Adversarial Deep Learning for Robust Detection of Binary Encoded Malware

Abdullah Al-Dujaili
CSAIL, MIT
Cambridge, USA
aldujail@mit.edu

Alex Huang
CSAIL, MIT
Cambridge, USA
alhuang@mit.edu

Erik Hemberg
CSAIL, MIT
Cambridge, USA
hembergerik@csail.mit.edu

Una-May O'Reilly
CSAIL, MIT
Cambridge, USA
unamay@csail.mit.edu

***Abstract*—Malware is constantly adapting in order to avoid detection. Model based malware detectors, such as SVM and neural networks, are vulnerable to so-called *adversarial examples* which are modest changes to detectable malware that allows the resulting malware to evade detection. Continuous-valued methods that are robust to adversarial examples of images have been developed using saddle-point optimization formulations. We are inspired by them to develop similar methods for the discrete, e.g. binary, domain which characterizes the features of malware. A specific extra challenge of malware is that the adversarial examples must be generated in a way that preserves their malicious functionality. We introduce methods capable of generating functionally preserved adversarial malware examples in the binary domain. Using the saddle-point formulation, we incorporate the adversarial examples into the training of models that are robust to them. We evaluate the effectiveness of the methods and others in the literature on a set of Portable Execution (PE) files. Comparison prompts our introduction of an online measure computed during training to assess general expectation of robustness.**

***Index Terms*—Neural Networks, Malware**

I. Introduction

Deep neural networks (DNN) started as extensions of neural networks in artificial intelligence approaches to computer vision and speech recognition. They are also used in computer security applications such as malware detection. A large challenge in developing malware detection models is the intelligent adversaries who actively try to evade them by judiciously perturbing the detectable malware to create what are called *Adversarial Examples* (AEs), i.e. malware variants that evade detection.

Much of the work done to understand and counter AEs has occurred in the image classification domain. An adversarial attack on an image classifier perturbs an image so that it is perceptually no different to a human but now classified incorrectly by the classifier. To counter them, researchers have demonstrated how DNN models can be trained more robustly. These methods assume a continuous input domain [18].

Our interest is malware detection where, in contrast to images, detectors often use features represented as binary $(0, 1)$ inputs. Malware AEs must not only fool the detector, they must also ensure that their perturbations do not alter the malicious payload. Our preliminary goal is to develop a method that, as is done in the continuous space, can generate (binary) perturbations of correctly classified malware that evade the detector. Our central goal is to investigate how the robust adversarial training methods for continuous domains can be transformed to serve the discrete or categorical feature domains that include malware. We can measure the effectiveness of a robust adversarial malware method on training a classifier by the evasion rate of AEs and we also seek an online training measure that expresses the general expectation of model robustness.

This leads to the following contributions at the intersection of security and adversarial machine learning: 1) We present 4 methods to generate binary-encoded AEs of malware with preserved malicious functionality 2) We present the SLEIPNIR framework for training robust adversarial malware detectors. SLEIPNIR employs saddle-point optimization (hence its name[1]) to learn malware detection models for executable files represented by binary-encoded features. 3) We demonstrate the framework on a set of Portable Executables (PEs), observing that incorporating randomization in the method is most effective. 4) We use the AEs of an adversarial crafting method [13] that does not conform to the saddle-point formulation of SLEIPNIR to evaluate the models from SLEIPNIR. We find that the model of the randomized method is also robust to them. 5) Finally, we provide the SLEIPNIR framework and dataset for public use.[2]

The paper is structured as follows. §.II presents background and related work. §.III describes the method. Experiments are in §.IV. Finally, conclusions are drawn and future work is outlined in §.V.

II. Background

Malware detection is moving away from hand-crafted rule-based approaches and towards machine learning techniques [25]. In this section we focus on malware detection with neural networks (§.II-A), adversarial machine learning (§.II-B) and adversarial malware versions (§.II-C).

A. Malware Detection using Neural Networks

Neural network methods for malware detection are increasingly being used. For features, one study combines DNN's with random projections [9] and another with two dimensional binary PE program features [24]. Research has also been

[1]https://en.wikipedia.org/wiki/Sleipnir
[2]https://github.com/ALFA-group/robust-adv-malware-detection.

DOI 10.1109/SPW.2018.00020

done on a variety of file types, such as Android and PE files [5], [16], [22], [29]. While the specifics can vary greatly, all machine learning approaches to malware detection share the same central vulnerability to AEs.

B. Adversarial Machine Learning

Finding effective techniques that robustly handle AEs is one focus of adversarial machine learning [6], [15]. An adversarial example is created by making a small, essentially non-detectable change to a data sample $\mathbf{x}$ to create $\mathbf{x}_{adv} = \mathbf{x} + \delta$. If the detector misclassifies $\mathbf{x}_{adv}$ despite having correctly classified $\mathbf{x}$, then $\mathbf{x}_{adv}$ is a successful adversarial example. Goodfellow *et al.* [11] provide a clear explanation for the existence of AEs.

