecting the response, for computing a step for making descendant the gradient function.

For reasons of simplicity, we adopted the direct method, since our aim was to provide evidence that feature-based deep learning models for malware detection could work with high accuracy even if the static analysis is performed; anyway, as other kinds of deep learning models, also performing a dynamic analysis of samples, they are affected by adversarial examples carefully designed.

So, the generators were trained by performing the comparison between the probability distribution of its generated samples with a "genuine" training set and back-propagating the difference (the error) through the network, at each iteration of the training process. To compute the distance (or similarity measure), we adopted the maximum mean discrepancy (MMD) [5, 6, 18], able to compare effectively two distributions.

Then, the training process of the generative networks develops as follows. Given a random variable with uniform probability distribution as input, we want the probability distribution of the generated output to be the "genuine data set probability distribution"; we considered two subcases:

- the first one in which the genuine dataset is the same adopted for training the discriminators;
- the second one, in which the genuine dataset is represented by a different and intersection fewer dataset from the one adopted for training discriminators.

The training process for each of the generators follows the basic idea to optimize its inner network by repeating the following steps:

- to generate some random inputs of the same size as the corresponding discriminator;
- to make these inputs go through the generator and the discriminator and collect both generated outputs;
- to compare the "genuine probability distribution" and the artificially generated one, by computing the MMD distance between the true samples and the generated ones; and
- to adopt backpropagation to make one step of gradient descent to lower the MMD distance between the truly genuine and artificially generated distributions.

We discuss here how to manipulate a source malware sample x into an adversarial malware binary x^* by slightly changing the surface feature values. Generators aim to minimize the confidence associated with the malicious class (i.e., it maximizes the probability of the adversarial malware sample being classified as benign), under the constraint that q_{max} is the maximum amount of noise (changes) that can be added to the original sample for being effective. The deep network implementing each generator produces the probability of the generic sample x being malware, denoted in the following with *f(x)*. If f(x) $\geq$ 0.5, the input file is thus classified as malware (and as benign, otherwise).

This can be characterized as the following constrained optimization problem:

$$\begin{aligned} min_x f(x) \\ s.t.d(x, x^*) \leq q_{max}, \end{aligned}$$

where

- x denotes the genuine sample distribution;
- x^* denotes the generated sample; and
- d(x, x^*) is the distance function computed as the MMD distance.

We solve this problem with a gradient-descent algorithm over the generator networks by adopting as loss function the distance between the true and the generated distributions at the current iteration.

We trained each generator, for both the genuine datasets, for 500 epochs, and we also adopted a learning rate set to 0.05. These hyperparameters for the training process were obtained after all the ten generators were able to converge and we stopped when the error reached the threshold value of 0.02 (2%) (corresponding to an accuracy rate in validation and testing of 98%). We were not able to reach lower error rates, because we trained generators for being able to produce just over 15.000 adversarial samples, in order to ensure the same numerosity of the genuine test examples when testing the discriminators. The overall time for training the ten generators until all of them converge to a similarity rate of 98%, estimated between the truly genuine and the adversarial generated samples distributions, lasted about 1 day and a half (about 37 h). We repeated the training process five times and we considered as assessed the generator models after a time of about 10 d.

In this way, we obtained 30,000 adversarial examples, divided into two sets G_{EQ} and G_{NEQ} comprising, respectively, 15,000 adversarial malicious samples generated (AEs$_{EQ}$)from comparison with the genuine training set adopted for discriminator models and 15,000 artificially generated samples (AEs$_{NEQ}$) computed by evaluating the difference from a different dataset from the one adopted for training discriminator models.

Finally, we addressed the adversarial attack to the ten discriminators with both the two sets of generated adversarial samples and we provide a brief discussion over the results we observed.

Like in Scenario 1, for computing performance metrics, we tested the discriminators with two variants of the test set denoted as $(X_{testAEs})$; each variant comprises 45,000 samples, divided into 15,000 genuine benign samples and 30,000 malicious samples, in turn divided into 15,000 genuine malware and 15,000 adversarial malware samples. The two classes of test sets differ only for the kind of adversarial malware samples included. In the first class, we included adversarial malware samples generated by comparison with the same training set adopted for training discriminator models; in the second class, we included adversarial malware samples generated by adopting a different training set from the one adopted for training discriminators.

