Improving Robustness of ML Classifiers against Realizable Evasion Attacks Using Conserved Features

Liang Tong, *Washington University in St. Louis;* Bo Li, *UIUC;* Chen Hajaj, *Ariel University;* Chaowei Xiao, *University of Michigan;* Ning Zhang and Yevgeniy Vorobeychik, *Washington University in St. Louis*

https://www.usenix.org/conference/usenixsecurity19/presentation/tong

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the 28th USENIX Security Symposium is sponsored by USENIX.

Improving Robustness of ML Classifiers against Realizable Evasion Attacks Using Conserved Features

Liang Tong
Washington University in St. Louis

Bo Li
UIUC

Chen Hajaj
Ariel University

Chaowei Xiao
University of Michigan

Ning Zhang
Washington University in St. Louis

Yevgeniy Vorobeychik
Washington University in St. Louis

Abstract

Machine learning (ML) techniques are increasingly common in security applications, such as malware and intrusion detection. However, ML models are often susceptible to *evasion attacks*, in which an adversary makes changes to the input (such as malware) in order to avoid being detected. A conventional approach to evaluate ML robustness to such attacks, as well as to design robust ML, is by considering simplified *feature-space* models of attacks, where the attacker changes ML features directly to effect evasion, while minimizing or constraining the magnitude of this change. We investigate the effectiveness of this approach to designing robust ML in the face of attacks that can be realized in actual malware (*realizable attacks*). We demonstrate that in the context of structure-based PDF malware detection, such techniques appear to have limited effectiveness, but they are effective with content-based detectors. In either case, we show that augmenting the feature space models with *conserved* features (those that cannot be unilaterally modified without compromising malicious functionality) significantly improves performance. Finally, we show that feature space models enable generalized robustness when faced with a variety of realizable attacks, as compared to classifiers which are tuned to be robust to a specific realizable attack.

1 Introduction

Machine learning (ML) has come to be widely used in a broad array of settings, including important security applications such as network intrusion, fraud, and malware detection, as well as other high-stakes settings, such as autonomous driving. A general approach is to extract a set of *features*, or numerical attributes, of entities in question, collect a training data set of labeled examples (for example, indicating which instances are malicious and which are benign), and learn a model which labels previously unseen instances, presented in terms of their extracted features. Success of ML in malware detection is particularly striking, with ML-based static detection of malicious entities at times exceeding 99% accuracy [36, 37].

Nevertheless, ML-based techniques are often susceptible to *adversarial examples*, an important special case of which are *evasion attacks*. In a prototypical case of an evasion attack, an adversary modifies malware code so that the resulting malware is categorized as benign by ML, but still successfully executes the malicious payload [12, 16, 26, 37, 44]. An even broader class of adversarial examples features attacks that manipulate an object, such as a stop sign, so that a computer vision pipeline misclassifies it as another object (such as a speed limit sign) [10, 15, 33].

In response, a host of methods emerged for making ML robust to adversarial examples, the most potent of which are those based on game-theoretic approaches, robust optimization (including certified robustness), and adversarial retraining [5, 15, 23, 25, 32, 42, 43, 46]. A fundamental ingredient in all of these are *feature-space models of attacks*. Specifically, the attacker is assumed to directly modify values of features, with either a constraint or a penalty on the aggregate feature change measured in terms of an l_p norm.

Such feature-space models of attacks are clearly abstractions of reality. First, arbitrary modifications of feature values may not be *realizable*. For example, adding a benign object to a malicious PDF (with no other changes) necessarily increases its size, and so setting the associated feature to 1 (from 0) and simultaneously reducing file size may not be practically feasible. Second, the key goal for an adversary is to create a target malicious effect, such as to execute a malicious payload. Limiting feature modifications to be small in some l_p norm clearly need not capture this: one can insert many no-ops (resulting in a large change according to an l_p norm) with no impact on malicious functionality, and conversely, minimal changes (such as removing a Javascript tag) may break malicious functionality. Nevertheless, an implicit assumption in robust ML approaches is that the feature-space models capture reality sufficiently to yield ML models that are robust even to realizable attacks. *The goal of our work is to evaluate the validity of this implicit assumption* in the context of PDF malware detection.

Our first contribution is to evaluate feature-space evasion attack models in the context of PDF malware detection, using EvadeML as a realizable attack [44]. Specifically, we consider four ML-based approaches for PDF malware detection: two based on features that capture PDF file structure (SL2013 [36] and Hidost [38]), and two based on PDF file content (two Mimicus variants of PDFRate [35, 37]). In all cases, we show that successful defense against a given realizable attack is feasible (by retraining with this attack). In the case of structure-based detectors, we demonstrate that adversarial retraining in the feature space does not lead to adequate robustness against realizable attacks. In contrast, adversarial retraining in the feature space is effective in the case of content-based detectors. In other words, the nature of the feature space can matter a great deal.

Our second contribution is a method for boosting robustness of feature-space models without compromising their mathematical convenience (crucial for most approaches for robust ML). The key idea is to identify *conserved features*, that is, features that cannot be unilaterally modified without compromising malicious functionality. We exhibit such features in our setting, show that they cannot be identified with traditional statistical methods, and develop an algorithm for automatically extracting them. Finally, we show that by simply constraining that these features remain unmodified in adversarial training, feature-space approaches become effective even for robust structure-based PDF malware detection.

Our third contribution is to explore the extent to which ML robustness is *generalizable* to multiple *distinct* realizable attacks. Specifically, we expose both a robust classifier that was retrained by using a realizable attack (EvadeML), and a model hardened using a feature-space attack (accounting for conserved features), to a series of realizable attacks. Our results reveal a stark difference between the two: ML models hardened using EvadeML are quite fragile; in contrast, ML models hardened using feature-space attacks exhibit uniformly high robustness to the other attacks. Remarkably, we demonstrate that ML models hardened using feature-space attacks remain robust *even against realizable attacks that defeat conserved features*.

2 Machine Learning in Security

2.1 Learning and Prediction

In the (supervised) machine learning literature, it is common to consider the problem abstractly. We are given a training dataset $\mathsf{D} = \{(x_i, y_i)\}$, where $x_i \in \mathsf{X} \subseteq \mathbb{R}^n$ are numeric feature vectors in some feature space X and $y_i \in \mathsf{L}$ are labels in a label space L. Each data point (or example) in D is assumed to be generated i.i.d. according to some unknown distribution P. We are also given a hypothesis (model) space, H, and our goal is to identify (*learn*) a good model $h \in \mathsf{H}$ in the sense that it yields a small expected error on new examples drawn from P. In practice, since P is unknown, one typically aims to find $h \in \mathsf{H}$ which (approximately) minimizes empirical error on training data D.

In security applications—as in others—one is not given numerical features; instead, we start with a collection of entities, such as executables, along with associated labels (we assume henceforth that these are available, as we focus here on supervised learning problems). We must then *design a collection of feature extractors*, where each feature extractor computes a numerical value of a corresponding feature from an input entity. For example, we extract a "size" feature by computing the size of an executable. Applying feature extractors to each entity in our dataset, and adding associated object labels, allow us to generate a dataset D to fit the conventional ML framework.

In this paper we focus on PDF malware detection, where the label space is binary: either a PDF file is benign (which we can code as -1), or malicious (which we can code as $+1$). In addition, several prior efforts presented techniques for defining *feature extractors* (commonly known simply as features) for PDF files [36, 37]. Applying such feature extractors to a PDF file dataset transforms this dataset into one comprised of numerical feature vectors and associated binary labels. The goal is to predict whether previously unseen PDFs (simulated by holding out a portion of our dataset as *test data*) are correctly labeled as malicious or benign.

2.2 Evasion Attacks

In an *evasion attack*, abstractly, one is given a learned model $h(x)$ (e.g., a SVM or neural network) which returns a label $y = h(x)$ (e.g., malicious or benign) for an arbitrary feature vector $x \in \mathsf{X}$ (e.g., extracted from a PDF file). The attacker additionally starts with an entity e (such as a malicious PDF file), from which we can extract a feature vector $\phi(e)$. The attacker then transforms e into another entity, e', with an associated feature vector $x' = \phi(e')$ so as to accomplish two goals: first, that $h(x')$ returns an erroneous label (in our running example, labels e' as benign based on its extracted features $\phi(e')$), and second, that e' preserves the functionality of the original entity e—which, in our example of PDF malware detection, entails preserving malicious functionality of e. The evasion attack as just described is presumed to transform the *entity itself*, such as the malicious PDF file, albeit accounting for the effect of such transformation on the extracted features $x' = \phi(e')$. We call attacks of this kind *realizable* evasion attacks. The process by which such realizable evasion attacks can be successfully accomplished is quite non-trivial, and typically warrants independent research contributions (e.g., [37, 44]).

In contrast, it is natural to short-circuit the complexity involved, and work directly in the *feature space*, as is conventional in the machine learning literature. In this case, the attacker is *modeled* as starting with a malicious feature vector x (*not the malicious entity* e), and *directly modifying the fea-*

tures to produce another feature vector $x' \in \mathsf{X}$, so as to yield erroneous predictions, i.e., $y' = h(x')$ (for example, being mislabeled as benign). Crucially, since we are no longer appealing to original entities, we must abstract away the notion of preserving (malicious) functionality. This is done through the use of a cost function, $c(x,x')$, whereby the attacker is penalized for greater modifications to the given feature vector x, commonly measured using an l_p norm difference between the original malicious instance and the modified feature vector [3, 23]. We term these the *feature-space models* of evasion attacks. Crucially, *essentially all approaches for robust ML, particularly the most successful ones, such as those based on robust optimization, leverage these models*.

2.3 Evasion Defense

A large number of approaches have been proposed for defending against evasion attacks or, more broadly, adversarial examples (e.g., [3, 5, 6, 29, 30, 32, 40, 42, 43]). While many have been shown inadequate [1, 7], the three generally effective approaches are: (a) game-theoretic reasoning, (b) robust optimization (a special case of (a) where the game is zero-sum), and (c) iterative adversarial retraining.[1] Game-theoretic methods in general, and robust optimization in particular, are not general-purpose, as solving these directly requires special structure, such as a continuous feature space and differentiability [3, 5, 6], and often additional structure of the learning model, such as linearity [43] or neural network architecture and activation functions [32, 42]. Finally, to date all have used the mathematical feature-space attack model at their core. In contrast, retraining can be performed without making assumptions about the nature of the learning algorithm or the adversarial model [23]. Since our study below involves realizable attacks (in addition to the mathematical models of attacks), non-linear SVM and, in all cases but one, binary features, iterative retraining is the sole defense that can be applied uniformly (which we require to ensure that our results are directly comparable).

