A survey on practical adversarial examples for malware classifiers

Daniel Park
Department of Computer Science
Rensselaer Polytechnic Institute
parkd5@rpi.edu

Bülent Yener
Department of Computer Science
Rensselaer Polytechnic Institute
yener@cs.rpi.edu

ABSTRACT

Machine learning based solutions have been very helpful in solving problems that deal with immense amounts of data, such as malware detection and classification. However, deep neural networks have been found to be vulnerable to adversarial examples, or inputs that have been purposefully perturbed to result in an incorrect label. Researchers have shown that this vulnerability can be exploited to create evasive malware samples. However, many proposed attacks do not generate an executable and instead generate a feature vector. To fully understand the impact of adversarial examples on malware detection, we review practical attacks against malware classifiers that generate executable adversarial malware examples. We also discuss current challenges in this area of research, as well as suggestions for improvement and future research directions.

CCS CONCEPTS

• **General and reference → Surveys and overviews**; • **Security and privacy → Malware and its mitigation**; • **Computing methodologies →** *Machine learning*.

KEYWORDS

SoK, survey, malware, adversarial examples, machine learning

ACM Reference Format:
Daniel Park and Bülent Yener. 2020. A survey on practical adversarial examples for malware classifiers. In *Reversing and Offensive-oriented Trends Symposium (ROOTS'20), November 19–20, 2020, Vienna, Austria.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3433667.3433670

1 INTRODUCTION

The field of malware detection and classification has grown considerably since the introduction of hash based file signature methods [51]. With malware authors incorporating evasion techniques, such as obfuscation, into their malicious code, detection methods using static and dynamic analysis were and continue to be developed. Increased computational power has also led to many machine learning based solutions being developed alongside these analysis methods and being deployed in commercial products [10, 73].

However, in 2014 Szegedy et. al. showed that deep neural networks (DNNs) are susceptible to adversarial attacks. Grosse et. al. went on to show that this vulnerability also held true for machine learning based malware detectors and classifiers [29]. Since this work, there have been many attacks developed against popular machine learning based models such as MalConv [62], however many of these attacks are not practical. Specifically, many attacks do not generate actual malware and instead generate a feature vector that represents what a possible perturbed malicious file should look like to evade detection. It is unpractical to generate an executable program given a feature vector due to the difficulty of the inverse feature-mapping [59]. This is to say that the feature extraction process is not uniquely invertible nor is there a guarantee that a found solution would contain the same program logic as the original malware sample.

In this work, we review practical attacks against machine learning based malware classifiers and detectors, or attacks against these ML models that result in executable malware. In Section 2, we introduce and define adversarial examples and the threat models in which they are considered. Then we review practical adversarial example research in the malware domain in Section 3. We offer suggestions for future directions in this field as well as discuss any challenges in Section 4. Lastly, we conclude in Section 5.

2 BACKGROUND

In this section, we begin by briefly discussing popular machine learning methods used in malware classification and detection. Then, we introduce and define adversarial examples and categorize threat models in which the adversarial examples are considered.

2.1 Machine learning for malware

With the increasing prevalence of the internet, we have seen an exponential increase in malware and attackers. To exacerbate the problem, malware authors obfuscate their malicious code to impede detection and evade various static and dynamic analysis methods [54].

The classic approach to detecting malware was to extract a file signature for malicious samples that were found on infected systems and add them to a signature database, also known as signature-based detection [51]. For this approach, the whole sample, as well as subsets of the sample in question, must be searched for known signatures because malicious behavior can be embedded and interleaved in otherwise benign software. However, because signature-based detection relies on capturing a malware sample and then analyzing it to generate a new signature, it can only defend against already known attacks and can only attempt to defend against new or obfuscated malware [67]. Machine learning based approaches have

ROOTS'20, November 19–20, 2020, Vienna, Austria

ACM ISBN 978-1-4503-8974-7/20/11…$15.00
https://doi.org/10.1145/3433667.3433670

Table 1: In this table, we list static features that are popularly used in machine learning based malware classifiers. We loosely categorize based on the data used to generate each feature.

Data	Static Feature
Bytes	• Extract *n*-grams from byte sequences • Convert bytes to black and white pixels • Use byte sequence or hex-dump as input
Opcodes	• Opcode frequency vector • Extract *n*-grams from disassembly • Build Markov chain from opcode sequences • Generate control flow graph
API Calls	• Indicator vector based on data mining • Frequency of API call based on data mining
System Calls	• Indicator vector based on disassembly • Frequency of system call
Environment	• Hard-coded network addresses

Table 2: In this table, we list dynamic features that are popularly used in machine learning based malware classifiers. Similar to Table 1, we loosely categorize by the data used to generate each feature.

Data	Dynamic Feature
Opcodes	• N-grams extracted from program trace
API Calls	• N-grams extracted from program trace • Frequency of API call during execution
System Calls	• N-grams based on program trace • Frequency of system call during execution • Taint analysis of system information • Behavioral profiles using system interaction
Environment	• Data transfers • Network traffic and communication • File and registry edits, creation, and deletion • New processes spawned during execution

been proposed as a solution to this problem because of their ability to predict labels for new inputs. Machine learning models, such as Support Vector Machines (SVM) and K-means clustering, are used in malware classification, detection, and analysis approaches. In the classification problem, we attempt to separate malware samples into predefined malware families. Based on the training data, which consists of labeled malware samples, the learning-based model infers the classification of new malware samples. The detection problem can be seen as a sub-problem of classification. For detection, the learning-based model is used to find or *detect* malware samples when given malicious and benign executables. As detection is a case of binary classification, learning-based detection models can also be called classifiers. Classification and detection are supervised algorithms as the training data is labeled. Machine learning can also be used to augment malware analysis. Non-supervised clustering algorithms can be used to learn new similarities between malware samples [32]. Additionally, we can reason over learning-based models to better understand what makes malware malicious [7, 20]. More recently, with increased research in deep learning approaches, researchers have begun to utilize convolutional neural networks to classify and detect malware [52, 62].

In Tables 1 and 2, we provide a brief overview of static and dynamic features that are popularly used in machine learning based malware classifiers. The rest of the section describes proposed models that utilize one or more of the listed features.

2.1.1 Static Features. N-grams are currently a popular feature used for the classification and detection of malware. Kolter and Maloof proposed extracting the most relevant *n*-grams of byte codes from PE malware for classification using various machine learning models, including naive bayes classifiers, decision trees, support vector machines (SVM), and boosted models [37]. They found that in addition to classification, their model could be used for malware detection.

McBoost was introduced as a tool to quickly analyze large amounts of binaries when searching for malware and utilized a three step process [58]. The first step is to detect packers using an ensemble of a heuristic-based classifiers and two different n-grams based classifiers. If a packer is detected, the binary is unpacked using QEMU and dynamic analysis. Lastly, a separate n-gram classifier is used to detect malware that should be forwarded for additional analysis.

Santos et. al. proposed the use of *n*-grams as an alternative to file signatures based methods in 2009. In doing so, they showed that machine learning, specifically a k-nearest neighbors model, and *n*-grams can successfully be used to detect new malware samples [67]. N-grams have also been used together with dynamic features to incorporate multiple views of malware simultaneously without actually conducting dynamic analysis [5]. Recent work such as solutions proposed for the Kaggle Microsoft Malware Challenge [64, 78], demonstrate continued usage of both byte and opcode *n*-grams with classification accuracies of almost 100%.

Similar to using sequences of opcodes to generate *n*-grams, Markov chains can be extracted from the opcode trace from a program. Such a Markov chain uses unique opcodes as its states and shows the transition probabilities from one opcode to another. Anderson et. al. used the similarity between programs' Markov chains as one feature for malware detection. Similarly, Runwal et. al. [53] proposed using graph similarity between Markov chains to detect embedded malware and Shafiq et. al. [45] proposed measuring the entropy using Markov chains to detect malware.

Jang et. al. presented BitShred, a tool for malware triage and analysis [32]. BitShred hashes the *n*-gram fingerprints extracted from samples using locality sensitive hashing to reduce the dimensionality of the feature space. The hashes are used in a k-nearest neighbors model to cluster the malware samples. Additionally, the authors showed that BitShred can be used to improve previous malware detection and classification models. For example, the authors showed that BitShred can be used to hash dynamic features, such as the behavior profiles generated in Bayer et. al. [12], to reduce the dimensionality of the feature space.