So, we discussed these two cases of AE attack and we compared them with the original accuracy scored by each discriminator when excluding AEs from its test set.

To verify the efficacy of the attack, for each test we measured beyond the accuracy and the sensitivity, also the evasion rate [10], computed as the percentage of malicious samples that managed to evade the network [30].

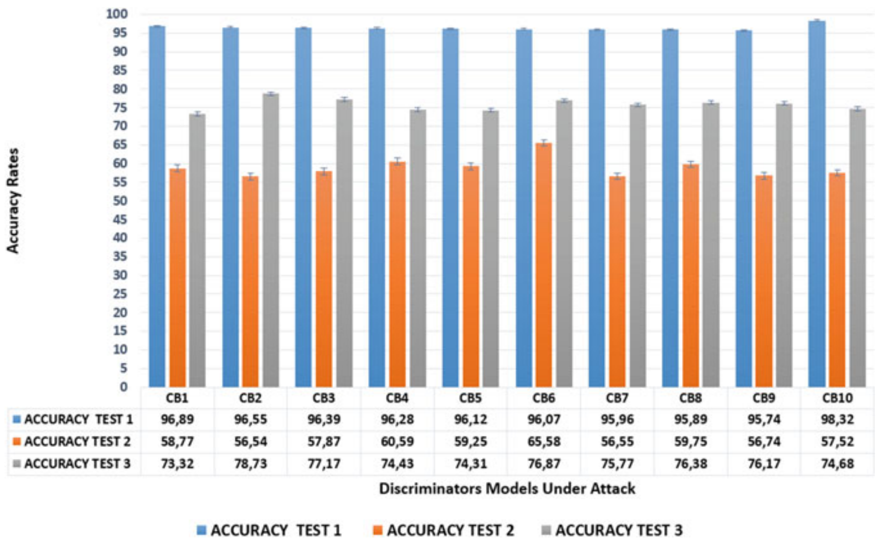
For each of the three cases shown in Table 5, accuracy was computed considering a test set comprising 45,000 samples; anyway, these tests were differently composed for allowing, respectively, the case excluding the adversarial examples and the two cases including AEs generated by comparing or not comparing to the genuine training set adopted for discriminators. These cases include

- Excluding AEs: No AEs attack is performed against the discriminators; the test set is made of 45,000 genuine samples only, divided into 15,000 goodware samples and 30,000 malware samples.
- Including AEs trained over the training set adopted by the discriminator: AEs attack is performed against the discriminators; the test set is made of 45,000 samples, among which 30,000 genuine samples are divided into 15,000 goodware and 15,000 malware; the remaining 15,000 represent adversarial examples; this test set will be called as follows: ***AEs$_{EQ}$***.
- Including AEs trained over a different training set from the one adopted by the discriminator: AEs attack is performed against the discriminators; the test set is made of 45,000 samples, among which 30,000 genuine samples are divided into 15,000 goodware and 15,000 malware; the remaining 15,000 represent adversarial examples; this test set will be called in the following as ***AEs$_{NEQ}$***.

The results that we obtained in terms of evasion rate and accuracy decay for each of the ten discriminators are summarized in Tables 5 and 6.

Table 5 Accuracy rate reached by attacking discriminators with adversarial examples from sets AEs$_{EQ}$ and AEs$_{NEQ}$

	TEST 1	TEST 2	TEST 3
Discriminator ID	Accuracy excluding AEs (%)	Accuracy including AEs$_{EQ}$ (%)	Accuracy including AEs$_{NEQ}$ (%)
C_{B_1}	**96.89**	58.77	73.32
C_{B_2}	96.55	56.54	78.73
C_{B_3}	96.39	57.87	77.17
C_{B_4}	96.28	60.59	74.43
C_{B_5}	96.12	59.25	74.31
C_{B_6}	96.07	62.58	76.87
C_{B_7}	95.96	56.55	75.77
C_{B_8}	95.89	59.75	76.38
C_{B_9}	95.74	56.74	76.17
C_{B_0}	**98.32**	57.52	74.68



	CB1	CB2	CB3	CB4	CB5	CB6	CB7	CB8	CB9	CB10
▪ ACCURACY TEST 1	96,89	96,55	96,39	96,28	96,12	96,07	95,96	95,89	95,74	98,32
▪ ACCURACY TEST 2	58,77	56,54	57,87	60,59	59,25	65,58	56,55	59,75	56,74	57,52
▪ ACCURACY TEST 3	73,32	78,73	77,17	74,43	74,31	76,87	75,77	76,38	76,17	74,68

Fig. 5 Accuracy rate distributions for discriminators under AEs attack

In Figs. 5, 6, and 7 are reported, respectively, the accuracy rate distributions and the trend line of the accuracy decay, computed over the ten discriminators and the two types of AEs considered, when discriminators are under AEs attack.