3 Validating Models of ML Evasion Attacks

We have two major goals: 1) *validation*: to evaluate whether robust ML approaches that make use of feature-space models of evasion attacks are, indeed, robust against *real*—realizable—attacks, and 2) *generalizability*: to study generalizability of evasion defenses.

We start with a conceptual model of defense and attack as a Stackelberg game between ML ("defender"), who first chooses a defense θ (in our case, the learned model $h(x)$) and the attacker, who finds an optimal attack that *reacts* to the particular defense θ. An *attack model* captures how the attacker changes behavior in response to the defense θ. The defender's goal is to choose the best defense θ against such a reactive attacker, as captured by the attack model. Indeed, this is a common way to model the adversarial evasion problem in prior literature [5, 22, 40]. This model has two useful features. First, the attack is treated as an oracle in the sense that it returns an attack for an arbitrary defense θ. This allows us, in principle, to design a defense against an arbitrary evasion attack, making no distinction between feature-space attack models and realizable attacks. Second, we can separately consider *defense* against a specific attack (for example, a feature-space attack), and evaluation, which can use another attack (e.g., a realizable attack).

To be more precise, let $O(h;\mathsf{D})$ be an arbitrary attack which returns evasions given a dataset D and a classifier h, and let $u(h;O(h;\mathsf{D}))$ be the measure that the defender wishes to optimize (for example, accuracy on data *after* evasions). Then defense against the attack $O(h;\mathsf{D})$ amounts to solving the following optimization problem:

$$\max_h u(h;O(h;\mathsf{D})). \tag{1}$$

In practice, we need a means for approximately solving the optimization problem in Equation (1) for an arbitrary attack. To this end, we make use of *iterative retraining*, an approach previously proposed for hardening classifiers against evasion attacks [21, 23]. In particular, we use a variant of iterative retraining with provable guarantees [23], which is outlined as follows:

1. Start with the initial classifier.
2. Execute the *evasion attack* for each malicious instance in training data to generate a new feature vector.
3. Add all new data points to training data (removing any duplicates), and retrain the classifier.
4. Terminate after either a fixed number of iterations, or when no new evasions can be added.

Now, we describe our approach to validation and generalizability evaluations.

In *validation*, consider a model of an evasion attack, $\tilde{O}(h;\mathsf{D})$ (e.g., a feature-space attack model), which is a proxy for a "real" (realizable) attack, $O(h;\mathsf{D})$; note that each attack evades a given ML model h. We first find the defense against $\tilde{O}$ using the retraining procedure above; let the resulting robust classifier be $\tilde{h}$. Next, we *evaluate* $\tilde{h}$ by running the target realizable attack $O(\tilde{h};\mathsf{D})$. Finally, we create a *baseline* h^*, which is a robust classifier against a target realizable attack O. We then evaluate how well $\tilde{h}$ performs, compared to h^*, against the target attack. For example, if we find that $\tilde{h}$ is ineffective against the target attack, we say that $\tilde{O}$ is a poor attack proxy, whereas if it remains robust, we view $\tilde{O}$ as a good proxy for the target attack O. We focus on validation in Sections 5 and 6.

[1] Otherwise known as adversarial training, it can be viewed as an approach for obtaining approximate game-theoretic or robust optimization solutions [23, 25, 40].

In evaluating *generalizability*, the approach is slightly different. Again, we consider a proxy attack $\tilde{O}$ (which may now be either a feature-space model, or some particular realizable attack), and find a defense $\tilde{h}$ against this attack. For evaluation, we consider a *collection* of target attacks $\{O_i)\}$, and run each of these attacks against $\tilde{h}$. We say that our proxy attack is generalizable if $\tilde{h}$ remains robust to all, or most of the attacks i; otherwise, it fails to generalize. We consider generalizability in Section 7.

4 Experimental Methodology

We use malicious PDF detection as a case study to investigate robustness of ML hardened using feature-space models of evasion attacks. We now describe our experimental methodology. We start with some background on PDF structure, and proceed to describe the specific ML-based detectors, evasion attacks (both realizable, and feature-space), datasets, and evaluation metrics used in our experiments.

4.1 PDF Document Structure

The Portable Document Format (PDF) is an open standard format used to present content and layout on different platforms. A PDF file structure consists of four parts: *header*, *body*, *cross-reference table* (CRT), and *trailer*. The header contains information such as the magic number and format version. The body is the most important element of a PDF file, which comprises multiple PDF objects that constitute the content of the file. These objects can be one of the eight basic types: Boolean, Numeric, String, Null, Name, Array, Dictionary, and Stream. They can be referenced from other objects via indirect references. There are other types of objects, such as JavaScript which contains executable JavaScript code. The CRT indexes objects in the body, while the trailer points to the CRT.

The relations between objects with cross-references can be described as a directed graph that presents their logical structure by using edges representing reference relations and nodes representing different objects.As an object can be referred to by its child node, the resulting logical structure is a directed cyclic graph. To eliminate the redundant references, the logical structure can be reduced to a structural tree with the breadth-first search procedure.

4.2 Target Classifiers

Several PDF malware classifiers have been proposed [8, 35, 36, 38]. For our study, we selected SL2013 [36], Hidost [38] and two variants of PDFRate [35] (termed PDFRate-R and PDFRate-B respectively), displayed in Table 1. SL2013 and its revised version, Hidost, are *structure-based* PDF classifiers, which use the logical structure of a PDF document to construct and extract features used in detecting malicious PDFs. PDFRate, on the other hand, is a *content-based* classifier, which constructs features based on *medadata* and *content* information in the PDF file to distinguish benign and malicious instances. Evasion attacks on both SL2013 and PDFRate classifiers, particularly of the realizable kind, have been developed in recent literature [36–38, 44], providing a natural evaluation framework for our purposes.

Classifier	Feature type	Number of features
SL2013	Binary	6,087
Hidost	Binary	961
PDFRate-R	Real-valued	135
PDFRate-B	Binary	135

Table 1: Target classifiers.

4.2.1 Structure-Based Classifiers

SL2013: SL2013 is a well-documented and open-source machine learning system using Support Vector Machines (SVM) with a radial basis function (RBF) kernel, and was shown to have state-of-the-art performance [36]. It employs structural properties of PDF files to discriminate between malicious and benign PDFs. Specifically, SL2013 uses the presence of particular *structural paths* as binary features to present PDF files in feature space. A structural path of an object is a sequence of edges in the reduced (tree) *logical structure*, starting from the catalog dictionary and ending at this object. Therefore, the structural path reveals the shortest reference path to an object. SL2013 uses 6,087 most common structural paths among 658,763 PDF files as a uniform set for classification.
Hidost: Hidost is an updated version of SL2013. It inherits all the characteristics of SL2013 and employs *structual path consolidation* (SPC), a technique to consolidate features which have the same or similar semantic meaning in a PDF. As the semantically equivalent structural paths are merged, Hidost reduces polymorphic paths and still preserves the semantics of logical structure, so as to improve evasion-robustness of SL2013 [38].

In our work, we employ the 961 features identified in the latest version of Hidost.

4.2.2 PDFRate: A Content-Based Classifier

The original PDFRate classifier uses a random forest algorithm, and employs PDF *metadata* and *content* features. The metadata features include the size of a file, author name, and creation date, while content-based features include position and counts of specific keywords. All features were manually defined by Smutz and Stavrou [35].

PDFRate uses a total of 202 features, but only 135 of these are publicly documented [34]. Consequently, in our work we employ the Mimicus implementation of PDFRate which was shown to be a close approximation [37]. Mimicus trained a surrogate SVM classifier with the documented 135 features

and the same dataset as PDFRate, using both the SVM and random forest classifiers, both performing comparably. We use the SVM implementation in our experiments to enable more direct comparisons with the structure-based classifiers that also use SVM. An important aspect of Mimicus is *feature standardization* on extracted data points performed by subtracting the mean of the feature value and dividing by standard deviation, transforming all features to be real-valued and zero-mean (henceforth, PDFRate-R). This surrogate was shown to have $\sim 99\%$ accuracy on the test data [35]. In addition, we construct a *binarized* variant of PDFRate (henceforth, PDFRate-B), where each feature is transformed into a binary feature by assigning 0 whenever the feature value is 0, and assigning 1 whenever the feature value is non-zero.

4.3 Realizable Evasion Attacks

4.3.1 EvadeML

The primary realizable attack in our study is EvadeML [44], which allows insertion, deletion, and swapping of objects, and is consequently a stronger attack than most other realizable attacks in the literature, which typically only allow insertion to ensure that malicious functionality is preserved. EvadeML assumes that the adversary has black-box access to the classifier and can only get classification scores of PDF files, and was shown to effectively evade both SL2013 and PDFRate [44]. It employs genetic programming (GP) to search the space of possible PDF instances to find ones that evade the classifier while maintaining malicious features. First, an initial population is produced by randomly manipulating a malicious PDF repeatedly. The manipulation is either a deletion, an insertion, or a swap operation on PDF objects. After the population is initialized, each variant is assessed by the Cuckoo sandbox [17] and the target classifier to evaluate its fitness. The sandbox is used to determine if a variant preserves malicious behavior, such as API or network anomalies. The target classifier provides a classification score for each variant. If a variant is classified as benign but displays malicious behavior, or if GP reaches the maximum number of generations, then GP terminates with the variant achieving the best fitness score and the corresponding mutation trace is stored in a pool for future population initialization. Otherwise, a subset of the population is selected for the next generation based on their fitness evaluation. Afterward, the variants selected are randomly manipulated to generate the next generation of the population.

We use EvadeML as the primary realizable evasion model for the first part of the paper. We set the GP parameters in EvadeML as the same as in the experiments by Xu et al. [44]. The population size in each generation is 48. The maximum number of generations is 20. The mutation rate for each PDF object is 0.1. The mutation traces that lead to successful evasion and promising variants are stored and applied in our experiments. The fitness threshold of a classifier is 0. We use the same external benign PDF files as Xu et al. [44] to provide ingredients for insertion and swap operations.

