2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA) | 978-1-7281-8543-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/TPS-ISA50397.2020.00025

From Blue-Sky to Practical Adversarial Learning

Aminollah Khormali[1], Ahmed Abusnaina[2], Songqing Chen[3], DaeHun Nyang[4], David Mohaisen[5]
[1,2,5]University of Central Florida [3]George Mason University [4]Ewha Womans University

***Abstract*—The state-of-the-art of adversarial machine learning on malware detection systems generally yield unexecutable samples. In this work, we make the case for understanding the robustness of visualization-based malware detection system against adversarial examples (AEs) that not only are able to fool models, but also maintain the executability of the original input. To motivate for our vision, we first investigate the application of existing off-the-shelf adversarial attack approaches on malware detection systems through which we found that those approaches do not necessarily maintain the functionality of the original inputs. Then, we discuss an approach for achieving a high misclassification rate and maintaining the executability and functionality of the original input. We use visualization-based malware detection as an example to highlight the gap between blue-sky research that focuses on aspect of the learning process, and call for more practical techniques that respect the semantics of the underlying applications.**

***Index Terms*—Malware Detection; Visualization; Adversarial Examples; Deep Learning**

I. Introduction

As the threat of malicious software (malware) is growing rapidly, malware detection is becoming a higher priority within the computer security research community and software security industry alike. To address this threat, there has been a wide variety of research works presenting various techniques for detection, using signatures [1], static analysis [2], and dynamic analysis [3]. Although these methods are powerful in analyzing malware, they have multiple shortcomings, such as their limited scalability [4]. The static analysis for malware detection is ineffective, since it is prone to obfuscation. Dynamic analysis, while addressing obfuscation by observing actual behavior of malware at the execution time, is often resources-intensive, and subject to dynamic analysis evasion (e.g., sandbox detection), making it insufficient alone. Those shortcomings and the need for an automation technique that extrapolates insight into software from features of behavior have opened a new avenue for malware analysis through visualization, often incorporated into automatic malware detectors that use machine and deep learning techniques [5], [6].

Visualization has been leveraged over both shallow and deep learning algorithms, and in many security applications, including computer forensics [7], network monitoring [8], and software security [5], [6], [9]. In malware analysis of visualization techniques, hard-to-analyze code information is transformed into an image, where features of different executable files are visualized. Compared to conventional methods, including those mentioned above, the detection process using visualization is simpler and faster (analogous to static analysis), due to the viability of deep learning approaches in recognizing patterns, even over obfuscated code [10].

The use of visualization techniques along with deep learning models has been shown to improve the performance of malware detection in multiple ways: reduced time and memory consumption, improved robustness and detection accuracy, and being non-intrusive [5], [11], [12], [13]. To this end, several studies have incorporated deep learning-based networks for detecting malicious executable files from benign ones using visualization techniques [14], [15].

Although applications of deep learning models are actively explored in a wide range of domains, *e.g.*, health-care, industry, and cyber-security [16], [17], deep learning itself has been shown to be vulnerable to AEs [18], [19]. AEs are created by adding a small perturbation to the original input of a machine learning algorithm to produce the adversaries' desirable outputs, such as misclassification [20]. Since the crafted samples are generated by applying limited changes on the original inputs, the crafted samples are very similar to the original ones, and are not necessarily outside of the training data manifold. Algorithms crafting adversarial samples are designed to minimize the perturbation, thus making it hard to distinguish adversarial samples from legitimate ones.

While generating adversarial samples, it is important to ensure executability of the resulting samples (code), *a common shortcoming of the prior work*, which make up the basis for our vision (the need for practical adversarial examples). Liu *et al.* [21] presented adversarial attacks on machine learning-based malware visualization detection methods, although they considered the entire sample as a candidate for perturbation, corrupting the adversarial sample's code. Of note in this direction of generating adversarial samples while not corrupting the sample is the seminal work of Grosse *et al.* [12], who investigated adversarial attacks on neural network-based android malware detection models. While a step in the right direction, perturbations were added into malware's none functional parts, producing a misclassification rate of only 63%, which is quite low. Moreover, although their method maintains the functionality of the code, it does not necessarily maintain its executability, since changing the application manifest will affect the permissions, software and hardware components access, intents, etc., which are necessary for the malicious applications to execute the code by the correct API calls.

While adversarial learning has been an active area, there is very little research done on understanding the impact of adversarial attacks on malware detection [12], [22], [23], especially using visualization. The work in [22], [23] suggested injection of a sequence of bytes for AE generation. Our vision is based on a work that differs in multiple points. First, our work tackles the visualization-based malware detection approach, which has

978-1-7281-8543-9/20/$31.00 ©2020 IEEE
DOI 10.1109/TPS-ISA50397.2020.00025

been proven to be faster and more efficient [12], [13]. Second, we want to highlight the targeted misclassification attacks, in addition to untargeted misclassification which was the focus of the work in [22], [23]. Finally, we want to advocate the generalization of the proposed method by evaluating it using various sets and algorithms (transferability).

Contributions. Motivated by the aforementioned issues, our main goal is generating *adversarial malware samples that (1) fool the classifier and (2) execute as intended.* First, we investigate the executability of the adversarial examples generated by leveraging adversarial attack methods into deep learning-based malware visualization detection systems. In particular, we explore multiple well-established adversarial attack methods with the visualization-based approach to generate adversarial samples with least L_2-distance from original input. We find that although the generated adversarial examples can successfully fool the classifier, none of them is still executable. Therefore, we further propose COPYCAT, which operates by padding binaries of a certain class into the binaries of another label, to maintain the executability of the original input, while forcing the model to act as desired by the adversary, with targeted and untargeted misclassification.

Organization. In section II we provide a brief background. We describe a realization of our main vision captured in COPYCAT in section III, and follow it with various evaluations and discussions in section IV. We review the related work in section V. Concluding remarks are presented in section VI.

II. Background and Preliminaries

We incorporate malware visualization from the computer-vision community, and the deep learning algorithms and adversarial attacks from the artificial intelligence community into a single process for generating adversarial malware samples. In this section, we describe some background information and basic models. First, we need to transform malware binaries into images from which features are extracted, as described in §II-B. In §II-C, we describe the high-level idea of the adversarial machine learning and its importance in the domain of deep learning-based malware detection systems. We note that we will provide more technical details of AEs generation methods in §III-A.