Drebin conducts large-scale static analysis of Android software to extract features such as hardware usage and requests, permissions, API calls, and networking information [7]. These features are used to map the sample to a joint vector space by generating an binary indicator vector. These binary indicator vectors are used as input to a SVM, which labels a sample as benign or malicious. Importantly, Drebin takes advantage of the simplicity of their model to attribute the model's decisions to specific features. This makes Drebin more explainable than malware classifiers and detectors based on complex architectures, such as convolutional neural networks.

Nataraj et. al. proposed using malware images (black and white image representations of binaries) to detect malware [52]. Since then, researchers and commercial antivirus have used malware images to detect malware with high accuracy [23, 26, 78]. Nataraj et. al. used the malware images to create a feature vector that would be used as input to a support vector machine. However, recent work has also shown that it is effective to use the raw images as input to a convolutional neural network [23, 34, 38, 82].

Research has also been done in developing static features that give some insight into how the program behaves during run-time using control flow graphs. This research mainly revolves around constructing a control flow graph and using graph matching techniques to detect malware [13, 16]. Ma et. al. take a similar approach in using control flow graphs, but extracts a sequences of API calls in an attempt to mimic dynamic analysis [46].

Raff et. al. takes a different approach and propose a convolutional neural network (CNN) model that takes in the whole binary as input [62]. In particular, the proposed model MalConv looks at the raw bytes of the file to determine maliciousness. MalConv borrows from neural network research in learning higher level representations from long sequences and relies on CNN's ability to capture high level local invariances. MalConv works by extracting the k bytes of a file. These k bytes are padded with $0xff$ bytes to create a feature vector of size d. If $k > d$, the first d bytes of the file are used without any additional padding. This d-length vector is mapped to a fixed length feature vector by an embedding that is jointly learned with the CNN.

2.1.2 Dynamic Features. Dynamic analysis is a technique for analyzing a binary by running it in a live environment. This environment is usually a secure sandbox or test environment, such as CWSandbox [80] and Cuckoo Sandbox [1], as to keep the host machine safe. Generally, these environments are heavily instrumented in order to keep record of executed and loaded code as well as any changes made to internal files, directories, and settings. These recorded features are called dynamic features.

The most general method of extracting dynamic features is to record the frequency and sequencing of system and API calls [3, 21]. For example, Accessminer records the systems calls trace during dynamic analysis and generates an n-grams representation of each sample [42]. Accessminer labels a sample as malware if it contains more multiple instances of "malicious" n-grams with respect to some predefined threshold. Another benefit of dynamic analysis is that network traffic and communication can be captured and analyzed, as proposed in Taintdroid [22]. These features can also be used to generate different representations of malware for additional features or to reduce dimensionality.

Bailey et. al. presented a dynamic analysis tool for automated classification and analysis of malware that used dynamic analysis to record new processes that were spawned, any files that were modified, any registry keys that were modified, as well as network access and usage [11]. These recorded features were used to create a malware fingerprint that focuses on state changes instead of code sequences. These dynamic features are used to create a hierarchical clustering of malware samples using a normalized compression distance metric.

Rieck et. al. uses CWSandbox to conducts dynamic analysis similar to the work by Bailey et. al., however, extracts features using strings from the resulting text report [63]. The string frequencies are used with a SVM to classify malware samples. The authors also show that their method can be extended to malware detection by introducing a new "unknown" class without introducing benign samples into the training set.

Bayer et. al. extended upon previous work by using taint analysis to learn how information from the operating system is used by the executable [12]. Additionally, the proposed method uses an abstraction of operating systems objects and operations to create a behavioral profile. The authors argue that the abstractions are more robust against evasion due to being able to abstract away or reason about the program without bogus system calls. The extracted behavioral profiles are then used with a clustering algorithm based on locality sensitive hashing to classify malware samples.

The behavior of a program can also be modeled into graphs as in the work of Kolbitsch et. al. [35]. The authors extended upon Malspecs [18] and generated behavior graphs of programs using system calls. Each behavior graph is a directed acyclic graph where the nodes are system calls and the directed edges denote information flow. Malware samples are detected using graph matching and similarity metrics against already known malware samples.

2.2 Adversarial examples

The notion of adversarial examples was first introduced in [72] and expanded upon in [28]. Assume f is the target classifier that an adversary plans to attack. This classifier can be represented as a function $f(\cdot)$ that takes an input and assigns it a label. An adversarial example x' targeting f is generated by perturbing an original input x with δ so that $f(x) \neq f(x')$.

$$x' = x + \delta,$$

$$f(x') \neq f(x)$$

Additionally, the perturbation is generally bounded by some value ϵ using an l_2, l_1, or l_∞ norm.

$$||x' - x||_p < \epsilon$$

In the natural image or sound domain, this bound is used to ensure that the perturbations are imperceptible to a discriminator, such as the human eye or ear.

There are many ways to find δ, the most popular being the Fast Gradient Sign Method [28] and the Carlini-Wagner (C&W) Attack [15] for white box models and substitute model attacks [55] for black box models. Most attacks will use the gradient of the loss function with respect to the input to find the direction in which the input must be perturbed for the wanted change to the output. This direction is then used to find δ. We briefly discuss these and other attacks in the Appendix.

2.2.1 Threat models. A threat model is a clear definition of the adversary that is considered in a study, including all capabilities and knowledge of the adversary. In this section, we define the white box and black box threat models that are popularly used in the machine learning domain. The threat models are made up of three parts: *threat vector and surface*, *knowledge*, and *capabilities.*

Threat Vector and Surface: The threat vector and surface correspond to the means in which the adversary interacts with the model under analysis. The threat vector is the allowable input space and locations that the adversary can use to attack the model. The threat surface, or attack surface, is the collection of all such threat vectors. In this case, the threat vector and surface consist of the input and output of the machine learning model. However, the adversary's access to these surfaces is further constrained by their knowledge and capabilities.

Knowledge: The adversary's knowledge represents what we assume the adversary knows about the target model. This knowledge is then used by the adversary to construct and mount an attack. In adversarial machine learning, the adversary's knowledge can be generalized into white and black box models.

In the white box model, the adversary is assumed to have complete knowledge of the system. Thus, we assume the adversary has complete access to the target machine learning model (with weights and parameters) as well as the data used to train the model. In the black box model, the adversary is assumed to only have access to the input and output of the model. Thus, the adversary has no knowledge of the internals or training process of the model (e.g., features extracted from executables and gradient information). An adversary can also be modeled as gray box. In a gray box model, the adversary has access to the input and output of the model and more. However, as gray box covers the entire spectrum between white and black box, it must be carefully defined.

The works reviewed in this study do not agree exactly on the adversary's knowledge. Specifically, some works may assume the adversary additionally has access to malware source code, while others do not. In Section 3, each works' deviations from this general definition of adversarial knowledge will be clearly denoted.

Capabilities: An adversary's capabilities corresponds to the abilities of the adversary and the types of attacks that can be mounted. In the case of adversarial examples, we may specify an attack algorithm that the adversary will use. Furthermore, in the case of adversarial examples in the malware domain, the adversary's capabilities are limited by their knowledge. For example, with access to malware source code, an adversary can easily apply specific transformations at compile time. However, this becomes more difficult without the source code.

2.2.2 Adversarial malware examples. Most adversarial example research is conducted using natural image datasets, such as MNIST, CIFAR10, and ImageNet. However, it is necessary to consider the set of allowable perturbations that preserve functionality of adversarial malware examples.

For natural images, the pixel values are perturbed to generate an adversarial example. Any negative or positive change to a pixel value will result in a slightly altered image as long as the resulting pixel is between 0 and 255. Executable programs can be represented in a similar way. Each byte of the binary, by definition, is between *0x00* and *0xff.* Each byte's hex representation can be translated to its decimal equivalent (between 0 and 255). In this state, a byte and pixel can be perturbed using the same methods. However, an arbitrary perturbation to a byte may not result in a valid executable because executable programs exist in a discrete space. Consider the simple case of altering one byte of an executable. If the byte comes from the *.text* section of an ELF, the new altered byte may break functionality of the program by changing function arguments or resulting in bad instructions. For this reason, applying adversarial example techniques to the malware domain requires special care in the binary's construction. Most importantly, an adversarial malware example must contain the same malicious program logic and functionality as the original.