4.3.2 The Mimicry Attack

Mimicry assumes that an attacker has full knowledge of the features employed by a target classifier. The mimicry attack then manipulates a malicious PDF file so that it mimics a particular selected benign PDF as much as possible. The implementation of Mimicry is simple and independent of any particular classification model.

Our mimicry attack uses the Mimicus [37] implementation, which was shown to successfully evade the PDFRate classifier. To improve its evasion effectiveness, Mimicus chooses 30 different target benign PDF files for each attack file. It then produces one instance in feature space for each target-attack pair by merging the malicious features with the benign ones. The feature space instance is then transformed into a PDF file using a *content injection approach*. The resulting 30 files are evaluated by the target classifier, and only the PDF with the best evasion result is selected, which was submitted to WEPAWET [8] to verify malicious functionality. To make Mimicry consistent with our framework, we employ the Cuckoo sandbox [17] in place of WEPAWET (which was in any case discontinued) to validate maliciousness of the resulting PDF file.

In addition to the original version of Mimicry, we implement an enhanced variation, *Mimicry+*, with two modifications. First, Mimicry+ chooses the 30 most benign PDF files predicted by the target classifier as target files (instead of randomly selecting those, as in Mimicry). Second, for each attack file, all the resulting 30 files are evaluated by the sandbox and only those verified to have malicious functionality are selected to evade the target classifier.

4.3.3 MalGAN

MalGAN [19] is a Generative Adversarial Network [14] framework to generate malware examples which can evade a black-box malware detector with binary features. It assumes that an attacker knows the features, but has only black-box access to the detector decisions. MalGAN comprises three main components: a generator which transforms malware to its adversarial version, a black-box detector which returns detection results, and a substitute detector which is used to fit the black-box detector and train the generator. The generator and substitute detector are feed-forward neural networks which work together to evade the black-box detector. The results of [19] show that MalGAN is able to decrease the *True Positive Rate* on the generated examples from $> 90\%$ to 0%. We note that strictly speaking, MalGAN variants are not implemented as actual PDF files; however, we still treat it as a realizable attack since it only adds features to a malicious

Entry	Hexadecimal Representation
/Action	/#41#63#74#69#6f#6e
/Filter	/#46#69#6c#74#65#72
/Length	/#4c#65#6e#67#74#68
/JavaScript	/#4a#61#76#61#53#63#72#69#70#74
/JS	/#4a#53
/S	/#53
/Type	/#54#79#70#65

Table 2: Transformation of entry names in the custom attack.

file, which can be implemented (at least in structure-based detection) by adding the associated objects into the PDF file.

4.3.4 Reverse Mimicry

The *Reverse Mimicry* attack assumes that an attacker has zero knowledge of the target classifier. The basic idea is to inject malicious payloads into target benign files to minimize the structural difference between the resulting examples and targets. Our Reverse Mimicry attack employs the adversarial examples provided by Maiorca et al. [26] which was shown to successfully evade PDF classifiers based on structural analysis. Specifically, we use the 500 PDF files produced by injecting a malicious JavaScript code that does not contain references to other objects to target benign PDF files. We selected the 376 files out of 500 that display malicious behaviors detected by the Cuckoo sandbox.

4.3.5 The Custom Attack

We implemented a custom attack which exploits a feature extraction vulnerability in the Mimicus implementation of PDFRate. Normally, the characters used in the Name objects of a PDF file are limited to a specific set. Since PDF specification version 1.2, a lexical convention has been added to represent a character with its hexadecimal ANSI-code, e.g., #xx. Such a modification enables us to create an arbitrary string in the form of #xx#xx#xx. In our implementation, we replaced a set of entries in the attack PDF files with their hexadecimal representations (see Table 2). These features were selected with the goal to obfuscate tags crucial to the code execution in PDF, which are frequently used for feature extraction. With this technique, the scanner would not be able to detect malicious code without dynamically reconstructing the PDF structure. While it is theoretically possible to replace all the ASCII text inside the document, we chose not to do that due to the concern on the expansion of file size.

4.4 Feature-Space Evasion Model

In typical realizable attacks, including EvadeML, a consideration is not merely to move to the benign side of the classifier decision boundary, but to appear as benign as possible. This naturally translates into the following multi-objective optimization in feature space:

$$\underset{x}{\text{minimize}} \quad Q(x) = f(x) + \lambda c(x_M, x), \tag{2}$$

where $f(x)$ is the score of a feature vector x, with the actual classifier (such as SVM) $g(x) = \text{sgn}(f(x))$, x_M the malicious seed, x an evasion instance, $c(x_M, x)$ the cost of transforming x_M into x, and λ a parameter which determines the feature transformation cost. We use l_2 norm distance between x_M and x as the cost function: $c(x_M, x) = \sum_i |x_i - x_{M,i}|^2$. Since in most of our experiments features are binary, the choice of l_2 norm (as opposed to another l_p norm) is not critical.

As the optimization problem in Equation (2) is non-convex and variables are binary in three of the four cases we consider, we use a stochastic local search method designed for combinatorial search domains, *Coordinate Greedy* (alternatively known as iterative improvement), to compute a local optimum (the binary nature of the features is why we eschew gradient-based approaches) [18, 23]. In this method, we optimize one randomly chosen coordinate of the feature vector at a time, until a local optimum is reached. To improve the quality of the resulting solution, we repeat this process from several random starting points. This approach has been shown to be extremely effective for computing evasion instances in binary domains [23].

4.5 Datasets

The dataset we use is from the *Contagio Archive*.[2] We use 5,586 malicious and 4,476 benign PDF files for training, and another 5,276 malicious and 4,459 benign files as the non-adversarial test dataset. The training and test datasets also contain 500 seeds selected by Xu et al. [44], with 400 in the training data and 100 in the test dataset. These seeds are filtered from 10,980 PDF malware samples and are suitable for evaluation since they are detected with reliable malware signatures by the Cuckoo sandbox [17]. We randomly select 40 seeds from the training data as the retraining seeds and use the 100 seeds in the test data as the test seeds.

4.6 Implementation of Iterative Adversarial Retraining

We made a small modification to the general iterative retraining approach described in Section 3 when it uses EvadeML as the realizable attack $O(h; \mathsf{D})$. Specifically, we used only 40 malicious seeds to EvadeML to generate evasions, to reduce running time and make the experiment more consistent with realistic settings where a large proportion of malicious data is not adapting to the classifier. As shown below, this set of 40 instances was sufficient to generate a model robust to evasions from held out 100 malicious seed PDFs.

[2] Available at the following URL: http://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html.

We distribute both retraining and adversarial test tasks on two servers (Intel(R) Xeon(R) CPU E5-2695 v4 @ 2.10GHz, 18 cores and 64 GB memory, running Ubuntu 16.04). For retraining using EvadeML as the attack, we assign each server 20 seeds; each seed is processed by EvadeML to produce the adversarial evasion instances. We then add the 40 examples obtained to the training data, retrain the classifier, and then split the seeds between the two servers in the next iteration. In the evaluation phase, we assign each server 50 seeds from the 100 test instances, and each seed is further used to evade the classifier by using EvadeML.

4.7 Evaluation Metrics

We evaluate performance in two ways: 1) evaluation of evasion robustness (which is central to our specific inquiry), and 2) traditional evaluation. To evaluate robustness, we compute the proportion of 100 malicious test seed PDFs for which EvadeML successfully evades the classifier; this is our metric of *evasion robustness*, evaluated with respect to EvadeML. Thus, evasion robustness of 0% means that the classifier is successfully evaded in every instance, while evasion robustness of 100% means that evasion fails every time. Our traditional evaluation metric uses test data of malicious and benign PDFs, where no evasions are attempted. On this data, we compute the ROC (receiver operating characteristic) curve and the corresponding AUC (area under the curve).

5 Efficacy of Feature-Space Attack Models

We now undertake our first task: evaluation of the effectiveness of robust ML obtained by using the abstract feature-space models of attack. We compare to a baseline classifier obtained by retraining with the most potent attack on our menu, EvadeML (which, in addition to inserting content, as done by other attacks [19, 26, 37], also allows the attacker to delete and swap PDF objects). We can think of our baseline as assuming that the defender knows that EvadeML is employed by the attacker, along with its hyperparameters. Throught this and next section, we also use EvadeML to evaluate the effectiveness of classifiers hardened using a feature-space model, in comparison with the above baseline.

5.1 Structure-Based PDF Malware Classification

Our first case study uses a state-of-the-art PDF malware classifier which engineers features based on PDF *structure*. Indeed, we evaluate two versions of this classifier: an earlier version, which we call *SL2013*, and a more recent version, which we call *Hidost*. The experiments by Xu et al. [44] demonstrate that SL2013 can be successfully evaded. Since Hidost was a recent redesign attempting in part to address its vulnerability to mimicry attacks by significantly reducing the feature space,

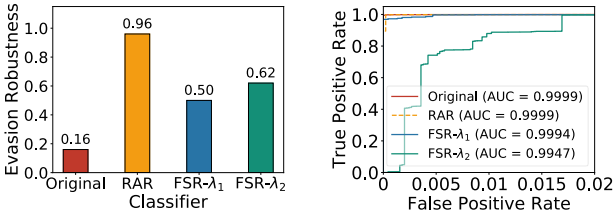

Figure 1: Evasion robustness under EvadeML test (left) and performance on non-adversarial data (right) of different classifiers for SL2013.

no data exists on its vulnerability to evasion attacks. Below we demonstrate that Hidost is also vulnerable to evasion attacks (indeed, more so than SL2013).

From the perspective of defense, we show that it is possible to harden both SL2013 and Hidost against a powerful realizable EvadeML attack by simply retraining with this attack (*RAR*, for *realizable-attack retraining*, henceforth refers to a model hardened using EvadeML). This serves as a baseline we use to evaluate the efficacy of a retraining defense with a feature-space attack model (henceforth, FSR for *feature-space retraining*). We then show that for both SL2013 and Hidost, FSR significantly underperforms RAR.

In our experiments, we empirically set the *RBF* parameters for training both SL2013 and Hidost to $C = 12$ and $\gamma = 0.0025$.