A. Deep learning-based Malware Detection

In COPYCAT, and as a baseline, we design a deep learning-based classifier based on the Convolutional Neural Network (CNN) architecture. CNNs are a type of deep, feed-forward artificial neural networks, widely used for image classification tasks [24], [25]. Deep learning models offer several advantages compared to traditional machine learning algorithms. For example, they extract and learn representative features automatically, which in turn means that they require minimal pre-processing effort compared to traditional machine learning methods where features have to be extracted through feature-engineering algorithms using domain knowledge [26].

The CNN architecture consists of three consecutive convolutional layers with ReLU activation, followed by a fully connected layer outputting softmax function values for each class. We tested different values for the number of convolutional layers, batch size, and epochs to improve the classification accuracy of the model over test samples of the two malware datasets. For the Windows-based malware binaries we achieved 99.12% accuracy rate after 50-epoch training and batch size of 150. Similarly, we trained another CNN model with the same architecture for IoT malware binaries and achieved 99.67% accuracy rate on test samples after 50-epoch training and a batch size of 150. More detailed information about convolutional neural networks can be found in [24].

B. Visualization

The different sections of code containing specific information about the software (malware) can be converted and organized as a two-dimensional array of unsigned integers, which is then transformed into a grayscale image. Recently, various visualization methods [5], [27], [11] have been proposed to analyze malware using this idea. The main goal of malware visualization techniques is conversion of hard-to-identify code information into an image, and acceleration of malware detection process [28]. The visualization process starts reading the malware samples as 8-bit unsigned integer vector, and categorizing it into a set of two-dimensional arrays. The values in the array are transformed into a range of 0 and 255 that represents a grayscale image. The pixels corresponding to 0 result in black pixels, while 255 result in white pixels. It is worth noting that the images' width is determined and fixed based on the characteristics of the dataset, while the height of the image can change based on the size of the malware sample. The output of the visualization step would be images with the same width and probably different heights. However, before applying the learning algorithm, we need to re-scale the images to a fixed width and height.

Illustration. A high-level flowchart of the malware visualization process is illustrated in Figure 1. In this process, each malware binary is first converted into an 8-bit vector and then transformed into a grayscale image with a fixed width of w_1 (vector to matrix conversion). The height of the image depends on the original malware size. However, the re-scaled images of all binaries are all of the same width and height, where w_1 and w_2 are equal while h_1 and h_2 are not necessarily equal.

We note that each binary file consists of various sections, such as `.text`, `.rdata`, `.data`, `.rsrc` [29]. The `.text` section contains the executable code, the `.data` section includes both uninitialized code and initialized data, while the `.rsrc` section contains all resources of the module. All sections are combined to construct the image corresponding to the binary, and four different samples from our benign and malware samples are visually illustrated in Figure 2, where 2(a) and 2(b) show benign and malicious Windows binaries, while 2(c) and 2(d) show benign and malicious IoT binaries, respectively. In Figure 2, different fragments of benign and malware samples represent unique image textures, which are leveraged in the deep learning to detect malicious binaries [30].

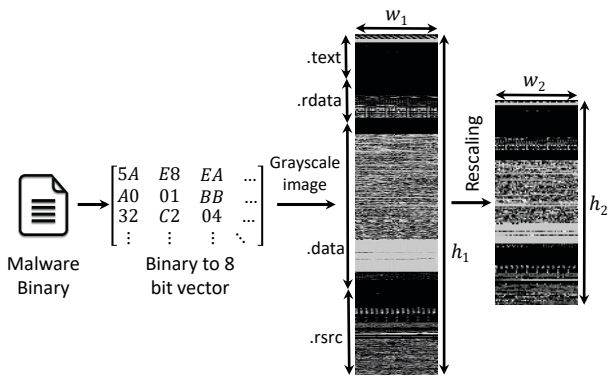
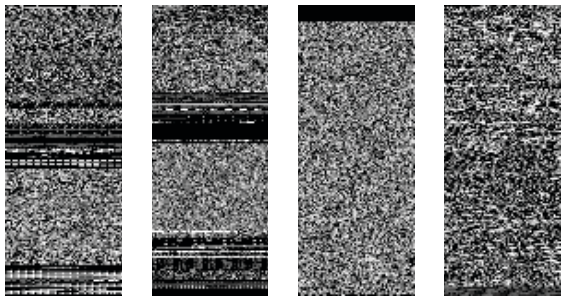


Fig. 1. The flowchart of the visualization process. Note that $w_1 = w_2$, while not necessarily the case for h_1 and h_2.



(a) Win. ben. (b) Win. mal. (c) IoT ben. (d) IoT mal.

Fig. 2. Sample visualization of benign and malicious binaries. Here, 2(a) and 2(b) show benign and malicious Windows binaries, respectively. While, benign and malicious IoT binaries are shown in 2(c) and 2(d), respectively.

C. Attacks in Deep Learning (Threat Model)

The main goal of the attacks against deep learning is to modify an input sample x such that it is misclassified by the model f, where it has been shown that an attacker is able to craft AEs, e.g., x', by applying a small perturbation to the original input such that the model is forced to produce an output of the attacker's choice [31], [32], [33]. Adversarial learning attacks on deep learning models can be categorized from multiple point of view, and based on the adversary's goals and capabilities [34], as highlighted in the following.

Goals. The adversary in deep learning aims to force the model into incorrect results. The main goal of the adversary can be represented based on the nature of the incorrectness.

- **Confidence reduction.** The goal is to reduce the classifier's confidence, by presenting ambiguity in the predictions of the model. This attack may not result in incorrect outputs.
- **Untargeted misclassification.** Producing AEs, e.g., x', that force the model output $f(.)$ to be any class other than the original one; this is, $f(x') \neq f(x)$.
- **Targeted misclassification.** Crafting AEs, e.g., x', that force $f(.)$ to be any specific class $f(x') = t$. Note that this type of attack is more complex than the confidence reduction and the untargeted misclassification attacks.