We can observe that AEs perform worse (test set AEs$_{NEQ}$) than the other AEs adversarial set, producing, over the ten tested models for discriminators, an average decay of accuracy valued to Delta $(a_{AEs_{EQ}}) = 20.63$ points. Minimum loss $min_{loss} = 19.20$ points and maximum loss $max_{loss} = 23.57$ points.

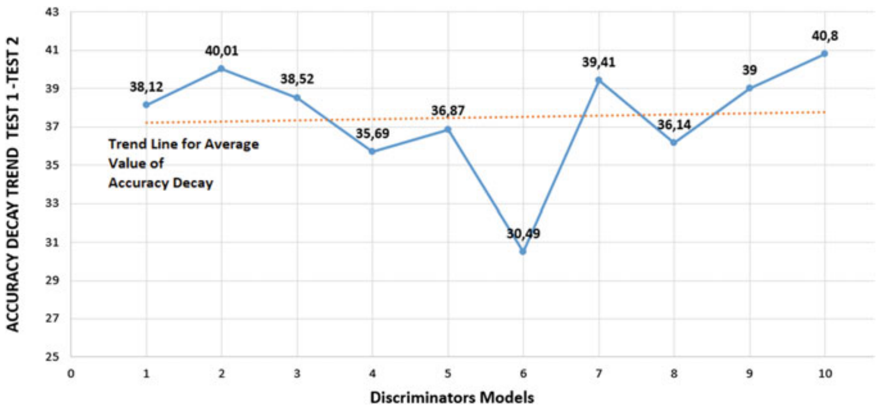


Fig. 6 Trend of accuracy decay of discriminators under AEs attack (TEST 1–TEST 2)

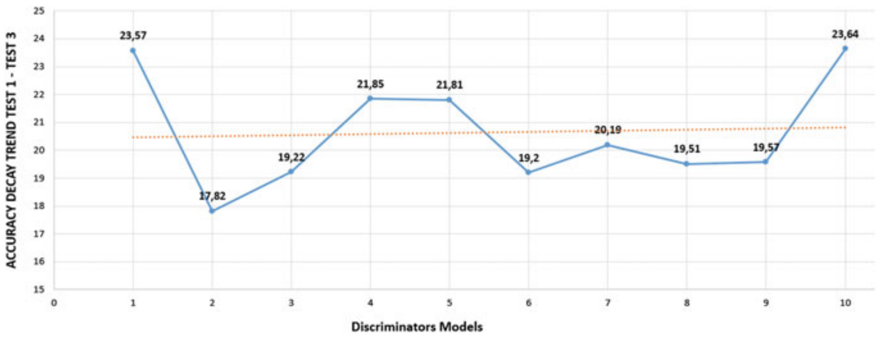


Fig. 7 Trend of accuracy decay of discriminators under AEs attack (TEST 1–TEST 3)

Instead, when adversarial attack is performed by adopting adversarial examples produced by generators trained over the same dataset adopted for training the discriminators, AEs perform better (test set AEs$_{EQ}$) than the other AEs adversarial set, affecting over the ten tested models for discriminators an average decay of accuracy valued to Delta ($a_{AEs_{EQ}}$) = 37.50 points. Minimum loss $min_{loss} = 30.49$ points and maximum loss $max_{loss} = 40.80$ points.

6 Conclusions

Attacks and defenses on adversarial examples draw great attention. The vulnerability to adversarial examples becomes one of the major risks for applying DNNs in safety-critical environments.