There are a variety of techniques that generate AEs [11], [26]. One efficient and widely used technique is the fast gradient sign method (FGSM) [11]. With respect to an input, this method finds the directions that move the outputs of the neural network the greatest degree and moves the inputs along these directions by small amounts, or perturbations. Let $\mathbf{x}$ represent an input, θ the parameters of the model, y the labels, and $L(\theta, \mathbf{x}, y)$ be the associated loss generated by the network. Maintaining the restriction of ϵ-max perturbation, we can obtain a max-norm output change using $\eta = \epsilon \text{sgn}(\nabla_{\mathbf{x}} L(\theta, \mathbf{x}, y))$. Because the technique references the detector's parameters, it is known as a *white-box* attack model [8], [11], [20].

There have been multiple studies focused on advancing model performance against AEs, e.g. [19], [30]. One obvious approach is retraining with the AEs incorporated into the training set. We are attracted to the approach of [18]. It casts model learning as a robust optimization problem with a saddle-point formulation where the outer minimization of detector (defensive) loss is tied to the inner maximization of detector loss (via AEs) [18]. The approach successfully demonstrated robustness against adversarial images by incorporating, while training, AEs generated using projected gradient descent.

C. Adversarial Malware

Security researchers have generated malware AEs using an array of machine learning approaches such as reinforcement learning, genetic algorithms and supervised learning including neural networks, decision trees and SVM [5], [10], [12]–[14], [22], [24], [27], [28]. These approaches, with the exception of [12], [13], are black box. They assume no knowledge of the detector though the detector can be queried for detection decisions. Multiple studies use binary features, typically where each index acts as an indicator to express the presence or absence of an API call, e.g. [23]. One study also includes byte/entropy histogram features [24]. Studies to date have only retrained with AEs.

Uniquely, this work generates functional white-box AEs in the discrete, binary domain while incorporating them into the training of a malware classifier that is robust to AEs.

III. Method

To address the problem of hardening machine learning anti-malware detectors via adversarial learning, we formulate the adversarial learning procedure as a saddle-point problem in line with [18]. Before describing the problem formally and presenting our proposed approach to tackle the same, we introduce the notation and terminology used in the rest of the paper.

A. Notation

This paper considers a malware classification task with an underlying data distribution $\mathcal{D}$ over pairs of binary executable representations and their corresponding labels (i.e., benign or malignant). For brevity, we use *malicious binary executable* and *malware* interchangeably. We denote the representation space of the executables and their label space by $\mathcal{X}$ and $\mathcal{Y}$, respectively. Based on extracted static features, each binary executable is represented by a binary indicator vector $\mathbf{x} = [x_1, \ldots, x_m] \in \mathcal{X}$. That is, $\mathcal{X} = \{0,1\}^m$ and x_j is a binary value that indicates whether the jth feature is present or not. On the other hand, labels are denoted by $y \in \mathcal{Y} = \{0, 1\}$, where 0 and 1 denote benign and malignant executables, respectively. We would like to learn the parameters $\theta \in \mathbb{R}^p$ of a binary classifier model such that it correctly classifies samples drawn from $\mathcal{D}$. Typically, the model's performance is measured by a scalar loss function $L(\theta, \mathbf{x}, y)$ (e.g., the cross entropy loss). The task then is to find the optimal model parameters θ^* that minimize the risk $\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[L(\theta, \mathbf{x}, y)]$. Mathematically, we have

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

B. Malware Adversarial Learning as a Saddle Point Problem

Blind spots are regions in a model's decision space, on either side of the decision boundary, where, because no training example was provided, the decision boundary is inaccurate. Blind spots of malware detection models—such as the one learned in (1)—can be exploited to craft misclassified adversarial malware samples from a correctly classified malware, while still preserving malicious functionality. An adversarial malware version $\mathbf{x}_{adv}$ (which may or may not be misclassified) of a correctly classified malware $\mathbf{x}$ can be generated by perturbing $\mathbf{x}$ in a way that maximizes the loss L, i.e.,

$$\mathbf{x}_{adv} \in \mathcal{S}^*(\mathbf{x}) = \arg\max_{\bar{\mathbf{x}} \in \mathcal{S}(\mathbf{x})} L(\theta, \bar{\mathbf{x}}, y) \ , \tag{2}$$

where $\mathcal{S}(\mathbf{x}) \subseteq \mathcal{X}$ is the set of binary indicator vectors that preserve the functionality of malware $\mathbf{x}$, and $\mathcal{S}^*(\mathbf{x}) \subseteq \mathcal{S}(\mathbf{x})$ is the set of adversarial malware versions that maximize the adversarial loss.

