Adversarially Robust Malware Detection Using Monotonic Classification

Inigo Incer*
UC Berkeley
inigo@eecs.berkeley.edu

Michael Theodorides*
UC Berkeley
theodorides@cs.berkeley.edu

Sadia Afroz
UC Berkeley
International Computer Science Institute
sadia@icsi.berkeley.edu

David Wagner
UC Berkeley
daw@cs.berkeley.edu

ABSTRACT

We propose monotonic classification with selection of monotonic features as a defense against evasion attacks on classifiers for malware detection. The monotonicity property of our classifier ensures that an adversary will not be able to evade the classifier by adding more features. We train and test our classifier on over one million executables collected from VirusTotal. Our secure classifier has 62% temporal detection rate at a 1% false positive rate. In comparison with a regular classifier with unrestricted features, the secure malware classifier results in a drop of approximately 13% in detection rate. Since this degradation in performance is a result of using a classifier that cannot be evaded, we interpret this performance hit as the cost of security in classifying malware.

ACM Reference Format:
Inigo Incer, Michael Theodorides, Sadia Afroz, and David Wagner. 2018. Adversarially Robust Malware Detection Using Monotonic Classification. In *IWSPA'18: 4th ACM International Workshop on Security And Privacy Analytics, March 19–21, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3180445.3180449

1 INTRODUCTION

Malware has been a long-running problem in computer security, and improvements in malware detection can benefit the field and society at large. Malware detection has advanced significantly over the last decade, yet deployed systems often rely heavily on blacklisting known-bad malware and struggle to detect new malware that has not been previously detected [20].

Recently, researchers have shown that machine learning can be used to improve detection of malware; for instance, Miller et al. show that machine learning can achieve better results than commercial antivirus software on their dataset [26]. However, existing machine learning methods are fragile: it is easy for attackers to make small changes to their malware to fool the classifier and evade detection [34]. In general, many machine learning methods are known to be susceptible to evasion attacks, and we lack effective defenses.

In this paper we introduce a new method for hardening malware classifiers against these attacks, and we show that our method makes it harder for an attacker to evade detection. Our approach is based on an understanding of the attacker threat model in this domain, including what kinds of changes attackers can easily and economically make to their malware, and the attackers' goals.

Our goal is to increase the cost to the attacker of fooling the classifier. To support this, we analyzed the features used for malware detection, to see which ones can be trivially or cheaply manipulated by an attacker. We discovered that some features are easy for an attacker to modify, without disrupting the malicious functionality of the malware, while others may be more expensive to change. During this analysis, we found many features that are easy to change in one direction, but hard to change in the reverse direction. For instance, it is easy to strip the signature from an application signed by Microsoft, but difficult for a malware author to add such a signature to his malware. Attacker goals also exhibit asymmetry. We expect that attackers will seek to have their malware be misclassified as benign, as there is less incentive to make benign applications be misclassified as malware.

Our approach takes advantage of this fundamental asymmetry in attacker goals and in features, by constructing a monotonic classifier. A boolean function $f : \{0,1\}^d \to \{0,1\}$ is *monotonically increasing* if $x \leq x'$ (i.e. $x_i \leq x'_i$ for each i) implies $f(x) \leq f(x')$. If f represents a malware classifier operating on d features, where $f(x) = 1$ indicates a prediction that the sample is malware, then monotonicity is a powerful security property: it says that an attacker who can only increase the feature values (but not decrease them) cannot make modifications that fool the classifier into misclassifying malware as benign. With this in mind, our approach is simple: we identify features where this holds, and train a monotonic classifier. This lets us argue that the classifier is robust against all of the modifications that our analysis found to be easy for an attacker to make; to fool the classifier, the attacker must make some other kind of change to her malware, which will presumably be more expensive for malware authors. Thus, while not a perfect defense against evasion, we hope that our approach raises the bar.

The remainder of this paper is structured as follows: in section 2 we give a brief overview of malware detection; we discuss previous machine learning applications to malware detection in section 3;

*Equal contribution.

IWSPA'18, March 19–21, 2018, Tempe, AZ, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5634-3/18/03…$15.00
https://doi.org/10.1145/3180445.3180449

in section 4 we define classification with monotonicity constraints and explain its usefulness for security; in section 5 we define and justify our threat model; we present the results of our experiments in section 6 and conclude the paper in section 7.

2 OVERVIEW OF MALWARE DETECTION

Malware, or malicious software, has been a persistent problem in computer security during the last few decades [32]. Traditional malware detection involves maintaining a blacklist of malware and whitelist of valid software that is used for detection [20]. In particular, traditional schemes identify malware by computing signatures on software and its behavior and checking whether the signatures exist in a database of known malicious signatures. This approach to malware detection has been effective at detecting known malware; however, signature-based malware detection is unable to detect new, unknown malware.

Advances in machine learning techniques, as well as improvements in computing power, have sparked the question of whether machine learning can be effective at classifying malware. Miller et al. demonstrated that it can: by applying logistic regression to features extracted from input executables [25, 26], they are able to detect 72% of malware at a false positive rate of 0.5%. They also show that this can be improved to 89% detection, if the machine learning model can suggest a small number of samples to be evaluated by a human analyst. This achieved a better detection rate than existing antivirus software, on the dataset they used.

While Miller et al. show that machine learning can be effective at detecting malware, their scheme makes no attempt to build a robust strategy that will continue to detect malware even if attackers modify their malware with this defense in mind. Zhong et al. showed that an attacker with knowledge of the model could modify her malware to evade detection by changing (on average) just 5 features [34]. In this paper, we build on their work and develop strategies for hardening the machine learning against attack, so that attackers cannot easily modify malware samples to prevent them from being detected by the classifier. We use the same datasets and features from that work to evaluate our approach.

3 RELATED WORK

A number of prior works have demonstrated adversarial attacks and defenses on various machine learning models. Adversarial robustness can be improved by reducing overfitting [13], using an ensemble of classifiers for prediction instead of a fixed one [1], using robust features that are costly for an adversary to change [33], adding noise to the training data [21], retraining with adversarial examples [22], discarding adversarial examples after detecting them [12, 14, 24], and reducing dimensionality [5, 18, 30]. Unfortunately, all of these papers evaluate against weak adversarial examples and fail to protect against strong adversarial attacks [6, 7]. Moreover, the effectiveness of these defenses are only evaluated for the image recognition task using MNIST or CIFAR dataset.