5.1.1 SL2013

Retraining with a Powerful Realizable Attack First, we replicated the EvadeML attack on the original SL2013; the classifier achieves only a 16% evasion robustness.[3] Next, to create a baseline, we conduct experiments in which EvadeML is employed to retrain SL2013. The process terminated after 10 iterations at which point no evasive variants of the 40 retraining seeds could be generated. We observe (Figure 1 (left)) that the retrained classifier (RAR) obtained by this approach achieves a 96% evasion robustness. Moreover, RAR is essentially as accurate as the baseline SL2013 on non-adversarial data (Figure 1 (right)). Thus, it is clearly possible to be highly robust to this evasion attack without significantly compromising effectiveness on data not featuring explicit evasion attacks.

Figure 2 (left) shows the gradual improvement of evasion robustness over the 10 retraining iterations. This plot demonstrates non-trivial effectiveness of EvadeML: the first few iterations are clearly insufficient, as re-running EvadeML creates many new evasions that cannot be correctly detected by

[3] This result differs from the experiments in [44] which show a 0% evasion robustness. We found a flaw in the implementation of feature extraction in EvadeML which causes evaluation to be performed using the wrong feature vectors. This bug has been fixed in the GitHub version of EvadeML.

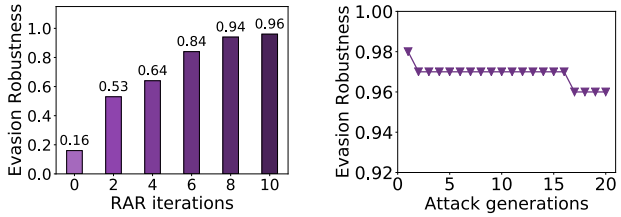

Figure 2: Evasion robustness with retraining iterations (left) and generations of the EvadeML attack test (right).

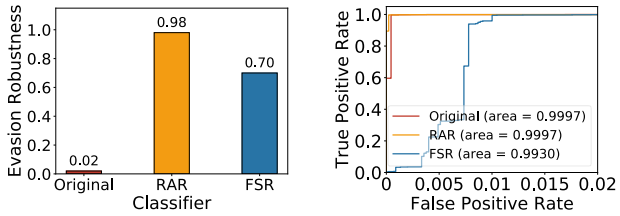

Figure 3: Evasion robustness under EvadeML test (left) and performance on non-adversarial data (right) of different classifiers for Hidost.

the classifier. Only after 6 iterations does EvadeML optimization loop begin to show significant signs of failing. Figure 2 (right) shows how increasing the number of generations in EvadeML attacks affects robustness of the RAR classifier. At this point, we can see that increasing the capability of the attack has minimal impact.

Feature-Space Retraining Next, we experimentally evaluate the effectiveness of retraining with a feature-space model of evasion attacks in obtaining robust ML in the face of the EvadeML realizable attack. We consider the setting with $\lambda = 0.05$ and $\lambda = 0.005$ in Equation 2 (henceforth, FSR-λ_1 and FSR-λ_2).

The robustness results are shown in Figure 1 (left). Compared to the SL2013 baseline, feature-space retraining (FSR) boosts evasion robustness from 16% to 62%. Crucially, *the robustness of the resulting classifier is far below the classifier achieved by RAR.* This illustrates that defense that relies on feature-space models of adversarial examples may not in fact lead to robustness when it is faced with a real attack.

We again consider performance of FSR classifier on non-adversarial test data (Figure 1 (right)). We can see that robustness boosting again does not much degrade performance, with AUC remaining above 99%. However, we do see a substantial degradation as we move from $\lambda = 0.05$ to 0.005; thus, as we increase adversarial power in the feature-space model, while we do obtain a slightly more robust model, we incur a nontrivial hit in performance on non-adversarial data.

5.1.2 Hidost

We now repeat our experiments above with another structure-based classifier, Hidost. We set the retraining parameter $\lambda = 0.005$, which appears to strike a reasonable balance between robustness and accuracy on non-adversarial data. As before, we first evaluated the robustness of the original Hidost [38] by EvadeML. The result shows a 2% robustness—remarkably, significantly worse than SL2013.

Evasion robustness of Hidost, as well as improvements achieved by RAR and FSR, are shown in Figure 3 (left), and the results are consistent with our observations for SL2013. First, by retraining with the realizable attack, evasion robustness is boosted to 98%, a rather dramatic improvement, and clear demonstration that successful defense is possible. In contrast, FSR achieves a 70% evasion robustness, a significant boost over the original Hidost, to be sure, but far below the evasion robustness of RAR.

Evaluating these classifiers on non-adversarial test data in terms of ROC curves (Figure 3 (right)), we can observe that RAR achieves comparable accuracy ($> 99.9\%$ AUC) with the original Hidost classifier on non-adversarial data, and provides even better *True Positive Rate (TPR)* when *False Positive Rate (FPR)* is close to zero. On the other hand, FSR achieves $> 99\%$ AUC, but yields a significant degradation of TPR when FPR< 0.01.

5.2 Content-Based PDF Malware Classification

Our next case study concerns another two PDF malware classifiers which use features based on PDF file content, rather than logical structure. We trained both real-valued and binarized PDFRate (henceforth, PDFRate-R and PDFRate-B) on the same dataset as SL2013 and Hidost, and achieved $> 99.9\%$ AUC for both classifiers on test data. In our experiments, we empirically set the SVM *RBF* parameters for training to $C = 10$ and $\gamma = 0.01$. In our evaluation of ML robustness, we again set the feature-space model parameter λ to be 0.005.

5.2.1 PDFRate with Real-Valued Features

We begin with the variant of PDFRate—PDFRate-R—which has been constructed in previous evaluations and shown comparable in performance to the original implementation [37]. We again begin by replicating the EvadeML evasion robustness evaluation of the baseline classifier. As expected, we find the classifier quite vulnerable, with only 2% evasion robustness.

Next, we retrain PDFRate-R with EvadeML for 10 iterations (RAR baseline), and perform feature-space retraining using the conventional feature space model above. Our results are shown in Figure 4 (left). Observe that while RAR

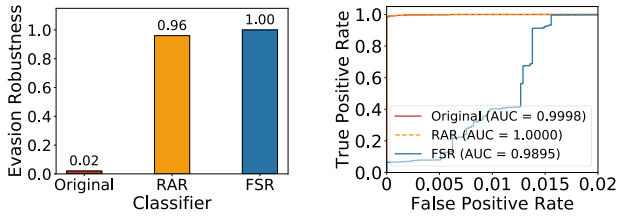

Figure 4: Evasion robustness under EvadeML test (left) and performance on non-adversarial data (right) of different classifiers for PDFRate-R.

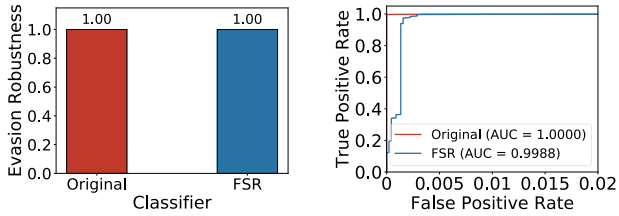

Figure 5: Evasion robustness under EvadeML test (left) and performance on non-adversarial data (right) of different classifiers for PDFRate-B.

indeed achieves a highly robust classifier (96% robustness), FSR actually performs *even better*, with 100% robustness.

Comparing RAR and FSR performance on non-adversarial data (Figure 4 (right)), we observe that the high robustness of FSR does incur a cost: while RAR remains exceptionally effective (>99.99% AUC), FSR achieves AUC slightly lower than 99%, although most significantly, the degradation is rather pronounced for low FPR regions (below 0.015).

5.2.2 PDFRate with Binarized Features

One of our great surprises is the robustness of the binarized PDFRate: despite the fact that the real-valued PDFRate is quite vulnerable, *the same classifier using binary features was 100% robust to EvadeML* (Figure 5 (left)). Consequently, this will serve as our robust baseline (equivalently, RAR would terminate with no iterations). Feature-space retrained PDFRate-B also exhibits 100% evasion robustness, although it does require a number of iterations to converge.

Considering now the performance of PDFRate-B and FSR on non-adversarial test data (Figure 5 (right)), we can make two interesting observations. First, the baseline PDFRate-B is remarkably good even on this data; in a sense, it appears to hit the sweet spot of adversarial robustness and non-adversarial performance. Second, FSR retrained classifier is competitive in terms of AUC ($\sim$ 99.9%), but is observably worse than the baseline classifier for very low false positive rates.

6 Evasion-Robust Classification with Conserved Features

Thus far, we had observed that ML hardened with the standard mathematically convenient feature-space evasion attack model may in some cases not yield satisfactory robustness against real attacks. The key issue is that feature-space models are entirely disembodied from the domain. This is crucial to enable us to have mathematical formulations of attacks, but clearly has limitations. The key question is whether we can devise a simple way of anchoring feature-space attacks in the application domain to allow us to meaningfully and minimally constrain abstract attacks to reflect some of the constraints that real attacks face. Next, we propose a refinement of the feature-space model that aims to do just that.

Specifically, we introduce the idea of *conserved features*, which we define to be *features, the unilateral modification of which compromises malicious functionality.* We develop this idea specifically for *binary features*, as this notion is particularly crisp in such a case (e.g., such features tend to correspond to the existence of particular objects in PDF).

Next, we present three major findings. First, conserved features do exist in all three of our classifiers over the binary feature space, and can be effectively identified (see our algorithm for identifying conserved features in Appendix A). Second, conserved features cannot be recovered using statistical feature reduction (in our case, sparse regularization), and feature reduction methods do not lead to robust classifiers. The reason is that conservation is connected to the relationship between features and malicious functionality, rather than statistical properties of non-evasion data; for example, features which are strongly correlated with malicious behavior are often a consequence of attacker "laziness" (such as whether a PDF file has an author), and are easy for attackers to change. Third, we demonstrate that the limitations of feature-space robust ML can be substantially alleviated by incorporating conserved features as attack invariants in the feature-space evasion model.