Capabilities. Considering attacks conducted at the test time, the adversarial attacks can be categorized based on the information and capabilities made available to the adversary into white-box attacks and black-box attacks.

- **Model architecture and training data.** The adversary has full knowledge and access to the training data x and deep learning model f used for classification, including the training functions and algorithms, the number and type of layers, neurons' activation functions, link weights, etc.
- **Model architecture** The adversary has full knowledge about the model architecture and associated activation functions, the number of layers, link weights, bias, etc.
- **Training data** The adversary builds a new deep learning model based on his collected dataset that is representative of the original dataset in both characteristics and distribution. Note that the adversary has no information regarding the original model and its properties.
- **Oracle** The adversary has oracle access to the model and no prior knowledge about the architecture and its internals. Moreover, we assume that the adversary can learn about the relationship between supplied inputs and the corresponding output of the model through rate-limited/unlimited queries.
- **Sample** The adversary is able to collect pairs of inputs and outputs associated with an unknown deep learning model. However, the adversary cannot change the input value to investigate its impact on the model output. Therefore, it is only practical when a large number of samples are available.

Threat Model. In this work, we designed two sets of configurations for COPYCAT depending on the utilized approach. While the AE padding approach assumes that the adversary has full knowledge of the topology of the model, the link weights, etc., the sample injection configuration is a black-box attack. Moreover, we assume that the adversary's goal is to conduct both targeted and untargeted attacks. Based on that, part of this study falls into the standard white-box attack, while the other part falls into black-box attack.

III. Methods and Techniques

There are several studies that incorporate deep learning-based classifiers for detecting malicious binaries from benign ones. In addition, it has been shown that deep learning models are not immune to adversarial attacks, and can be easily fooled. Although the concept of adversarial machine learning is an active research area, there are very few research works that have investigated the impact of adversarial attacks on deep learning-based malware detection systems [21]. Therefore, the key goals of our study are to investigate the impact of adversarial attacks on the performance of deep learning-based malware detectors, as well as generating adversarial malware samples that are executable. In particular, we aim to generate adversarial samples that can fool the deep learning model such that malware binaries are classified as benign, and vice versa.

To do so, we design two different configurations: an adversarial attack configuration, and executability configuration. The first configuration utilizes several existing attack

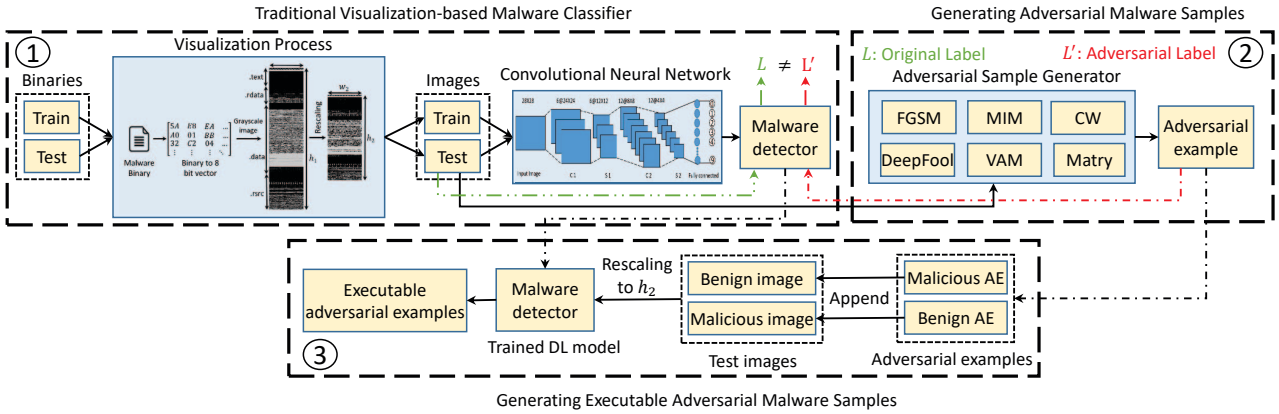


Fig. 3. General structure of the visualization-based generic adversarial attacks. As shown, it consists of three modules. Module 1 is for the visualization-based malware detector, module 2 is for generating adversarial samples based on generic adversarial learning algorithms. Note that the AEs generated at module 2 are not necessarily executable. Thus, to maintain the executability of the crafted AE, the generated AEs from 2 are being appended to the original test images in the third module.

methods—previously unexplored for this problem domain—to generate adversarial binary samples. In the executability configuration, we focus on generating adversarial samples that are not only successful in fooling the classifier, but also executable. In the following, we will describe a step-by-step aspect of these configurations.

A. Generic Adversarial Attacks

To investigate off-the-shelf adversarial attacks on malware detection, we utilized five generic algorithms capable of generating AEs. The primary goal of this configuration is to generate AEs that successfully fool the traditional visualization-based malware detection designed using deep learning networks. This attack configuration consists of two modules, the traditional visualization-based malware detection system (① in Figure 3) and adversarial example generator module (② in Figure 3), reviewed in the following.

Visualization-based Malware Detection System. This module consists of visualization and deep learning sub-modules. The visualization sub-module transforms the binary of each (potential) malware sample into a grayscale image, while the deep learning sub-module, which is based on a convolutional neural network with multiple layers, is designed and trained on a training set of grayscale images and evaluated using the test images for tuning. Note that this model builds the baseline model for malware detection, which will be mainly used for adversarial binary generation in the subsequent modules.

Adversarial Example Generator. This module aims to create AEs using off-the-shelf adversarial machine learning algorithms. The adversarial sample generator module, shown in Figure 3, takes the trained model and test set of grayscale images as inputs and generates adversarial binaries using one of five different algorithms: the Fast Gradient Sign Method (FGSM) [35], Carlini and Wagner method (C&W) [31], DeepFool [18], Momentum Iterative Method (MIM) [36], and Projected Gradient Descent (PGD) [37].

Shortcomings. This attack configuration assumes the availability of an intermediary state, rather than the malware binary, where binaries are already transformed into grayscale images (visualized) accessible by the adversary, and the adversary tries to force the trained DL model into misclassification. While this constitutes a major shortcoming of this configuration, it is not the only one: another shortcoming is that the generated AEs in this configuration are not guaranteed to be executable since the applied perturbation into the grayscale image may result in corrupted binary code.