A few works have examined adversarial attacks on machine learning in the malware domain. Srndic et al. [29] examined PDFrate, a deployed machine-learning-based PDF malware detector. They demonstrated that an attacker can modify PDF malware to evade detection by adding dummy features to the header of PDF malware. Their attack can reduce the detection rate of the modified malware from 100% to 33%. Similarly, Xu et al. [31] showed that genetic algorithms can be used to automatically create evading PDF malware. They successfully defeat two malware detectors, PDFrate and Hidost. Grosse et al. [15] examined Android malware and showed that a highly accurate deep learning model can be evaded by adding less than 20 features to the app manifest file. Hu et al. [19] used a generative adversarial network (GAN) to construct adversarial malware examples that can evade several ML models including random forest, logistic regression, SVM, and a voting ensemble of these classifiers.

Our approach is fundamentally different from all the prior approaches. Instead of fixing an existing weak classifier, we build a classifier that is robust against certain kinds of adversarial attacks by construction. The monotonicity property of our classifier guarantees protection from evasion attacks. To achieve this security guarantee, we focus on features that are hard to change and that are monotonic. Although some prior work suggested identifying adversarially robust features [33], our paper is the first to focus on the monotonicity property of the features.

Monotonic classification has been used to learn ordinal classes and to improve the interpretability of an ML model [16]. We use monotonic classification to improve the robustness of a classifier. Monotonicity can be achieved in classes, features, and training data [4, 16]. We apply monotonic classification considering only monotonic features, while our training data is non-monotonic. Monotonicity can be achieved in many different ways [16]: by using monotonic constrain linear and polynomial functions [2, 9, 23], by reducing monotonic violation after training [27], by penalizing monotonicity violations during training [3] and by relabeling samples to be monotonic before training [10, 11]. We use the monotonicity functionality implemented in XGBoost. XGBoost uses a greedy algorithm to achieve monotonicity.

Ben-David et al. [4] showed that monotonicity reduces the accuracy of a classifier, a behavior which we also noticed. We are the first to use monotonicity in feature selection and as a constraint to the classifier in order to obtain a security property.

4 SECURE MALWARE CLASSIFICATION WITH MONOTONIC CONSTRAINTS

Consider a classifier f which operates on vectors $x \in \mathbb{R}^d$ of d features $x_1, x_2, \cdots, x_d$. We write $x \leq x'$ if $x_i \leq x'_i$ for each i. A monotone increasing classifier is one where $x \leq x'$ implies $f(x) \leq f(x')$. Equivalently, if $x_i \leq x'_i$, we demand

$$f(x_1, x_2, \cdots, x_i, x_{i+1}, \cdots, x_d) \leq f(x_1, x_2, \cdots, x'_i, x_{i+1}, \cdots, x_d),$$

i.e. increasing the value of an argument should only increase the value of the output, regardless of the value of the other variables. Figure 1 shows an example of a classification with a single feature in which a gradient boosting classifier is used to predict the function's value. When monotonicity constraints are imposed on the classifier, the classifier's output is an increasing function of the feature value.

Monotonicity is desirable in a classifier used for malware detection because attackers can readily change some features used in malware detection in one direction (e.g. from false to true), but changing them in other direction (e.g. from true to false) can be much more difficult. Consider, for instance, the case of software

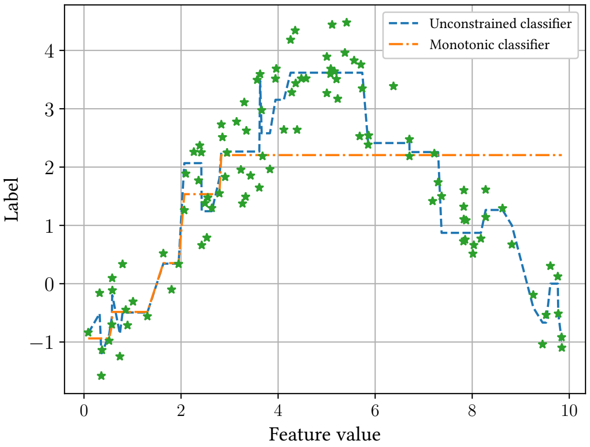


Figure 1: Example of a classifier with monotonicity constraints.

signing: removing a signature is easy, but an attacker would have a hard time getting his malware signed by a trusted entity. More generally, monotonicity in a classifier prevents evasion attacks in applications in which we can state that *the greater the amount of suspicious activity someone generates, the more likely he should be to get caught*. The malware detection application with monotonic features is one such application.

Our purpose in this work is to research the relative effectiveness of monotone classification on a malware dataset, compared to traditional classification without monotonic constraints. To study this question, we adopted an off-the-shell classifier for our tasks. XGBoost, a tree boosting classifier described in Chen et al. [8], suited our needs due its support for sparse feature vectors, its speed on large datasets, and its implementation of monotonicity constraints.

We provide an overview of tree boosting following the description in Hastie et al. [17]. Suppose we have N samples $(x_i)_{i=1}^N$ $(x_i \in \mathbb{R}^d)$ and labels $y \in \{-1, 1\}^N$. We wish to compute an M-additive function $f : \mathbb{R}^d \to \mathbb{R}$ of the form

$$\hat{y}_i = f(x_i) = \sum_{m=1}^{M} \beta_m \, b(x_i; \gamma_m),$$

where b is a basis function with parameters γ_m and weights β_m. The classifier is found by minimizing

$$\min_{\{\beta_m, \gamma_m\}^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m \, b(x_i; \gamma_m)\right), \qquad (1)$$

where L is a loss function. The solution to (1) can be computationally taxing, and a technique called Forward Stagewise Additive Modeling is used to assuage the computational burden. In this technique, instead of computing f directly, $M + 1$ functions f_m are computed iteratively as follows:

$$(\beta_m, \gamma_m) = \arg\min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta \, b(x_i; \gamma)) \qquad (2)$$

$$f_m(x) = f_{m-1}(x) + \beta_m \, b(x; \gamma_m).$$

In tree boosting, the function $b(x; \gamma)$ is a classification tree. Finally, to solve (2) in the Forward Stagewise Additive Modeling iteration, XGBoost implements a numerical optimization technique called gradient boosting.

The enforcement of monotonicity constraints in XGBoost occurs through a greedy algorithm. This algorithm places an upper and a lower bound on the predicted values for each tree, and propagates these bounds to each subtree when a split is made; this is done to ensure that the branch taken at a node when the value of a feature is negative is never larger than the value before the split occurs.

5 THREAT MODEL

We think of the problem as a two-player game between the attacker and defender. The attacker has a malicious executable that would be detected by the defender's malware classifier, and seeks to modify it so it is no longer detected. If there is an easy modification the attacker can make to render it undetected, she will do so. However, the attacker's resources are not unlimited. In particular, our goal is not to provide a perfect defense, but rather to increase the attacker's costs, hopefully sufficiently to make attacks unprofitable.

More formally, we envision that the attacker has a sample x that is detected as malicious by the classifier, i.e., $f(x) = 1$. The attacker seeks to somehow modify x into a variant x' that won't be detected (i.e., $f(x') = 0$), yet without disrupting the malicious functionality embedded in x. If there is an easy, inexpensive modification that achieves this goal, the attacker wins and we consider the classifier insecure; otherwise, we consider the classifier (adequately) secure.