Adversarial malware examples are an immediate threat as they are evasive and malicious executables that can take advantage of many commercial antivirus software's persistent vulnerability to obfuscation and mutation [61]. This differentiates practical adversarial malware examples from an adversarial feature vector. While an adversarial feature vector also evades detection or classification, there is no immediate threat. Pierazzi et. al argue that generating an executable given an adversarial feature vector is difficult and call this the *inverse feature-mapping* problem. There is no unique solution to the inverse feature-mapping problem. In the simple case of an *n*-gram classifier, the addition of an *n*-gram can be done in multiple ways. However, they are not all guaranteed to result in an executable that contains the same program logic or executability as the original malware sample. This problem becomes more difficult when dealing with black box models, where the attacker has no knowledge of the classifier's input and internals. Pierazzi et. al. explain that there are two ways practical adversarial malware examples circumvent this: (1) A gradient-driven approach where the code perturbations' effect on the gradient is approximated and used to follow the direction of the gradient and (2) a problem-driven approach where mutations are first applied randomly before taking on an evolutionary approach.

3 PRACTICAL ATTACKS

In this section, we review practical attacks in the adversarial malware example literature, or attacks that result in executable binaries. In Table 3, we give an overview of the practical attacks in this work recording (1) if the work was evaluated against malware classifiers that use static features (2) if the work was evaluated against malware classifiers that use dynamic features, (3) the target models in their evaluation, (4) available transformations in the attack, and (5) whether the approach is gradient-driven or problem-driven.

We use terminology from [59] and organize our review into gradient-driven and problem-drive approaches as defined in Section 2.2.2. For both approaches, we further organize the literature into attacks that mainly edit bytes and metadata in Sections 3.1.1 & 3.2.1 and attacks that utilize code transformations in Sections 3.1.2 & 3.2.2.

3.1 Gradient-driven approaches

In this section, we review gradient-driven approaches for generating adversarial malware examples. We further organize the review using the attacks' available transformations.

Table 3: In this table, we summarize practical adversarial malware example algorithms by showing (1) if the work was evaluated with static features, (2) if the work was evaluated with dynamic features, (3) the target learning-based models in the evaluation, (4) available transformations in the attack, and (5) whether the attack is gradient-driven or problem-driven. We note a general trend in the use of obfuscation and further discuss this in Section 4.

Attack	Static	Dynamic	Target model	Transformation	Approach
GADGET [65]	✓	✓	Ensemble of custom machine learning and deep learning models	add API calls	gradient-driven
Anderson et. al. [6]	✓		Gradient boosted decision trees	Edit byte features and PE metadata	problem-driven
Kolosnjaji et. al. [36]	✓		MalConv [62]	Edit padding bytes	gradient-driven
Kruek et. al. [39]	✓		MalConv [62]	Edit padding bytes	gradient-driven
Demetrio et. al. [20]	✓		MalConv [62]	Edit PE header bytes	gradient-driven
Park et. al. [57]	✓		Custom CNNs and gradient boosted decision trees	semantic *NOP* insertion	gradient-driven
Song et. al. [68]	✓	✓	Commercial antivirus	Edit byte features and PE metadata, and instruction substitution	problem-driven
Yang et. al. [83]	✓		AppContext [84] and Drebin [7]	Mutates and transplants context features [84]	problem-driven
Kucuk et. al. [40]	✓	✓	Two custom classifiers using static features and one custom classifier using dynamic features	Control flow obfuscation, bogus code blocks, and add API calls	problem-driven
Pierazzi et. al. [59]	✓		Drebin [7]	Bogus code blocks and opaque constructs	problem-driven
HideNoSeek [24]	✓		Custom Bayesian classifier	JavaScript AST transforms	problem-driven

3.1.1 *Editing bytes and metadata.*

A popular method for creating practical adversarial malware examples is to add or alter bytes in unused space in the binary. Additionally, this can be done in the header to change header metadata without affecting functionality. In this section, we will review proposed attacks that use this type of transformation. Because these attacks focus on unused or "unimportant" (for execution) bytes, they do not require source code for generating their evasive malware samples. However, with the exception of GADGET [65], these attacks are still white box attacks as they require complete access to the target model to compute gradients.

In 2018, Rosenberg et. al. proposed GADGET, a software framework to transform PE malware into evasive variants taking advantage of the transferability property of adversarial examples between DNNs [65]. The proposed attack assumes a black box threat model with no access to the malware source code. However, the attack assumes the target model takes a sequence of API calls as input. To generate adversarial examples, GADGET constructs a surrogate or substitute model that is trained with Jacobian-based dataset augmentation, introduced by Papernot et. al. as an attack against natural image classifiers [55]. The dataset augmentation creates synthetic inputs that help the substitute model better approximate the target black box model's decision boundaries. This increases the probability of the transferability of the attack as both the substitute and target model will have learned similar distributions. Once the substitute model is trained, adversarial malware examples are generated by adding dummy API calls to the original malware's API call sequence. The authors call these dummy API calls semantic *nops* as the chosen API call or their corresponding arguments have no affect on the original program logic. It is important to note that the authors only add API calls, as removing an API call can break the functionality of the program. Let us say that the original API call sequence is an array w_0 where each index $j \in [0, n]$ contains an API call. Each iteration i of this process returns a new array w_i. At iteration i, an API call d is added to w_{i-1} at some index j that pushes it towards the direction indicated by the gradient as the most impactful for the substitute model's decision. This results in w_i where $w_i[j] = d$ and $w_i[j+1:] = w_{i-1}[j:]$ because all API calls in the previous sequence after index j are essentially "pushed back". This method of perturbing the input by adding dummy API calls ensures that functionality is not broken. To generate the actual executable from this adversarial API call sequence, GADGET implements a wrapper that hooks all API calls. The hooks call the original APIs as well as dummy APIs as necessary from the adversarial API call sequence. These hooks ensure that the resulting adversarial malware example maintains the functionality and behavior of the original sample, as the original sample is being executed in a sense. GADGET was evaluated against Custom models including variants of logistic regression, recurrent neural networks (RNN), fully connected deep neural networks (DNN), convolutional neural networks (CNN), SVM, boosted decision trees, and random forest classifiers. The authors also showed that their attack produces malware that is able to evade classifiers that use static features, such as printable strings.

Kolosnjaji et. al. proposed a white box attack against MalConv that generated adversarial PE malware examples by iteratively manipulating padding bytes at the end of the file [36]. Although the authors note that bytes at any location in the PE can be altered, it requires precise knowledge of the file architecture as a simple change can break file integrity. For this reason, the proposed attack focused only on byte appending. A challenge faced by the authors was the non-differentiability of MalConv due to its embedding layer. To circumvent this, the authors proposed computing the gradient of the objective function with respect to the embedding representation z instead of the input. Each padding byte is replaced with an embedded byte m that is closest to the line $g(\eta) = z + \eta n$ where n is the normalized gradient direction. However, if m's projection on the line $g(\eta)$ is not aligned with n, the next closest embedded byte is selected. By only altering the padding at the end of the file, the proposed attack does not change the program logic nor the functionality of the original malware sample. However, this also limits the total number of perturbations allowed by the attack. As explained in Section 2, MalConv extracts up to d bytes from a binary. If the size of the binary is less than d, the extracted k bytes have $(d - k)$ *0xff* padding bytes appended to it. This means that the proposed attack is limited by the size of the original malware sample.

Kruek et. al. [39] extended the work of Kolosnjaji et. al. by proposing a method for reconstructing the PE malware sample given the adversarial example's embedding. The authors found that reconstructing bytes from the perturbed embedding $z*$ is often non-trivial as $z*$ can lose resemblance to embeddings $z \in Z$ used to learn M, the function mapping padding bytes to embedding bytes. Thus, they presented a novel loss function to ensure that perturbed embeddings $z*$ will be close to an actual embedding in M. This is done by introducing a distance term in the loss function between generated embeddings and M.