In conclusion, this configuration is well-suited to cases where the detection design is based on the visualization approach alone, where malware binary code is readily transformed into grayscale images. This configuration, however, will not be effective with end-to-end detection (i.e., where the detector starts with the binary, does the transformation via visualization, then learning and detection).

B. COPYCAT

In a nutshell, generating adversarial samples while affecting the executability of the application is counter-intuitive, and is an easy way to infer whether a piece of software is malicious or benign (by attempting to execute it). Moreover, altering a malware sample while not paying attention to modifications in the malicious piece of code (by, for example, overriding it), while may fool the classifier, goes against the whole point of adversarial example generation from a practical standpoint. Therefore, we propose two approaches to generate samples that 1) fool the classifier, and 2) maintain the executability of the original binary (malicious or benign).

COPYCAT is inspired by the work of Cha *et al.*'s [38], where binaries of an application specified by a certain architecture are padded into an application from another architecture to change the behavior and application logic depending on the system used for running it. As such, we consider the machine learning model as an additional system on which

a sample needs to operate, along with the actual system on which the binary will run, and customize our sample to operate successfully on both as intended. Note that, and unlike the work of Cha *et al.*'s [38], our program sizes do not need to be identical in size: we have the advantage of constructing a sample of a minimal size only to fool the classifier, while not altering the actual sample for a guaranteed executability.

Executable Adversarial Examples. COPYCAT contributes two methods for executable adversarial samples generation:

- **AE Padding.** Our first method is called "AE Padding", in which COPYCAT generates an AE x' for an original sample x using one of the five previous attack methods. Then, we convert x' to an adversarial image with the same dimensions of the original sample image, and append the adversarial image to the end of the original sample image. Appending pixels at the end is equivalent to appending the code into a non-reachable code area, which will not affect the executability but will change the output pattern of the visualization, and likely leading to a misclassification (which needs to be confirmed). The general architecture of this configuration of COPYCAT is shown in Figure 3, where the third module in ③ is responsible for appending AEs, generated in ②, to the end of the original sample image. Note that the appended images were not necessarily executable, whereas the output of the third module is executable.
- **Sample Injection.** The second method is called "Sample Injection", in which we inject the binaries of a sample from the targeted class into an unreachable area of the binaries (code) of the attacked sample, usually at the end of the code. This process will ensure the executability of the original binary while changing the pattern of the visualization, leading to a model misclassification.

Concurent Work. Two other concurrent studies [22], [23] proposed similar approaches to ours. However, unlike those concurrent studies which produced untargeted misclassification attacks, COPYCAT produces both targeted and untargeted misclassification. In other words, COPYCAT can force the model to label the original input as any other desired class. This is a particularly important objective in malware family classification applications, since classification, for example, is used for attribution, designing the correct remedies, and risk management procedures, etc. Moreover, although a minor contribution, we explored and confirmed that our approach generalized equally well to IoT malware. Moreover, we show that COPYCAT is able to achieve an untargeted misclassification rate of 98.90%, compared to the untargeted classification results of 60% in the concurrently published work [22]. Finally, COPYCAT was able to achieve a targeted misclassification rate of 73.50% and 96.5% for Windows and IoT samples, respectively, which are promising. We use a CNN model with a general architecture that can accommodate multiple types of malware, unlike MalConv [22].

IV. Evaluation and Discussion

We evaluate the performance of COPYCAT for generating executable AEs. To do so, we assume that the adversary aims to conduct both targeted and untargeted attacks, with white-box access to the model. We evaluate the performance of COPYCAT against two sets of datasets, namely Windows and IoT binaries. In the following, we review the datasets, the experimental setup, the results and findings, and a discussion.

A. Dataset

We used two datasets: Microsoft Windows malware binaries and Internet of Things malware binaries. We used those two types of malware for the following reason. First, while Windows malware is popular, IoT malware is on the rise, and deserves consideration in evaluations. Second, because machine learning algorithms outcomes depend on the used dataset, having multiple datasets would be better than one.

We note no prior work used such emerging IoT malware dataset, although our work can be easily applied to other malicious binaries, such as Android applications as well. To this end, in the following, we review the used datasets.

Windows Malware Binaries. To carry out our experiment on representative Windows malware samples and facilitate reproducibility of results, we used open-source malware dataset, called BIG 2015. BIG 2015 is introduced in the Kaggle Microsoft Malware Classification Challenge [30]. The dataset consists of 10,678 labeled malware samples classified into nine different malware families, and 1,000 benign samples randomly selected from valid executables of Windows 10 binaries. Table I shows the distribution of the samples across the nine malware families.

IoT Malware Binaries. We select IoT malware for their prevalence, as they are very popular for the increasing use of IoT devices, their flexibility (Linux applications), and portability (developed for multiple platforms). For our IoT binaries, we obtained a set of 2,899 randomly selected IoT malware samples in binary format from the IoTPOT project [39]. In addition, we collected a set of 207 benign IoT binaries, for a total of 3,106 IoT binaries.

Table II shows the distribution of the malware samples across families, the variety in malware IoT samples helps with creating more robust models exposed to more patterns and behaviors which increases the decision confidence of the classifier. Table III shows the labels used in the training process of the DL model, summing up the number of benign and malicious samples outlined above.

B. Experimental Setup

For a transparent evaluation, in the following we describe the experimental setup we use for generating the AEs.

Algorithms implementation. All attack methods are implemented using Cleverhans [40], a python library with state-of-the-art implementations for various adversarial methods.

Parameters. Each attack method has different parameters. We conducted several experiments with different values to achieve

TABLE I
FAMILY-LEVEL DISTRIBUTION OF WINDOWS MALWARE.

Malware Family	Number of Samples	
	#	%[1]
Ramnit	1,534	13.14%
Lollipop	2,470	21.15%
Kelihos ver3	2,942	25.19%
Vundo	451	3.86%
Simda	41	0.35%
Tracur	685	5.87%
Kelihos ver1	386	3.30%
Obfuscator.ACY	1,158	9.91%
Gatak	1,011	8.65%
Benign Sample	1,000	8.56%
Total	11,678	100%

TABLE II
DISTRIBUTION OF IOT MALWARE SAMPLES ACROSS FAMILIES (LABELS ASSIGNED BY ANTIVIRUS SCANNERS). %[1] REFERS TO THE REPRESENTATION OF FAMILIES AMONG MALWARE SAMPLES.