With this in mind, in this section, we systematically analyze the set of modifications a resource-bounded attacker might be able to make. For each modification, we determine whether it would be easy and inexpensive to apply, and whether it risks disrupting the malicious functionality of the malware. We consider a modification admissible if it is easy to make and doesn't risk breaking the malicious payload of the malware sample. Our defense is aimed at adversaries that only make admissible modifications.

We structure our analysis of which modifications are admissible by analyzing each feature from Miller et al. [26], which are derived from information provided by VirusTotal about the binary. We identify which features can be controlled by an easy change to the malicious executable. Since all the features are boolean, there are four cases: (1) the feature cannot be easily changed; (2) the feature can be easily changed from false to true, but not vice versa; (3) the feature can be easily changed from true to false, but not vice versa; (4) the feature can be easily changed to any desired value. Features of type (1), (2), or (3) can be safely used with our monotonic classifier. Features of type (4) are not safe to use, and we drop them. Of the 13 feature categories used by Miller et al. [26], we were able to keep 8 categories for our monotoinc classifier. Table 1 lists categories of features that we kept and omitted from our classifier. The rest of this section describes our analysis in detail.

We assume that the attacker has full knowledge of the classifier and its parameters. We also allow the attacker to make any number of admissible modifications. The security property guaranteed by our scheme is that, if the original sample x is detected by our classifier as malicious, there is no set of admissible modifications that will prevent it from being detected. Therefore, to avoid detection,

Feature Category	Kept/Omitted	Reasoning
Binary Metadata	Removed	Easy to change metadata in an executable's header section
Digital Signing	Kept	Difficult to get malware signed by trusted entity
Heuristic Tools	Partially Kept	Some heuristic tools can fit our monotonic model
Packer Detection	Kept	Attacker using a packer has a reason to use a packer.
PE Properties	Omitted	Easy to change entropies, section hashes, and resources lists
Static Imports	Kept	Not trivial to remote imported function
Dynamic Imports	Kept	Not trivial to remote imported function
File Operations	Kept	Not trivial to eliminate a file access.
Mutex Operations	Omitted	Mutex names are easy for an attacker to change or remove
Network Operations	Omitted	IP Addresses can be easily changed
Processes	Kept	Removing an existing operation may disrupt functionality of malware
Registry Operations	Kept	Difficult to remove existing set/delete operation
Windows API Calls	Omitted	Could not fit to monotonic model

Table 1: Summary of feature categories kept from Miller et al. [26]

the attacker must either risk breaking the malicious payload or incur the cost of making a more substantial modification.

Training set pollution attacks are beyond the scope of this paper: we assume that the instances in the training set have not been influenced by the attacker.

5.1 Unsafe Features

In our analysis of possible attacker modifications, we encountered a few feature categories used in previous classifiers that are easy for an attacker to modify. These includes features about network traffic, mutexes, metadata in the executable, and Windows API calls. The network features captured the IP addresses that are contacted when the executable runs; attackers can presumably easily change these IP addresses by using proxies or simply hosting their C&C infrastructure on a different host. Some malware binaries use mutexes to avoid re-infecting the same machine by always grabbing a specific mutex on first infection. Miller et al. [26] used mutex names as a feature. However, since mutex names are easy for an attacker to change or remove completely in most cases, we consider them not safe to use. Similarly, in most cases it would be easy to change metadata in the executable's header section. Finally, Miller et al. [26] used n-grams from the sequence of Windows APIs called by the executable; we were unable to fit these within our monotonic model, and we suspect it would be easy for an attacker to disrupt these features by inserting extra Windows API calls that have no effect but that change the n-grams. We did not use any of these unsafe features in our classifier.

5.2 Imports

VirusTotal lists the library functions statically imported in the data segment of the binary. Miller et al. [26] create a boolean feature for each library function encountered in the dataset and set the value to 0 for a program that does not import the function and to 1 for a program that does import the library function. VirusTotal also runs each binary in a sandbox and provides a list of all library functions that were dynamically loaded at run-time through Windows API calls or at load-time. Miller et al. [26] create a boolean feature for each dynamically linked library.

Imports are monotonic features because it is in general not trivial to remove an imported library. Typically a library is imported because it is used by the malware, so removing this feature would require the malware author to find another way to access the same functionality, which is likely possible but might not be trivial. In contrast, attackers can trivially import additional functions in their malicious programs without using them or changing the malicious actions of their programs, i.e., the binary can be modified to include extra library functions that do not fulfill any purpose besides attempting to evade malware detection. Consequently, we assume that attackers can add extra unnecessary static or dynamic imports to their malware, but not remove existing imports.

Moreover, we observe that it is easy for an attacker to swap a static import for the identical dynamic import; as a result, we merged the static and dynamic import features used by Miller et al. into a single import feature for each imported library.

5.3 File Operations

VirusTotal reports all of the filesystem operations performed by the program when run in the sandbox, including attempts to open, read, copy, delete, move, and replace files. Miller et al. [26] create a boolean feature for each pair of file operation and file path encountered in the training set. Its value is set to 1 if the particular file operation is attempted on that file path. For example the FileReadFeature:c:\windows\Dcache.bin feature would be 1 if the program attempts to read the Dcache.bin file; otherwise, it will be set to 0. We use features for each of the following file operations: copy, delete, move (rename), open, read, and write. For copy and move, we have a separate feature for the source and the target.

These features are monotonic because it is not trivial to eliminate a file access. If a file is accessed, typically that will be because it is part of the functionality of the malware. Removing this file access might disrupt the behavior of the malware: for instance, ransomware fundamentally requires reading and writing most files on the disk, so there is no easy way for an attacker to avoid such operations. It may be possible to find an alternative way to achieve the attacker's ends, but we anticipate that it will not be trivial. On the other hand, it would be easy to add extra file accesses (e.g., to read a file and then ignore what was read), so we assume that attackers can add extra file operations but not remove existing ones.

Miller et al. [26] also use features that count the number of file operations of each type that a program includes, including a count of the number of files replaced, written, copied, opened, moved, downloaded, and deleted. We transform these into monotonic, boolean features. We create a feature for each power of two (say, 2^k) and each operation; the feature is set to 1 if the number of such operations is at least 2^k, or 0 if not. For example the FileOpenedCountFeature_32 equals 1 if the program opened at least 32 files. These features are monotonic as well, since attackers can add extra file operations (increasing the count) but not remove existing operations (they cannot decrease the count).