Demetrio et. al. proposed *feature attribution* as a explainable machine learning algorithm to understand decisions made by machine learning models [20]. Feature attribution was based off of a technique called *integrated gradients* introduced in 2017 by Sundararajan et. al [71]. Given the target model f, an input x, and a baseline x', integrated gradients compute the attribution for the ith feature of x as

$$IG_i(x) = (x_i - x_i') \int_0^1 \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha$$

As this integral is computed on all points in the line between x and x', each point should contribute to f's classification of x as long as x' is a *good* baseline. The authors approximate this integral using summations, pulling from the suggestions in [71]. It is important to note that these contributions are calculated with respect to the chosen baseline x'. The authors selected an empty file to be the baseline for the proposed feature attribution technique. Another option for the baseline was a file with only zero bytes. However, this option was labeled malicious with a 20% probability by MalConv, going against baseline constraints laid out in [71]. Using feature attribution, Demetrio et. al. observed the attribution to each byte of input executables and found that MalConv heavily weighs the PE header section of binaries. The authors exploited this and presented a white box attack against MalConv that only alters bytes in the malware sample's header. This attack used the same algorithm presented in [36] but perturbed unused and editable bytes inside the header instead of padding at the end of the file.

3.1.2 *Code transformations.*

Many of the works above note that the proposed methods can be used to alter the *.text* section of malicious binaries as long as the program's functionality and malicious behavior is not altered. The following attacks make use of obfuscation techniques to alter the *.text* section.

Park et. al. proposed a white box attack that utilized *semantic nops*, such as *mov eax, eax* in x86 assembly, to create adversarial PE malware examples [57]. The authors attacked convolutional neural networks that used the image representation of an executable [52] as input. The image representation of an executable treats each byte as a pixel and uses the byte's decimal value as the pixel value. The proposed attack has two steps. First, an adversarial example is generated using FGSM. This adversarial example is an image and may not have the same functionality or malicious behavior as the original malware sample. In the second step, the original malware sample and the generated adversarial image are used as input to a dynamic programming algorithm that inserts *semantic nops* using LLVM passes. Similar to how API calls are added to resemble the generated adversarial feature vector in [65], the dynamic programming algorithm adds *semantic nops* in a way such that the resulting malware sample's image representation resembles the generated adversarial image from step 1. The authors went on to show that this attack can be used against a black box model because of the transferability property of adversarial examples and perturbations [50, 55]. Using a simple 2-layer CNN as a substitute model, the authors generated adversarial malware examples that also evaded black box models, one of which being a gradient boosted decision tree using byte-level features. The authors also mention that their attack works best given the malware's source code. However, in the absence of source code, binary translation and rewriting techniques can be used to insert the necessary *semantic nops*. It is important to note that introducing these techniques also introduces artifacts from binary lifting process.

3.2 Problem-driven approaches

In this section, we review adversarial malware example algorithms that take problem-driven approaches. Similar to Section 3.1, we further organize the review using the attacks' available transformations. Problem-driven approaches do not require white box access to the target for gradient information. As such, the following approaches are black box attacks.

3.2.1 *Editing bytes and metadata.*

Anderson et. al. proposed a particularly interesting attack in which a reinforcement learning (RL) agent is equipped with a set of PE functionality preserving operations [6]. The RL agent is rewarded for actions that produce malware that evades detection. Through this game, the agent learns a policy for creating evasive malware. The proposed attack makes use of the following actions that do not change the original program logic:

- Adding functions to the import table that are never used.
- Changing section names.
- Creating new but unused sections.
- Adding bytes to unused space in sections.
- Removing signer information.
- Alter debugging information.
- Packing or unpacking the binary.
- Modifying the header.

Using these actions, the RL agent is able to alter features such as PE metadata, human readable strings, and byte histograms. After up to 50,000 mutations during the training phase, the RL agent is evaluated against a gradient boosted decision tree model, shown to be successful in classifying malware [78]. The authors note that their adversarial examples should be functional by construction. However, they found that their attack breaks functionality in certain Windows PE's that make use of less common uses of the file format or obfuscation tricks that violate the PE standard. The authors claim that this can simply be fixed by ensuring the original malware samples can be correctly parsed by binary instrumentation frameworks.

Song et. al. took a different approach in generating adversarial malware examples [68]. The proposed attack randomly generates a sequence of *macro-actions* and applies them to the original PE malware sample. This is repeated until the resulting transformed malware evades detection. Once the malware sample is evasive, unnecessary macro-actions are removed from the sequence of macro-actions applied to it. This is done to minimize the probability of accidentally breaking functionality due to some obfuscation tricks. The remaining macro-actions are then broken down into *micro-actions* for a finer detailed trace of transformations leading to the adversarial malware sample. We suggest the reader peruse the original paper for greater detail on each macro and micro action, however, we briefly describe them here. Macro-actions consist of the following:

- Append bytes to the end of the binary.
- Append bytes to unused space at the end of a section.
- Add a new section.
- Rename a section.
- Zero out signed certificate.
- Remove debugging information.
- Zero out the checksum value in the header.
- Substitute instructions with semantically equivalent instructions.

Some of these macro-actions can be broken down to a sequence of smaller actions, named micro-actions. For example, the action of appending bytes can be broken down to a sequence of adding one byte at a time. The authors claim that by breaking down each macro-action, it is possible to gain insights into why a particular action caused evasion. Instead of utilizing adversarial example generation algorithms such as FGSM or the C&W attack, the proposed method instead seeks to provide a more explainable attack against machine learning models. This method was evaluated against commercial antivirus and was also found to be effective against classifiers that incorporate both static and dynamic analysis.

3.2.2 *Code transformations.*

Yang et. al. proposed two attacks to construct Android malware samples to evade detection by machine learning models, but did not use machine learning algorithms [83]. Instead of targeting misclassification, the proposed *evolution attack* focuses on mimicking the natural evolution of Android malware based on mutating contextual features (made up of temporal features, locale features, and dependency features) [84]. This is done by automating these mutation strategies through an obfuscation tool OCTOPUS and employing them on a large scale to identify "blind spots" on target classifiers. Malware families are organized into phylogenetic evolutionary trees [69] to analyze commonly shared features and divergent features within the family. Each feature mutation is then ranked by feasibility and frequency, and sorted. The top x mutations are then used to generate new malware variants. The authors also proposed a *feature confusion attack* to complement the evolution attack. The goal of the feature confusion attack is to modify the malware sample such that certain features are similar to those of benign samples. The attack begins by collecting a set of *confusing features*, or a set of features that both malware and benign samples share. For each feature in the *confusing feature* set, the number of benign and malicious samples containing that feature is recorded. If there are more benign samples, that feature is added to the target features list. The attack then mutates malware samples to include the found target features to increase probability of evasion. The proposed method was evaluated against Android learning-based malware classifiers AppContext [84] and Drebin [7]. It is important to note that while the attack does not require white box access to the target model, it does assume (1) malware source code and (2) knowledge of features used by the model.

Kucuk et. al. argued that adversarial malware examples must evade both static and dynamic machine learning based classifiers [40]. As such, they proposed an attack for PE malware utilizing bogus control flow obfuscation and API obfuscation to evade detection by models using both static and dynamic features. The applied control flow obfuscation is based off of the LLVM-Obfuscator [33]. LLVM-Obfuscator alters the control flow of a program at the LLVM-IR level by utilizing opaque predicates and never-executed bogus basic blocks with arbitrary instructions. Using differential analysis, the authors find the optimal control flow obfuscation and bogus basic blocks to generate an adversarial malware example. This perturbs static features, such as n-grams, opcode frequency, and imported API calls. The attack uses a genetic algorithm minimizing the Kullback-Leibler (KL) divergence between the frequency feature vectors of the desired target class and the adversarial malware sample. To evade a dynamic API call based malware classifier, the authors use the same genetic algorithm to determine which API calls must be obfuscated and then obfuscates them using the techniques laid out in [70]. Additionally, the same genetic algorithm is used again to determine additional API call sequences that should be added to the original malware sample, similar to the approach taken by [65].

Pierazzi et. al. proposed a black box attack targeting the Android malware classifier Drebin [59]. The authors proposed a problem-space approach that repeatedly inserts benign code blocks using opaque predicates to change features extracted by Drebin. These benign code blocks are initialized before the attack by analyzing samples in the training set for code sequences that contribute to a negative or benign label. The attack is bounded by a feasibility check to avoid excessive transformations, which may lead to increased suspicion. Additionally, the code blocks are inserted using FlowDroid [8] and Soot [75] to minimize side-effects or artifacts.