Malware Family	Number of Malware	
	#	%[1]
Gafgyt	2,609	89.99%
Mirai	185	6.38%
Tsunami	67	2.32%
Singleton	32	1.10%
Hajime	6	0.21%
Total	2,899	100%

a high misclassification rate. For example, DeepFool has two parameters to be tuned, the overshooting value and the number of iterations, which were tuned to 100 and 0.05 to achieve 96.43% accuracy with the Windows malware model. The parameters of each attack method are reported in Table IV, while the obtained results for other attacks are reported in Table V and Table VI for the Windows and IoT datasets, respectively.

Evaluation System. All experiments are conducted using Python 3.6 running over Ubuntu 16.04, using a system of i5-8500 CPU operating at 3.00GHz, 32GB DDR4 RAM, 512GB SSD, and NVIDIA GTX980 Ti GPU.

C. Evaluation Metrics

We use three evaluation metrics: the mislcassification rate (MR), perturbation magnitude (PM), and running time (RT). MR is used as an indication of the success rate for fooling the detection system. When the predicted label $\text{argmax}_k\, P(y_k|\bar{x}_i)$ for the adversarial sample $\bar{x}_i$ is not same as the correct class label y_i of the original sample x_i, this results in misclassification, and the rate is calculated as:

$$\frac{1}{n}\sum_{i} I(\underset{k}{\text{argmax}}\, P(y_k|\bar{x}_i) \neq y_i).$$

TABLE III
IOT DATASET: MALWARE AND BENIGN BINARIES.

Label	Number of Samples	
	#	%[1]
Benign	207	6.66%
Malware	2899	93.34%
Total	3,106	100%

TABLE IV
PARAMETERS USED FOR EACH ADVERSARIAL ATTACK METHODS. HERE ϵ REFERS TO THE DISTORTION PARAMETER, L.R. REFERS TO LEARNING RATE, # ITER. REFERS TO THE NUMBER OF ITERATIONS, AND O. REFERS TO OVERSHOOTING PARAMETER IN DEEPFOOL METHOD. THE SIGN (–) MEANS THE PARAMETER IS NOT APPLICABLE TO THE METHOD.

Method	ϵ	# Iter.	L.R.	O.
FGSM	0.3	-	-	-
C&W	-	100	0.1	-
DeepFool	-	100	-	0.05
PGD	0.3	250	-	-
MIM	0.3	250	-	-

We use L_p to measure the PM by p–$norm$ distance as:

$$\|\delta\|_p = \left(\sum_{i=1}^{n} \|\bar{x}_i - x_i\|^p\right)^{\frac{1}{p}}$$

For p–$norm$, studying the L_0, L_2 and L_∞ is very common [31]. In this study, we measure the count of changes in the adversarial example compared to the original sample and L_2 distance of the adversarial example and the original one. Note that these two metrics indicate magnitude of the perturbations. Finally, we measure the running time (RT) required to generate adversarial example (in seconds) as a metric to compare the complexity of various attacks.

D. Results and Discussion

In this section, we discuss the results for both the Microsoft Windows and IoT malware datasets.

Generic Attacks Configuration. Our goal is to generate AEs that are able to fool the DL-based malware detector. Note that this configuration is based on applying the perturbations on the images generated from malware binaries. Thus there is no guarantee regarding the executability of the crafted AEs.

Windows. We generate AEs using five generic adversarial attack methods mentioned in §III-A. Table V shows the results for the different attack methods in terms of misclassification rate, perturbation magnitude, and running time. As shown in Table V, PGD achieved the highest misclassification rate of 99.63%, while perturbing 89.14% of the pixels. However, PGD is expensive in terms of its running time. On the other hand, FGSM and C&W are the top two methods with the lowest running times, while reaching reasonable misclassification rate of 99.08% and 99.45%, respectively.

We observe from Table V that all of these generic adversarial attack algorithms change a huge portion of the original

TABLE V
GENERATING AES FOR WINDOWS BINARIES USING GENERIC ADVERSARIAL ATTACK METHODS.

Method	MR (%)	Pixels		L_2 Dist.	RT (s)
		#	%		
FGSM	99.08	8,594	83.92	25.86	**112.10**
PGD	**99.63**	9,128	89.14	23.11	1,339.60
C&W	99.45	**7,316**	**71.44**	4.09	2,071
MIM	99.54	8,732	85.27	25.08	6,875
DeepFool	99.45	7,750	75.68	**3.23**	15,545.70

TABLE VI
GENERATING AES FOR IOT BINARIES USING DIFFERENT ADVERSARIAL ATTACK METHODS

Method	MR (%)	Pixels		L_2 Dist.	RT (s)
		#	%		
PGD	**100**	9,452	92.30	23.36	3,979.80
FGSM	93.19	9,558	93.33	26.98	**51**
C&W	**100**	**8,932**	**87.22**	**2.44**	975.30
MIM	**100**	9,296	90.78	24.77	3,926.50
DeepFool	**100**	9,258	90.41	3.68	1,798.10

image's pixels (e.g., more than 71%), which in turn correspond to parts of the original binaries. Thus, it can be easily expected that the applied perturbation on the grayscale images may result in a corrupted binary that will not be executable.

IoT. The same set of attacks as in §IV-D are applied to the IoTPOT dataset. As such, the results with the IoTPOT dataset are shown in Table VI. As shown, we can observe that all adversarial attack methods achieved a misclassification rate of 100%, except for the FGSM method. Here the C&W method is shown to outperform other methods considering the L_2 distance between the generated AEs and original inputs, although with a larger number of changed pixels. In particular, we notice that the number of changed pixels varies from 87.22% in C&W to 93.33% in the FGSM approach. The huge number of changed pixels in the generic adversarial approaches may make the crafted adversarial binaries unexecutable, which is not desirable in effective evasion techniques for software.