5.4 Packer Detection

Packers are tools used in the creation of malicious programs that hide the contents of executables. VirusTotal includes reports from three heuristic packer detection tools that list which packers (if any) were used to create the binary. Miller et al. [26] introduce a separate boolean feature for each packer, but we believe that this is not monotonic: it would be easy for attackers to change which packer they use to pack their binary, which changes one packer feature from 1 to 0 and another from 0 to 1. Instead, to model packer detection as a monotonic feature, we aggregate the results of all packer reports for a program and include a single feature that is 0 if no packers are detected for a program and 1 if any packer was detected.

We treat this feature as monotonic, because an attacker that is not using a packer tool can easily begin using a packer if it will help avoid detection. However, an attacker that is already using a packer presumably has a reason to use a packer (e.g., to avoid being detected by antivirus software); switching to an unpacked binary would come at a significant cost (e.g., his malware might be easier for antivirus software to detect).

5.5 Digital Signatures

VirusTotal lists all signers who have digitally signed each binary. We believe that signatures from trusted signers are valuable for determining whether a program is malware. For instance, a binary that is signed by Microsoft or Dell is probably unlikely to be malware. Therefore, we include a feature for each trusted signer we encounter in our dataset. The feature is set to 0 for programs that are signed by that signer and 1 for programs not signed by that particular signer. It is trivial for attackers to remove an existing signature from a binary, but presumably infeasible for them to gain a signature from a trusted signer on their malware. Therefore, we model these features as monotonic.

To ensure monotonicity, we deviate from Miller et al. [26]'s approach: they include a boolean feature for every signer seen in the training set, whereas we limit it to signers who appear to be trustworthy. It would be easy for attackers to create a new public key and sign their binary with it, but hard for them to get it signed by some existing trusted signer, so monotonicity requires that we limit to trusted signers. We obtain our list of trusted signers by identifying the signers in our training set who have never signed a malicious program and have signed at least 100 benign programs. We then manually inspect each signer in the list of verify that the signer is a well-known trusted entity.

5.6 Processes

VirusTotal runs each binary in a Cuckoo sandbox that records the executable path or process name of every process that a binary creates, injects, or terminates. Miller et al. [26] create a boolean feature for each executable path or name encountered in the training set and each operation (create, inject, or terminate); it is set to 1 if the binary performs that operation on that process path/name.

We consider this feature monotonic: it is easy for an attacker with sufficient permission to create, inject, or terminate additional processes, but not easy to remove an existing operation, as that may disrupt the functionality of the malware.

5.7 Registry

On Windows, the registry is a system-wide key-value store used by both the system and applications. VirusTotal lists all registry keys set and deleted by each program. Since it is common for malware to both set and delete keys in the registry, Miller et al. [26] include both a set and deleted feature for each registry key encountered in our dataset. The feature is set to 1 for programs that set or delete the key respectively and 0 otherwise.

We treat this feature as monotonic: it is trivial to set or delete additional registry keys, but may not be easy to remove an existing set/delete operation, as that operation might be crucial to the operation of malware (e.g., its persistence).

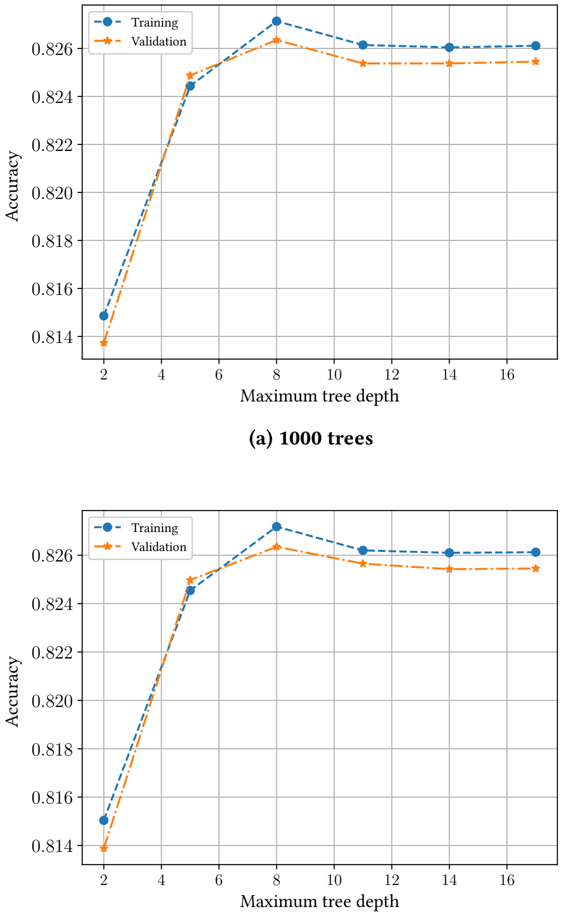
5.8 Symantec Suspicious Insights

VirusTotal also reports data from Symantec's Suspicious Insights program, which computes a reputation score for the binary based on how often it has been executed on other end-user machines. We include a single feature for this information. Because gaining reputation for a program requires installing the program on many end user machines, we assume it is hard for an attacker to clear this feature (but it is easy to set it). Therefore, we treat this feature as monotonic as well.

6 EXPERIMENTS AND RESULTS

We applied our methods to a dataset containing over 700 GB of binaries and scans carried out by VirusTotal between 2011 and 2014. This dataset has over 1.1 million binaries, 5.7 million scans (some binaries were scanned multiple times on different dates), and 200,000 features extracted by Miller et al. [26]. Each scan reports the number of AV vendors which, on the date of the scan, classify the binary as malicious. If at least 4 AV vendors declare the binary malicious, we label it malicious; otherwise, we label it benign. In our dataset, 78% of binaries have a scan in which over 4 AVs declared the binary malicious; that is, 78% of our binaries can be regarded as malicious.

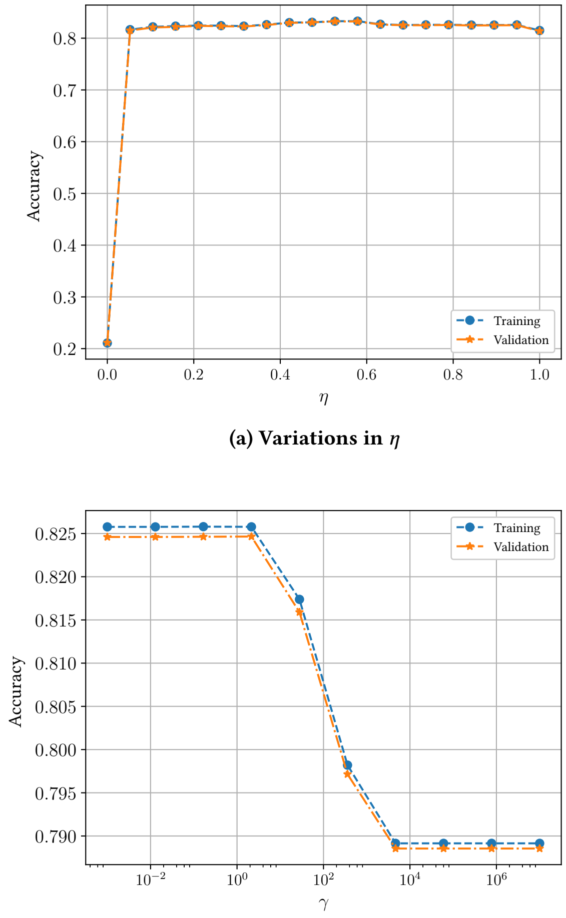
We split the data set into a training set and validation set, trained a classifier using our methods, and evaluated its effectiveness by measuring the following three metrics. Accuracy is defined as the fraction of samples which the classifier classified correctly; the false positive rate is the fraction of benign samples that the classifier classified as malicious; sensitivity is the fraction of malicious samples that the classifier classified as malicious. Sensitivity is also called true positive rate or detection rate. All metrics are measured on the validation set. The sensitivity, S, validation accuracy, A, and false



