Adversarial Approximation of a Black-Box Malware Detector

by

Abdullah Ali

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer and Information Science)
in the University of Michigan-Dearborn
2019

Master's Thesis Committee:

Assistant Professor Birhanu Eshete, Chair
Associate Professor Di Ma
Assistant Professor Probir Roy

2019

ACKNOWLEDGEMENTS

This work was done thanks to the efforts and immense help of Professor Birhanu Eshete, who I would like to thank for staying behind me throughout the semester. I would like to thank my wife Iman, who helped during the toughest moments of this work both emotionally and technically.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ML: Machine Learning

DNN: Deep Neural Network

CNN: Convolutional Neural Network

RNN: Recurrent Neural Network

EN: Entropy

CH: Color Hilbert

MLaaS: Machine Learning as-a-Service

API: Application Programming Interface

DoS: Denial of Service

ABSTRACT

A deployed machine learning-based malware detection model is effectively a black-box for an adversary whose objective is evading the model. In such a set- ting, the adversary has no access to details of the black-box except its prediction on a given input. With such limited leverage, the adversary has no choice but to explore avenues to infer the model's decision boundary, based on which adversarial inputs are crafted to evade it.

Inferring the best approximation of a black-box model's decision boundary is a non-trivial exercise for which an exact solution is unattainable. This is because there are exponentially many combinations of model architectures, parameters, and training examples to explore. In this context, the adversary prefers an optimal strategy that yields the best approximation of the black-box with minimal effort. This thesis presents a novel adversarial approximation approach for a black-box malware detector. Beginning with publicly accessible input-set for the black-box model, our approach leverages the recent advances in image transformation for deep neural networks and transferability of knowledge from publicly available pre-trained models to obtain an acceptable approximation of a black-box malware detector.

Experimental evaluation of our approach against a 93% black-box model trained on raw-byte sequence features of benign and malware Windows executables achieves up to 92% accurate approximator that leverages the Inception V3 pre- trained model. On a comparison dataset disjoint with the black-box's and the approximator's training sets, our approach achieved 90.1% similarity between the target black-box and the approximated model, showing the viability of our approach for approximation of black-box malware detectors with optimal effort.

CHAPTER 1: INTRODUCTION

Recent advances in Machine Learning (ML), such as deep neural networks (DNNs) have been demonstrated to achieve impressive accuracy on typical vision tasks, such as image classification and object recognition. The success of DNNs inspired adoption in other do- mains such as machine translation, speech processing, healthcare, and malware detection [3]. Despite their impressive accuracy, DNNs and other traditional machine learning techniques such as logistic regression, support vector machines, and decision trees have also been shown to be vulnerable to (training-time) poisoning —where an adversary modifies the decision boundary of a model and (test-time) evasion — where an adversary crafts an input that bypasses a model's decision boundary [4]. At training-time, an adversary injects training examples with the purpose of skewing a model towards a desired class. Once a model is deployed, an adversary probes it with carefully perturbed input instances that are intended to cause the model make prediction mistakes. These subtly perturbed variations of input instances are called adversarial examples [5, 6].

The real-life potential consequences of poisoning and evasion attacks are worrisome. A malware detector could be evaded to result in attacks that slip under the radar and make their way to critical infrastructure (e.g., power grids, nuclear reactors) and services (e.g., banks, hospitals). A road-side traffic sign detector of an autonomous vehicle could be mislead to make the wrong decisions that lead to traffic accidents. A medical image classifier could be misguided via adversarial perturbations to give the wrong diagnosis. A voice- based home assistant could be tricked to execute adversary-induced actions (e.g., open door for intruder). All these examples of adversarial

attacks on deployed ML-models are consequential and will be even more so as machine learning continues to be pervasively deployed in safety-, security-, privacy-critical settings.

Depending on the adversary's knowledge, evasion attacks may be white-box, black- box, or gray-box. In a white-box setting, the adversary has access to model architecture, parameters, features and training set. In a black-box setting, the adversary only knows the output (e.g. label = 'benign') for a given input (e.g., notepad.exe). Gray-box is an intermediate case where the adversary has some knowledge about model details.

In a black-box setting, an adversary may leverage publicly available training examples to approximate the target black-box model to pave the way for crafting adversarial examples to evade it. Getting the best approximation of a target black-box model is a non- trivial exercise for which an exact solution is unattainable. What combination of model architectures, parameters, and training examples is effective enough for the best-effort ap- proximation of the black-box model? How can an adversary begin with zero knowledge about the internals of a black-box model and reach to an acceptable substitute model? What strategy works when all the adversary has access to is a small set of samples that the black- box can accept? Any attempt to answer these questions comes down to some form of an approximation strategy on the side of the adversary.

To address the aforementioned questions, prior work has leveraged the notion of transferability [7, 8] in machine learning to approximate a black-box model. For instance, in the image domain, a substitute model is trained to fit a black-box image classifier, and the substitute model was then used in a white-box setting to craft adversarial examples [8] to evade the black-box model. In a similar vein, in [7], it is demonstrated that a generative adversarial network (GAN)-based model approximation enables crafting of adversarial examples that evade an API call-based malware detection model.

While prior work [7, 8, 9, 10] has explored interesting directions to approximate a black-box model, this thesis focuses on adversarial approximation of a black-box malware detector with minimal efforts desirable for the adversary. In pursuit of answers the questions raised earlier, an average-skilled adversary may adopt a brute-force approach to explore numerous combinations of model architectures, parameters, features and training data. A skilled adversary, on the other hand, is more likely to prefer a strategy that yields the best approximation of the black-box model while minimizing efforts. This thesis argues that beginning with publicly accessible input-set for the black-box model, a skilled adversary can leverage the representation and knowledge of publicly accessible pre-trained models to obtain an acceptable approximation of the black-box model with optimal efforts.

In particular, this thesis presents an adversarial approximation approach that leverages a curated training data of Windows executables (benign and malware), their equivalent image representation, and a pre-trained image classification model to approximate a black- box malware detector trained on byte-sequence features of executables. While prior work (e.g., [7, 8, 9]) mostly assumes similar feature sets for the black-box and the approximated model, this thesis, in the strict sense of black-box setting (hence most challenging for the adversary), assumes different feature representations of the black-box and the approximated model. In addition, unlike prior work ([7, 8, 9, 10]), this work ensures no overlap among the training data for the black-box model, the approximator training data, and the comparison dataset used to evaluate the similarity of the approximated model to the black- box model.

We evaluated our approach against a 93% accurate Convolutional Neural Network (CNN) black-box model [3] trained on raw-byte sequence features. Our approximation approach obtained up to 92% accurate CNN on features transformed from bytes to images and trained based on the

Inception V3 [11] pre-trained model. On a comparison dataset disjoint with the black-box's and the approximator's training sets, our approach achieved 90.1% similarity between the black-box and the approximated model. The results indicate that, even if the target model is a black-box, a skilled adversary can still take advantage of publicly available training data and the underlying knowledge of pre-trained models (Inception V3 in this case) to successfully approximate the decision boundary of a black-box model. An intriguing observation of our results is that, even if the adversary ends up obtaining training samples that don't overlap with the samples used to train the black-box model, the adversary can still achieve an acceptable approximation of the black-box model with minimal efforts. Another intriguing observation is even though prior work ([7, 8, 9]) assume and use the same feature representations for the black-box model and the approximated model, in our experiments, we intentionally explored the more challenging scenario for which the representation of the black-box is raw byte sequences and that of the approximated model is pixels. Interestingly, our approximation approach still managed to achieve above 90% similarity between the target black-box and the approximated model.

The rest of this thesis is organized as follows. Chapter 2 presents background on machine learning with focus on deep learning and attack surface in the learning pipeline. In chapter 3, we present discussion of related work and comparison of this work with closely related ones. Our approach is presented in chapter 4. The experimental evaluations appear in chapter 5. Chapter 6 concludes the thesis and highlights future research directions.

CHAPTER 2: BACKGROUND

In this chapter, we briefly introduce machine learning focusing on deep learning, followed by an overview of the machine learning attack surface.

2.1 Deep Neural Networks

Deep learning is a type of machine learning that is used to teach a model made of layered neurons how to identify a specific objective. A typical DNN has an input layer which is comprised of neurons –functions that accept input, process them, and produce output. These neurons store values extracted from the input given to the model. For example, passing an image to a DNN would mean that these neurons will each store a part of the image be it a pixel or a small concentration of pixels, then these inputs are moved into the hidden layers [12], interconnected neurons that will pass the information between each other and apply activation functions dictated by the engineers of the model. Each move in these hidden layers is accompanied by a weight that is calculated by the previous layer which will affect the information sent forward. This keeps going until we reach the last layer called the output layer.