Executability. We note that *all* AEs resulting from both malware datasets, using all methods above were unexecutable.

COPYCAT. The results of our proposed approach, COPYCAT, for generating executable adversarial binaries are reported as followed. **Windows.** we introduced AE padding and sample injection. Figure 4 shows the results of the AE padding method, by appending the adversary example, generated using generic adversarial attack methods, to the binaries of the original executable sample. As it can be seen, we achieved the highest misclassification of 79.9% using MIM as our adversarial example generator. Table VII and Table IX show the results of the sample injection method, by injecting benign sample binaries at the end of malware sample binaries (M2B), and vice versa (B2M)—when we inject a benign sample into a malware sample we are in fact converting a

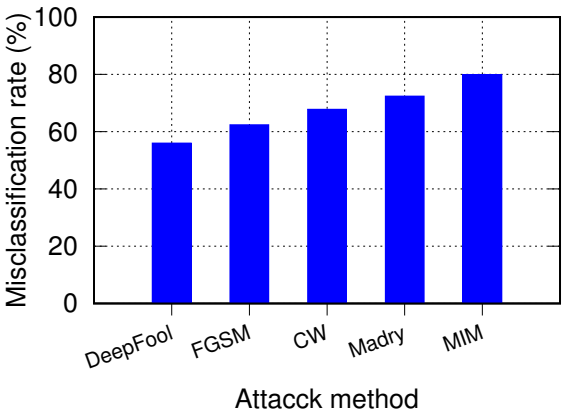


Fig. 4. Misclassification rate of benign executable files due to appended malicious adversarial samples. We found that the misclassification rate is highly correlated with the # of distorted pixels and the L_2 distance.

malware sample into a benign sample (M2B). In the first case, we achieved an overall misclassification rate of 98.9%, while the misclassification rate of M2B was 73.5% by injecting a benign sample with 4.4MB size. In the second case, we achieve a misclassification rate of B2M (regardless of the class) of 98.6% with a malware sample of size 11.5MB.

IoT. We show results for the sample injection method, which consists of injecting benign binaries into malware and vice versa. Table VII shows the results of injecting benign binaries into malware samples, by using a benign sample of size 11.5MB, we achieve a misclassification of 96.5%. Whereas Table VII shows the results of injecting malware binaries into benign samples, achieving a misclassification accuracy of 95.65% using a malware of size 2.4MB.

Executability. *All* AEs that resulted from both malware datasets, using the two methods above, were executable.

E. Generalization with Transferability

To investigate the generalization of COPYCAT, several experiments are conducted. To do so, we train another DL model with the following network structure: we use three consecutive fully connected layers of size 64 connected to the input vector, followed by a dropout with a probability of 0.5. Similarly, the output of the dropout function is fully connected with another softmax dense layer. Throughout our experiments, we found that the injection of malware samples can force this Deep Neural Network (DNN)-based model to misclassification, with a similar performance to that of the CNN-based malware detection system. The results of the benign-to-malware misclassification attack are shown in Table VIII, where it is concluded that the generated adversarial binaries are general, and can be used to possibly fool other models with different structures. For instance, the sample injection configuration of COPYCAT is able to force the DNN-based malware detection system to misclassify benign samples to malicious with a varying misclassification rate from

TABLE VII
MISCLASSIFICATION RATE FOR VARIOUS SETTINGS WITH BOTH WINDOWS AND IOT MALWARE SETS

Size vs. MR (B2M; Win)		Size vs. MR (M2B; IoT)		Size vs. MR (B2M; IoT)	
Malware size	MR(%)	Benign size	MR (%)	Malware Size	MR (%)
0.82 MB	82.1	1.0 MB	67.5	256 KB	56.50
1.10 MB	88.5	1.9 MB	76.7	553 KB	92.27
2.20 MB	96.4	3.6 MB	95.5	1.1 MB	95.17
11.50 MB	98.6	11.5 MB	96.5	2.4 MB	95.65

TABLE VIII
TRANSFERABILITY RESULTS FOR VARIOUS SETTINGS WITH BOTH WINDOWS AND IOT MALWARE SETS

Transf. (B2M; Win)		Transf. (M2B; IoT)		Transf. (B2M; IoT)	
Size (M)	MR(%)	Size (B)	MR (%)	Size (M)	MR (%)
0.82 MB	90.09	1.0 MB	56.00	256 KB	85.02
1.10 MB	94.06	1.9 MB	70.50	553 KB	82.60
2.20 MB	78.21	3.6 MB	36.75	1.1 MB	95.17
11.50 MB	100	11.5 MB	80.00	2.4 MB	97.58

TABLE IX
Size vs. MR of M2B and overall (Windows).

Benign Size	Misclassification (%)	
	Overall	Benign
1.2 MB	76.2	56.1
1.4 MB	90.2	37.8
3.5 MB	89.3	63.0
4.4 MB	98.9	73.5

TABLE X
TRANSFERABILITY: SIZE VS. MR OF M2B AND OVERALL (WINDOWS).

Benign Size	Transferability (%)	
	Overall	Benign
1.2 MB	37.5	23.7
1.4 MB	72.4	18.1
3.5 MB	66.0	42.3
4.4 MB	**89.3**	**34.4**

78.21% to 100% for samples injected with a size of 2.2MB and 11.5MB, respectively. In addition, we found that the injection of a benign sample into a malicious sample can result in an evasion rate of up to 89.28% in a DNN-based malware detection system, as the detailed results are shown in Table X.

We also observed a similar set of results for the IoT dataset, as shown in Table VIII and Table VIII for benign and malicious sample injection, respectively. For example, we were able to achieve a misclassification rate of 80% and 97.58% by injecting a benign and malicious file, respectively, in DNN-based IoT malware detection.

F. Size Trade-off

Injection method's outcomes (misclassification) depend on the size of the injected sample, due to the behavior of NN in general. Neural networks are well known for recognizing patterns, and increasing the size of the injected sample will increase the portion of sample's pattern against the overall layout. In general, a larger injected sample results in a higher effect on misclassification.