(a) 1000 trees

(b) 10000 trees

Figure 2: Training and validation accuracy of the monotonic classifier operating on the reduced dataset for various values of the maximum local tree depth of the gradient boosting classifier. The two plots correspond to 1000 and 10000 weak learners, as indicated.



(a) Variations in η

(b) Variations in γ

Figure 3: Training and validation accuracy of the monotonic classifier operating on the reduced dataset for various values of hyperparameters. A classifier composed of 1000 weak classifiers was fit for each experiment, with each tree having a maximum depth of 2. The hyperparameter η is the step shrinkage parameter; it takes values in the interval $[0, 1]$ and is tuned to prevent overfitting. Hyperparameter γ controls the value by which the loss must decrease in order for the classifier to accept a split; this hyperparameter takes non-negative real values.

positive rate, *FP*, are related as follows:

$$S = A(1 + \eta) - \eta(1 - FP),$$

where η is the ratio of benign samples to malicious samples.

When applying our monotonic classifier, we filtered the features as explained in Section 5 to retain only those features that we treat as monotonic, yielding approximately 100,000 features; we refer to datasets using these features as "reduced features" datasets. We compare to a conventional classifier that is trained on all features.

6.1 Non-temporal tests

In the first part of our tests, we disregard the temporal information provided by the timestamped scans and declare a binary malicious if there exists any scan for which at least 4 AVs declared the binary malicious. As a result, for this part of the tests, our dataset is an array of features and a label per binary for each of the 1.1 million binaries. To reduce training time, we selected a random subset of 200,000 binaries and randomly split into training and validation sets, with 80% of the binaries used for training and 20% for validation. The primary limitation of this methodology is that results may be biased by training on the future and testing on the past; we will address those limitations in the next subsection.

We evaluated the performance of a monotone classifier (i.e. one which enforces monotonicity of the trained model in all features with which it is presented) operating on the reduced dataset. We

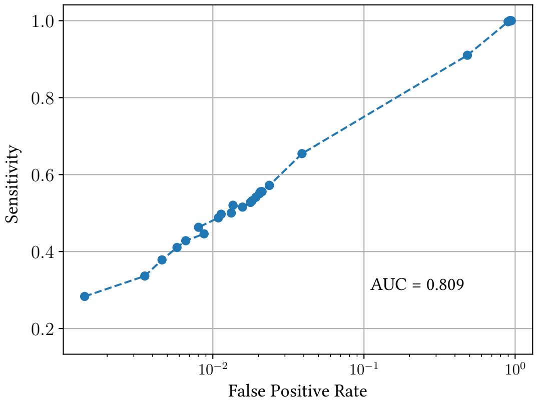


Figure 4: ROC of the monotonic classifier on the reduced features, with a training-test split that ignores temporal information. Each sample corresponds to the performance of a classifier composed of 1000 weak learners with a maximum tree depth of 7.

experimented with hyperparameters, including the number of trees, tree depth, learning rate, and minimum gain, to select values that are suitable for this domain. We found little difference in performance between using 1000 and 10000 trees in the XGBoost classifier (see Figure 2), so we trained a classifier with 1000 trees and maximum tree depth 7. Similarly, we selected reasonable choices for the learning rate (η) and minimum gain required to make a split in the tree (γ); as Figure 3 shows, it is not difficult to choose reasonable values for these hyperparameters.

Figure 4 shows the ROC for the monotone classifier of 1000 learners. We observe that the classifier has approximately 50% sensitivity at a 1% false positive rate.

6.2 Detecting malware over time

In practice, a malware detector will be trained on data received in the past and must classify samples received after training has occurred. We apply an evaluation methodology that reflects this. In particular, we sorted the 5.7 million scans in our dataset by date and chose five dates in the range of the timestamps such that the number of scans between consecutive dates was the same (up to rounding). For each chosen date, D, we constructed a training set by randomly choosing 500k scans that occurred before that date. The labels on training set instances correspond to the best knowledge available regarding the scanned binary up to time D. The validation set was created by taking the next 40k scans in our dataset that occurred immediately after the chosen date D. We labeled binaries in the validation malicious if there has been any scan in the entire dataset that reported it as malicious (i.e., at least 4 AV products report it as malicious). In this way, we construct 5 training/validation splits. This procedure ensures both temporal sample consistency and temporal label consistency, in the language of Miller et al. [26].

We train two classifiers: a monotonic XGBoost classifier, applied to the reduced features, and an ordinary XGBoost classifier, applied to all features. Figure 5 shows the detection rate obtained for both

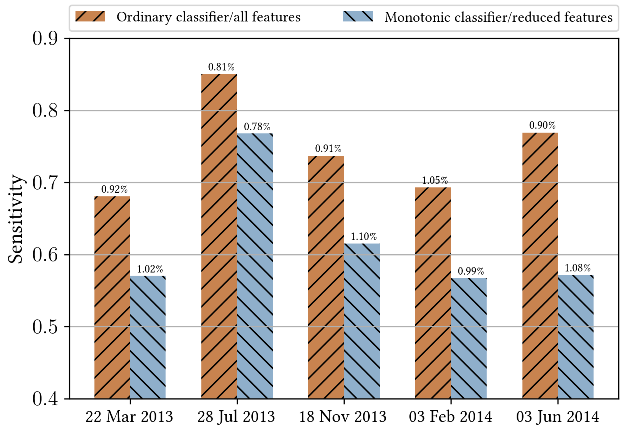


Figure 5: Detection rate of both the ordinary classifier operating on all features and the monotone classifier operating over the dataset containing the reduced set of features. The classifiers are trained using 500k scans which occurred before the indicated date; the reported sensitivity corresponds to the performance of the classifiers over the 40k binaries with scans occurring immediately after the given dates. The labels used to train the classifiers correspond to the best knowledge of the label associated with the binary of the corresponding scan up to the given date, while the label used for validation corresponds to the best knowledge we have of the maliciousness of the binary. Each classifier consists of 1000 weak learners with a maximum depth of 5. The classifiers are trained to target a false positive rate of 1%. On top of each bar in the figure is the false positive rate of the classifier. The fraction of scans labeled malware in all validations sets was close to 50%.