The output layer will point us into the direction of the label it thinks our input is part of. Continuing with our image example, our input layer will have passed neuron information that represent the picture as a whole. This information will move through the hidden layers and get changed based on the weights that the network decided on during the training phase. This will help the model by the time it gets to the output layer to have a better idea of what label it thinks this image

is part of, and then apply the last activation function that will sum up the results of the last hidden layer and give us the prediction. These architectures can be very complicated depending on the specific task the DNN is used for, and one DNN architecture doesn't fit all tasks. In this thesis, we use a type of DNN called Convolutional Neural Network (CNN). This type is widely used for image recognition and serves the purpose of this thesis. The main difference between a DNN and CNN is that DNNs are fully connected networks, meaning that the hidden layers are all connected where each neuron in a layer will feed information to at least one neuron in the next layer. Whereas CNNs are made up of both fully connected layers as well as convolutional layers –layers that help a CNN detect patterns in an input (the very reason why a CNN is very effective with images). Each one of the convolutional layers will have filters, these filters are defined with a size (say 3x3). Now, the filter will slide over the input picture and capture all 3x3 pixel windows available in the image, and the output will be the results of this scan placed on a new layer as seen in Figure 2.1.

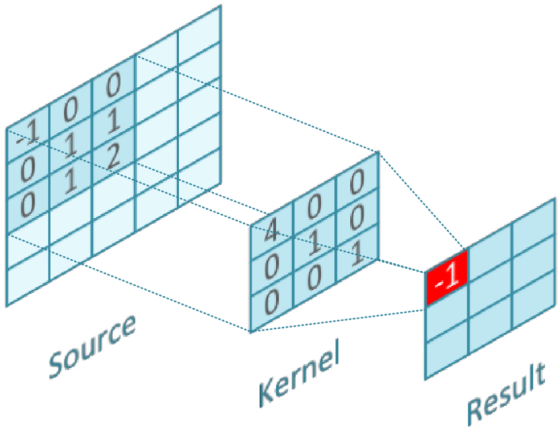


Figure 2.1: An illustration of how a convolutional layer filter works in a CNN (from [1]).

The filtering will happen in every convolutional layer in the model. When the input passes all of these layers it reaches a layer which flattens the results and pass it on to other layers which will begin the process of prediction as seen in Figure 2.2.

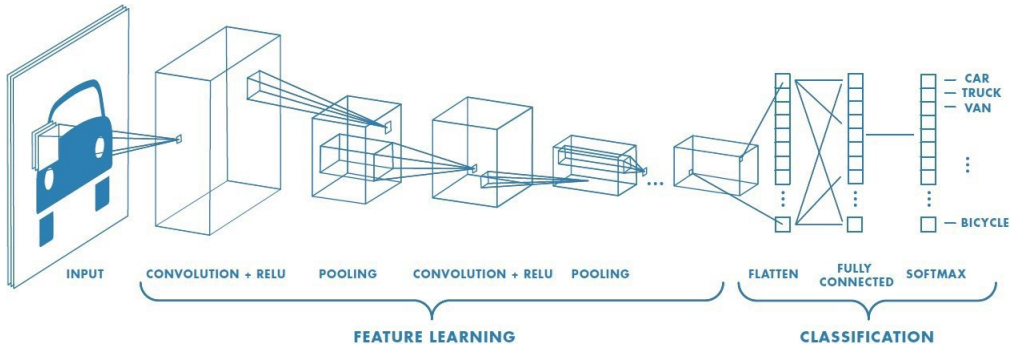


Figure 2.2: An illustration of a CNN architecture (from [2]).

2.2 Machine Learning Attack Surface

In the machine learning pipeline, the attack surface extends from the training data to the prediction output of the model. The surface is usually examined with respect to adversarial goals and capabilities (knowledge) [4].

2.2.1 Adversarial Goals

The adversary aims to violate the basic security properties of a machine learning system. These properties are confidentiality, integrity, and availability. When confidentiality is the target, the adversary typically aims to steal an already deployed model (e.g., intellectual property, top-secret business logic) [9]. For models trained on privacy-sensitive data (e.g., medical records), the adversary's goal is to perform membership inference attacks [13, 14] so as to determine whether or not a target individual has participated in the training data.

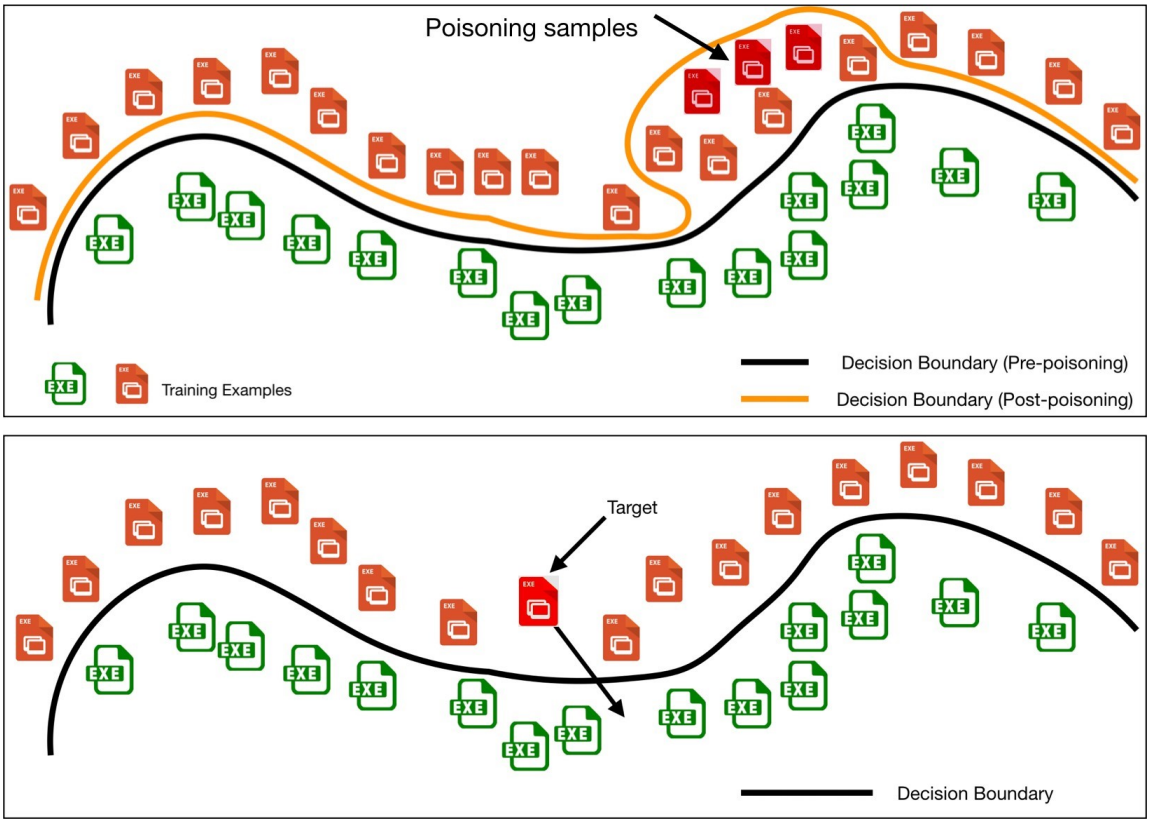


Figure 2.3: Illustration for training-time poisoning and test-time evasion.

An adversary aiming for integrity violations would have two broad targets –the training data or the deployed model (its predictions). By injecting training examples that advance the goals of the adversary, the resulting model after training could be made ready for future manipulations –this is called training-time poisoning [15]. For example, the adversary may carefully prepare inputs (training examples) that would change the decision boundary of the model in favor of the adversary's goals (e.g., insert backdoor or trojan in models). Another common motivation for poisoning is to skew the model's predictions towards a targeted or erroneous output that serves malicious intent (e.g., make malware to always bypass malware detection model).

Once a model is deployed, an adversary may also target the model's integrity and probe it with carefully perturbed input instances that are intended to cause the model make prediction mistakes –

this is called test-time evasion. Figure 2.3 illustrates training-time poisoning (top box) and test-time evasion (bottom box). These subtly perturbed variations of input instances are called adversarial examples [5, 6, 10, 16, 17, 18] (e.g., see "Target" sample in the bottom box of Figure 2.3). In image classifiers, an adversarial example could be crafted by applying minimal perturbation to produce visually imperceptible image, that can cause the model to make the wrong predictions (e.g., perturb a stop sign to be detected as a yield sign while it still looks like a stop sign to the human eye). For a malware classifier, an adversarial example could be created taking a malware sample that has been correctly classified as malware, and minimally perturbing it so that the classifier now mistakenly classifies it as benign. Note in the case of the malware, the perturbation need not break the malicious behavior of the sample.

Another adversarial target is the availability of ML model or the system that deploys it. The specific goal in this case is similar to classic Denial of Service (DoS) attacks. By over- whelming a deployed model (e.g, a remote prediction API) with a flood of prediction re- quests, the adversary can effectively deny access to legitimate users of the model. Imagine a remote medical image classifier used by physicians from multiple countries. If the classifier falls victim of a DoS attack, patients who depend on the results of the classifier would suffer. This threat model has become more relevant with the emerging Machine Learning as-a-Service (MLaaS) paradigm. Similar to distributed-DoS, adversaries can conspire to launch a coordinated DDoS attack against MLaaS platforms such as Amazon's ML[1] and Google's Cloud Prediction API[2].