To develop intuition about the nature of conserved features, consider SL2013, which employs structural paths as features to discriminate between malicious and benign PDFs. On the one hand, the structural paths like `/Type` are unessential to preserve malicious behaviors, and we do not expect them to be conserved. On the other hand, as the shellcode which triggers malicious functionality is embedded in certain PDF objects, those corresponding structural paths are likely to be conserved in each variant crafted from the same malicious seed (e.g., `/OpenAction/JS`). In addition, structural paths that facilitate embedded script in PDF files also can be conserved features as removing them can break the script (e.g., `/Names` and `/Pages`). This further illustrates that conserved features are not necessarily optimal for statistically distinguishing benign and malicious instances (indeed, these may be common to both); rather, they serve to anchor the feature-

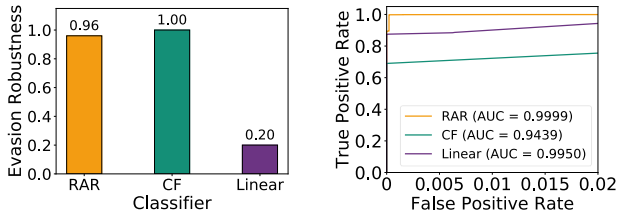

Figure 6: Classifying with conserved features: comparing evasion robustness (left) and ROC curves (right).

space attack model in the domain by connecting features to malicious functionality.

6.1 Classifying Using Only Conserved Features

We begin by exploring the effectiveness of using *only* conserved features for classification. We identified 8 conserved features for SL2013 (out of ∼6000), 7 for Hidost (out of ∼1000), and 4 for PDFRate-B (out of 135); these are detailed in Table 3 of the appendix, while our algorithm for identifying conserved features is presented in Appendix A.

We start by considering four natural questions pertaining to conserved features: 1) are they sufficient to make a classifier robust to evasions, 2) do they effectively discriminate between benign and malicious instances, 3) can they be identified using standard statistical methods (such as sparse regularization), and 4) are they just detecting the presence of JavaScript in PDF?

We explore these for SL2013. Specifically, we trained a classifier using *only* the 8 conserved features (*CF* henceforth). As we can see in Figure 6 (left), this classifier is 100% robust to EvadeML attacks, appearing to resolve the first question. However, we emphasize that conserved features alone need not capture the full spectrum of adversarial behavior and constraints. Indeed, in Section 7 we show that classifiers based solely on conserved features can also be evaded, particularly if attacks are *specifically designed to evade them*. Rather, as we show presently, they provide a *sufficient anchoring* in the problem domain for feature-space attack models to succeed.

To address question (2), consider Figure 6 (right): clearly, if we desire a low false positive rate, using only conserved features for classification yields subpar performance on non-adversarial data. To address the third question, we learn a linear SVM classifier for SL2013 with l_1 regularization (henceforth, Linear) where we empirically adjust the SVM parameter C to perform feature reduction until the number of the features is also 8; we find that only 3 of these are conserved features (see Appendix A.6 for a more detailed analysis of the relationship between statistically useful and conserved features). As we can see in Figure 6 (left), this classifier exhibits poor robustness; thus, statistical methods are insufficient to identify good conserved features.

To address the fourth question, we create a classifier using only one boolean feature which identifies the presence of JavaScript in a PDF file (henceforth, we refer to this feature as *JS*). We find that this classifier is also robust to EvadeML. On non-adversarial data, JS achieves FPR of 0.04 and FNR of 0.14 (in other words, 4% of the benign files in the non-adversarial dataset use JavaScript, while 14% of malicious instances use alternative attacks to Javascript).[4] To create an apples-to-apples comparison with the CF classifier, we empirically adjust the classification threshold of CF until we get the same FPR with JS. The resulting CF classifier exhibits FNR of 0.11, considerably better than JS. Nevertheless, it is clear that using either CF (only conserved features), or only JS, is impractical, since both FNR and FPR of these are quite high. Moreover, as we show in Section 7, classifiers based only on conserved features can be defeated by other realizable attacks. Next, we show that identification of conserved features is nevertheless crucial in creating highly effective feature-space attack models.

6.2 Feature-Space Model with Conserved Features

As discussed above, the feature-space evasion model in Equation (2) may not sufficiently boost ML robustness. Since conserved features allow us to minimally tie the abstract feature-space representation to malicious functionality, we offer a natural modification of the model in Equation (2), imposing the constraint that conserved features cannot be modified by the attacker. We formally capture this in the new optimization problem in Equation (3), where S is the set of conserved features:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & Q(x) = f(x) + \lambda c(x_M, x), \\ \text{subject to} \quad & x_i = x_{M,i},\ \forall i \in \mathsf{S}. \end{aligned} \tag{3}$$

Other than this modification, we use the same *Coordinate Greedy* algorithm with random restarts as before to compute adversarial examples. We adopt the evasion model in Equation (3) to retrain the target classifier using the retraining procedure from Section 4. We denote the classifier obtained by the retraining procedure using a feature-space model grounded by conserved features by *CFR*. We also study the effectiveness of our automated procedure for identifying conserved features as compared to using a subset that only considers Javascript features (we can think of these as expert-identified conserved features, as this is what an expert would naturally consider). To this end, we repeat the procedure above by replacing the conserved feature set S in Eq. 3 with a subset that involves Javascript. The classifier resulting from such restricted adver-

[4] We observe similar results for 5,000 benign PDFs obtained by using Google web searches [37], where 3% of benign files use Javacript.

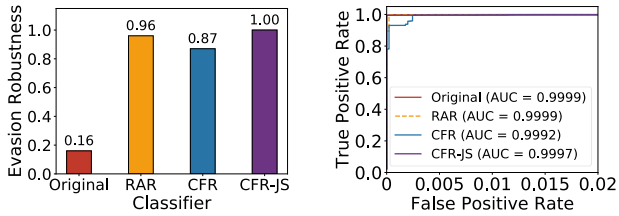

Figure 7: Evasion robustness (left) and performance on non-adversarial data (right) of different variants of SL2013.

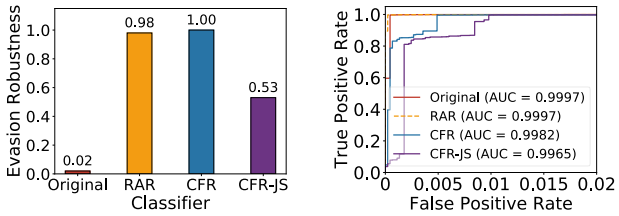

Figure 8: Evasion robustness (left) and performance on non-adversarial data (right) of different variants of Hidost.

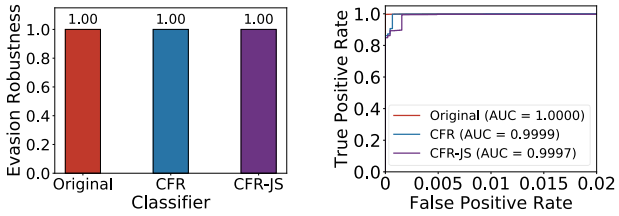

Figure 9: Evasion robustness (left) and performance on non-adversarial data (right) of different variants of PDFRate-B.

sarial retraining with "expert"-identified conserved features is termed *CFR-JS*.

6.2.1 SL2013

We now evaluate the robustness and effectiveness of the feature space retraining approach, which uses conserved features. We set the parameter $\lambda = 0.005$ as before. The robustness results are presented in Figure 7 (left). Observe that *CFR* now significantly improves robustness of the original classifier, with evasion robustness rising from 16% to 87%. Moreover, *CFR-JS* achieves a 100% evasion robustness against EvadeML. These results demonstrate that by leveraging the conserved features, the feature-space evasion models are now quite effective as a means to boost evasion robustness of SL2013.

In Figure 7 (right) we evaluate the quality of these classifiers on non-adversarial test data in terms of ROC curves. In all cases, be it original, RAR, CFR, and CFR-JS, AUC is $> 99.9\%$, although we can see a slight degradation of CFR for extremely low false positive rates compared to the others. It is noteworthy that CFR performs much better than FSR (robust ML using a standard feature-space approach, recall Figure 1 (right)).

6.2.2 Hidost

Next, we evaluate the effectiveness of CFR for Hidost. The results are shown in Figure 8 (left) and are largely consistent with SL2013. In particular, CFR boosts evasion robustness from 2% to 100% (slightly better than RAR), well above conventional FSR (recall Figure 3 (left)). In contrast, CFR-JS only boosts robustness to 53%, showing that our algorithmic approach can in some cases offer a considerable advantage to expert-chosen conserved features.

Evaluating the performance of CFR and CFR-JS on non-adversarial test data in terms of ROC curves in Figure 8 (right), we find that the CFR classifier can achieve $\sim 99.8\%$ AUC. This is somewhat worse than RAR, particularly for very low false positive rates, but better than CFR-JS—again, in this case using the full batch of conserved features exhibits a significant advantage over solely looking for Javascript.

6.2.3 Binarized PDFRate

Finally, we evaluate the effectiveness of the CFR variants of PDFRate-B. We observe that both the *CFR* and *CFR-JS* classifiers in the PDFRate-B family achieve 100% evasion robustness against EvadeML (Figure 9 (left)), just as the RAR and FSR counterparts had.

However, a close look at Figure 9 (right) demonstrates that CFR and CFR-JS achieve far better performance on non-adversarial data, with $>99.9\%$ AUC, where improvements are particularly significant for small false positive rates compared to FSR (recall Figure 5 (right)). Moreover, in this experiment, CFR achieves slightly higher TPR than CFR-JS for low FPR regions (below 0.003). The main takeaway here is that although the feature-space approach already yields high robustness in this setting, introducing conserved features significantly mitigates its degradation in performance on non-adversarial data.

7 Additional Realizable Evasion Attacks

So far we used EvadeML as the primary realizable attack in our experiments. This choice is defensible, as EvadeML explores a significantly larger attack space than many other evasion methods (e.g., Mimicry [37]), allowing deletions and swaps, in addition to insertions. Nevertheless, it is natural to wonder whether classifiers robust to EvadeML remain robust to other classes of evasion attacks. A particularly intriguing question is how the classifiers hardened against EvadeML fare in comparison with classifiers hardened against feature-space models, when faced with different realizable attacks.

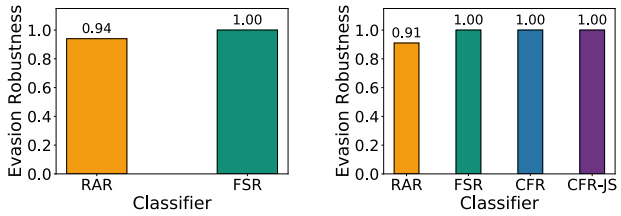

Figure 10: Robustness to Mimicry attack. Left: PDFRate-R (note that our notion of CFR is not applicable here). Right: PDFRate-B.