classifiers and for the five chosen dates, and Table 2 shows how the performance of classifiers trained at one point in time degrades when this classifier is used in the future without retraining. We infer from these results that retraining is necessary in order to maintain adequate detection performance; moreover, retraining reinforces our original assumption of monotonicity in some properties because any features which an attacker could have originally added for distraction are likely to be removed over time (since the attacker removed them to evade a detector), and after retraining the attacker cannot remove these features anymore. All classifiers consist of 1000 weak learners and use a loss function which targets a 1% false positive rate. We observe that for the last date shown in these results, 03 Jun 2014, the monotonic classifier operating on the reduced dataset achieved 57% detection rate at a false positive rate of 1.08%, and the ordinary classifier operating on all features had 77% sensitivity at 0.90% false positive rate.

Table 3 summarizes our results, showing the detection rate (sensitivity) averaged across the five dates, for each classifier/feature-set

	22 Mar 2013 Validation Dataset		28 Jul 2013 Validation Dataset		18 Nov 2013 Validation Dataset		03 Feb 2014 Validation Dataset		03 Jun 2014 Validation Dataset	
Training Dataset	Mon. classifier	Ord. classifier	Mon. classifier	Ord. classifier	Mon. classifier	Ord. classifier	Mon. classifier	Ord. classifier	Mon. classifier	Ord. classifier
22 Mar 2013	57.05% (1.02% FP)	68.05% (0.92% FP)	61.26% (1.95% FP)	71.14% (1.70% FP)	54.52% (4.62% FP)	67.47% (5.71% FP)	49.40% (9.11% FP)	57.99% (9.57% FP)	44.29% (4.57% FP)	46.61% (3.77% FP)
28 Jul 2013			76.78% (0.78% FP)	85.03% (0.81% FP)	63.96% (4.23% FP)	75.71% (9.23% FP)	57.33% (10.32% FP)	63.42% (10.67% FP)	45.05% (5.45% FP)	47.93% (4.03% FP)
18 Nov 2013					61.55% (1.10% FP)	73.70% (0.91% FP)	57.89% (4.47% FP)	60.92% (3.95% FP)	35.84% (2.10% FP)	47.29% (1.33% FP)
03 Feb 2014							56.70% (0.99% FP)	69.31% (1.05% FP)	32.72% (0.56% FP)	44.81% (0.42% FP)
03 Jun 2014									57.17% (1.08% FP)	76.90% (0.90% FP)

Table 2: Detection rate of both the ordinary classifier operating on all features and the monotone classifier operating over the dataset containing the reduced set of features, where the training and validation datasets and the training parameters are as described in the caption of Figure 5. The table shows the resulting detection rates (with associated false positive rates) when a model trained on one dataset is validated on datasets occurring at various times.

Classifier	Dataset	Sensitivity	FP rate
Ordinary	All features	74.6%	0.9%
Monotone	Reduced	61.9%	1.0%

Table 3: Average of detection rate and false positives for both classifier/dataset configurations over the five dates shown in Figure 5.

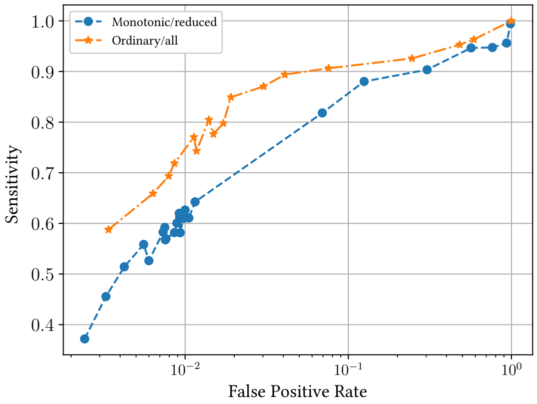


Figure 6: ROC curves for the ordinary classifier operating on all features and the secure classifier operating on the reduced features. Classifiers were trained with the training/validation split set to 18 Nov 2013, the date which splits our dataset in two equal numbers of scans. Each classifier consists of 1000 weak learners which have a maximum depth of 5.

configuration. As we can see, adopting our scheme causes approximately a 13 p.p. drop in detection rate. Of course, the benefit of our scheme is its security against a particular class of (low-cost) evasion attacks. Thus, we interpret the 13 p.p. difference as the cost of security for malware classification.

Figure 6 shows the ROCs of the monotonic classifier with reduced features and the ordinary classifier with all features using 18 Nov 2013 as the split date for the training/validation datasets. Finally, to help shed light on which features have the greatest impact, we computed feature importance scores and ranked the features; Tables 4 and 5 show the most important features.

Weight	Feature
415	SuspiciousInsightSparseFeature:True
213	Uses Packer
103	Import:gdi32.dll
102	Import:oleaut32.dll
93	Import:shlwapi.dll
90	Import:comctl32.dll
89	Import:ws2_32.dll
84	Import:ole32.dll
84	Import:msvcrt.dll
84	Import:comdlg32.dll
81	Import:shell32.dll
79	Import:riched20
73	Import:wininet.dll
72	Import:uxtheme.dll
71	RegKeySetFeature:
70	Import:advapi32.dll
68	Import:version.dll
67	FileWrittenCountFeature:FileWrittenCountFeature_2_4
66	FileOpenedFeature:exe
62	ProcessCreatedFeature:c:
62	FileWrittenCountFeature:FileWrittenCountFeature_4_8
62	FileReadFeature:c:
60	Import:ntdll.dll
58	FileReadCountFeature:FileReadCountFeature_8_16
57	Import:secur32.dll
57	FileWrittenCountFeature:FileWrittenCountFeature_1_2
56	Import:winmm.dll
56	Import:riched20.dll
56	FileReadCountFeature:FileReadCountFeature_1_2
56	FileOpenedFeature:system32
55	FileOpenedFeature:dll
53	Import:user32.dll
53	FileDeletedCountFeature:FileDeletedCountFeature_4_8
52	Import:psapi.dll
52	FileReadFeature:tmp
51	Import:winspool.drv
50	FileReadCountFeature:FileReadCountFeature_4_8
49	FileOpenedFeature:c:
48	Import:userenv.dll
48	FileOpenedFeature:windows

Table 4: Most important features of the monotonic classifier, ranked by the number of times a feature is used to make a decision in the trees. The classifier was trained and validated on the 18 Nov 2013 dataset and targeted a 1% false positive rate (shown in Figure 6).