[1] https://aws.amazon.com/machine-learning
[2] https://cloud.google.com/ai-platform/

2.2.2 Adversarial Capabilities

Whether the adversary's goal is training-time poisoning, test-time evasion, model stealing, membership inference, or DoS, the success of the attacks depend on adversarial capabilities what the adversary knows (has access to). The adversary may have different levels of knowledge about the training data, learning algorithm, model parameters/hyper- parameters, and feature set [4]. The more the adversary knows about the model details, the easier the attack, and vice-versa. In general, the adversarial knowledge can have three scenarios –white-box, black-box, and gray-box.

In a white-box (full knowledge) setting, the adversary knows the training data, details of the learning algorithm (e.g., whether it is SVM or DNN), parameters (e.g., weights and biases of a neural network), feature set (e.g., pixels for images, bytes for executable binaries, characters for natural language text). This setting is the easiest for the adversary, while the most exposing for the model.

In a black-box (zero/minimal knowledge) setting, the adversary has no (very limited) knowledge about the model. Typically the adversary only knows the prediction results of inputs. We note that the adversary may know a few more details that anyone would know. For example, the task the model is trained on (e.g., image classification, malware detection), possible features (e.g., pixels for images, API calls for executable binaries), and some training samples (e.g., access to a public image database or malware corpus). This is the most difficult setting for the adversary due to lack of access to the learning algorithm, model parameters, and features. One common strategy to fill the knowledge gap is for the adversary to repeatedly query the black-box model and use its prediction output as ground truth to train a substitute model that approximates the black-box model. This thesis operates in the black-box setting and for malware detector trained on raw bytes of Windows executables.

A gray-box (partial knowledge) setting is when the adversary has partial knowledge about the model, training data, and features. For instance, the adversary may only know a subset of training set and has some guess about features.

CHAPTER 3: RELATED WORK

This chapter discusses related work on black-box adversarial approximation of machine learning models, both in the image classification and malware detection domain.

3.1 Black-Box Approximation for Image Classifiers

A closely related work from the vision domain that we drew inspiration from is by Papernot *et al.* [12], where the goal is to use substitute models to approximate a CNN black-box model, and craft adversarial examples to evade the black-box. In this work, the authors assume that the adversary knows only two things: the original labelling of a sample that will be passed to the black-box and the black-box's labeling of the samples. The substitute model is trained by passing a batch of data based on the labeling of the black-box. Then they apply the Jacobian augmentation algorithm to synthetically generate more training examples for the next training iteration.

$$S_{\rho+1} = \overrightarrow{x} + \lambda_{\rho}\ \mathrm{sin}(J_f[O(\overrightarrow{x})] : \overrightarrow{x} ES_{\rho})\ \cup S_{\rho} \tag{3.1}$$

Equation 3.1 is explained in [8], where they draw from the training procedure used in [12] to apply on multiple architectures. They apply Equation 3.1 to their training set as follows. S_{ρ} and $S_{\rho+1}$ are the old set and the new set of data, $\overrightarrow{x}$ *is* the sample being used, λ is what defines the augmentation step size and O are the labels from the last training session of the substitute model. The way [8] and [12] differ is that [8] used a periodical step size or a different λ each time they notice that the current augmented training set is not improving the outcome.

3.2 Black-Box Approximation for Malware Detectors

In terms of malware detectors approximation, the main work that motivated this thesis is by Hu and Tan [7]. Like [12] and [8], this work also goes beyond approximation to craft adversarial examples based on an approximated black-box model. This work assumes that the only thing the adversary knows about the black-box is the type of features it uses. In our case, we assume a threat model where the adversary knows nothing about the details of the black-box. In addition, our approach differs from [7] in the assumption about the underlying feature representations of training examples (Windows executables). While [7] assumes the same (API calls precisely) feature representation of the black-box and the substitute, in our work we make no assumption about the similarity or difference of the black-box and the substitute model.

Related to [7], [19] adopts Equation 3.1 to synthesize training examples to approximate a target black-box malware detector based on API call features. Like [8] and [7], [12] also crafts adversarial examples to evade the black-box model it approximates. Next, we make approach-level comparisons between our work and these closely related works.

3.3 Comparison with Related Work

Table 3.1 shows multi-criteria comparison of this work with the state of the art in black- box adversarial approximation of machine learning models both in the image and malware domain. Our comparison is based on assumptions about features, model architecture, initial training set size, and dataset overlap between among black-box and substitute model.

Features compares assumptions about features of the black-box and the substitute (e.g., same, different). While prior work [8, 7] assumed similar features for both the black-box and the substitute model, in this work we consider the more challenging scenario where the black-box is trained based on raw-byte sequences of Windows executables, whereas the substitute is based on

pixels of the image representations of the raw bytes. Table 3.1 shows that features used for training the substitute in [7] are the same as the ones the black-box was trained on. Similarly, [12] uses images on an image recognition black-box which is vaguely similar, whereas the rest are completely different from the Black Box they are trying to approximate.

Model architecture captures assumptions about model architecture for the black-box and the substitute model (e.g., same, different). Unlike most prior work [7, 9, 8, 12], we make no assumption about the similarity of model architectures for the black-box and the substitute, our current implementation of our approximation strategy uses CNN for both. It is worth noting that most prior work leverage the popular adoption of DNNs for malware detection to make an assumption of the same, i.e., DNN, architecture for the black-box and the substitute model.

Initial set size compares assumptions on availability of initial set to begin approximation (e.g., limited, enough, big). In [8], the assumption is that training data is scarce and hence the adversary explores augmentation techniques to generate more training samples to train the substitute. In [7], the approach assumes an adversary has collected enough samples to train a substitute. In this work, we explore both cases where the adversary has access to a limited initial set and the case when the adversary has prepared enough samples to explore adversarial approximation opportunities.

Data overlap examines whether dataset used for the black-box, the substitute, and comparison of black-box with substitute have overlap (e.g., overlap, disjoint). In this work, we use disjoint datasets for training the black-box, approximation of the substitute, and comparison of the black-box with the substitute. Doing so ensures the effectiveness of the substitute model approximated with a completely new dataset. In the closely related works we review here [7, 12, 9, 19], there is no explicit statement on whether or not the datasets are completely disjoint from

the black-box (that is the reason why we mark “N/A” for related work and “disjoint” for our work in the last row of Table 3.1).

Table 3.1: Comparison with closely related work.

	[7]	[12]	[9]	[19]	**This work**
Features	same	same*	different	different	different
Model architecture	different	same*	different	different	same*
Initial set size	big	enough	enough	enough	limited/big
Data overlap	N/A	N/A	N/A	N/A	disjoint

CHAPTER 4: APPROACH

This chapter presents the threat model, problem formulation, challenges, and details of our approach.

4.1 Threat Model, Problem, and Challenges

Threat Model: In this work, we operate under a specific threat model with a specific adversarial goal and capabilities. The adversary's goal is to approximate the decision boundary of a deployed malware detector (let's call it *B*). The adversary doesn't know *B*'s architecture, parameters, features, or training set. The adversary only knows that *B* accepts Windows executables as input and returns labels (benign or malicious) as output. It is as- summed that the adversary has access to a seed-set of benign and malicious PEs but can't tell whether or not the PEs in the seed-set intersect with *B*'s training set. Moreover, the adversary knows the task for which *B* is trained (malware detection in our case).

Problem Formulation: Given a deployed malware detection model, *B*, under the threat model stated earlier, the adversary's goal is to find *B*'s approximation, *S*, with minimal number of interactions with *B*. The ultimate goal of the adversary is to use *S* (i.e., *B*'s approximation) in a white-box setting to mount evasion attacks against *B*. A white-box setting gives the adversary the leverage to manipulate the internals of *S* so as to infer *B*'s vulnerabilities to evasion attacks (e.g., to craft adversarial examples). An approximation of *B* could also allow the adversary to steal the model (e.g., if *B* is an intellectual property or some top-secret business logic).

Challenges: The following are the challenges tackled by this work:

- **Challenge-1:** Where does the adversary start with the approximation? A natural direction would be to use a brute-force strategy on combinations of learning algorithms, feature sets, and training sets. However, this strategy yields infinitely large number of possibilities to explore (hence time-consuming for the adversary).
- **Challenge-2:** How does the adversary expand an initial seed-set of samples that *B* accepts?
- **Challenge-3:** How does the adversary ensure that the approximation is progressing in the right direction (getting close to *B*)?