To harden the model learned in (1) against the adversarial versions generated in (2), one needs to incorporate them into the learning process. We choose to do so by making use of the saddle-point formulation presented in [18]. Thus, our adversarial learning composes (1) and (2) as:

$$\theta^* \in \underbrace{\arg\min_{\theta \in \mathbb{R}^p} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\overbrace{\max_{\bar{\mathbf{x}} \in \mathcal{S}(\mathbf{x})} L(\theta, \bar{\mathbf{x}}, y)}^{\text{adversarial loss}} \right]}_{\text{adversarial learning}} \ . \tag{3}$$

Solving (3) involves an inner non-concave maximization problem and an outer non-convex minimization problem. Nevertheless, this formulation is particularly interesting because of two reasons. First, in the case of a continuous differentiable loss function (in the model parameters θ), Danskin's theorem states that gradients at inner maximizers correspond to descent directions for the saddle-point problem—see [18] for a formal proof. Second, it has been shown empirically that one can still reliably optimize the saddle-point problem for learning tasks with continuous feature space—i.e., $\mathcal{X} \subseteq \mathbb{R}^m$—even with *i)* loss functions that are not continuously differentiable (e.g., ReLU units); and *ii)* using gradients at approximate maximizers of the inner problem [18]. To find these maximizers, prior work has used variants of projected gradient descent on the negative loss function such as the Fast Gradient Sign Method (`FGSM`) [11] and its multi-step variant `FGSM`^{k} [17]. Finding maximizers (or approximations) of the inner problem for a given malware involves moving from continuous to constrained binary optimization: flipping bits of the malware's binary feature vector $\mathbf{x}$ while preserving its functionality.

C. Adapting Gradient-Based Inner Maximization Methods for Binary Feature Spaces

Our malware-suited methods step off from the empirical success of gradient-based methods like `FGSM`^{k} in approximating inner maximizers for continuous feature spaces [18]. The bulk of prior work has focused on adversarial attacks against images. In such setups, pixel-level perturbations are often constrained to ℓ_∞-ball around the image at hand [11]. In the case of malware, perturbations that preserve malicious functionality correspond to setting unset bits in the binary feature vector $\mathbf{x}$ of the malware at hand. As depicted in Fig. 1 (a), we can only add features that are not present in the binary executable and never remove those otherwise. Thus, $\mathcal{S}(\mathbf{x}) = \{\bar{\mathbf{x}} \in \{0,1\}^m \mid \mathbf{x} \wedge \bar{\mathbf{x}} = \mathbf{x}\}$ and $|\mathcal{S}(\mathbf{x})| = 2^{m - \mathbf{x}^T\mathbf{1}}$. One could incorporate all the adversarial malware versions in the training through brute force enumeration but they grow exponentially in number and blind spots could be redundantly visited. On the other hand, with gradient-based methods, we aim to introduce adversarial malware versions in an online manner based on their difficulty in terms of model accuracy.

In the continuous space of images, projected gradient descent can be used to incorporate the ℓ_∞-ball constraint (e.g., the `Clip` operator [17]). Inspired by linear programming relaxation and rounding schemes for integer problems, we extend the projection operator to make use of gradient-based methods for the malware binary space via **deterministic** or **randomized rounding** giving rise to two discrete, binary-encoded, constraint-based variants of `FGSM`^{k}, namely **`dFGSM`^{k}** and **`rFGSM`^{k}**, respectively. It is interesting to note that `MalGAN`'s black-box system [14] used a deterministic rounding scheme—with $\alpha = 0.5$—to craft adversarial malware versions.

With `FGSM`^{k} in continuous space, AEs are generated by moving iteratively in the feasible space (e.g., the ℓ_∞-ball around around an image). In contrast, the crafted adversarial malware versions are situated at the vertices of the binary feature space. Instead of multi-stepping through the continuous space to generate just one adversarial malware version (i.e., `dFGSM`^{k} or `rFGSM`^{k}), we can use the gradient to visit multiple feasible vertices (i.e., adversarial malware versions) and choose the one with the maximum loss, see Fig. 1 (a). This suggests a third method: **multi-step Bit Gradient Ascent (`BGA`^{k})**, see Fig. 1 (b). This method sets the bit of the jth feature if the corresponding partial derivative of the loss is greater than or equal to the loss gradient's ℓ_2-norm divided by $\sqrt{m}$. The rationale behind this is that the projection of a unit vector with equal components onto any coordinate equals $1/\sqrt{m}$. Therefore we set bits (features) whose corresponding partial derivative contribute more or equally to the ℓ_2-norm of the gradient in comparison to the rest of the features. After k steps, the binary indicator vector that corresponds to the vertex with the maximum loss among the visited vertices is chosen as the adversarial malware version. A final method: **multi-step Bit Coordinate Ascent (`BCA`^{k})** updates one bit in each step by considering the feature with the maximum corresponding partial derivative of the loss. A similar approach has been shown effective for Android malware evasion in [12], [13]. Table I presents a formal definition of the methods. In the next section, we propose a metric to measure their effectiveness in covering the model's blind spots.