HideNoSeek differs from other attacks that apply code transformations in that it attempts to hide malicious JavaScript by transforming the abstract syntax tree (AST) to appear benign [24]. The attack begins by building ASTs of malicious and benign files to detect sub-ASTs or sub-graphs that are shared between the two classes. To create the adversarial example, HideNoSeek utilizes randomization, data obfuscation, and opaque constructs to insert benign-appearing sub-ASTs. The attack can also rewrite existing ASTs to appear benign. These attacks were conducted in a black box model against custom classifiers based on Zozzle, a Bayesian classifier that uses features extracted from JavaScript ASTs [19].

4 DISCUSSION

In this section, we discuss challenges in practical adversarial malware sample research as well as possible research directions.

4.1 Challenges

First, it should also be noted that we are in no way diminishing or downplaying the contributions of adversarial example research in the malware domain that do not result in an executable malware sample. However, we believe that extending or including a discussion about possible ways to extend the attack to result in an executable malware sample is necessary to better frame the proposed attack in a realistic adversarial environment. As adversarial example research is growing at a rapid pace, it is necessary to fully understand how these attacks can transition to the malware detection and cybersecurity field. A fully developed or proof of concept attack would also aid in the development of models robust against adversarial malware samples.

4.1.1 *Threat models.*

One challenge in this area of research is inconsistency in threat models. We believe it is necessary to clearly define the threat model considered in each study to better understand the limitations of the attacks as well as any assumptions made by the authors. In addition to the general white box and black box threat models used in the adversarial example literature, we recommend including (1) assumptions on source code availability and (2) feasibility of attack due to time or computational constraints on the adversary. Similar to the work of Papernot et. al. [55], it would be interesting to see the effects of varying the adversary's resources, e.g., limiting the allowed number of queries to the target model or incurring a cost for each iteration of the adversary's attack.

4.1.2 *Establishing baselines.*

Another challenge is in establishing baselines and ground truth. There is no consistent dataset nor are there consistent (ML or commercial) malware classifiers throughout the reviewed papers. Although all works considered in this survey boast high evasion rates against top classifiers, we are unable to fairly evaluate them against each other. Having consistency between proposed attacks and their experimental evaluation would allow for better comparisons between the attacks. However, maintaining consistent datasets and malware classifiers and conducting a fair evaluation both pose their own challenges as shown in [76]. This would also help extend the evaluation set forth by Quarta et. al., who used their framework crAVe to show that simply obfuscating or mutating malware samples can be enough to evade detection as not all anti-virus software conduct some form of dynamic analysis.

Dataset: Although transforming old malware to be evasive does show a vulnerability in malware detection, exclusion of more recent malware samples poses a risk in concept drift. For example, if malware changes drastically to target new platforms, old malware datasets may not correctly reflect malicious features and behavior. The same can be said for benign program samples. Traditionally, benign samples are scraped from fresh installations of an operating system. However, it is unclear whether these pre-installed programs are reflective of programs that a user downloads and/or scans for malicious behavior.

Malware classifiers: It is currently unclear which malware classifier is the best for evaluating an attack. As Song et. al. stated, it is also unrealistic to assume any prior knowledge of the model. We do not believe there currently is nor will there be a consistent malware detection model baseline as research in this area is still growing. However, we do suggest that future work evaluate their attacks against multiple classifiers under a black box threat model. This would be helpful in understanding the attack's transferability between various detection models that use different features in their decision-making process.

4.2 Possible research directions

In this section, we will briefly discuss currently open research questions.

4.2.1 *Defending against practical adversarial malware examples.*

Some research has already been done in evaluating the use of adversarial training in the malware domain [4, 43]. However, robust machine learning research includes many other defense strategies such as smoothing [2] and randomization [60]. It is unclear whether these approaches would transfer and defend against adversarial malware examples.

4.2.2 *Relationships between obfuscation and adversarial examples.*

Obfuscation and adversarial examples share a common goal: evade detection. Additionally, a majority of the practical adversarial malware example algorithms incorporated popular obfuscation strategies into their attack. One possible research problem is evaluating the feasibility of using more advanced obfuscation methods, such as virtualization, for adversarial example generation. It is also currently unclear what the benefit of adversarial malware examples is when compared against more traditional malware evasion techniques, such as those summarized in Bulazel et. al. [14]. It would also be interesting to extend upon the work of Song et. al. [68] and Demetrio et. al. [20] in the explainability of adversarial malware examples and use this to further develop evasive transformations.

4.2.3 Integration of static and dynamic analysis techniques. Many of the reviewed works assume that no advanced analysis is done on the malware samples prior to being tested. However, this does not always have to be the case. For example, a pre-processing step can be used to deobfuscate the evasive malware samples produced by [57] and [40] using a deobfuscation framework such as SATURN [25]. It would be interesting to see future work in attack and defense that considers using classification and detection pipelines instead of a sole machine learning model or commercial antivirus product.

4.3 Other Survey and systematization of knowledge papers

In this section, we provide other survey and systematization of knowledge papers that cover related topics.

Yuan et. al. survey adversarial attacks and defenses for deep learning [86]. They also provide applications and problem domains in which adversarial attacks can be used. Similar to this work, Maiorca et. al. provide a survey on adversarial attacks against machine learning based PDF malware detection systems [48]. Bulazel and Yener survey dynamic malware analysis evasion and mitigation strategies [14]. Ye et. al. survey the application of data mining techniques to malware detection [85]. Ucci et. al. provides a survey on malware analysis using machine learning [74]. Lastly, van der Kuowe et. al. survey popular benchmarking flaws that must be considered to fairly and accurately evaluate security research.

5 CONCLUSION

We have presented a survey of practical adversarial examples in the malware domain. The study of adversarial examples and their affect on the cybersecurity field is incredibly important as machine learning based solutions begin to be adopted in both industry and academia. We hope that this survey will provide to be useful in future research in this field.

ACKNOWLEDGMENTS

We would like to thank Adrian Dabrowski and the anonymous reviewers for their helpful comments and suggestions for improving the paper.

REFERENCES

[1] [n.d.]. Cuckoo Sandbox. https://cuckoosandbox.org/

[2] 2019. Certified Adversarial Robustness via Randomized Smoothing *(Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 1310–1320. http://proceedings.mlr.press/v97/cohen19c.html

[3] Vitor Afonso, Matheus Amorim, André Grégio, Glauco Junquera, and Paulo De Geus. 2014. Identifying Android malware using dynamically obtained features. *Journal of Computer Virology and Hacking Techniques* 11 (02 2014), 9–17. https://doi.org/10.1007/s11416-014-0226-7

[4] A. Al-Dujaili, A. Huang, E. Hemberg, and U. O'Reilly. 2018. Adversarial Deep Learning for Robust Detection of Binary Encoded Malware. In *2018 IEEE Security and Privacy Workshops (SPW)*. 76–82. https://doi.org/10.1109/SPW.2018.00020

[5] Blake Anderson, Curtis Storlie, and Terran Lane. 2012. Improving Malware Classification: Bridging the Static/Dynamic Gap. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence* (Raleigh, North Carolina, USA) *(AISec '12)*. ACM, New York, NY, USA, 3–14. https://doi.org/10.1145/2381896.2381900

[6] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. arXiv:1801.08917 [cs.CR]

[7] D. Arp, Michael Spreitzenbarth, M. Hubner, Hugo Gascon, and K. Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *The Network and Distributed System Security Symposium (NDSS)*.

[8] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Edinburgh, United Kingdom) *(PLDI '14)*. Association for Computing Machinery, New York, NY, USA, 259–269. https://doi.org/10.1145/2594291.2594299

[9] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples *(Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 274–283. http://proceedings.mlr.press/v80/athalye18a.html ICML 2018.

[10] Avast. [n.d.]. AI & machine learning. https://www.avast.com/en-us/technology/ai-and-machine-learning.

[11] Michael Bailey, Jon Andersen, Z. Morleymao, and Farnam Jahanian. 2007. *Automated classification and analysis of internet malware.* Technical Report. In Proceedings of Recent Advances in Intrusion Detection (RAID'07.

[12] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. 2009. Scalable, Behavior-Based Malware Clustering. In *In Proceedings of Symposium on Network and Distributed System Security (NDSS)*.

[13] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. 2006. Detecting Self-mutating Malware Using Control-Flow Graph Matching. In *Detection of Intrusions and Malware & Vulnerability Assessment*, Roland Büschkes and Pavel Laskov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 129–143.