4.2 Approach Overview

Figure 4.1 depicts an overview of our black-box adversarial approximation approach. Given a black-box model *B* trained on raw byte features of executables in the black-box training set, the adversary collects benign and malware executables from public sources to maintain the approximator training set. The adversary then uses *B* as an oracle to label samples in the approximator training set (details in Section 4.3.1). Next, the labeled approximator set is transformed from raw bytes to image representation to make it ready for the adversarial approximation (details in Section 4.3.2). The adversarial approximation step uses the labeled-and-transformed approximator set to systematically approximate *B*. The approximation begins by training an initial approximator model *S* on the labeled-and-transformed approximator set and *S* is refined until it achieves acceptable accuracy (details Section in 4.3.3). Using a separate dataset disjoint with the approximator training set, the last stage of the approach compares the similarity between *B* and *S* (details in Section 4.3.4). The key insight is that a higher similarity score for *B* and *S* is an indication of the effectiveness of the approach in Figure 4.1

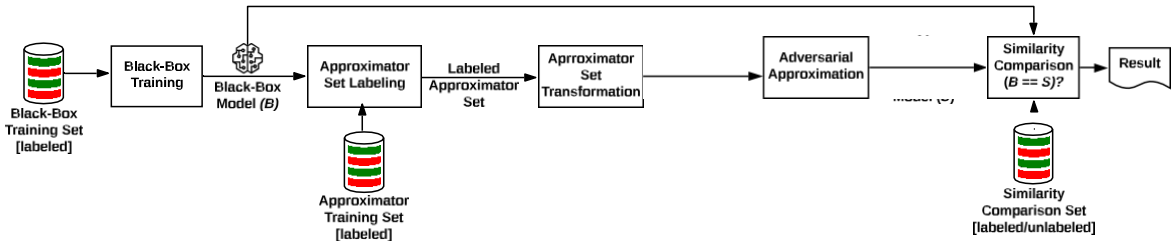


Figure 4.1: Approach overview.

For the sake of presenting an end-to-end approximation framework, Figure 4.1 includes the black-box training at the beginning. In practice, the adversary doesn't have access to the black-box model. It is also worth noting that the three datasets shown in Figure 4.1 are assumed to be disjoint in order to illustrate a truly black-box setting (the most challenging for the adversary). Again, in reality, the adversary has no easy way to determine if the substitute training set or the model similarity comparison set have intersection among themselves or with the black-box training set. In our approach, taking advantage of the fact that we trained our own black-box model, we made sure that these three sets are dis- joint in order to bring the scenario close to a true black-box setting. Note, however, that in the adversarial approximation, our approach operates in line with the threat model stated earlier.

4.3 Approach Details

In this section, present the details of our approach and justify our choices with respect to the threat model and the challenges stated earlier. We will first discuss the labeling of the approximation set, followed by how we transform the samples in the approximation set to fit our approach goals. Next, we explain the core component of our approach, the adversarial approximation. Finally, we describe the similarity comparison component.

4.3.1 Labeling of Approximation Set

An implicit question in **Challenge-1** is "given a potential set of benign and malware executable samples, how does the adversary label them towards effective approximation?". To tackle this challenge, a naive labeling approach would be to take the "ground truth" labels that the samples come with. Had our end goal been training a malware detector, this naive labeling approach would have sufficed for training a typical binary classifier for malware detection. Our goal, however, is to approximate a black-box malware detector. In other words, we expect our labeling strategy to guide our approximation strategy towards B.

Given a set of samples $x^{(1)}, ..., x^{(n)}$ and a hypothesis function h learned by B, by query- ing B as $h(x^{(i)})$, we obtain $y^{(i)}$ (i in [1, n]) as labels. The $y^{(i)}$'s may or may not match the ground truth labels. If B misclassifies some $x^{(i)}$'s, the misclassified $x^{(i)}$'s will not match the ground truth counter-parts. What should be done with the misclassified samples in the approximator training set? The alternatives we have are a) drop the misclassified $x^{(i)}$'s and explore approximation with the correctly labeled $x^{(i)}$'s, b) reinstate labels to ground truth labels and proceed with approximation, or c) take the labels assigned by B for what they are. Alternative a) is no different from training the approximator without querying B. Alternative b) tries to "cor- rect" the "imperfections" of B (note that if we pursue this path, "correct" could mean lower accuracy because we are essentially changing the underlying distribution of B). Alternative c) is the most realistic for our threat model, because it takes B for what it is and uses labels returned by B' (i.e., the $y^{(i)}$'s) to prepare the labeled approximation set. It is worth noting that alternative c) tackles **Challenge-3** by guiding the adversarial approximation strategy in the right direction.

4.3.2 Approximation Set Transformation

Given our black-box threat model, the adversary doesn't know what features are used to train *B*. To address **Challenge-1**, the adversary may pursue different possibilities of features used to train malware detectors based on executables. However, the space of possible features is exponentially large. For example, if we only consider static analysis-based features of a given executable sample, we end up with numerous possible features such as meta-data, DLL imports, byte sequences, and so on. Similarly, considering the dynamic analysis-based features results in several candidate feature sets such as API/system call traces, instruction sequences, and call graphs. Therefore, such a strategy of feature guessing would be neither effective nor preferred by a skilled adversary simply because it is resource-intensive.

A strategy we propose in this approach is to transform the raw byte representation of each executable to an image representation (pixel values precisely), analogous to taking a photograph of the executable's raw bytes. The main rationale here is that instead of fishing for the best combination of a set of features to train the approximator, it is more plausible to capture the whole executable's bytes via image representations such that the learning algorithm is able to "see" and learn from all the distinguishing features of an executable in one place (i.e., the image). Past work has also explored the viability of bytes-to-pixels conversion for malware detection [20, 21]. Another equally important rationale for bytes- to-pixels transformation is the public accessibility of acceptably accurate pre-trained image classification models (e.g., Inception V3 [11]) which, by the notion of transfer learning [8] could feed knowledge to the approximator.

To realize the bytes-to-pixels transformation, we leverage BinViz [22], a library that provides algorithms to transform the binary executables into a colored canvas, such that the colors (pixel intensities) represent the bytes in the executable, and these color intensities are used as the features

to train the approximator. In this work, we use two types of image representation, the Entropy (EN) representation and the Color Hilbert (CH) representation. CH scans the bytes of a binary executable, and assigns color based on the value of each byte. The assigned colors are then mapped on a canvas of a chosen dimension. EN uses Equation 4.1 (originally proposed by [23]) to compute the randomness of bytes in a specific location of the executable:

$$-\sum_{i=1}^{n} \rho_i \, log_2 \rho_i \qquad (4.1)$$

where p_i refers to the probability of appearances of a byte value i, and n is the number of possible values (n = 256 for possible byte values). The values provided by the algorithm are given a color from bright pink to black. An example that illustrates EN and CH representation for a benign executable is shown in Figure 4.2 (a) and 4.2 (b), respectively. Similarly, 4.3 (a) and 4.3 (b) show the EN and CH representations of a malware executable. Notice the clear visual difference between benign and malware samples, which seems to support our rational of mapping bytes to pixels. Looking at the CH representation we would hope that the combination of these colors in the images serves best to give the model a clear idea of some features that can be used to learn from. As for the EN representation, we can see that the focus is more on areas instead of specific pixels, which we would like to evaluate how it would affect our training since it is not as detailed as the CH representation. In Chapter 5, we will evaluate the utility of the image representations by examining the approximator's accuracy. Next, we describe how the canvas is filled out with colors.

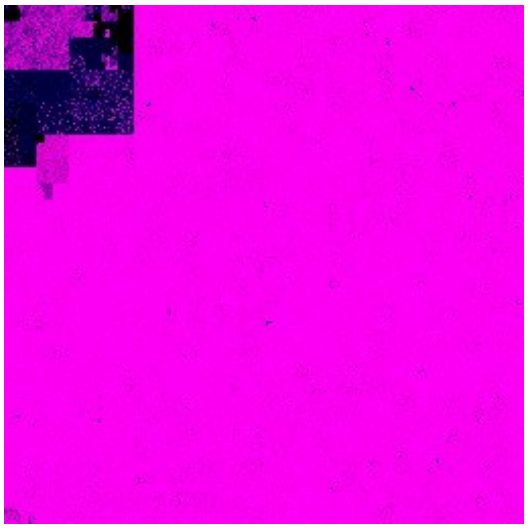
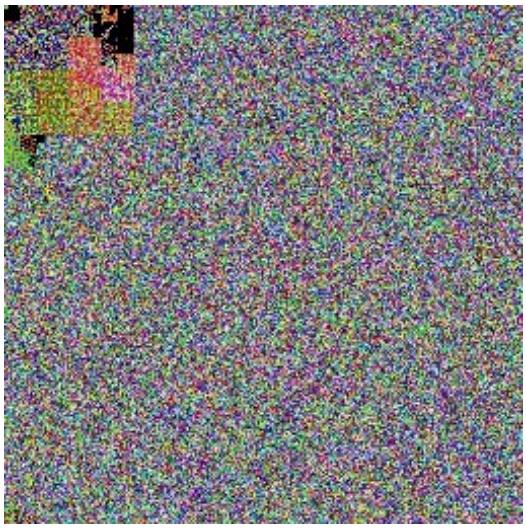
(a)  (b) 

Figure 4.2: EN and CH rendering of a benign executable (02Mi-cro Card Reader Driver 3.11.exe).