D. Blind Spots Coverage

With adversarial learning, we aim to discover and address blind spots of the model while learning its parameters simultaneously. In other words, we would like to incorporate as many members of $\mathcal{S}^*(\mathbf{x})$ as possible in training the model. In line with this notion, we propose a new measure called the **blind spots covering number**, denoted $\mathcal{N}_{BS}$, which measures

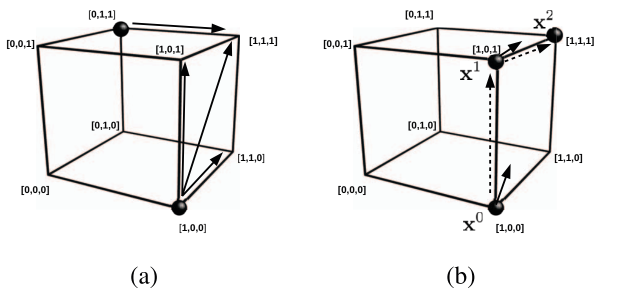


Fig. 1. (a) Two malicious binary executables (malwares) in the 3-dimensional binary indicator vector space. The set of adversarial malware versions for the malware at $[1,0,0]$ is $\mathcal{S}([1,0,0]) = \{[1,0,0],[1,1,0],[1,0,1],[1,1,1]\}$, and for the malware at $[0,1,1]$ is $\mathcal{S}([0,1,1]) = \{[0,1,1],[1,1,1]\}$. The arrows point to the set of allowed perturbations. (b) Two-step bit gradient ascent (`BGA`^{2}). The solid arrows represent the loss gradient at the arrows end points, while the dashed arrows represent the bit gradient ascent update step. At step 1, the contribution to the magnitude of the loss gradient (ℓ_2-norm) is predominantly towards setting the 3rd feature. Thus, $\mathbf{x}_1$ is obtained by setting $\mathbf{x}_0$'s 3rd bit. Similarly, $\mathbf{x}_2$ is obtained by setting $\mathbf{x}_1$'s 2nd bit. After visiting 2 vertices besides our starting vertex, we choose $\arg\max_{\mathbf{x} \in \{\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2\}} L(\theta, \mathbf{x}, 1)$ as the adversarial malware version of $\mathbf{x}^0$. Note that this a special case of `BGA`^{k}, where only one bit is set at a step. Two-step bit coordinate ascent (`BCA`^{2}) would generate the same adversarial malware version.

TABLE I
Proposed inner maximizers for the saddle-point problem (3).

Considered inner maximization methods for crafting adversarial versions of a malware given its binary indicator vector $\mathbf{x}$. Denote $\partial L(\theta, \mathbf{x}^t, y)/\partial x_j^t$ by $\partial_{x_j^t} L$, the adversarial malware version by $\bar{\mathbf{x}}^k$, the projection operator into the interval $[a, b]$ by $\Pi_{[a,b]}$ such that $\Pi_{[a,b]} = \max(\min(x, b), a)$, the OR operator by $\vee$, and the XOR operator by $\oplus$. Furthermore, for all the methods, the initial starting point $\mathbf{x}^0$ can be any point from $\mathcal{S}(\mathbf{x})$, i.e., $\mathbf{x}^0 \in \mathcal{S}(\mathbf{x})$. In our setup, $\mathbf{x}^0$ is set to $\mathbf{x}$. For $\texttt{BGA}^k$ and $\texttt{BCA}^k$, the $(1 - 2x_j^t)$ term is used to enforce that the gradient is towards 0 if $x_j^t = 1$, and vice versa.

Method	**Definition**
$\texttt{FGSM}^k$ with deterministic rounding ($\texttt{dFGSM}^k$)	$x_j^{t+1} = \Pi_{[0,1]}\left(x_j^t + \epsilon \text{sgn}(\partial_{x_j^t} L)\right)\ ,\ 0 \leq j < m\ ,\ 0 \leq t < k$ $\bar{x}_j^k = \mathbf{1}\left\{x_j^k > \alpha\right\} \vee x_j\ ,\ \alpha \in [0,1]\ ,\ 0 \leq j < m$ $\mathbf{x}_{adv} \in \arg\max\{L(\theta, \mathbf{x}^*, 1) \mid \mathbf{x}^* \in \{\bar{\mathbf{x}}^k, \mathbf{x}\}\}$
$\texttt{FGSM}^k$ with randomized rounding ($\texttt{rFGSM}^k$)	$x_j^{t+1} = \Pi_{[0,1]}\left(x_j^t + \epsilon \text{sgn}(\partial_{x_j^t} L)\right)\ ,\ 0 \leq j < m\ ,\ 0 \leq t < k$ $\bar{x}_j^k = \mathbf{1}\left\{x_j^k > \alpha_j\right\} \vee x_j\ ,\ \alpha_j \in \mathcal{U}(0,1)\ ,\ 0 \leq j < m$ $\mathbf{x}_{adv} \in \arg\max\{L(\theta, \mathbf{x}^*, 1) \mid \mathbf{x}^* \in \{\bar{\mathbf{x}}^k, \mathbf{x}\}\}$
Multi-Step Bit Gradient Ascent ($\texttt{BGA}^k$)	$x_j^{t+1} = \left(x_j^t \oplus \mathbf{1}\left\{(1 - 2x_j^t)\ \partial_{x_j^t} L \geq \frac{1}{\sqrt{m}} \lvert\lvert \nabla_{\mathbf{x}} L(\theta, \mathbf{x}^t, y) \rvert\rvert_2\right\}\right) \vee x_j\ ,\ 0 \leq j < m\ ,\ 0 \leq t < k$ $\mathbf{x}_{adv} \in \arg\max\{L(\theta, \mathbf{x}^*, 1) \mid \mathbf{x}^* \in \{\mathbf{x}^t\}_{0 \leq t \leq k} \cup \{\mathbf{x}\}\}$
Multi-Step Bit Coordinate Ascent ($\texttt{BCA}^k$)	$j^{t+1} \in \arg\max_{1 \leq j \leq m} (1 - 2x_j^t)\ \partial_{x_j^t} L$ $x_j^{t+1} = (x_j^t \oplus \mathbf{1}\{j = j^{t+1}\}) \vee x_j\ ,\ 0 \leq j < m\ ,\ 0 \leq t < k$ $\mathbf{x}_{adv} \in \arg\max\{L(\theta, \mathbf{x}^*, 1) \mid \mathbf{x}^* \in \{\mathbf{x}^t\}_{0 \leq t \leq k} \cup \{\mathbf{x}\}\}$