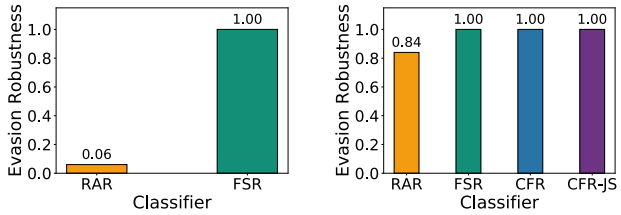

Figure 11: Robustness to Mimicry+ attack. Left: PDFRate-R (note that our notion of CFR is not applicable here). Right: PDFRate-B.

To answer these questions, we consider *five* additional realizable attacks: *Mimicry* [37], which was one of the first realizable attacks on PDF malware detectors, *Mimicry+*, an enhanced variant of Mimicry, *MalGAN* [19], which uses Generative Adversarial Networks (GANs) to create evasion attacks (but only targets binary classifiers), *Reverse Mimicry* [26], which inserts malicious payloads into target benign files, and a new custom attack aimed at defeating PDFRate-B conserved features. The Mimicry/Mimicy+ attacks are designed specifically for PDFRate, and cannot be usefully applied to SL2013 or Hidost, whereas the Reverse Mimicry attack and our custom attack require *zero knowledge* of target classifiers.

7.1 Mimicry and Mimicry+ Attacks

We start by considering the Mimicry and Mimicry+ attacks for both real-valued and binarized variants of PDFRate, with the same 100 malicious seeds employed in Section 5 and 6 as attack files.

The results are shown in Figures 10 and 11, and offer two noteworthy findings. First, as can be seen in Figure 11, RAR classifiers (hardened specifically against EvadeML, recall that the original PDFRate-B classifier is equivalent to RAR) can be quite vulnerable to the Mimicry+ attack, whereas both FSR and CFR classifiers remain robust. Second, Mimicry+ is indeed a much stronger attack than Mimicry: the original Mimicry fails to significantly degrade RAR performance, whereas Mimicry+ largely evades the RAR variant of PDFRate-R, and is somewhat more potent against PDFRate-B

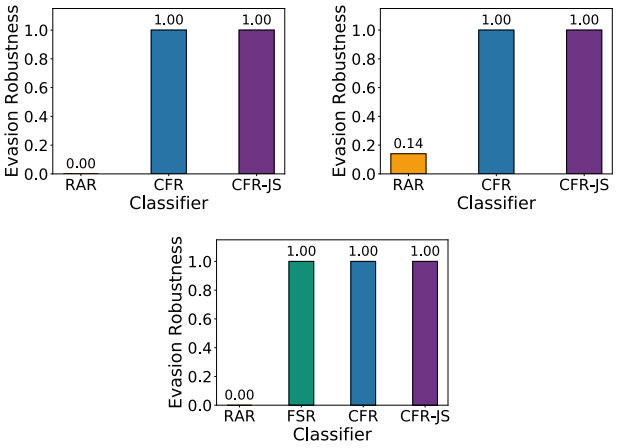

Figure 12: Robustness to MalGAN attack. SL2013 (top left), Hidost (top right), PDFRate-B (bottom).

than Mimicry. This demonstrates that besides its mathematical elegance, the abstract feature-space evasion models, once appropriately anchored to the domain, are rather generally robust to evasion attacks.

7.2 MalGAN Attack

Next, we consider the MalGAN attack on the three classifiers over binary feature space we have previously studied: SL2013, Hidost, and PDFRate-B, with RAR and FSR/CFR versions that have been shown robust to EvadeML.

The results, shown in Figure 12, demonstrate that despite EvadeML being a powerful attack, the RAR approaches which use it for hardening (with resulting classifiers no longer very vulnerable to EvadeML) are *highly* vulnerable to MalGAN, with evasion robustness of 0% in most cases. In contrast, CFR models which use conserved features remain highly robust (100% in all cases), just as we had observed earlier.

7.3 Reverse Mimicry Attack

Next, we employ the *Reverse Mimicry* attack on the EvadeML-robust variants of all the classifier types (SL2013, Hidost, PDFRate-R, and PDFRate-B).

Figure 13 presents the results, which are revealing in several ways. First, we again observe that RAR (hardened specifically against EvadeML) is roundly defeated in most cases. Second, consider the robustness results for the classifier using only the conserved features (CF), we can see that reverse mimicry succeeds in defeating conserved features for a non-trivial proportion of instances. It does so by including Javascript tags in structural paths that are not used as features by SL2013/Hidost (since these classifiers only consider commonly occurring sets of structural paths). Thus, this attack reveals an important vulnerability in the feature extraction

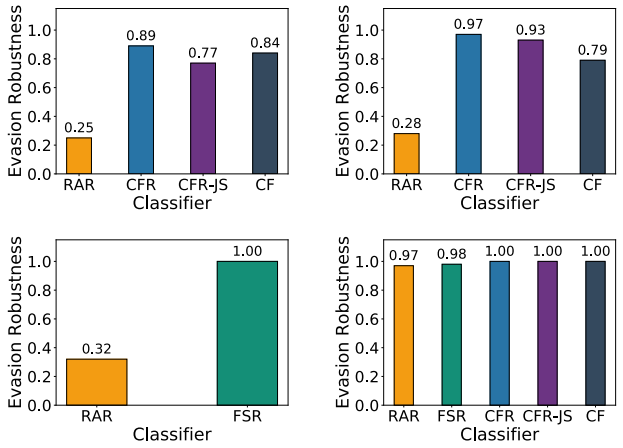

Figure 13: Robustness to Reverse Mimicry attack. SL2013 (top left), Hidost (top right), PDFRate-R (bottom left), PDFRate-B (bottom right). Note that our notions of CFR and CF for PDFRate-R is not applicable here.

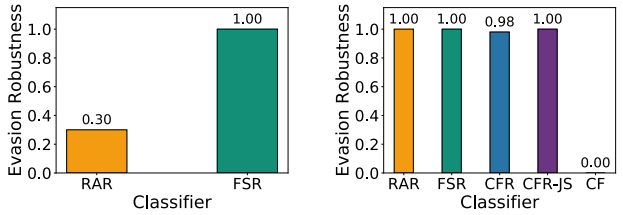

Figure 14: Robustness to the custom attack. Left: PDFRate-R (note that our notions of CFR and CF are not applicable here). Right: PDFRate-B.

approach employed by these classifiers; indeed, it suggests that structure-based classifiers may be *inherently* difficult to harden. Remarkably, CFR remains more robust than CF despite these vulnerabilities. The case of Hidost is particularly stark: CFR is nearly 20% more robust than CF!

7.4 The Custom Attack

Our final attack specifically targets a feature extraction bug in the Mimicus implementation of PDFRate in order to defeat the corresponding CF classifier.

The results are shown in Figure 14. We find that after this attack, CF robustness is 0. We also observe that the robustness of RAR classifier for PDFRate-R also drops, although to 0.3 rather than 0. Significantly, the FSR classifiers for both PDFRate-R and PDFRate-B remain 100% robust, and the CFR variant of PDFRate-B has nearly perfect robustness (0.98) against this attack. Our latter observation is particularly remarkable: although the conserved features are roundly defeated by this attack, the use of these as a part of a holistic retraining approach yields a classifier that remains robust. Thus, not only is it possible to construct a robust malware classifier without unduly relying on conserved features, but we can accomplish this through iterative retraining in feature space.

8 Related Work

Below we briefly describe some of the related literature on adversarial evasion or adversarial example attacks and defenses; we refer readers to Vorobeychik and Kantarcioglu [40] for a broader and more in-depth treatment of the subject of ML attacks and defenses.

Evasion and Adversarial Example Attacks: An early realizable evasion attack on machine learning was devised by Fogla et al. [11, 12], who developed an attack on anomaly-based intrusion detection systems. Šrndic and Laskov [37] present a case study of an evasion attack on a state-of-the-art PDF malware classifier, PDFRate. Xu et al. [44] propose EvadeML, a fully realizable attack on PDF malware classifiers which generates evasion instances by using genetic programming to modify PDF source directly, using a sandbox to ensure that malicious functionality is preserved. Grosse et al. [16] develop a method for generating evasion attacks against a deep learning-based Android malware classifier, using a gradient-based approach which is also a form of iterative improvement heuristics, but chooses the best coordinate to improve in each iteration as evaluated by the gradient, rather than random coordinate as in our case. Their approach likely requires fewer steps than coordinate greedy, but since we run coordinate greedy until convergence, this difference isn't important in our study. Moreover, we also optimize among several local optima through random restarts, which is likely to obtain better evasion solutions (Grosse et al. [16] stop as soon as an evasion is found, rather than trying to identify the most benign looking malware). This particular attack can be viewed as realizable, even though it wasn't implemented and evaluated in actual malware, since the attack space is significantly restricted to only add features that do not interfere with others already present. Similarly, MalGAN, an evasion attack based on generative adversarial networks developed by Hu and Tan [19], only adds features from benign to malicious malware, and we treat it as a realizable attack (since it's not difficult to implement).

In addition to classifier evasion methods which change the actual malicious instances (or are relatively direct to implement as such), a number of techniques have sprouted for modeling adversarial examples in feature space [1–4, 7, 9, 15, 21, 21–24, 28, 41, 45]. Moreover, a series of efforts explore evasion in the context of image classification by deep neural networks [15, 20, 31, 33], although Gilmer et al. [13] question the common threat models used in these works. Several recent approaches attempt to generate adversarial examples against computer vision systems in physical space, such as

adding stickers to a stop sign to cause misclassification, or wearing printed glass frames to fool face recognition, and are therefore somewhat analogous to our notion of realizable attacks [10, 33].

Evasion-Robust Classification: Dalvi et al. [9] presented the first approach for evasion-robust classification. A series of approaches formulate robust classification as minimizing maximum loss (i.e., following a robust optimization paradigm), where maximization is attributed to the evading attacker aiming to maximize the learner's loss through small feature-space transformations [25, 32, 39, 42, 46]. A number of alternative methods for designing classifiers consider the interaction as a non-zero-sum game [5, 6, 21–23]. Finally, a series of iterative retraining procedures have been proposed, both for general adversarial evasion [21, 23], and specifically for deep learning methods for vision [15, 20, 25] (note that Madry et al. [25] fall into both robust optimization and retraining buckets, since their approach is equivalent to retraining if stochastic gradient descent simply continues by processing adversarial examples as they are added). These diverse efforts share one common property: attack models that they leverage use feature-space manipulations, which are only a proxy for realizable attacks on ML.