[14] Alexei Bulazel and Bülent Yener. 2017. 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* (Vienna, Austria) *(ROOTS)*. Association for Computing Machinery, New York, NY, USA, Article 2, 21 pages. https://doi.org/10.1145/3150376.3150378

[15] N. Carlini and D. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*. 39–57. https://doi.org/10.1109/SP.2017.49

[16] S. Cesare and Y. Xiang. 2011. Malware Variant Detection Using Similarity Search over Sets of Control Flow Graphs. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*. 181–189.

[17] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. ZOO: Zeroth Order Optimization Based Black-Box Attacks to Deep Neural Networks without Training Substitute Models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security* (Dallas, Texas, USA) *(AISec '17)*. Association for Computing Machinery, New York, NY, USA, 15–26. https://doi.org/10.1145/3128572.3140448

[18] Mihai Christodorescu, Somesh Jha, and Christopher Kruegel. 2007. Mining Specifications of Malicious Behavior. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering* (Dubrovnik, Croatia) *(ESEC-FSE '07)*. Association for Computing Machinery, New York, NY, USA, 5–14. https://doi.org/10.1145/1287624.1287628

[19] Charlie Curtsinger, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. 2011. ZOZZLE: Fast and Precise in-Browser JavaScript Malware Detection. In *Proceedings of the 20th USENIX Conference on Security* (San Francisco, CA) *(SEC'11)*. USENIX Association, USA, 3.

[20] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2019. Explaining Vulnerabilities of Deep Learning to Adversarial Malware Binaries. arXiv:1901.03583 [cs.CR]

[21] Marko Dimjašević, Simone Atzeni, Ivo Ugrina, and Zvonimir Rakamaric. 2016. Evaluation of Android Malware Detection Based on System Calls. In *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics* (New Orleans, Louisiana, USA) *(IWSPA '16)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/2875475.2875487

[22] William Enck, Peter Gilbert, Byung-Gon Chun, Landon Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol Sheth. 2010. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *Commun. ACM* 57, 393–407. https://doi.org/10.1145/2494522

[23] Kabanga Espoir Kamundala and Chang Kim. 2018. Malware Images Classification Using Convolutional Neural Network. *Journal of Computer and Communications* 06 (01 2018), 153–158. https://doi.org/10.4236/jcc.2018.61016

[24] Aurore Fass, Michael Backes, and Ben Stock. 2019. HideNoSeek: Camouflaging Malicious JavaScript in Benign ASTs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1899–1913. https://doi.org/10.1145/3319535.3345656

[25] Peter Garba and Matteo Favaro. 2019. SATURN - Software Deobfuscation Framework Based On LLVM. In *Proceedings of the 3rd ACM Workshop on Software Protection* (London, United Kingdom) *(SPRO'19)*. Association for Computing Machinery, New York, NY, USA, 27–38. https://doi.org/10.1145/3338503.3357721

[26] Lahouari Ghouti. 2020. Malware Classification Using Compact Image Features and Multiclass Support Vector Machines. *IET Information Security* (01 2020). https://doi.org/10.1049/iet-ifs.2019.0189

[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[28] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*. http://arxiv.org/abs/1412.6572

[29] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial Examples for Malware Detection. In *Computer Security – ESORICS 2017*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer International Publishing, 62–79.

[30] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-box Adversarial Attacks with Limited Queries and Information *(Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 2137–2146. http://proceedings.mlr.press/v80/ilyas18a.html

[31] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. 2018. Prior Convictions: Black-Box Adversarial Attacks with Bandits and Priors. *International Conference on Learning Representations* (2018). https://arxiv.org/abs/1807.07978

[32] Jiyong Jang, David Brumley, and Shobha Venkataraman. 2011. BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '11)*. Association for Computing Machinery, New York, NY, USA, 309–320. https://doi.org/10.1145/2046707.2046742

[33] Pascal Junod, Julien Rinaldini, Johan Wehrli, and Julie Michielin. 2015. Obfuscator-LLVM – Software Protection for the Masses. In *Proceedings of the IEEE/ACM 1st International Workshop on Software Protection, SPRO'15, Firenze, Italy, May 19th, 2015*, Brecht Wyseur (Ed.). IEEE, 3–9. https://doi.org/10.1109/SPRO.2015.10

[34] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal. 2018. Malware Classification with Deep Convolutional Neural Networks. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 1–5. https://doi.org/10.1109/NTMS.2018.8328749

[35] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. 2009. Effective and Efficient Malware Detection at the End Host. In *Proceedings of the 18th Conference on USENIX Security Symposium* (Montreal, Canada) *(SSYM'09)*. USENIX Association, USA, 351–366.

[36] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli. 2018. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*. 533–537. https://doi.org/10.23919/EUSIPCO.2018.8553214

[37] J. Zico Kolter and Marcus A. Maloof. 2006. Learning to Detect and Classify Malicious Executables in the Wild. *Journal of Machine Learning Research* 7 (Dec. 2006), 2721–2744.

[38] D. Kornish, J. Geary, V. Sansing, S. Ezekiel, L. Pearlstein, and L. Njilla. 2018. Malware Classification using Deep Convolutional Neural Networks. In *2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. 1–6. https://doi.org/10.1109/AIPR.2018.8707429

[39] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. 2018. Deceiving End-to-End Deep Learning Malware Detectors using Adversarial Examples. arXiv:1802.04528 [cs.LG]

[40] Yunus Kucuk and Guanhua Yan. 2020. Deceiving Portable Executable Malware Classifiers into Targeted Misclassification with Practical Adversarial Examples. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy* (New Orleans, LA, USA) *(CODASPY '20)*. Association for Computing Machinery, New York, NY, USA, 341–352. https://doi.org/10.1145/3374664.3375741

[41] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. *ICLR Workshop* (2017). https://arxiv.org/abs/1607.02533

[42] Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. 2010. AccessMiner: Using System-Centric Models for Malware Protection. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '10)*. Association for Computing Machinery, New York, NY, USA, 399–412. https://doi.org/10.1145/1866307.1866353

[43] Deqiang Li and Qianmu Li. 2020. Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection. *IEEE Transactions on Information Forensics and Security* (2020), 1–1. https://doi.org/10.1109/tifs.2020.3003571

[44] Yandong Li, Lijun Li, Liqiang Wang, Tong Zhang, and Boqing Gong. 2019. NATTACK: Learning the Distributions of Adversarial Examples for an Improved Black-Box Attack on Deep Neural Networks *(Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 3866–3876. http://proceedings.mlr.press/v97/li19g.html ICML 2019.

[45] M. Zubair Shafiq Syed Ali Khayan and Muddassar Farooq. 2008. Embedded Malware Detection Using Markov n-grams. In *Detection of Intrusions and Malware & Vulnerability Assessment*.

[46] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma. 2019. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms. *IEEE Access* 7 (2019), 21235–21245.

[47] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. *International Conference on Learning Representations* (2018). https://openreview.net/forum?id=rJzIBfZAb accepted as poster.

[48] Davide Maiorca, Battista Biggio, and Giorgio Giacinto. 2019. Towards Adversarial Malware Detection: Lessons Learned from PDF-Based Attacks. *ACM Computing Survey* 52, 4, Article 78 (Aug. 2019), 36 pages. https://doi.org/10.1145/3332184

[49] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2574–2582. https://doi.org/10.1109/CVPR.2016.282

[50] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations.

[51] Peter Morley. 2001. Processing virus collections. *VIRUS* 129 (2001), 129–134.

[52] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. 2011. Malware Images: Visualization and Automatic Classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security* (Pittsburgh, Pennsylvania, USA) *(VizSec '11)*. ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/2016904.2016908

[53] Neha Runwal Richard M. Low and Mark Stamp. 2012. Opcode Graph Similarity and Metamorphic Detection. *Journal of Computer Virology* 8 (2012), 37–52.