The attacks discussed in §IV-D do not guarantee the executablity of the generated sample, as each pixel in the visualization corresponds to some of the data in the original sample. Changing one bit (or pixel) may corrupt the sample, leading to unexecutable adversarial example. Our proposed methods preserve the functionality and executability by padding data into unreachable point of the application, which is usually after the exit code. These methods will not affect the application in any way except by increasing its size, which we consider as a trade-off cost of executability.

V. RELATED WORK

Machine learning algorithms for malware detection have been actively pursued. For example, malware visualization is a technique that transforms binaries into images, which are then used to build machine/deep learning-based malware detection systems [28], [27]. Cui *et al.* [14] introduced a malware detection method using deep learning, and by transferring the malicious code into grayscale images. They achieved an accuracy rate of 94.5% on the Vision Research Lab dataset [41], claiming better accuracy rate and execution time compared to static [2] and dynamic feature analysis [4].

Ni *et al.* [42] proposed a malware detection system, called MCSC, built over 10,805 grayscale images consisting of nine different malware families. Su *et al.* [13] presented a lightweight IoT malware classifier that works based on visualization and CNN model, and achieves a detection rate of 94% over goodware and DDoS malware classification tasks [43]. Fu *et al.* [15] presented a fine-grained malware detector, through RF, KNN, and SVM models, that was trained over colored

images generated from malware binaries. Their method was able to achieve a malware detection accuracy rate of 97.47%, and a family classification accuracy rate of 96.85% using random forest model. Moreover, Fan *et al.* [44] presented a Metagraph2vec based malware detection system. Hou *et al.* [45] investigated android malware detection system based on structured heterogeneous information network.

Arp *et al.* [46] presented DREBIN for detecting malicious android applications based on machine learning models trained over features extracted from static analysis. McDermott *et al.* [47] introduced a Mirai detector based on deep learning with 99% accuracy. Pajouh *et al.* [48] explored using recurrent neural networks to detect ARM-based IoT malware, achieving detection accuracy of 98.18% using LSTM structures to analyze OpCode sequences. Furthermore, Wang *et al.* [49] proposed an adversary resistant neural network with an application to malware detection based on feature nullification.

Although addressing a different problem, and using a different approach, COPYCAT is inspired in part by Cha *et al.*'s [38], where they propose the idea of platform-independent programs. Moreover, the approach in this work is in line with, although concurrent to the work of Kolosnjaji *et al.* [22] and kreuk *et al.* [23]. However, COPYCAT is different in multiple aspects, as highlighted earlier; e.g., performing both untargeted and targeted misclassification, tackling the visualization-based malware detection approach, providing higher evasion rate, and targeting both windows and IoT malware. COPYCAT is able to achieve high attack success rate while generating executable adversarial binaries, unlike the work of Liu *et al.* [21], where the main focus is maximizing the transferability regardless of the executability of the generated binaries.

VI. Concluding Remarks

We have explored two methods to generate AEs that fool the visualization-based malware detection system, while preserving executability and functionality of the binaries. However, both of the proposed methods require appending data into the original sample. Generalizing this over multi-class classifiers, requires padding of at least one AE of each target class to ensure targeted misclassification. For example, to support a 10-class classifier would require multiplying the size of the original sample by approximately ten times. This, in turn, constitutes the main shortcoming of the AE padding method. In addition, our results indicate that the performance of the sample injection method positively correlates with the size of the injected sample. Furthermore, the adversarial training method depends on the injected AEs similar to the ones used by the adversary, which may not be the case in practice. It is computationally expensive and almost impossible to find adversarial inputs by all attack models, which results in that the adversarial training is not-adaptive to unknown attacks.

In this work, we have studied several generic adversarial attacking methods and incorporated them into deep learning malware detection, showing great success in fooling classifiers. To allow misclassification while maintaining executability, a shortcoming of state-of-the-art, we proposed AE padding and sample injection, two new adversarial methods to manipulate the output of the deep model while guaranteeing the functionality and executability of the generated sample. By designing three configurations; an adversarial attack configuration, and two executability configurations, we have evaluated the performance of the proposed attacks through extensive experiments. The executable configuration is capable of generating AEs that successfully force the deep learning-based model to a misclassification rate of 98.9% and 96.5% on Windows and IoT malware datasets, respectively, while preserving executability and maintaining the same functionality of the original malware.

Overall, our work makes the case for understanding the robustness of visualization-based malware detection system against adversarial examples (AEs) that not only are able to fool models, but also maintain the executability of the original input, and calls for exploring such properties in other domains of machine learning.

Acknowledgement

This work was supported in part by CyberFlorida Collaborative Seed Award (2020), NSF CNS-2007153, and the National Research Foundation of South Korea under grant NRF-2016K1A1A2912757.

References

[1] P. Faruki, V. Laxmi, A. Bharmal, M. S. Gaur, and V. Ganmoor, "Androsimilar: Robust signature for detecting variants of android malware," *Journal of Information Security and Applications*, vol. 22, pp. 66–80, 2015.

[2] H. Kang, J.-w. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 479174, 2015.

[3] A. Bulazel and B. Yener, "A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web," in *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*. ACM, 2017, p. 2.

[4] A. Al-Dujaili, A. Huang, E. Hemberg, and U. O'Reilly, "Adversarial deep learning for robust detection of binary encoded malware," in *Proceedings of the IEEE Security and Privacy Workshops, SP Workshops*, 2018, pp. 76–82.

[5] K. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," *International Journal of Information Security*, vol. 14, no. 1, pp. 1–14, 2015.

[6] B. C. M. Cappers, P. N. Meessen, S. Etalle, and J. J. van Wijk, "Eventpad: Rapid malware analysis and reverse engineering using visual analytics," in *IEEE Symposium on Visualization for Cyber Security, VizSec 2018, Berlin, Germany, October 22, 2018*, 2018, pp. 1–8.

[7] S. Chen, S. Chen, N. V. Andrienko, G. L. Andrienko, P. H. Nguyen, C. Turkay, O. Thonnard, and X. Yuan, "User behavior map: Visual exploration for cyber security session data," in *IEEE Symposium on Visualization for Cyber Security, VizSec 2018, Berlin, Germany, October 22, 2018*, 2018, pp. 1–4.