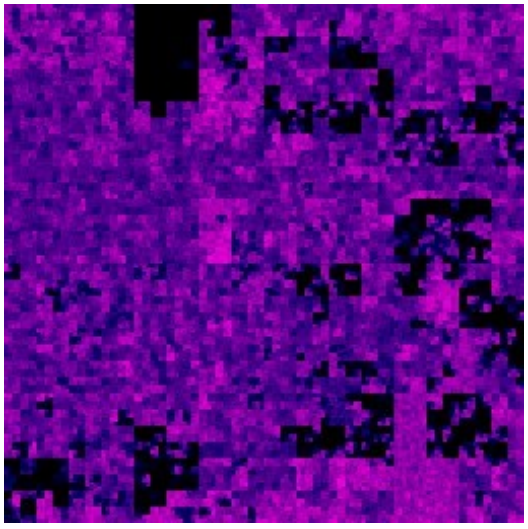
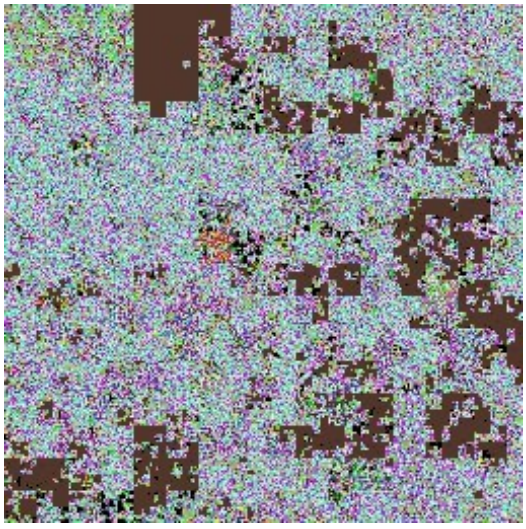
(a)  (b) 

Figure 4.3: EN and CH rendering of a malware executable (Trojan.GenericKDZ.58985).

To fill out the canvas of the image, we leveraged a well-known mapping technique called Hilbert curve [24], which makes sure that if two bytes are close to each other in the executable they should be close to each other in the image representation as well. This insight is essential in preserving the semantic structure of the executables when mapping bytes form the byte-space to the corresponding pixels in the color-space. This aspect of the Hilbert curve provides the

approximator model an accurate representation of the executable file so that it explores the discriminating utility of all possible features. The next question would be, how does the Hilbert curve function? Intuitively, the idea of Hilbert curve[3] is to find a line to fill a canvas that will keep the points which are close to each other on that line at the same distance when it fills the needed space. This, in our case, keeps the features of the executables intact since separating them would lead to breaking the semantics of a string of bytes which represents a part of our feature set in the images we would like to generate.

4.3.3 Adversarial Approximation

The adversary has two broad choices for adversarial approximation, brute-force or systematic. A brute-force strategy, where by the adversary keeps trying a combination of approximation dataset, feature set, model architecture, and model parameters until an acceptable approximator is trained. A systematic strategy would minimize the space of exploration of the numerous combinations to maximize the chance of obtaining an acceptable approximator with minimal effort. Next, we describe these two options and justify why a systematic approximation approach is more desirable for the adversary.

4.3.3.1 Brute-Force

The idea of brute force stems from the fact that the adversary doesn't know what the best combination of approximation dataset, feature set, model architecture, and model parameters. As a result, the adversary tries potentially infinite possibilities until an *S* close enough to *B* is obtained. Applying this approach is unreasonably tiresome and is such a waste of time. The number of tries and hours the adversary will have to sink in to get to their goal is very large, and will defeat the

[3] https://www.youtube.com/watch?v=3s7h2MHQtxc

purpose of the attack since by the time any significant ap- proximation advances can be made, *B* might have been retrained and the process will have to start all over again. For the adversary, it comes down to timeliness and approximation effectiveness (quickly obtain *S* close enough to *B*) with minimal effort, which motivates a systematic approximation strategy.

4.3.3.2 Systematic

On the one hand, the adversary races against time and limited resources (e.g., training examples). On the other hand, the adversary has access to pre-trained and acceptably accurate models for images (e.g., Inception V3 [11]). To obtain an acceptably accurate approxi- mation of the black-box model, the adversary takes advantage of the advances in image classification models to quickly train a candidate approximator on the transformed features of the executables at the disposal of the adversary. Notice doing this cuts the effort on feature engineering down to zero because the candidate approximator (e.g., a CNN) would automatically learn the features from its training data. However, training the candidate approximator based on a pre-trained model highly de- pends on the presence of a large corpus of training examples, but the adversary may not have the leverage to collect relevant training examples. This challenge is specially true when one attempts to collect benign executables from the wild. We observe that, while it is relatively easy to obtain a dataset of hundreds of thousands of malware executables from malware repositories, it takes weeks and sometimes months to collect benign executables due to the lack of publicly curated benign executables dataset.

To address the scarcity of training examples, the adversary exploits the notion of data augmentation, which aims to synthesize minimally manipulated yet semantically intact variants of an image representation of samples (executables in our case) [8, 12]. Intu- itively, augmentation involves slightly altering an image while keeping the main features of the image intact. Applying

augmentation methods (e.g., slight rotation, flipping) in vi- sion tasks (e.g., object detection) is relatively easier because as long as the augmentation method doesn't visually alter the object, the model can still detect it. In our approach, however, augmentation needs to be done with care because augmentation methods that work fine in the vision domain may not necessarily apply to the malware domain due to strict domain-specific constraints, i.e., malware behavior should remain intact when ap- plying augmentation. In the process of slightly altering an executable's image, we don't want to end up with a mutated image that semantically diverges from the original image representation of the executable.

Due to the need to preserve an executable's behavior, we carefully choose the augmentation methods for our approximation strategy. In particular, we selected flipping and rotation to synthesize semantically equivalent variants of the training examples. These two augmentation methods are proved to keep the original structure intact [12]. Figures 4.4 and 4.5, respectively, show flipping and rotation of a benign and malware sample in our dataset. Through augmentation, we managed to increase our approximator training set by three-fold (details in Section 5.2.2). The impact, on the approximator's accuracy, of extending the initial approximator training set through augmentation is evaluated in Section 5.3.2. Note that the use of augmentation to expand the training data directly addresses **Challenge-2**.

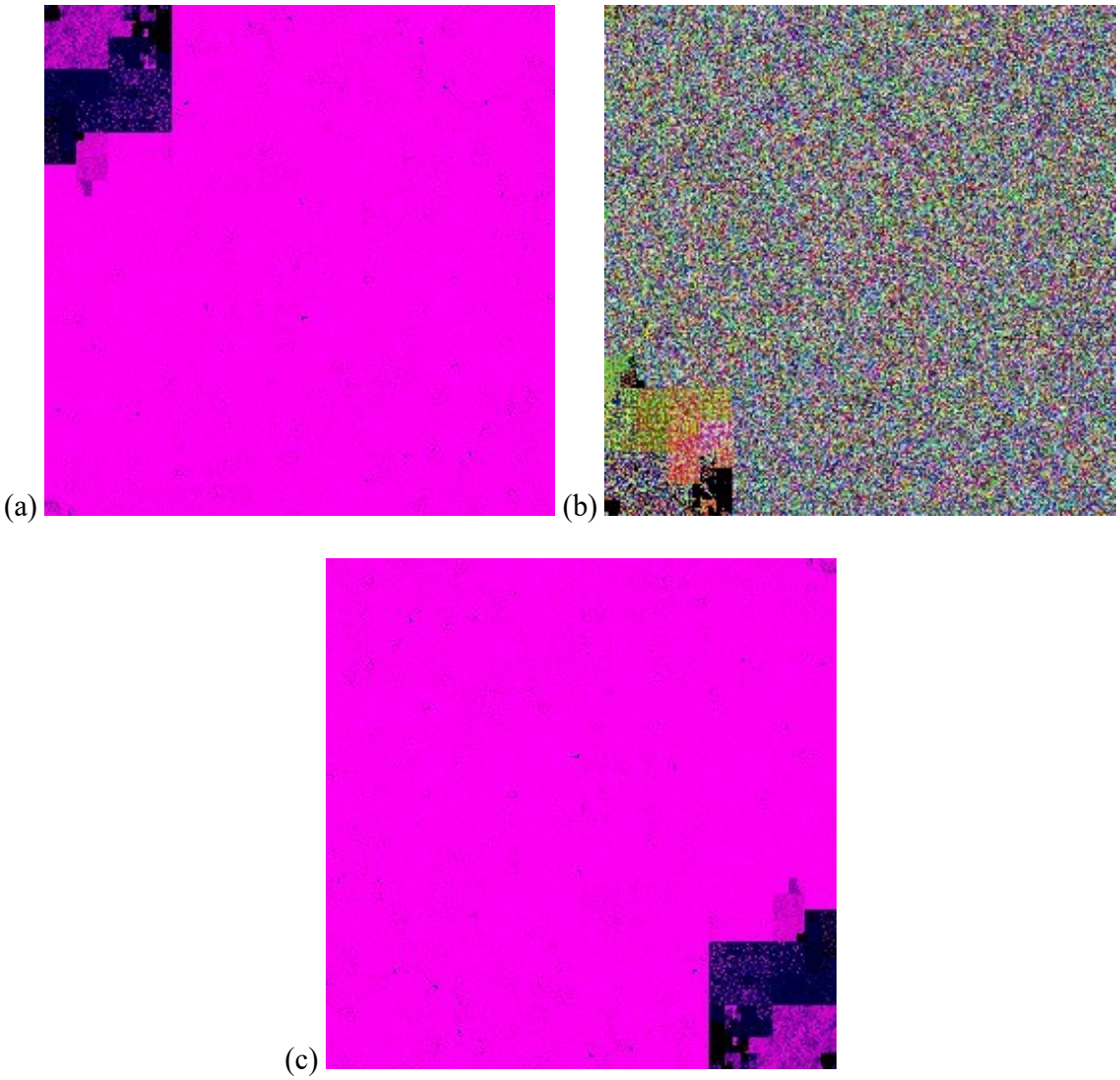


Figure 4.4: Augmentation example of a benign executable (02Mi-cro Card Reader Driver 3.11.exe) (a): original EN, (b): flipped CH, (c): rotated EN.