the effectiveness of an algorithm $\mathcal{A}$ in computing the inner maximizers of (3). The measure is defined as the expected ratio of the number of adversarial malware versions crafted by $\mathcal{A}$ during training, denoted by $\mathcal{S}^*_{\mathcal{A}}(\mathbf{x})$, to the maximum possible number of the same. Formally, it can be written as follows.

$$\mathcal{N}_{BS}(\mathcal{A}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\frac{y|\mathcal{S}^*_{\mathcal{A}}(\mathbf{x})|}{2^{m-\mathbf{x}^T\mathbf{1}}}\right] \tag{4}$$

Models trained with high $\mathcal{N}_{BS}$ have seen more AEs in training, and because training against multiple AEs implies more exhaustive approximations of the inner maximization problem, they are expected to be more robust against adversarial attacks [18]. While it may be computationally expensive to compute (4) exactly, we provide a probabilistic approximation of it in §.IV-B.

E. Adversarial Learning Framework

Having specified four methods for approximating the inner maximizers of (3) and a measure of their effectiveness, we can now describe SLEIPNIR, our adversarial learning framework for robust malware detection. Consider a training dataset D of n independent and identically distributed samples drawn from $\mathcal{D}$. As outlined in Algorithm 1 and depicted in Fig. 2, SLEIPNIR groups D into minibatches B of s examples similar to [17]. However, the grouping here is governed by the examples' labels: the first $r < s$ examples are malicious, followed by $s - r$ benign examples. At each training step, the model's parameters θ are optimized with respect to the adversarial loss (2) of malware executables and the natural loss (1) of benign executables. This is motivated by the fact that authors of benign applications have no interest in having their binaries misclassified as malwares [12]. However, one should note that a malware author might wish to create adversarial benign applications to poison the training dataset. This possibility is considered for future work. As an equation, our empirical saddle-point problem at each training step has the form

$$\min_{\theta\in\mathbb{R}^p} \frac{1}{s}\left[\sum_{i=1}^{r} \max_{\bar{\mathbf{x}}^{(i)}\in\mathcal{S}(\mathbf{x}^{(i)})} L(\theta, \bar{\mathbf{x}}^{(i)}, 1) + \sum_{i=r+1}^{s} L(\theta, \mathbf{x}^{(i)}, 0)\right]. \tag{5}$$

IV. EXPERIMENTS

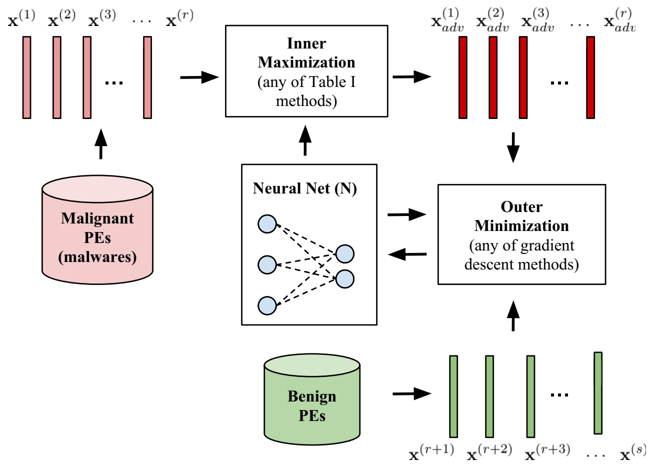
This section provides an empirical evaluation of our proposition in §.III. We conduct experiments to validate and compare the efficacy of the proposed methods in terms of classification accuracy, evasion rates, and blind spots coverage. First, the setup of our experiments is described in §.IV-A, followed by a presentation of the results in §.IV-B.