6.3 A robust and effective malware detector

The ordinary classifier provides the best accuracy if attackers do not attempt to evade the classifier, but very poor accuracy if attackers do attempt to evade it. The monotonic classifier provides somewhat worse accuracy if attackers do not attempt to evade, but much better accuracy if attackers do attempt to evade. Can we construct a scheme that provides the "best of both worlds"? We investigate a construction that is a hybrid of these two methods.

Let f_o denote the ordinary classifier, and f_m the monotonic classifier. We construct a new classifier as

$$f(x) = f_o(x) \vee f_m(x),$$

Gain	Feature
3680.39	FileWrittenFeature:lua
1403.96	FileOpenedFeature:C:\Program Files\SAItest.txt
1169.97	FileMoveSrcFeature:ocsetuphlp
1054.40	RegKeySetFeature:globaluserid
1051.98	FileWrittenFeature:C:\DOCUME 1\<USER> 1\LOCALS 1\Temp\$
949.79	FileOpenedFeature:css
940.36	Import:mpr
780.88	FileWrittenFeature:ftp
763.11	RegKeySetFeature:dotgetright
721.58	Import:icmp.dll
704.58	FileWrittenFeature:tp
579.27	FileWrittenFeature:loading
566.45	FileOpenedFeature:C:\DOCUME 1\<USER> 1\LOCALS 1\Temp\7z
553.54	RegKeyDeleteFeature:hardware
548.70	FileOpenedFeature:<user> 1
529.33	FileOpenedFeature:dup2patcher
390.79	Import:pstorec.dll
377.15	FileOpenedFeature:captura
357.08	FileOpenedFeature:C:\lua\ltn12\init.lua
347.41	ProcessCreatedFeature:1
344.44	FileMoveSrcFeature:data
307.83	FileCopySrcFeature:data
291.94	FileOpenedFeature:_setupx
283.58	FileCopyDstFeature:data
280.94	FileReadFeature:python27
271.76	ProcessCreatedFeature:svchost
261.89	FileReadFeature:C:\WINDOWS\system32\kernel32.dll
233.08	RegKeySetFeature:shell
223.47	RegKeySetFeature:HKEY_CURRENT_USER\SOFTWARE\1ClickDownl
217.59	RegKeyDeleteFeature:versionindependentprogid
207.93	RegKeySetFeature:notepad
207.13	FileDeletedFeature:nshelper
193.58	FileOpenedFeature:7
188.51	FileWrittenFeature:xp
186.19	FileMoveDstFeature:data
173.94	FileOpenedFeature:C:\Documents and Settings\<USER>\Loca
169.33	FileOpenedFeature:C:\DOCUME 1\<USER> 1\LOCALS 1\Temp\lo
167.16	FileDeletedFeature:install
148.13	ProcessCreatedFeature:cmd
146.78	RegKeySetFeature:runonce

Table 5: Most important features of the monotonic classifier, ranked by the average gain obtained when using this feature in the trees. The classifier was trained and validated on the 18 Nov 2013 dataset and targeted a 1% false positive rate (shown in Figure 6).

i.e., classify the executable x as malware if either f_o or f_m does.

Our experiments show that this technique allows the classifier f to keep the high performance of the ordinary classifier (in the absence of evasion) with much of the robustness of the monotonic classifier. For instance, we can form a combined classifier which has a detection rate of 73% at a 1% false positive rate. This is obtained by combining a monotonic classifier with a 56% detection rate with an ordinary classifier having a detection rate of 69%. It follows that it is impossible for an attacker to evade detection in 56% of the malicious binaries in our validation dataset; our detector catches an additional 17% of the malicious binaries in the validation dataset, but it is possible for an attacker to evade this detection (because it occurs through the ordinary classifier). Thus, in the absence of evasion, this is as effective as the ordinary classifier (which attains a 74% detection rate at 1% false positive rate). In the presence of evasion, this scheme can detect many—but not all—malicious binaries; whereas the ordinary classifier has essentially no security against evasion attacks.

In general, we can trade off detection rate in the absence of evasion vs robustness against evasion. Table 6 shows the FP rates and detection rates of the combined classifier for various combinations of FP rates and detection rates of the two classifiers which compose the combined classifier.

Monotonic Classifier		Ordinary Classifier		Combined Classifier	
FP Rate	Sensitivity	FP Rate	Sensitivity	FP Rate	Sensitivity
0.75%	59.21%	0.64%	65.91%	1.11%	71.50%
0.76%	56.98%	0.64%	65.91%	1.13%	70.90%
0.73%	58.27%	0.64%	65.91%	1.15%	71.08%
0.76%	56.75%	0.64%	65.91%	1.14%	70.84%
0.60%	52.63%	0.64%	65.91%	1.00%	69.66%
0.56%	55.85%	0.64%	65.91%	0.94%	70.67%
0.42%	51.43%	0.79%	69.36%	1.00%	71.89%
0.33%	45.56%	0.79%	69.36%	0.93%	70.81%
0.24%	37.19%	0.79%	69.36%	0.91%	70.02%
0.56%	55.85%	0.79%	69.36%	1.10%	72.87%
0.42%	51.43%	0.86%	71.88%	1.08%	73.84%
0.33%	45.56%	0.86%	71.88%	1.03%	73.12%
0.24%	37.19%	0.86%	71.88%	0.98%	72.21%
0.75%	59.21%	0.34%	58.77%	0.90%	67.99%
0.93%	61.21%	0.34%	58.77%	1.05%	69.75%
0.86%	58.19%	0.34%	58.77%	1.03%	67.66%
0.76%	56.98%	0.34%	58.77%	0.95%	67.23%
0.93%	58.16%	0.34%	58.77%	1.09%	68.28%
0.73%	58.27%	0.34%	58.77%	0.91%	67.33%
0.89%	60.06%	0.34%	58.77%	1.07%	68.42%
0.92%	61.99%	0.34%	58.77%	1.10%	70.24%
0.76%	56.75%	0.34%	58.77%	0.92%	67.07%

Table 6: We create a new classifier, which we call a combined classifier, by instantiating an ordinary classifier operating on the full set of features, and a monotonic classifier operating on the reduced feature set. The table shows the detection rate for the combined classifier for various detection rates of each of the components (the monotonic and ordinary classifiers). The classifiers were trained and validated using 18 Nov 2013 as the split date. 500k samples are used for training, and the 40k scans which occur immediately after the split date are used to create the validation set.