4.3.4 Similarity Comparison

Once the approximator is trained with an acceptable accuracy, its true effectiveness can only be assessed when compared with the black-box model on a separate dataset, disjoint with both the training set of the black-box and the approximator.

Algorithm 1 shows the pseudo-code for the similarity comparison of B and S. The similarity score is the percentage of matching predictions between B and S. The higher the similarity score, the closer S is to B, which means S effectively mirrors the decision boundary of B. The adversary

can then probe *S* for further attacks in a white-box setting. Attacks that succeed on *S*, would, by transitivity, work on *B*. This is the essence of having an accurate approximate model which would be used as a substitute for the black-box. By crafting adversarial examples using techniques such as the Fast Gradient Sign Method [5], the adversary can now transitively re-target the black-box using the approximated model, *S*, as a surrogate.

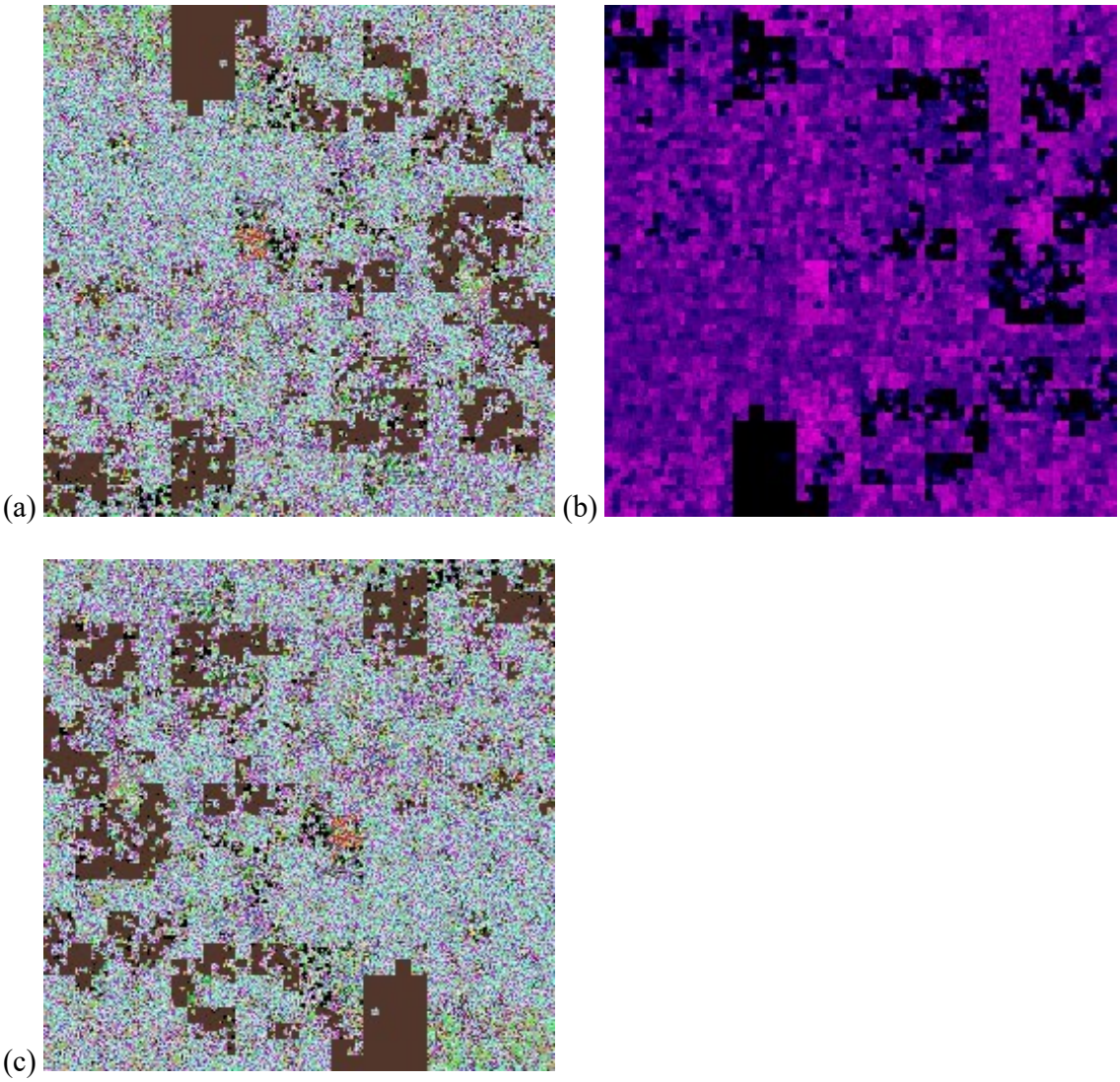


Figure 4.5:Augmentation example of a malware executable (Trojan.GenericKDZ.58985). (a): original CH, (b): flipped EN, (c): rotated CH.

Algorithm 1 Similarity comparison between black-box (B) and approximator (S).

1: $matches \leftarrow 0$

2: **for** $i = 1 \rightarrow num\ comparison\ samples$ **do**

3: $y^{(i)}_{B} \leftarrow blackbox(x^{(i)})$

4: $y^{(i)}_{S} \leftarrow approximator(x^{(i)})$

5: **if** $y^{(i)}_{B} == y^{(i)}_{S}$ **then**

6: $matches \leftarrow matches + 1$

7: **end if**

8: **end for**

9: $similarity\ score \leftarrow matches/num\ comparison\ samples$

CHAPTER 5: EVALUATION AND RESULTS

This chapter presents the dataset, experimental setup, and evaluation results of our approach.

5.1 Dataset

Our dataset shown in Table 5.1 is collected from two sources. The benign executables are collected from a free Windows software store[4], while the malware executables are obtained from VirusShare[5]. These two sources are widely used by prior work [3, 25, 16, 7, 19] for adversarial examples generation or and malware detection. Overall, we collect 61, 330 executables with 52% benign and 48% malware.

The collection of malware executables is fairly easy due to a continuously curated repository of malware executables maintained by sites such as VirusShare and VirusTotal[6]. The collection of benign executables is more challenging due lack of publicly curated benign executables. To curb this challenge, we resorted to websites that give access to a huge amount of freeware. We implemented a crawling script to automatically download benign executables which account for 52% the dataset shown in Table 5.1.

[4] https://download.cnet.com/s/software/windows/?licenseType=Free
[5] https://virusshare.com
[6] https://www.virustotal.com/gui/

Table 5.1: Summary of datasets used.

Dataset	Benign	Malware	Total
Black-Box Training Set	20, 000	20, 000	40, 000
Approximator Training Set	8, 000	8, 000	16, 000
Similarity Comparison Set 1	2, 556	766	3, 316
Similarity Comparison Set 2	1, 349	665	2, 014
Total			**61, 330**

5.2 Setup

Our experimental setup consists of a CNN black-box, *B*, trained on a portion of the dataset; an approximation model, *S*, trained on a separate portion of the training set; and a compar- ison step that uses yet another separate chunk of the dataset to evaluate how close *S* is to *B*. In what follows, we describe the details of our setup for *B* and *S*.

5.2.1 Black-Box Model Training

B is trained on 20K benign and 20K malware executables. The model is based on MalConv [3]. For the sake of quick reproduction, we use the same architecture in MalConv with some tweaks needed to fit our hardware limitations (NVIDIA 1080 with 8GB of memory). MalConv is based on raw bytes of an executable, and the algorithm reads the first megabyte of each executable, turn it into an array and use that array with others in batches to train the model.

Due to memory limitations on hardware, instead of reading 1MB (as was done in the original MalConv), we fed only 1/3MB of each executable to our custom CNN. After suc- cessfully training our model, we got an accuracy of 93% which is acceptable for use since the data set used in [3] to achieve an accuracy of 98% was about 100K. Also, given the fact we truncated 2/3MB of the byte

features used in MalConv, the accuracy we obtained is acceptable to set the black-box model as the target for adversarial approximation.

5.2.2 Approximate Model Training

The approximation model we used is Inception V3 [11]. There are a couple of reasons to go with this specific architecture. First, the fact that our approach is going to rely on image recognition, and Inception V3 is known to be a great way to achieve that goal. Second, the prospect of using a widely available and acceptably accurate pre-trained model for anyone to use is in favor of the constraints the adversary has on collecting enough training samples for the approximation and come up with an effective model architecture tuned to adversarial goals. Third, when using Inception V3 we are only retraining the last layer of an already pre-trained model which not only saves us a lot of time, but also gives us confidence in the accuracy of the final model because of transferability.