9 Discussion and Conclusion

We undertook an extensive exploration of the extent to which robust ML that uses the conventional feature-space models of evasion attacks remains robust to "real" attacks that can be implemented in actual malware and preserve malicious functionality (what we call realizable attacks). Our first intriguing observation is that defense based on feature-space models can fail to achieve satisfactory robustness. This in itself raises some doubts about the nearly universal focus on such models as a means for ML defense, and suggests that practical usefulness of such approaches cannot be taken for granted. However, we also show that changing the nature of the feature space can make a difference: robust ML with feature-space models is quite robust in content-based detection (which uses content, rather than structural paths, as features). Additionally, we presented a refined version of the feature-space model that makes use of conserved features (which we can identify automatically, as shown in the Appendix), and showed that where feature-space defense previously failed, it now succeeds. Our final finding may well be the most intriguing: feature-space approaches exhibit generalized robustness, in that the resulting robust ML (after appropriate refinement using conserved features) exhibits robustness to multiple realizable attacks. This contrasts with defense that is hardened using a *specific* realizable attack—even one quite powerful on the surface (EvadeML)—which can fail dramatically when faced with a different attack. These findings demonstrate the power of effective mathematical abstractions in security.

It is natural to wonder how our approach and results are applied to other domains. In computer vision, the analog of realizable malware attacks are physical attacks, whereby the physical environment is modified, rather than the digital object, such as an image. Here, the corresponding foundational question is whether common robust ML methods based on small-l_p attacks successfully protect against physical attacks. The notion of conserved features can also be seen as more generally applicable. For example, in a bag-of-words representation for spam filtering, these could correspond to the existence of URL or file attachments, and in SQL injection attacks, these may refer to the existence of specific SQL commands, such as `Select`.

The main limitation of our study is in the specific choices we had to make to ensure that it is tractable. We chose a particular defensive paradigm—iterative retraining. As we have argued, it is the only paradigm that can fit every case that we investigate; for example, there is no other general approach for learning a robust SVM with non-linear kernels. However, it is possible that approaches based on robust optimization, if they were developed, can improve performance by taking advantage of the special structure of this problem. We implemented a particular class of feature-space attacks, using l_2 norm to measure the attacker's cost of feature manipulations, and stochastic local search to compute evasions. It is possible that better attack algorithms for generating attacks over binary domains will be developed, and, indeed, some alternatives exist. However, prior work suggests that this approach yields attacks that are close to optimal [23], with the use of random restarts playing a crucial role. Finally, our study was specific to PDF malware detection. However, our framework is quite general, and could be used in the future to consider other similar questions, such as the effectiveness of robust deep learning against physical attacks. Several additional limitations offer further opportunities for future work. One example is the fact that we only define conserved features when these are binary; it may be that finding meaningful conserved features in continuous feature spaces is inherently more difficult. Another issue is the surprising finding that sufficient anchoring of feature-space defense in the domain using conserved features allows us to achieve robustness, *even when conserved features can be circumvented*. It may be that conserved features are ultimately only a part of the solution, and only help if they adequately capture the attack surface in the abstract feature space. The extent to which small variations in the set of identified conserved features matters is also an open question: our evidence is mixed, with "expert"-defined features usually, but not always, sufficient for robustness.

Acknowledgments

This work was partially supported by the Army Research Office (W911NF1610069) and NSF CAREER award (IIS-1649972).

Appendix

A Identifying Conserved Features

We now describe a systematic automated procedure for identifying these. We first introduce how to identify conserved features of SL2013, and then describe how to generalize the approach to extract conserved features of Hidost.

The key to identifying the conserved features of a malicious PDF is to discriminate them from non-conserved ones. Since merely applying statistical approaches on training data is insufficient to discriminate between these two classes of features, as demonstrated above, we need a qualitatively different approach which relies on the nature of evasions (as implemented in EvadeML) and the sandbox (which determines whether malicious functionality is preserved) to identify features that are conserved.

We use a modified version of pdfrw [27][5] to parse the objects of PDF file and repack them to produce a new PDF file. We use Cuckoo [17] as the sandbox to evaluate malicious functionality. In the discussion below, we define x_i to be the malicious file, S_i the conserved feature set of x_i, and O_i the set of its non-conserved features. Initially, $\mathsf{S}_i = \mathsf{O}_i = \emptyset$.

At the high level, our first step is to sequentially delete each object of a malicious file and eliminate non-conserved features by evaluating the existence of a malware signature in a sandbox for each resulting PDF, which provides a preliminary set of conserved features. Then, we replace the object of each corresponding structural path in the resulting preliminary set with an external benign object and assess the corresponding functionality, which allows us to further prune non-conserved features. Next, we describe these procedures in detail.

A.1 Structural Path Deletion

In the first step, we filter out non-conserved features by deleting each object and its corresponding structural path, and then checking whether this eliminates malicious functionality (and should therefore be conserved). First, we obtain all the structural paths (objects) by parsing a PDF file. These objects are organized as a tree-topology and are sequentially deleted. Each time an object is removed, we produce a resulting PDF file by repacking the remaining objects. Then, we employ the sandbox to detect malicious functionality of the PDF after the object deletion. If any malware signature is captured, the corresponding structural path of the object is deleted as a non-conserved feature, and added to O_i. On the other hand, if no malware signature is detected, the corresponding feature is added in S_i as a *possibly* conserved feature.

One important challenge in this process is that features are not necessarily independent. Thus, in addition to identifying S_i and O_i, we explore *interdependence* between features by deleting objects. As the logic structure of a PDF file is with a tree-topology, the presence of some structural path depends on the presence of other structural paths whose object refers to the object of the prior one. We define that a structural path is a dependent of another if unilateral deleting the object associated with the latter causes a flip from 1 to 0 on the feature value of the former. For any feature j of x_i, we denote the set of features that depend on j by D_i^j. Note that for a given structural path (feature), there could be multiple corresponding PDF objects. In such a case, these objects are deleted simultaneously, so as the corresponding feature value is shifted from 1 to 0.

A.2 Structural Path Replacement

In the second step, we subtract the remaining non-conserved features in the preliminary S_i and move them to O_i. Similar to the prior step, we first obtain all the structural paths and objects of the malicious PDF file. Then for each object of the PDF that is in S_i, we replace it with an external object from a benign PDF file and produce the resulting PDF, which is further evaluated in the sandbox. If the sandbox detects any malware signature, then the corresponding structural path of the object replaced is moved from S_i to O_i. Otherwise, the structural path is a conserved feature since both deletion and replacement of the corresponding object removes the malicious functionality of the PDF file. Note that in the case of multiple corresponding and identical objects of a structural path, all of these objects are replaced simultaneously.

After structural path deletion and replacement, for each malicious PDF file x_i, we can get its conserved feature set S_i, non-conserved feature set O_i, and dependent feature set D_j for any feature $j \in \mathsf{S}_i \cup \mathsf{O}_i$, which could be further leveraged to design evasion-robust classifiers.

A.3 Obtaining a Uniform Conserved Feature Set

The systematic approach discussed above provides a conserved feature set for each malicious seed to retrain a classifier. Our goal, however, is to identify a single set of conserved features which is *independent* of the specific malicious PDF seed file. We now develop an approach for transforming a collection of S_i, O_i, and D_i^j for a set of malicious seeds i into a *uniform* set of conserved features.

Obtaining a uniform set of conserved features faces two challenges: 1) minimizing conflicts among different conserved features, as a conserved feature for one malicious instance could be a non-conserved feature for another, and 2) abiding by feature interdependence if a conserved feature should be further eliminated.

To address these challenges, we propose a *Forward Elimination* algorithm to compute the uniform conserved feature

[5] The modified version is available at https://github.com/mzweilin/pdfrw.

Algorithm 1 Forward Elimination for uniform conserved feature set.

Input:
The set of conserved features for $x_i (i \in [1,n])$, S_i;
The set of non-conserved features for $x_i (i \in [1,n])$, O_i;
The set of dependent features for $j \in \mathsf{S}_i \cup \mathsf{O}_i$, D_i^j;
Output:
The uniform conserved feature set for $\{x_1, x_2, ..., x_n\}$, S;

$\mathsf{S} \leftarrow \bigcup_{i=1}^{n} \mathsf{S}_i$;
$\mathsf{S}' \leftarrow \mathsf{S}$;
$\mathsf{Q} \leftarrow \emptyset$;
$\mathsf{D}^j = \bigcup_{i=1}^{n} \mathsf{D}_i^j$;
for each $j \in \mathsf{S}'$ **do**
  if $j \notin \mathsf{Q}$ **then**
    if $\sum_{i=1}^{n} \mathbb{1}_{j \in \mathsf{O}_i} \geq \beta \cdot \sum_{i=1}^{n} \mathbb{1}_{j \in \mathsf{S}_i}$ **then**
      $\mathsf{S} \leftarrow \mathsf{S} \setminus (\{j\} \cup \mathsf{D}^j)$;
      $\mathsf{Q} \leftarrow \mathsf{Q} \cup (\{j\} \cup \mathsf{D}^j)$;
    end if
  end if
end for
return S;

set for a set of malicious seeds $\{x_1, x_2, ..., x_n\}$, given the conserved feature sets, non-conserved feature sets and dependent sets for each seed. As Algorithm 1 shows, we first obtain a union of the conserved feature sets. Then, we explore the contradiction of each feature in the union with the others, by comparing the total number of the feature being selected as a non-conserved feature and conserved feature. If the former one is greater than β times the latter one, then this feature, together with its dependents, are eliminated from the union. Otherwise, the feature is added to the uniform feature set. We use β as a parameter to adjust the balance between conserved and non-conserved features. Typically, $\beta > 1$ as we are inclined to preserve malicious functionality associated with a conserved feature, even it could be a non-conserved feature of another PDF file. We set $\beta = 3$ in our experiments.

A.4 Identifying Conserved Features for Other Classifiers

Once we obtain conserved features of SL2013 for each malicious seeds, we can employ these features to identify conserved features for other classifiers using binary features. As our approach relies on the existence of malicious functionality and corresponding features, such a relation is not obvious for real-valued features; we therefore leave the question of how to define and identify conserved features in real space for future work.