Table 6 Evasion rate computed attacking discriminators with adversarial examples (AEs) from sets A_{EQ} and A_{NEQ}

Discriminator ID	Evasion rate with AEs_{EQ} (%)	Evasion rate with AEs_{NEQ} (%)
C_{B_1}	**35.84**	**26.69**
C_{B_2}	33.72	27.38
C_{B_3}	34.65	25.44
C_{B_4}	32.61	26.14
C_{B_5}	34.89	25.98
C_{B_6}	33.77	26.29
$C)_{B_7}$	31.52	27.66
C_{B_8}	29.05	26.39
C_{B_9}	27.39	24.61
C_{B_0}	**39.12**	**32.45**

Adversarial perturbations can easily fool deep neural networks (DNNs) in the testing/deploying stage exploiting blind spots in the ML engine. The effectiveness of an adversarial system is measured in terms of *evasion rate* and it depends upon a specific group of features considered for the input set. Applied to the creation of malware, GANs are able to generate a new instance of a malware family without knowing an explicit model of the initial distribution of the data.

So an attacker could use GANs to fool detection systems, just by sampling the provided data. On the other hand, GANs are also useful to build more robust machine learning models helping in the development of a better training set. Real defense technologies such as AV or EDR must take into account an acceptable trade-off among the detection accuracy, short learning times, and limit the size of data obtainable by selecting a convenient combination of the sensitive feature. The effectiveness of an attack on the ML model also depends on the knowledge of the system by the attacker. In this case study, we conducted a gray-box attack in which the features of the training set are known: this permits us to reach a very high evasion rate (about 98%).

References

1. Alazab, Mamoun, Sitalakshmi Venkatraman, Paul Watters, and Moutaz Alazab. 2013. Information security governance: the art of detecting hidden malware. In *IT security governance innovations: theory and research*, 293–315. IGI Global.
2. Alzantot, Moustafa, Bharathan Balaji, and Mani Srivastava. 2018. Did you hear that? adversarial examples against automatic speech recognition. arXiv:1801.00554.
3. Anderson, Hyrum S., and Phil Roth. 2018. Ember: an open dataset for training static pe malware machine learning models. arXiv:1804.04637.

4. Apruzzese, Giovanni, Michele Colajanni, Luca Ferretti, Alessandro Guido, and Mirco Marchetti. 2018. On the effectiveness of machine and deep learning for cyber security. In *2018 10th international conference on cyber Conflict (CyCon)*, pages 371–390. IEEE, 2018.
5. Arbel, Michael, Dougal Sutherland, Mikołaj Bińkowski, and Arthur Gretton. 2018. On gradient regularizers for mmd gans. *Advances in neural information processing systems* 6700–6710.
6. Arjovsky, Martin, and Léon Bottou. 2017. Towards principled methods for training generative adversarial networks. arXiv:1701.04862.
7. Azab, Ahmad, Mamoun Alazab, and Mahdi Aiash. 2016. Machine learning based botnet identification traffic. In *2016 IEEE Trustcom/BigDataSE/ISPA*, 1788–1794. IEEE.
8. Azab, Ahmad, Robert Layton, Mamoun Alazab, and Jonathan Oliver. 2014. Mining malware to detect variants. In *2014 fifth cybercrime and trustworthy computing conference*, 44–53. IEEE.
9. Benchea, Răzvan, and Dragoş Teodor Gavriluţ. 2014. Combining restricted boltzmann machine and one side perceptron for malware detection. In *International conference on conceptual structures*, 93–103. Springer.
10. Biggio, Battista, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, 387–402. Springer.
11. Biggio, Battista, Paolo Russu, Luca Didaci, Fabio Roli, et al. 2015. Adversarial biometric recognition: A review on biometric system security from the adversarial machine-learning perspective. *IEEE Signal Processing Magazine* 32 (5): 31–41.
12. Brown, Tom B., Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. 2017. Adversarial patch. arXiv:1712.09665.
13. Carlini, Nicholas, and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE symposium on security and privacy (sp)*, 39–57. IEEE.
14. Chen, Liang-Chieh, George Papandreou, Florian Schroff, and Hartwig Adam. 2017. Rethinking atrous convolution for semantic image segmentation. arXiv:1706.05587.
15. Chen, Pin-Yu, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, 15–26.
16. Chen, Xinyun, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. arXiv:1712.05526.
17. Damodaran, Anusha, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H. Austin, and Mark Stamp. 2017. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques* 13 (1): 1–12.
18. Dziugaite, Gintare Karolina, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. arXiv:1505.03906.
19. Firdausi, Ivan, Alva Erwin, Anto Satriyo Nugroho, et al. 2010. Analysis of machine learning techniques used in behavior-based malware detection. In *2010 second international conference on advances in computing, control, and telecommunication technologies*, 201–203. IEEE.
20. Gibert, Daniel. 2016. *Convolutional neural networks for malware classification*. Tarragona, Spain: University Rovira i Virgili.
21. Goodfellow, Ian, Patrick McDaniel, and Nicolas Papernot. 2018. Making machine learning robust against adversarial inputs. *Communications of the ACM* 61 (7): 56–66.
22. Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. arXiv:1412.6572.
23. Grosse, Kathrin, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *European symposium on research in computer security*, 62–79. Springer.
24. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In*Proceedings of the IEEE international conference on computer vision* 1026–1034.