[8] R. R. Gómez, Y. Nadji, and M. Antonakakis, "Towards designing effective visualizations for dns-based network threat analysis," in *2017 IEEE Symposium on Visualization for Cyber Security, VizSec 2017, Phoenix, AZ, USA, October 2, 2017*, 2017, pp. 1–8.

[9] M. Angelini, G. Blasilli, P. Borrello, E. Coppa, D. C. D'Elia, S. Ferracci, S. Lenti, and G. Santucci, "Ropmate: Visually assisting the creation of rop-based exploits," in *IEEE Symposium on Visualization for Cyber Security, VizSec 2018, Berlin, Germany, October 22, 2018*, 2018, pp. 1–8.

[10] M. Abuhamad, T. AbuHmed, A. Mohaisen, and D. Nyang, "Large-scale and language-oblivious code authorship identification," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, 2018, pp. 101–114.

[11] A. Makandar and A. Patrot, "Malware class recognition using image processing techniques," in *Proceedings of the 2017 International Conference on Data Management, Analytics and Innovation (ICDMAI)*, 2017, pp. 76–80.

[12] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel, "Adversarial examples for malware detection," in *Proceedings of the 22nd European Symposium on Research Computer Security - ESORICS, Part II*, 2017, pp. 62–79.

[13] J. Su, D. V. Vargas, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of iot malware based on image recognition," in *Proceedings of the 2018 IEEE Annual Computer Software and Applications Conference, COMPSAC*, 2018, pp. 664–669.

[14] Z. Cui, F. Xue, X. Cai, Y. Cao, G. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Trans. Industrial Informatics*, vol. 14, no. 7, pp. 3187–3196, 2018.

[15] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, "Malware visualization for fine-grained classification," *IEEE Access*, vol. 6, pp. 14 510–14 523, 2018.

[16] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, "Predicting the sequence specificities of dna-and rna-binding proteins by deep learning," *Nature biotechnology*, vol. 33, no. 8, p. 831, 2015.

[17] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: high-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, pp. 251–266, 2015.

[18] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016, pp. 2574–2582.

[19] A. Abusnaina, A. Khormali, H. Alasmary, J. Park, A. Anwar, and A. Mohaisen, "Adversarial learning attacks on graph-based IoT malware detection systems," in *Proceedings of the 39th IEEE International Conference on Distributed Computing Systems, ICDCS*, 2019.

[20] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proceedings of the IEEE Symposium on Security and Privacy, SP Workshops*, 2016, pp. 582–597.

[21] X. Liu, J. Zhang, Y. Lin, and H. Li, "Atmpa: Attacking machine learning-based malware visualization detection methods via adversarial examples," 2019.

[22] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *Proceedings of the 26th European Signal Processing Conference, EUSIPCO*, 2018, pp. 533–537.

[23] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," in *Proceedings of the NeurIPS Workshop on Security in Machine Learning*, 2018.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of Advances in neural information processing systems*, 2012, pp. 1097–1105.

[25] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, 2012, pp. 3642–3649.

[26] "Deep learning and machine learning differences: Recent views in an ongoing debate," Mar 2017. [Online]. Available: http://www.dataversity.net/deep-learning-machine-learning-differences-recent-views-ongoing-debate/

[27] J. Zhang, Z. Qin, H. Yin, L. Ou, and Y. Hu, "IRMD: malware variant detection using opcode image recognition," in *Proceedings of the 22nd IEEE International Conference on Parallel and Distributed Systems, ICPADS*, 2016, pp. 1175–1180.

[28] D. Lee, I. S. Song, K. J. Kim, and J.-h. Jeong, "A study on malicious codes pattern analysis using visualization," in *Proceedings of the International Conference on Information Science and Applications (ICISA)*, 2011, pp. 1–5.

[29] G. Conti, S. Bratus, A. Shubina, A. Lichtenberg, R. Ragsdale, R. Perez-Alemany, B. Sangster, and M. Supan, "A visual study of primitive binary fragment types," *White Paper, Black Hat USA*, 2010.

[30] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv preprint arXiv:1802.10135*, 2018.

[31] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy, SP 2017*, 2017, pp. 39–57.

[32] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With great training comes great vulnerability: practical attacks against transfer learning," in *Proceedings of the 27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1281–1297.

[33] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the ACM on Asia Conference on Computer and Communications Security, AsiaCCS*, 2017, pp. 506–519.

[34] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, 2016, pp. 372–387.

[35] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proceedings of the 2015 International Conference on Learning Representations.*, 2015.

[36] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[37] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.

[38] S. K. Cha, B. Pak, D. Brumley, and R. J. Lipton, "Platform-independent programs," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 547–558.

[39] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: A novel honeypot for revealing current iot threats," *JIP*, vol. 24, no. 3, pp. 522–533, 2016.

[40] N. Papernot, N. Carlini, I. Goodfellow, R. Feinman, F. Faghri, A. Matyasko, K. Hambardzumyan, Y.-L. Juang, A. Kurakin, R. Sheatsley *et al.*, "cleverhans v2. 0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, 2016.

[41] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, p. 4.

[42] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Computers & Security*, vol. 77, pp. 871–885, 2018.

[43] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: analysing the rise of iot compromises," *EMU*, vol. 9, p. 1, 2015.

[44] Y. Fan, S. Hou, Y. Zhang, Y. Ye, and M. Abdulhayoglu, "Gotcha - sly malware!: Scorpion A metagraph2vec based malware detection system," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, 2018, pp. 253–262.

[45] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1507–1515.

[46] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: effective and explainable detection of android malware in your pocket," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium, NDSS*, 2014.

[47] C. D. McDermott, F. Majdani, and A. Petrovski, "Botnet detection in the internet of things using deep learning approaches." in *Proceedings of the International joint conference on neural networks 2018 (IJCNN 2018)*. ACM, 2018.

[48] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "A deep recurrent neural network based approach for internet of things malware threat hunting," *Future Generation Computer Systems*, vol. 85, pp. 88–96, 2018.

[49] Q. Wang, W. Guo, K. Zhang, A. G. O. II, X. Xing, X. Liu, and C. L. Giles, "Adversary resistant deep neural networks with an application to malware detection," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1145–1153.