[54] P. OKane, S. Sezer, and K. McLaughlin. 2011. Obfuscation: The Hidden Malware. *IEEE Security Privacy* 9, 5 (Sept 2011), 41–47. https://doi.org/10.1109/MSP.2011.98

[55] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (Abu Dhabi, United Arab Emirates) *(ASIA CCS '17)*. Association for Computing Machinery, New York, NY, USA, 506–519. https://doi.org/10.1145/3052973.3053009

[56] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*. 372–387. https://doi.org/10.1109/EuroSP.2016.36

[57] Daniel Park, Haidar Khan, and Bulent Yener. 2019. Generation & Evaluation of Adversarial Examples for Malware Obfuscation. *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)* (Dec 2019). https://doi.org/10.1109/icmla.2019.00210

[58] R. Perdisci, A. Lanzi, and W. Lee. 2008. McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables. In *2008 Annual Computer Security Applications Conference (ACSAC)*. 301–310. https://doi.org/10.1109/ACSAC.2008.22

[59] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *2020 IEEE Symposium on Security and Privacy (SP)*. 1332–1349. https://doi.org/10.1109/SP40000.2020.00073

[60] Rafael Pinot, Raphael Ettedgui, Geovani Rizk, Yann Chevaleyre, and Jamal Atif. 2020. Randomization matters. How to defend against strong adversarial attacks.

[61] Davide Quarta, Federico Salvioni, Andrea Continella, and Stefano Zanero. 2018. Extended Abstract: Toward Systematically Exploring Antivirus Engines. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Cristiano Giuffrida, Sébastien Bardin, and Gregory Blanc (Eds.). Springer International Publishing, Cham, 393–403.

[62] Edward Raff, J. Barker, J. Sylvester, Robert Brandon, Bryan Catanzaro, and C. Nicholas. 2018. Malware Detection by Eating a Whole EXE. In *AAAI Workshops*.

[63] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. 2008. Learning and Classification of Malware Behavior. In *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (Paris, France) *(DIMVA '08)*. Springer-Verlag, Berlin, Heidelberg, 108–125. https://doi.org/10.1007/978-3-540-70542-0_6

[64] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. 2018. Microsoft Malware Classification Challenge. *CoRR* abs/1802.10135 (2018). arXiv:1802.10135 http://arxiv.org/abs/1802.10135

[65] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. 2018. Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers. In *Research in Attacks, Intrusions, and Defenses*, Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis (Eds.). Springer International Publishing, Cham, 490–510.

[66] Sara Sabour, Yanshuai Cao, Fartash Faghri, and David J. Fleet. 2016. Adversarial Manipulation of Deep Representations. https://www.cs.toronto.edu/~fleet/research/Papers/iclr2016_conference.pdf accepted as conference paper.

[67] Igor Santos, Yoseba K. Penya, Jaime Devesa, and Pablo García Bringas. 2009. N-grams-based File Signatures for Malware Detection. In *ICEIS*.

[68] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. 2020. Automatic Generation of Adversarial Examples for Interpreting Malware Classifiers. arXiv:2003.03100 [cs.CR]

[69] Chen Su. 2006. Bioinformatics: A Practical Guide to the Analysis of Genes. *Briefings in Bioinformatics* 7, 1 (03 2006), 123–124. https://doi.org/10.1093/bib/bbk012 arXiv:https://academic.oup.com/bib/article-pdf/7/1/123/23993075/bbk012.pdf

[70] M. Suenga. 2009. A museum of API obfuscation on win32.

[71] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 3319–3328.

[72] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. *CoRR* abs/1312.6199 (2014).

[73] Microsoft Defender ATP Research Team. [n.d.]. New machine learning model sifts through the good to unearth the bad in evasive malware. https://www.microsoft.com/security/blog/2019/07/25/new-machine-learning-model-sifts-through-the-good-to-unearth-the-bad-in-evasive-malware/.

[74] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Computers & Security* 81 (2019), 123 – 147. https://doi.org/10.1016/j.cose.2018.11.001

[75] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. 1999. Soot - a Java Bytecode Optimization Framework. In *Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research* (Mississauga, Ontario, Canada) *(CASCON '99)*. IBM Press, 13.

[76] E. van der Kouwe, G. Heiser, D. Andriesse, H. Bos, and C. Giuffrida. 2020. Benchmarking Flaws Undermine Security Research. *IEEE Security Privacy* 18, 3 (2020), 48–57. https://doi.org/10.1109/MSEC.2020.2969862

[77] Xiaosen Wang, Kun He, Chuanbiao Song, Liwei Wang, and John E. Hopcroft. 2019. AT-GAN: An Adversarial Generator Model for Non-constrained Adversarial Examples. arXiv:1904.07793 [cs.CV] preprint.

[78] Xiaozhou Wang, Jiwei Liu, and Xueer Chen. 2015. Big 2015 1st Place. https://github.com/xiaozhouwang/kaggle_Microsoft_Malware.

[79] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. 2014. Natural evolution strategies. *The Journal of Machine Learning Research* 15, 1 (2014), 949–980.

[80] C. Willems, T. Holz, and F. Freiling. 2007. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security Privacy* 5, 2 (2007), 32–39. https://doi.org/10.1109/MSP.2007.45

[81] Chaowei Xiao, Bo Li, Jun yan Zhu, Warren He, Mingyan Liu, and Dawn Song. 2018. Generating Adversarial Examples with Adversarial Networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 3905–3911. https://doi.org/10.24963/ijcai.2018/543

[82] Jinpei Yan, Yong Qi, and Qifan Rao. 2018. Detecting Malware with an Ensemble Method Based on Deep Neural Network. *Security and Communication Networks* 2018 (03 2018), 1–16. https://doi.org/10.1155/2018/7247095

[83] Wei Yang, Deguang Kong, Tao Xie, and Carl A. Gunter. 2017. Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (Orlando, FL, USA) *(ACSAC 2017)*. Association for Computing Machinery, New York, NY, USA, 288–302. https://doi.org/10.1145/3134600.3134642

[84] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. 2015. AppContext: Differentiating Malicious and Benign Mobile App Behaviors Using Context. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 303–313. https://doi.org/10.1109/ICSE.2015.50

[85] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. 2017. A Survey on Malware Detection Using Data Mining Techniques. *ACM Computing Survey* 50, 3, Article 41 (June 2017), 40 pages. https://doi.org/10.1145/3073559

[86] X. Yuan, P. He, Q. Zhu, and X. Li. 2019. Adversarial Examples: Attacks and Defenses for Deep Learning. *IEEE Transactions on Neural Networks and Learning Systems* 30, 9 (2019), 2805–2824. https://doi.org/10.1109/TNNLS.2018.2886017

[87] Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating Natural Adversarial Examples. https://arxiv.org/pdf/1710.11342.pdf accepted as conference paper.

A ADVERSARIAL EXAMPLES

In this section, we gently present high level descriptions of popular white and black box attacks against DNN's. These descriptions are included to showcase the expansiveness of this research area and to hopefully spark novel applications to the malware domain.

A.1 Preliminaries

Many of the papers in this field use slightly differing terminology and variables to denote the same thing. In this section, we will provide definitions that will be used for the rest of the document for consistency.

$F(x, \theta) = y$	Neural network with parameters θ that accepts input $x \in \mathbb{R}^n$ and produces output $y \in \mathbb{R}^m$. Often, θ will be omitted if it is fixed.
$Z_F(x)$	Similar to $F(x)$ but does not include the softmax layer (i.e., the logits). $F(x) = \text{softmax}(Z_F(x))$.
$C_F(x) = l$	Assigned label for x by neural network F, i.e. $argmax_i\ F(x)_i$. F will be omitted if the network in question is clear.
$C^*(x)$	The correct label for x.
$\text{loss}_{F,t}$	Generic loss function for neural network F and target label t. Takes an input image x.

A.2 White box attacks

In this section, we summarize the most popular white box attacks in the literature.

A.2.1 L-BFGS. Szegedy et. al. [72] introduced the generation of adversarial examples using a box-constrained L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) algorithm. Given an image x and target label t, the proposed method searches for a similar image x' (measured by the l_2 distance) that is classified as t. The problem is modeled as follows:

$$\text{minimize}||x - x'||_2^2$$
$$\text{such that } C(x') = t$$
$$x' \in [0, 1]^n$$

Because finding a solution to this problem can be difficult, they instead solve the following problem:

$$\text{minimize } c \cdot ||x - x'||_2^2 + \text{loss}_{F,t}(x')$$
$$\text{such that } x' \in [0, 1]^n$$

where $\text{loss}_{F,t}$ is any loss function, such as cross-entropy loss.

A.2.2 FGSM. Goodfellow et. al. [28] introduced the Fast Gradient Sign Method (FGSM) for generating adversarial examples quickly. Given an image x, FGSM produces an adversarial x' such that

$$x' = x + \epsilon \cdot \text{sign}(\Delta\text{loss}_{F,t}(x))$$

where ϵ is chosen to be sufficiently small to avoid detection.