25. Hu, Weiwei, and Ying Tan. 2017. Generating adversarial malware examples for black-box attacks based on gan. arXiv:1702.05983.
26. Huang, Ling, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 43–58.
27. Jung, Wookhyun, Sangwon Kim, and Sangyong Choi. 2015. Poster: deep learning for zero-day flash malware detection. In *36th IEEE symposium on security and privacy*, vol. 10, 2809695–2817880.
28. Kawai, Masataka, Kaoru Ota, and Mianxing Dong. 2019. Improved malgan: Avoiding malware detector by leaning cleanware features. In *2019 international conference on artificial intelligence in information and communication (ICAIIC)*, 040–045. IEEE.
29. Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems* 3146–3154.
30. Kolosnjaji, Bojan, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European signal processing conference (EUSIPCO)*, 533–537. IEEE.
31. Kolosnjaji, Bojan, Apostolis Zarras, George Webster, and Claudia Eckert. 2016. Deep learning for classification of malware system call sequences. In *Australasian joint conference on artificial intelligence*, 137–149. Springer.
32. Kreuk, Felix, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. 2018. Deceiving end-to-end deep learning malware detectors using adversarial examples. arXiv:1802.04528.
33. Moosavi-Dezfooli, Seyed-Mohsen, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In*Proceedings of the IEEE conference on computer vision and pattern recognition* 2574–2582.
34. Muñoz-González, Luis, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, 27–38.
35. Obeis, Turki, and Wesam Bhaya Nawfal. 2016. Review of data mining techniques for malicious detetion. *Research Journal of Applied Sciences* 11 (10): 942–947.
36. Oyama, Yoshihiro, Takumi Miyashita, and Hirotaka Kokubo. 2019. Identifying useful features for malware detection in the ember dataset. In *2019 seventh international symposium on computing and networking workshops (CANDARW)*, 360–366. IEEE.
37. Papernot, Nicolas, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv:1605.07277.
38. Papernot, Nicolas, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 506–519.
39. Pascanu, Razvan, Jack W Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. 2015. Malware classification with recurrent networks. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 1916–1920. IEEE.
40. Pendlebury, Feargus, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 729–746.
41. Pietrek, Matt. 2002. Inside windows-an in-depth look into the win32 portable executable file format. *MSDN Magazine* 17 (2): 80–90.
42. Puranik, Piyush Aniruddha. 2019. Static malware detection using deep neural networks on portable executables.
43. Raff, Edward, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. 2017. Malware detection by eating a whole exe. arXiv:1710.09435.

44. Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* 779–788.
45. Saxe, Joshua, and Konstantin Berlin. 2015. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th international conference on malicious and unwanted software (MALWARE)*, 11–20. IEEE.
46. Su, Jiawei, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation* 23 (5): 828–841.
47. Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. arXiv:1312.6199.
48. Ucci, Daniele, Leonardo Aniello, and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Computers & Security* 81: 123–147.
49. Ye, Yanfang, Tao Li, S. Donald Adjeroh, and Sitharama, and Iyengar. 2017. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)* 50 (3): 1–40.
50. Yuan, Xiaoyong, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems* 30 (9): 2805–2824.
51. Zhang, Jinlan, Qiao Yan, and Mingde Wang. 2019. Evasion attacks based on wasserstein generative adversarial network. In *2019 Computing, communications and IoT applications (ComComAp)*, 454–459. IEEE.
52. Zhong, Wei, and Gu Feng. 2019. A multi-level deep learning system for malware detection. *Expert Systems with Applications* 133: 151–162.