A. Setup

Dataset. The Portable Executable (PE) format [2] is a file format for executables in Windows operating systems. The format encapsulates information necessary for Windows OS to manage the wrapped code. PE files have widespread use as malware. We created a corpus of malicious and benign PE files from VirusShare [3] and internet download sites, respectively.

Algorithm 1 SLEIPNIR

```
Input:
    $N$ : neural network model, $D$ : training dataset,
    $s$ : minibatch size, $r$ : number of malwares in minibatch,
    $\mathcal{A}$ : inner maximizer algorithm (any of Table I)
1: Randomly initialize network $N$
2: repeat
3:     Read minibatch $B$ from dataset $D$
           $B = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(s)} \mid y^{i \leq r} = 1,\ y^{i > r} = 0\}$
4:     Generate $r$ adversarial versions $\{\mathbf{x}^{(1)}_{adv}, \ldots, \mathbf{x}^{(r)}_{adv}\}$
       from feasible sets of corresponding malware examples
       $\{\mathcal{S}(\mathbf{x}^{(1)}), \ldots, \mathcal{S}(\mathbf{x}^{(r)})\}$ by $\mathcal{A}$ using current state of $N$
5:     Make new minibatch
           $B' = \{\mathbf{x}^{(1)}_{adv}, \ldots, \mathbf{x}^{(r)}_{adv}, \mathbf{x}^{(r+1)}, \ldots, \mathbf{x}^{(s)}\}$
6:     Do one training step of network $N$ with minibatch $B'$
7: until training converged
```



$\mathbf{x}^{(1)}$ $\mathbf{x}^{(2)}$ $\mathbf{x}^{(3)}$ … $\mathbf{x}^{(r)}$
Inner Maximization (any of Table I methods)
$\mathbf{x}^{(1)}_{adv}$ $\mathbf{x}^{(2)}_{adv}$ $\mathbf{x}^{(3)}_{adv}$ … $\mathbf{x}^{(r)}_{adv}$
Malignant PEs (malwares)
Neural Net (N)
Outer Minimization (any of gradient descent methods)
Benign PEs
$\mathbf{x}^{(r+1)}$ $\mathbf{x}^{(r+2)}$ $\mathbf{x}^{(r+3)}$ … $\mathbf{x}^{(s)}$

Fig. 2. Overview of the SLEIPNIR framework. Malware is perturbed by an inner maximization method (any of Table I) to create AEs. The generated adversarial malware versions and benign examples are used in an outer minimization of the adversarial and natural loss (5), which can be solved in minibatches using any variant of the gradient descent algorithm.

To label the collected PEs, we use VirusTotal's [4] ensemble of virus detectors. We require benign files to have 0% positive detections from the ensemble and malicious files to have greater than 50% positive detections to avoid false positives. At the time of writing this paper, we have 34,995 malicious and 19,696 benign PEs.

Feature Representation. As mentioned earlier, each portable executable is represented as a binary indicator feature vector. Each index of the feature vector represents a unique Windows API call and a "1" in a location represents the presence of the corresponding API call. In our dataset of PEs, we found a total of 22,761 unique API calls. Thus, each PE file is represented by a binary indicator vector $\mathbf{x} \in \{0,1\}^m$, with $m = 22,761$. We use the `LIEF` [1] library to parse each PE and turn it into its representative binary feature vector.[3]

Neural Net (N) Architecture. We use a feed-forward network for our malware classifier N with 3 hidden layers of 300 neurons each. The ReLU activation function is applied to all the 3×300 hidden neurons. The LogSoftMax function is applied to the output layer's two neurons which correspond to the two labels at hand: benign and malicious. The model is implemented in `PyTorch` [21].

Learning Setup. We use $19,000$ benign PEs and $19,000$ malicious PEs to construct our training (60%), validation (20%), and test (20%) sets. The training set is grouped into minibatches of 16 PE samples according to Line 3 of Algorithm 1. The classifier N's parameters θ are tuned with respect to (5), where L is the negative log likelihood loss, using the `ADAM` optimization algorithm with a 0.001 learning rate over 150 epochs. Note that one step of `ADAM` corresponds to Line 6 of Algorithm 1. To avoid overfitting, model parameters at the minimum validation loss are used as the final learned parameters θ^*. With regard to the inner maximizers algorithms (Table I), all were set to perform 50 steps, i.e., $k = 50$. This makes the step size ϵ for $\texttt{dFGSM}^k$ and $\texttt{rFGSM}^k$ small enough (we set it to $\epsilon = 0.02$) to follow the gradient accurately while also ensuring that multi-steps could reach close to other vertices of the binary feature space (Fig. 1) and not be suppressed by rounding. With 50 steps and 0.02 step size, both these conditions are met. We run Algorithm 1 with $\mathcal{A}$ being set to each of the inner maximizers from Table I to obtain 4 adversarially trained models in addition to the model trained naturally. We also used the adversarial sample crafting method presented by Grosse *et al.* [13, Algorithm 1] which trains a model adversarially without using a saddle-point formulation: the AEs in [13] are tuned with respect to the value of the benign output neuron rather than the loss L. Though not directly, this does maximize the adversarial loss value. All experiments for the six models were run on a CUDA-enabled GTX 1080 Ti GPU.