First, on the 16K (8K benign + 8K malware) approximation training set labeled by B, we use the two types of image representations we obtained from the transformation: Color Hilbert and Entropy representation, to separately test which representation achieves better approximation accuracy. We then followed the following steps to do our intensive training:

Progressive: We train S progressively on 4K, 8K, 12K, and 16K executables separately, and then compare the results we obtained from both Color Hilbert and Entropy represen- tations. This step is to emulate what an adversary would do since they will be actively collecting data, they will surly attempt to retrain their model each time they get more training examples until an acceptable accuracy is obtained or the training data is exhausted.

Augmentation-Based: In this step, we focus on the augmentation of the approximator training dataset. The goal here is emulating an adversary with scarce training data and aims to expand to more training data with less time. In particular, the augmentation methods discussed in Section

4.3.3 are tested in practice. We begin with the 8K executables for each label (16K overall) and employed (a) rotation (by 90 degrees), (b) flipping, and (c) rotation + flipping to obtain twice as much (32K) for rotation and flipping each, and three times (48K) by merging the samples synthesized by rotation and flipping, with the original 16K samples we started with. The next section discusses the effectiveness evaluation of the augmentation techniques.

5.3 Results and Discussion

We present the results in three parts. We will first evaluate the effectiveness of the progressive training of *S*. Next, we examine the effectiveness of the augmentation methods. Lastly, we compare the similarity of *B* with candidate approximators.

5.3.1 Progressive Accuracy of Approximated Model

Figure5.1 shows the training accuracy and validation accuracy of each progressive step in training *S* with the Color Hilbert representation. When examine the results in Figure 5.1, it can be seen that the difference between training and validation accuracy narrows as the model progresses with more training data, which is an indication of the real-life effectiveness of the model on training examples it has never seen during training. We pick the model trained on 16K data points as our first approximator model.

Similarly, Figure5.2 shows the progressive training of *S* using the Entropy representation. Here, the trend persists, where we see a very high training accuracy across the board but as we train on bigger data sets we get a more efficient model (validation accuracy is close to training accuracy). This leads us to the same conclusion we made earlier and we choose the model trained on 16K data points as our representative for this batch of models. These results are not surprising since it is intuitive to see that the more data we trained our model on the better it will be at doing

its job. Obviously, the accuracy values could get higher with bigger data sets but since we are limited to these original images we can only train with what we have at the time of this evaluation.

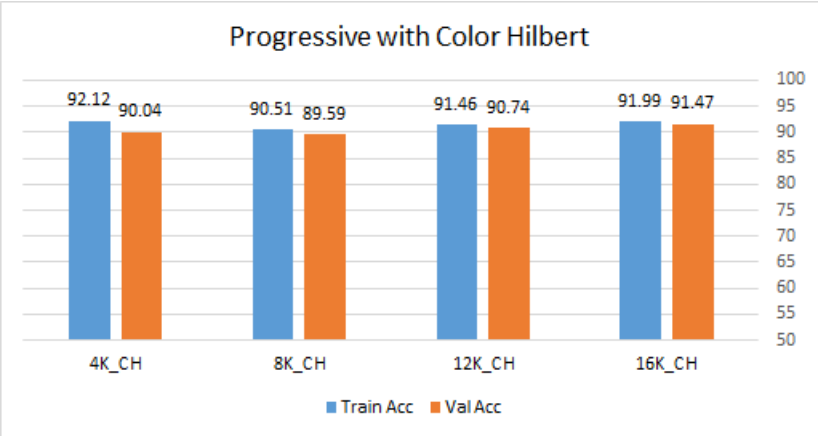


Figure 5.1: Training and validation accuracy for progressive training of S with Color Hilbert representation.

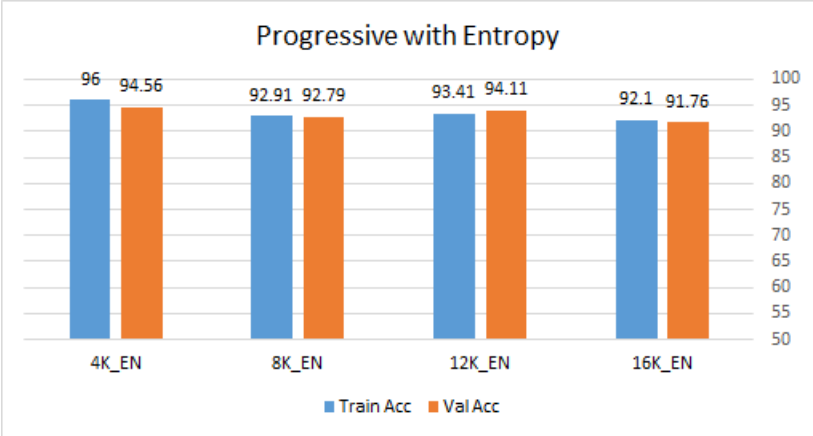


Figure 5.2: Training and validation accuracy for progressive training of S with Entropy representation.

5.3.2 Augmentation Effectiveness

Now that we have exhausted all of our original approximator training set, let us shift our focus to evaluating the effectiveness of the augmentation algorithms discussed earlier.First, let us start with the flipped images added to our approximator training set. As stated earlier, the flipping

augmentation doubles our training set. The results are split into two figures, Figure5.3 (for Color Hilbert) and Figure 5.4 (for Entropy). From Figure 5.3, it can be seen that it doesn't show much improvement over the results from Figure 5.1 and 5.2. This doesn't mean that the augmentation strategy is not useful, but it gives us visibility into the effectiveness of the specific representation+augmentation we used. More importantly, we could expand more on this and try different combinations to examine if doing so would give us better results. Our conclusion is our experiments demonstrate the viability of this strategy since the time it takes to produce these augmented images is much shorter than collecting new data to expand the data set.

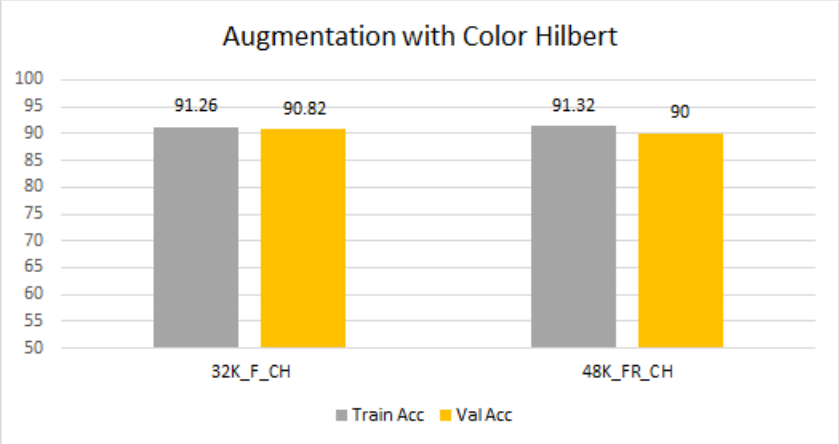


Figure 5.3: Approximation results for Color Hilbert representation with image flipping augmentation (32K_F_CH) and rotation augmentation (48K_FR_CH).

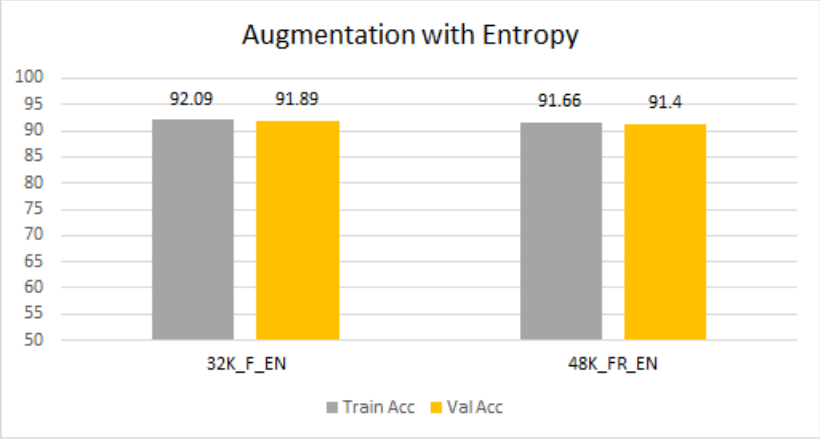


Figure 5.4: Approximation results for Entropy representation with image flipping augmentation (32K_F_EN) and rotation augmentation (48K_FR_EN).

In Figure 5.4, however, we see a slightly different trend. In particular, we notice a small improvement when using the flipped images with our original set. This could be attributed to the different features in the picture representation of entropy that the model looks to train from. However, we see a small drop of accuracy when we add the rotated images. In general, the accuracy values obtained for the augmentation methods are overall acceptably comparable to the progressive training presented earlier. More importantly, the augmentation methods can be done fast and hence will help fill the data scarcity gap for the adversary.

5.3.3 Similarity Comparison

Finally, using algorithm 1, we compare the similarity of the black-box model with the candidate approximators obtained earlier. As shown in Table 5.2, we have 6 approximators, the first 2 from the progressive approximation and the last 4 obtained through the augmentation methods. The comparison of these approximators with the black-box is done on a separate comparison set, disjoint with the black-bot training set and the approximator training set.