Iterative extension. Kurakin et. al. [41] extended FGSM to be iterative with the goal of produce adversarial examples that are closer to or more indistinguishable from the original image. This was done by taking multiple steps of size α in the direction of the gradient-sign instead of taking a single step of size ϵ. This iterative process begins with

$$x'_0 = 0$$

and each step x'_i is as follows:

$$x'_i = x'_{i-1} - \text{clip}_\epsilon(\alpha \cdot \text{sign}(\Delta\text{loss}_{F,t}(x'_{i-1})))$$

A.2.3 JSMA. Papernot et. al. [56] introduced the Jacobian-based Saliency Map Attack (JSMA) for generating adversarial examples based on understanding the mapping between input features and the computed output label. For a neural network F and a target label t, this method uses the gradient $\Delta Z_F(x)_t$ to compute a saliency map to model each input pixel's impact on the the computed label of x, $C(x)$. Using the saliency map, the most important pixel is modified. This is repeated until an adversarial example is generated or until a set threshold for pixel modifications is reached. For the exact formulation of the saliency map, we recommend reading the original paper.

A.2.4 DeepFool. Moosavi-Dezfooli et. al. [49] acknowledged the difficulty of understanding the decision making process of a DNN and instead approached a simpler version of the problem. The proposed method assumes a simplified model where the target neural network is completely linear with linearly separable classes. The proposed attack, DeepFool, generates an adversarial example for this simplified problem. This is then repeated until an adversarial example is found for the simpler model that transfers to the non-simplified problem. Similar to our suggestion for JSMA, the full formulation is best found in the original paper.

A.2.5 C&W. [15] proposed a new approach to generating adversarial examples, popularly called the Carlini-Wagner (C&W) method or attack. The C&W attack finds a small perturbation δ by solving

$$\min_{\delta \in \Re^n} \quad ||\delta||_p + c \cdot f(x + \delta) \tag{1}$$
$$\text{s.t.} \quad x + \delta \in [0, 1]^n$$

where f is an objective function that leads to $C(x) = t$ and $|| \cdot ||_p$ is the l_p norm. To solve the box constrain problem, instead of optimizing over δ as in Equation 1, the authors proposed omtimizting over ω by setting

$$\delta_i = \frac{1}{2}(\tanh(\omega_i) + 1) - x_i \tag{2}$$

There are three proposed attacks, an l_2 attack, $_0$ attack, and an l_∞ attack, but we only describe the l_2 attack as the authors reported it to be the strongest.

The l_2 attack attempts to minimize the distortion by searching for an ω that minimizes

$$||\frac{1}{2}(\tanh(\omega) + 1) - x||_2^2 + c \cdot f(\frac{1}{2}(\tanh(\omega) + 1) \tag{3}$$

where f is the function

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa) \tag{4}$$

where t is a chosen target class, κ is a parameter to adjust misclassification confidence, x' is the adversarial example, and $Z(\cdot)$ is the output of the classifier.

A.2.6 BPDA. More recently, Athalye et. al. introduced the Backward Pass Differentiable Approximation (BPDA) attack in response to defenses that obfuscated the target neural network's gradients. Simply put, the gradient of $F(x)$ is approximated by finding a differentiable approximation $G(x)$. To estimate the gradient, the adversary does backpropagation with $F(x)$ during the forward pass but with $G(x)$ during the backward pass.

A.2.7 Deep representation manipulation. Sabour et. al. [66] proposed generating adversarial examples guided by the internal representation of a deep neural network. The proposed attack uses a guide image x_g and its internal representation, $Ir_F(x_g)$, with respect to $F(\cdot, \theta)$, and applies perturbations to a different sample image x such that $Ir_F(x) \sim Ir_F(x_g)$. The actual perturbations to push $Ir_F(x)$ towards $Ir_F(x_g)$ are found using L-BFGS from [72].

A.3 Black box attacks

In this section, we summarize the most popular black box attacks in the literature.

A.3.1 Finite difference estimation. In finite difference estimation black box attacks, the adversary has access to the class probabilities output by the target model F. In this section, we describe black box attacks that take advantage of this information.

ZOO. The Zeroth Order Optimization based black box (ZOO) attack [17] is a method for creating adversarial examples that uses finite difference estimates of the gradients. ZOO adopts an iterative optimization approach, similar to the C&W attack [15]. The attack begins with a correctly classified input x. Then the adversary defines an adversarial loss function that scores perturbations δ applied to the input and optimizes over the adversarial loss function using the estimated gradients to find δ^* that creates an adversarial example. ZOO uses "zeroth order stochastic coordinate descent" to optimize the input with respect to the adversarial loss directly.

Limited Queries and Information. A similar finite difference based approach is adopted by Ilyas et. al. [30] in a query limited (QL) setting. Like ZOO, the QL attack estimates the adversarial loss's gradients using a finite difference estimation. However, the QL attack attempts to minimize the number of queries needed by the adversary to estimate the gradients using a search distribution. The QL attack updates the adversarial example using Projected Gradient Descent (PGD) as outlined in Madry et. al. [47] based on the gradient estimates, which are evaluated using Natural Evolutionary Strategies (NES) [79].

This method was later extended upon by the original authors by using a bandit optimization-based algorithm (BAND) with *gradient priors* [31]. The authors defined two types of gradient priors: (1) time-dependent priors or information from successive gradient estimations and (2) data-dependent priors or information from the structure of the inputs. BAND achieves similar success rates to the QL attack using 90% less queries.

A.3.2 Transferability. Transferability of adversarial examples was introduced by Papernot et. al. [55]. This property can be used to mount a black box attack on a target DNN by instead attacking a substitute model and *transferring* the resulting adversarial examples to the target black box model. This substitute model is generally trained for the same classification problem as the target model. Because this model is trained by the adversary, the adversary can use any of the white box attacks described above to generate an adversarial example.

Universal perturbations. Moosavi-Dezfooli et. al. [50] also demonstrated this transferability property through a universal adversarial attack. In their experimentation, the authors used DeepFool [49] to generate adversarial examples and found that an attack on model

A transfers to a model B trained for the same classification task. The authors also provide a formulation and explanation for the transferability and universality of perturbations.

Substitution attack with Jacobian-based dataset augmentation. This transferability property was extended by Papernot et. al. [55] in a substitution attack. Instead of solely relying on transferability, the authors proposed training a substitute model using data labeled by the target model. Additionally, the authors introduced Jacobian-based Dataset Augmentation, which uses a similar idea of a saliency map in JSMA [56], to approximate the target model's decision boundaries. The goal of this attack is to increase the probability of transferable adversarial examples by pushing the substitute model to learn the same decision boundaries as the target model. Using Jacobian-based dataset augmentation decreases the number of queries to the target model during the training phase of the substitute model because the training dataset is *augmented* to heavily weigh exploring decision boundaries of the model. They experimentally showed their attack successfully evading known image classifiers as well as online black box services.

Generative Adversarial Network (GAN) approach. Zhao et. al. [87] proposed using a generative adversarial network GAN [27] to generate adversarial examples. The proposed method trained a GAN with a generator G that maps latent space vectors Z to natural image samples X. An *inverter* I is separately trained to map natural image samples X to latent space vectors Z, or invert the mapping of G. Instead of generating and applying perturbations in the X space, the latent space vector $z_i = I(x_i)$ is perturbed to produce $\hat{z}_i$. An adversarial example in this model is a $\hat{z}_i$ such that $F(G(\hat{z}_i)) \neq F(x)$. This attack relies on the transferability property as the GAN is trained separately from the target model. However, the GAN can be trained using data labeled using the target model, depending on the adversary's knowledge, to incorporate the target model into the attack. Similar approaches using GANs have been proposed by Wang et. al. [77] and Xiao et. al. [81].

$\aleph$*-Attack.* Motivated by NES [79], Li et. al. [44] introduced a powerful black box attack called $\aleph$-Attack. Unlike other existing attacks that optimize perturbations specifically for an input x, the proposed attack estimates a probability density distribution centered around the input x such that a sample drawn from the estimated distribution is an adversarial example. $\aleph$-Attack essentially learns the distribution of adversarial examples and samples that distribution to generate a black box attack. The authors experimentally showed that $\aleph$-Attack was more effective than the BPDA attack [9] against ML models trained on the CIFAR10 and Imagenet datasets.