		Monotonic Classifier		Combined Classifier	
Date	Baseline	FP Rate	Sensitivity	FP Rate	Sensitivity
22 Mar 2013	47.53%	0.41%	55.34%	1.01%	77.93%
28 Jul 2013	55.80%	0.51%	74.27%	0.99%	85.12%
18 Nov 2013	58.74%	0.56%	53.09%	1.04%	72.43%
03 Feb 2014	50.60%	0.50%	47.82%	1.01%	66.32%
03 Jun 2014	55.18%	0.41%	49.49%	1.01%	73.99%

Table 7: We trained a combined classifier of a 1% false positive rate by using a secure classifier with a 0.5% false positive rate for each of the datasets resulting from the split dates shown in Figure 5. Each of the monotonic and ordinary classifiers comprising the combined classifier was trained on 500k samples occurring before the split date and validated on the 40k samples occurring immediately after the split date. The baseline indicates the fraction of malware present in the validation set. Each classifier consists of 1000 boosted trees with a maximum depth of 5.

We ran further tests to verify whether the combined classifier keeps its higher detection rates over time. Table 7 shows the detection rate for a 1% false positive classifier which uses a 0.5% false positive monotonic classifier for five split dates. Comparing these results with those in Figure 5, we observe that the combined classifiers outperform the secure classifiers for all dates.

7 CONCLUSION

In this paper, we focus on a specific type of evasion attack where an adversary attempts to evade a classifier by adding more features. Real world attacks are more complex than adding already known features. Our proposal is the first step at designing a robust ML model by construction, which can be used as a building block to handle complex adversarial attacks.

Care has to be exercised when applying machine learning to security-related applications [28], because of the potential for adversaries to mislead the classifier using evasion attacks. We argued that a more secure way to use machine learning techniques in malware classification is to train a malware classifier that enforces monotonicity on certain carefully-chosen features. In particular, we identify features where it is hard for attackers to modify their malware to change those features in a particular direction, and use those features with the monotonic classifier. In effect, we use domain knowledge about the malware domain to categorize what kinds of influence an attacker can have over the features, and then select a classifier that will be secure by design against those kinds of manipulations.

We evaluated our approach on over 700GB of binaries from VirusTotal. Our results indicate that the approach is effective: we can achieve comparable detection performance to a traditional, unprotected classifier; yet the cost of modifying malware to evade detection is increased. In particular, on average the detection rate drops from approximately 75% (without hardening against adversarial examples) to approximately 62% (with our hardening scheme), a tax of approximately 13%. We interpret this as the cost of security for applying machine learning in this domain.

ACKNOWLEDGMENTS

We thank Qi Zhong and the authors of Miller et al. [26] for the data and features. We are also grateful for the generous support from the Center for Long-Term Cybersecurity (CLTC) at UC Berkeley and the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of any funding sponsors.

REFERENCES

[1] Ibrahim M Alabdulmohsin, Xin Gao, and Xiangliang Zhang. 2014. Adding robustness to support vector machines against adversarial reverse engineering. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 231–240.

[2] Norman P Archer and Shouhong Wang. 1993. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences* 24, 1 (1993), 60–75.

[3] Arie Ben-David. 1995. Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning* 19, 1 (1995), 29–43.

[4] Arie Ben-David, Leon Sterling, and TriDat Tran. 2009. Adding monotonicity to learning algorithms may impair their accuracy. *Expert Systems with Applications* 36, 3 (2009), 6627–6634.

[5] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. 2017. Dimensionality Reduction as a Defense against Evasion Attacks on Machine Learning Classifiers. *arXiv preprint arXiv:1704.02654* (2017).

[6] Nicholas Carlini and David Wagner. 2017. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. *arXiv preprint arXiv:1705.07263* (2017).

[7] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy*.

[8] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[9] Hennie Daniels and Marina Velikova. 2010. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks* 21, 6 (2010), 906–917.

[10] Wouter Duivesteijn and Ad Feelders. 2008. Nearest neighbour classification with monotonicity constraints. *Machine Learning and Knowledge Discovery in Databases* (2008), 301–316.

[11] Ad Feelders. 2010. Monotone relabeling in ordinal classification. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 803–808.

[12] Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. 2017. Adversarial and Clean Data Are Not Twins. *arXiv preprint arXiv:1704.04960* (2017).

[13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2015).

[14] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280* (2017).

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

[16] Maya Gupta, Andrew Cotter, Jan Pfeifer, Konstantin Voevodski, Kevin Canini, Alexander Mangylov, Wojciech Moczydlowski, and Alexander Van Esbroeck. 2016. Monotonic calibrated interpolated look-up tables. *The Journal of Machine Learning Research* 17, 1 (2016), 3790–3836.

[17] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *Boosting and Additive Trees*. Springer New York, New York, NY, 337–387. https://doi.org/10.1007/978-0-387-84858-7_10

[18] Dan Hendrycks and Kevin Gimpel. 2017. Early Methods for Detecting Adversarial Images. (2017).

[19] Weiwei Hu and Ying Tan. 2017. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *arXiv preprint arXiv:1702.05983* (2017).

[20] Nwokedi Idika and Aditya P Mathur. 2007. A survey of malware detection techniques. *Purdue University* 48 (2007).

[21] Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. 2015. Robust convolutional neural networks under adversarial noise. *arXiv preprint arXiv:1511.06306* (2015).

[22] Alex Kantchelian, JD Tygar, and Anthony D Joseph. 2016. Evasion and hardening of tree ensemble classifiers. *33rd ICML* 48 (2016), 2387–2396.

[23] Herbert Kay and Lyle H Ungar. 2000. Estimating monotonic functions and their bounds. *AIChE Journal* 46, 12 (2000), 2426–2434.

[24] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. 2017. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267* (2017).

[25] Bradley Miller. 2015. *Scalable Platform for Malicious Content Detection Integrating Machine Learning and Manual Review*. Ph.D. Dissertation. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-194.html

[26] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. 2016. Reviewer integration and performance measurement for malware detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 122–141.

[27] Hari Mukarjee and Steven Stern. 1994. Feasible nonparametric estimation of multiargument monotone functions. *J. Amer. Statist. Assoc.* 89, 425 (1994), 77–80.

[28] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 305–316.

[29] Nedim Srndic and Pavel Laskov. 2014. Practical evasion of a learning-based classifier: A case study. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 197–211.

[30] Weilin Xu, David Evans, and Yanjun Qi. 2017. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. *arXiv preprint arXiv:1704.01155* (2017).

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

[32] Ilsun You and Kangbin Yim. 2010. Malware obfuscation techniques: A brief survey. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*. IEEE, 297–300.

[33] Fei Zhang, Patrick PK Chan, Battista Biggio, Daniel S Yeung, and Fabio Roli. 2016. Adversarial feature selection against evasion attacks. *IEEE transactions on cybernetics* 46, 3 (2016), 766–777.

[34] Qi Zhong, Alex Kantchelian, Sadia Afroz, Doug Tygar, and Anthony D. Joseph. 2017. Hardening Malware Pipeline Against Evasion Attacks. (May 2017). Research presentation.