Hidost Hidost and SL2013 are similar in nature in such a way that they employ structural paths as features. The only difference is that Hidost consolidates features of SL2013 as described in Section 4. Therefore, once the conserved features of SL2013 are identified, we can simply apply the *PDF structural path consolidation rules* described in Srndic and Laskov [38] to transform these features to the corresponding conserved features for Hidost.

Binarized PDFRate We identify the conserved features for PDFRate-B by using the conserved feature set S_i of each seed x_i. For each x_i, we generate $|\mathsf{S}_i|$ PDF files, each of which corresponds to the PDF file when an element (structural path) in S_i is deleted. We then compare PDFRate-B features of these PDFs to the original x_i. If any feature value of x_i is flipped from 1 to 0, then this feature will be added in the conserved feature set of x_i for PDFRate-B. Afterward, we use Algorithm 1 to obtain the uniform conserved feature set. This approach can in fact be used for arbitrary PDF malware detectors over binary features (leveraging conserved structural paths identified using SL2013).

Classifier	Conserved features	Involve JS?
SL2013	/Names	No
	/Names/JavaScript	Yes
	/Names/JavaScript/Names	Yes
	/Names/JavaScript/Names/JS	Yes
	/OpenAction	No
	/OpenAction/JS	Yes
	/OpenAction/S	No
	/Pages	No
Hidost	/Names	No
	/Names/JavaScript	Yes
	/Names/JavaScript/Names	Yes
	/Names/JavaScript/Names/JS	Yes
	/OpenAction	No
	/OpenAction/JS	Yes
	/Pages	No
PDFRate-B	count_box_other	No
	count_javascript	Yes
	count_js	Yes
	count_page	No

Table 3: Conserved features and their relevance to JavaScript.

A.5 Conserved Features

Table 3 presents the full list of conserved features we identified for each classifier.

A.6 Conserved vs. Regularized Features

In our experiments, we empirically adjust the SVM parameter C to study the overlap between *conserved features* and those selected by l_1 regularization. We first adjust C to perform feature reduction until the number of features is identical to the number of conserved features. In this case, sparse versions of both SL2013 and Hidost include only 3 of the conserved features, while sparse PDFRate-B includes only 1. In another experiment, we adjusted C until all conserved features were selected. In this case, SL2013 requires 510 features, Hidost needs 154, and PDFRate-B needs 83.

References

[1] ATHALYE, A., CARLINI, N., AND WAGNER, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning* (2018), pp. 274–283.

[2] BARRENO, M., NELSON, B., SEARS, R., JOSEPH, A. D., AND TYGAR, J. D. Can machine learning be secure? In *ACM Asia Conference Computer and Communications Security* (2006), pp. 16–25.

[3] BIGGIO, B., CORONA, I., MAIORCA, D., NELSON, B., SRNDIC, N., LASKOV, P., GIACINTO, G., AND ROLI, F. Evasion attacks against machine learning at test time. In *European Conference on Machine Learning and Knowledge Discovery in Databases* (2013), pp. 387–402.

[4] BIGGIO, B., FUMERA, G., AND ROLI, F. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering 26*, 4 (2014), 984–996.

[5] BRÜCKNER, M., AND SCHEFFER, T. Stackelberg games for adversarial prediction problems. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2011), pp. 547–555.

[6] BRÜCKNER, M., AND SCHEFFER, T. Static prediction games for adversarial learning problems. *Journal of Machine Learning Research*, 13 (2012), 2617–2654.

[7] CARLINI, N., AND WAGNER, D. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy* (2017), pp. 39–57.

[8] COVA, M., KRUEGEL, C., AND VIGNA, G. Detection and analysis of drive- by-download attacks and malicious javascript code. In *International Conference on World Wide Web* (2010), pp. 281–290.

[9] DALVI, N., DOMINGOS, P., MAUSAM, SANGHAI, S., AND VERMA, D. Adversarial classification. In *SIGKDD International Conference on Knowledge Discovery and Data Mining* (2004), pp. 99–108.

[10] EYKHOLT, K., EVTIMOV, I., FERNANDES, E., LI, B., RAHMATI, A., XIAO, C., PRAKASH, A., KOHNO, T., AND SONG, D. Robust physical-world attacks on deep learning visual classification. In *Computer Vision and Pattern Recognition* (2018).

[11] FOGLA, P., AND LEE, W. Evading network anomaly detection systems: Formal reasoning and practical techniques. In *ACM Conference on Computer and Communications Security* (2006), pp. 59–68.

[12] FOGLA, P., SHARIF, M., PERDISCI, R., KOLESNIKOV, O., AND LEE, W. Polymorphic blending attacks. In *USENIX Security Symposium* (2006).

[13] GILMER, J., ADAMS, R. P., GOODFELLOW, I. J., ANDERSEN, D., AND DAHL, G. E. Motivating the rules of the game for adversarial example research. arXiv preprint.

[14] GOODFELLOW, I., POUGET, J., MIRZA, M., XU, B., WARDE, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Neural Information Processing Systems* (2014), pp. 2672–2680.

[15] GOODFELLOW, I. J., SHLENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations* (2015).

[16] GROSSE, K., PAPERNOT, N., MANOHARAN, P., BACKES, M., AND MCDANIEL, P. Adversarial perturbations against deep neural networks for malware classification. In *European Symposium on Research in Computer Security* (2017).

[17] GUARNIERI, C., TANASI, A., BREMER, J., AND SCHLOESSER, M. Cuckoo sandbox: A malware analysis system, 2012. http://www.cuckoosandbox.org/.

[18] HOOS, H. H., AND STUTZLE, T. *Stochastic Local Search : Foundations & Applications*. Morgan Kaufmann, 2004.

[19] HU, W., AND TAN, Y. Generating adversarial malware examples for black-box attacks based on GAN. arXiv preprint.

[20] HUANG, R., XU, B., SCHUURMANS, D., AND SZEPESVÁRI, C. Learning with a strong adversary. In *International Conference on Learning Representations* (2016).

[21] KANTCHELIAN, A., TYGAR, J. D., AND JOSEPH, A. D. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning* (2016), pp. 2387–2396.

[22] LI, B., AND VOROBEYCHIK, Y. Feature cross-substitution in adversarial classification. In *Neural Information Processing Systems* (2014), pp. 2087–2095.

[23] LI, B., AND VOROBEYCHIK, Y. Evasion-robust classification on binary domains. *ACM Transactions on Knowledge Discovery from Data* (2018).

[24] LOWD, D., AND MEEK, C. Adversarial learning. In *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining* (2005), pp. 641–647.

[25] MADRY, A., MAKELOV, A., SCHMIDT, L., TSIPRAS, D., AND VLADU, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations* (2018).

[26] MAIORCA, D., CORONA, I., AND GIACINTO, G. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious PDF files detection. In *ACM Asia Conference on Computer and Communications Security* (2013), pp. 119–130.

[27] MAUPIN, P. Pdfrw: A pure python library that reads and writes pdfs. https://github.com/pmaupin/pdfrw, 2017. Accessed: 2017-05-18.

[28] NELSON, B., RUBINSTEIN, B. I., HUANG, L., JOSEPH, A. D., LEE, S. J., RAO, S., AND TYGAR, J. Query strategies for evading convex-inducing classifiers. *Journal of Machine Learning Research* (2012), 1293–1332.

[29] PAPERNOT, N., MCDANIEL, P., SINHA, A., AND WELLMAN, M. Towards the science of security and privacy in machine learning. In *IEEE European Symposium on Security and Privacy* (2018).

[30] PAPERNOT, N., MCDANIEL, P., WU, X., AND JHA, S. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy,* (2016).

[31] PAPERNOT, N., MCDANIEL, P. D., AND GOODFELLOW, I. J. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, 2016. arxiv preprint.

[32] RAGHUNATHAN, A., STEINHARDT, J., AND LIANG, P. Certified defenses against adversarial examples. In *International Conference on Learning Representations* (2018).

[33] SHARIF, M., BHAGAVATULA, S., BAUER, L., AND REITER, M. K. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 1528–1540.

[34] SMUTZ, C., AND STAVROU, A. Malicious pdf detection using matadata and structural features. Tech. rep., 2012.

[35] SMUTZ, C., AND STAVROU, A. Malicious pdf detection using matadata structural features. In *Annual Computer Security Applications Conference* (2012), pp. 239–248.

[36] ŠRNDIC, N., AND LASKOV, P. Detection of malicious PDF files based on hierarchical document structure. In *Network and Distributed System Security Symposium* (2013).

[37] ŠRNDIC, N., AND LASKOV, P. Practical evasion of a learning-based classifier: A case study. In *IEEE Symposium on Security and Privacy* (2014), pp. 197–211.

[38] ŠRNDIĆ, N., AND LASKOV, P. Hidost: a static machine-learning-based detector of malicious files. *EURASIP Journal on Information Security 2016*, 1 (2016), 22.

[39] TEO, C. H., GLOBERSON, A., ROWEIS, S., AND SMOLA, A. J. Convex learning with invariances. In *Neural Information Processing Systems* (2007).

[40] VOROBEYCHIK, Y., AND KANTARCIOGLU, M. *Adversarial Machine Learning*. Morgan and Claypool, 2018.

[41] VOROBEYCHIK, Y., AND LI, B. Optimal randomized classification in adversarial settings. In *International Conference on Autonomous Agents and Multiagent Systems* (2014), pp. 485–492.

[42] WONG, E., AND KOLTER, J. Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning* (2018).

[43] XU, H., CARAMANIS, C., AND MANNOR, S. Robustness and regularization of support vector machines. *Journal of Machine Learning Research 10* (2009), 1485–1510.

[44] XU, W., QI, Y., AND EVANS, D. Automatically evading classifiers: A case study on PDF malware classifiers. In *Network and Distributed System Security Symposium* (2016).

[45] ZHANG, F., CHAN, P., BIGGIO, B., YEUNG, D., AND ROLI, F. Adversarial feature selection against evasion attacks. *IEEE Transactions on Cybernetics* (2015).

[46] ZHOU, Y., KANTARCIOGLU, M., THURAISINGHAM, B. M., AND XI, B. Adversarial support vector machine learning. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2012), pp. 1059–1067.