Table 5.2: Comparison results between the black-box and the approximator.

Approximation Method	**Similarity (black-box, approximator)**
16K using Color Hilbert (CH)	80.5
16K using Entropy (EN)	89.7
32K using CH and flipped	80.6
32K using EN and flipped	89.6
48K using CH flipped and rotated	79.8
48K using EN flipped and rotated	90.1

On average, our approach achieved 85.5% similarity score, with the highest similarity score of 90.1% (on the largest dataset obtained through flipping and rotation of entropy rep- resentations). When we compare Color Hilbert and Entropy approximators, the Entropy- based approximators outperformed Color Hilbert by about 9%. Comparing the different augmentation methods, it depends on the color representation, i.e., whether we use Color Hilbert or Entropy. The entropy-based representations consistently outperformed the Color Hilbert representations for all the augmentation techniques. As we discussed in Section 4.3.2, this difference is attributable to the different canvas coloring methods used in the respective representations (Color Hilbert and Entropy).

Finally, Figure 5.5 shows the details of similarity scores for the 6 approximators split into malware, benign, and overall. Having a malware detection rate higher than the benign detection rate serves us best, since our goal is to be able to identify malware identified as so by the black-box. From the figure, it can be seen that the split shows that Entropy-based approximators tend to match the black-box more on malware, while Color Hilbert-based approximators match the black-box more on the benign predictions. Again, this variation is rooted in the canvas coloring methods.

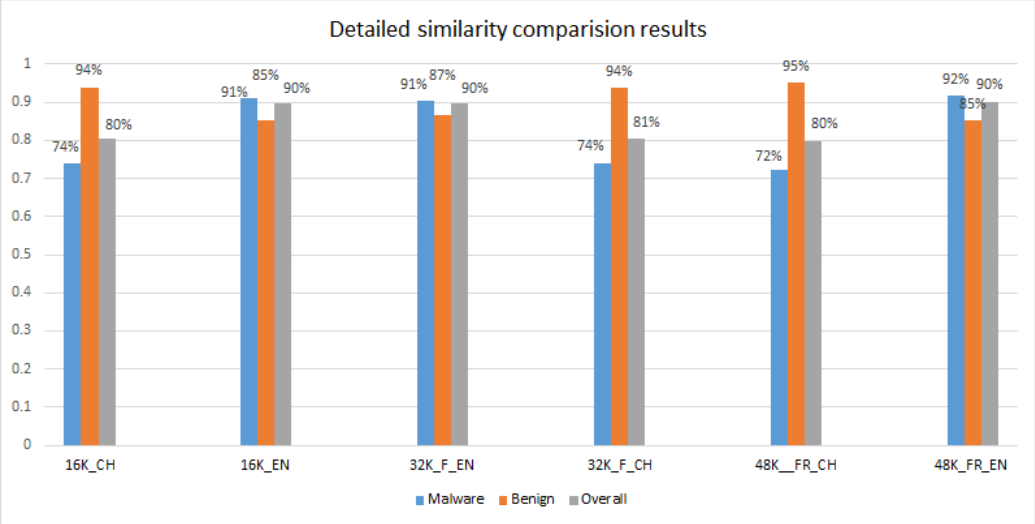


Figure 5.5: Similarity comparison results split into benign and malware for all approximated models with respect to the black-box.

C HAPTER 6: CONCLUSION AND FUTURE WORK

This thesis presented a novel adversarial approximation approach for a black-box malware detector. Beginning with publicly accessible input-set for the black-box model, our approach leverages the recent advances in image transformation (using Entropy and Color Hilbert) and the notion of transferability (using Inception V3's pre-trained model) to effectively approximate a black-box malware detector.

Our experimental evaluations suggested two intriguing insights. Firstly, even if the adversary obtains training samples that don't overlap with the black-box model's training set, our results show that the adversary can still achieve an acceptable approximation of the black-box model with minimal efforts. Secondly, even in the challenging scenario for which the representation of the black-box is different from that of the approximated model, interestingly, our approximation approach still managed to achieve above 90% similarity between the target black-box and the approximated model. In addition, we demonstrated the effectiveness of our approach for the case where the adversary has access to limited training examples (using augmentation), and also the opposite case, where the adversary has enough training examples (progressive training). Overall, the results that we got are very promising, and our experimental setup can be reproduced by other researchers to explore more about the attack surface of black-box machine learning models.

One avenue for future work is a more thorough exploration of augmentation techniques. In our case, we carefully picked the specific augmentation methods in the interest of pre- serving the semantics of the samples. We believe there is more to explore on the pros and cons of various augmentation techniques such as color blurring. Another avenue worth ex- ploring is applying our

approach to other model architectures (e.g., Recurrent Neural Nets, Support Vector Machines, Logistic Regression) to see the cross-architecture transferability of our adversarial approximation technique ([8] has more details on this aspect for multiple model architectures in the vision domain).

REFERENCES

[1] Wicht,B.,"Deep Learning feature Extraction for Image Processing, https://www.researchgate.net/figure/A-valid-convolution-of-a-5x5-image-with-a-3x3fig5_322505397, 2019.

[2] Saha, S.,"A Comprehensive Guide to Convolutional NeuralNetworks the ELI5 way," https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-wa2018.

[3] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., and Nicholas, C. K., "Malware Detection by Eating a Whole EXE," *The Workshops of the The Thirty- Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018, pp. 268–276.

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

[5] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R., "Intriguing properties of neural networks," *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[6] Goodfellow, I. J., Shlens, J., and Szegedy, C., "Explaining and Harnessing Adversarial Examples," *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[7] Hu, W. and Tan, Y., "Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN," *CoRR*, Vol. abs/1702.05983, 2017.

[8] Papernot, N., McDaniel, P. D., and Goodfellow, I. J., "Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples," *CoRR*, Vol. abs/1605.07277, 2016.

[9] Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T., "Stealing Machine Learning Models via Prediction APIs," *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, 2016, pp. 601–618.

[10] "Deceiving End-to-End Deep Learning Malware Detectors using Adversarial Examples," 2018.

[11] "Advanced Guide to Inception v3 on Cloud TPU," https://cloud.google.com/tpu/docs/inception-v3-advanced, 2019.

[12] Papernot, N., McDaniel, P. D., Goodfellow, I. J., Jha, S., Celik, Z. B., and Swami, A., "Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples," *CoRR*, Vol. abs/1602.02697, 2016.

[13] Shokri, R., Stronati, M., Song, C., and Shmatikov, V., "Membership Inference Attacks Against Machine Learning Models," *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, 2017, pp. 3–18.

[14] Carlini, N., Liu, C., Erlingsson, Ú., Kos, J., and Song, D., "The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks," *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, 2019, pp. 267–284.

[15] Biggio, B., Nelson, B., and Laskov, P., "Poisoning Attacks against Support Vector Machines," *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[16] Al-Dujaili, A., Huang, A., Hemberg, E., and O'Reilly, U., "Adversarial Deep Learning for Robust Detection of Binary Encoded Malware," *2018 IEEE Security and Privacy Workshops, SP Workshops 2018, San Francisco, CA, USA, May 24, 2018*,2018, pp. 76–82.

[17] Suciu, O., Coull, S. E., and Johns, J., "Exploring Adversarial Examples in Malware Detection," *Proceedings of the AAAI Symposium on Adversary-Aware Learning Techniques and Trends in Cybersecurity (ALEC 2018) co-located with the Association for the Advancement of Artificial Intelligence 2018 Fall Symposium Series (AAAI-FSS 2018), Arlington, Virginia, USA, October 18-20, 2018.*, 2018, pp. 11–16.

[18] Demetrio, L., Biggio, B., Lagorio, G., Roli, F., and Armando, A., "Explaining Vulnerabilities of Deep Learning to Adversarial Malware Binaries," *Proceedings of the Third Italian Conference on Cyber Security, Pisa, Italy, February 13-15, 2019.*, 2019.

[19] Rosenberg, I., Shabtai, A., Rokach, L., and Elovici, Y., "Generic Black-Box End-to- End Attack Against State of the Art API Call Based Malware Classifiers," *Research in Attacks, Intrusions, and Defenses - 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings*, 2018, pp. 490–510.

[20] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S., "Malware Images: Visualization and Automatic Classification," *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, VizSec '11, ACM, 2011, pp. 4:1–4:7.

[21] Han, K., Lim, J. H., Kang, B., and Im, E. G., "Malware analysis using visualized images and entropy graphs," *Int. J. Inf. Sec.*, Vol. 14, No. 1, 2015, pp. 1–14.

[22] Cortezi, A.,"binviz," https://github.com/cortesi/scurve/blob/ master/binvis, 2019.

[23] Shannon, C. E., "A mathematical theory of communication," *Mobile Computing and Communications Review*, Vol. 5, No. 1, 2001, pp. 3–55.

[24] Byrne, A. and Hilbert, D. R., "Color realism and color science," *Cambridge University Press*, Vol. 26, No. 1, 2003, pp. 3–64.

[25] Anderson, H. S. and Roth, P., "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," *CoRR*, Vol. abs/1804.04637, 2018.