B. Results

For brevity, we refer to the trained models by their inner maximizer methods. Experiment results are presented in Tables II and III as follows.

Classification Performance. Based on Table II, all the adversarially trained models achieve a classification accuracy comparable to the naturally trained counterpart. However, we observe that models trained using inner maximizers of Table I tend to have higher false positive rate (FPR) and lower false negative rate (FNR)—positive denotes malicious. The FPR increase can be explained by the transforming of malware samples when models are trained adversarially. Such transformations could turn malware feature vectors into ones more similar to those of the benign population, and subsequently the benign test set. Likewise, the FNR decrease can be attributed to the adversarial versions boosting the model's confidence on vertices with less original malicious samples compared to

[3]The generated feature vectors are available by request.

the benign samples. With [13]'s method, it is the other way around. Arguably, the reason is that its adversarial objective is to maximize just the benign (negative) neuron's output and it is indifferent to the malicious (positive) neuron. As a result, the crafted adversarial malware version does not necessarily end up at a vertex at which the model's confidence, with respect to the malicious label, is low, which consequently improves the FPR and worsens the FNR.

Robustness to Evasion Attacks. We tried the adversarial attackers generated by the inner maximizers and [13]'s method as inputs to each of the trained models to assess their robustness against the adversaries generated during training as well as other adversaries. It can be seen in Table III that $\texttt{rFGSM}^k$ is our most successful adversarial training method, achieving relatively low evasion rates across all attack methods. As expected, all training methods are resistant to attacks using the same method, but each method aside from $\texttt{rFGSM}^k$ has at least one adversarial method that it performs poorly against. Evasion rates for `Natural` training, which uses non-altered malicious samples, provide a baseline for comparison.

Blind Spots Coverage. Given the high-dimension feature vectors and the sizeable dataset, it was computationally expensive to compute $\mathcal{N}_{BS}$ exactly. Instead, we computed an approximate probabilistic measure $\bar{\mathcal{N}}_{BS}$ using a Bloom filter [7]. The computed measures are presented in the last column of Table II as the ratio of total adversarial malware versions to original samples over all the training epochs. `Natural` training has a ratio of 1.0 since we do not modify the malicious samples in any way. A coverage value of 4.0 for $\texttt{rFGSM}^k$ means that with *high probability* we explored 4 times as many malicious samples compared to `Natural` training. A high coverage value indicates that the adversarial training explored more of the valid region $\mathcal{S}(\mathbf{x})$ for malware sample $\mathbf{x}$, resulting in a more robust model. This observation is substantiated by the correlation between coverage values in Table II and evasion rates in Table III. Note that $\bar{\mathcal{N}}_{BS}$ is computed and updated after each training step. Thus, it can be used as an online measure to assess training methods' robustness to adversarial attacks.

TABLE II
PERFORMANCE METRICS OF THE TRAINED MODELS.

In percentage, **Accuracy**, False Positive Rate (**FPR**), and False Negative Rate (**FNR**) are of the test set: 3800 malicious PEs and 3800 bengin PEs, with $k = 50$. $\bar{\mathcal{N}}_{BS}$ denotes the probabilistic normalized measure computed during training to approximate the blind spots covering number $\mathcal{N}_{BS}$. This was obtained using a Bloom filter to track the number of distinct malware samples presented during training, be they from the original malware training samples or their adversarial versions. Models corresponding to bold cells are the best with regard to the corresponding measure/rate. The measures of the inner maximizers and [13]'s are reported in their relative difference to `Natural`'s counterparts.

Model	Accuracy	FPR	FNR	$\bar{\mathcal{N}}_{\mathbf{BS}}$
Natural	91.9	8.2	8.1	1.0
$\texttt{dFGSM}^k$	$+0.1$	$+1.4$	-1.7	$+1.6$
$\texttt{rFGSM}^k$	-0.6	$+3.6$	$\mathbf{-2.4}$	$\mathbf{+3.0}$
$\texttt{BGA}^k$	$\mathbf{+0.2}$	$+0.0$	-0.5	$+2.5$
$\texttt{BCA}^k$	-0.3	$+0.9$	-0.5	$+0.0$
[13]'s method	-1.1	$\mathbf{-3.9}$	$+5.9$	$+0.6$

TABLE III
EVASION RATES.

Evasion rates of adversaries on the test set against the trained models with $k = 50$. Models corresponding to bold cells are the most robust models with regard to the corresponding adversary. Adversaries corresponding to shaded cells are the (or one of the) most successful adversaries with regard to the corresponding model. Evasion rates of the proposed inner maximizers are the lowest on their corresponding expected adversary after the `Natural` adversary as shown by the corresponding framed cells along the diagonal. This conforms to their saddle-point formulation, in contrast to [13]'s method with $\texttt{BCA}^k$ being its weakest adversary after the `Nautral` adversary, as framed below. This is expected as training with [13]'s method does not follow an exact saddle-point formulation.

Model	Adversary					
	Natural	$\texttt{dFGSM}^k$	$\texttt{rFGSM}^k$	$\texttt{BGA}^k$	$\texttt{BCA}^k$	**[13]'s method**
Natural	8.1	99.7	99.7	99.7	41.7	99.7
$\texttt{dFGSM}^k$	6.4	$\boxed{6.4}$	21.1	7.3	27.4	99.2
$\texttt{rFGSM}^k$	**5.7**	7.0	$\boxed{\mathbf{5.9}}$	**5.9**	**6.8**	35.0
$\texttt{BGA}^k$	7.6	39.6	17.8	$\boxed{7.6}$	10.9	68.4
$\texttt{BCA}^k$	7.6	99.5	99.5	91.8	$\boxed{7.9}$	98.6
[13] 's method	14.0	69.3	69.3	37.5	$\boxed{14.1}$	**15.6**

V. CONCLUSIONS AND FUTURE WORK

We investigated methods that reduce the adversarial blind spots for neural network malware detectors. We approached this as a saddle-point optimization problem in the binary domain and used this to train DNNs via multiple inner maximization methods that are robust to adversarial malware versions of the dataset.

We used a dataset of PE files to assess the robustness against evasion attacks. Our experiments have demonstrated once again the power of randomization in addressing challenging problems, conforming to the conclusions provided by state-of-art attack papers [8]. Equipping projected gradient descent with randomness in rounding helped uncover roughly 4 times as many malicious samples in the binary feature space as those uncovered in natural training. This performance correlated with the online measure we introduced to assess the general expectation of robustness.

There are several future research questions. First, we would like to study the loss landscape of the adversarial malware versions and the effect of starting point $\mathbf{x}^0$ initialization for inner maximizers, in comparison to their continuous-domain counterparts. Second, $\mathcal{N}_{BS}$ quantifies how many different adversarial examples are generated but it does not capture how they are located with regard to the benign examples and, subsequently, their effect on the model's FPR and FNR. We hope that investigating these directions will lead towards fully resistant deep learning models for malware detection.

ACKNOWLEDGMENT

This work was supported by the MIT-IBM Watson AI Lab and CSAIL CyberSecurity Initiative.

References

[1] LIEF - Library to Instrument Executable Formats - Quarkslab. https://lief.quarkslab.com/. Accessed: 2018-01-05.

[2] PE Format (Windows). https://msdn.microsoft.com/en-us/library/windows/desktop/ms680547. Accessed: 2018-01-05.

[3] VirusShare.com. https://virusshare.com/. Accessed: 2018-01-05.

[4] VirusTotal. https://www.virustotal.com. Accessed: 2018-01-05.

[5] Hyrum S Anderson, Anant Kharkar, Bobby Filar, and Phil Roth. Evading machine learning malware detection. 2017.

[6] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *arXiv preprint arXiv:1712.03141*, 2017.

[7] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[8] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

[9] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3422–3426. IEEE, 2013.

[10] Hung Dang, Yue Huang, and Ee-Chien Chang. Evading classifiers by morphing in the dark. In *ACM CCS*, volume 17, pages 119–133, 2017.

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[12] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016.

[13] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, pages 62–79. Springer, 2017.

[14] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on GAN. *arXiv preprint arXiv:1702.05983*, 2017.

[15] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[16] TonTon Hsien-De Huang, Chia-Mu Yu, and Hung-Yu Kao. R2-d2: Color-inspired convolutional neural network (cnn)-based android malware detections. *arXiv preprint arXiv:1705.04448*, 2017.

[17] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICML 2017 Workshop on Principled Approaches to Deep Learning*, 2017.

[19] Taesik Na, Jong Hwan Ko, and Saibal Mukhopadhyay. Cascade adversarial machine learning regularized with a unified embedding. *arXiv preprint arXiv:1708.02582*, 2017.

[20] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[21] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[22] Edward Raff, Jared Sylvester, and Charles Nicholas. Learning the pe header, malware detection with minimal domain knowledge. *arXiv preprint arXiv:1709.01471*, 2017.

[23] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against rnns and other api calls based malware classifiers. *arXiv preprint arXiv:1707.05970*, 2017.

[24] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*, pages 11–20. IEEE, 2015.

[25] Matthew G Schultz, Eleazar Eskin, F Zadok, and Salvatore J Stolfo. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49. IEEE, 2001.

[26] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[27] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium*, 2016.

[28] Wei Yang, Deguang Kong, Tao Xie, and Carl A Gunter. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In *Proc. ACSAC*, 2017.

[29] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 371–372. ACM, 2014.

[30] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. Efficient defenses against adversarial attacks. *arXiv preprint arXiv:1707.06728*, 2017.