Evading Classifiers by Morphing in the Dark

Hung Dang
National University of Singapore
hungdang@comp.nus.edu.sg

Yue Huang
National University of Singapore
a0119416@u.nus.edu

Ee-Chien Chang
National University of Singapore
changec@comp.nus.edu.sg

ABSTRACT

Learning-based systems have been shown to be vulnerable to evasion through adversarial data manipulation. These attacks have been studied under assumptions that the adversary has certain knowledge of either the target model internals, its training dataset or at least classification scores it assigns to input samples. In this paper, we investigate a much more constrained and realistic attack scenario wherein the target classifier is minimally exposed to the adversary, revealing only its final classification decision (e.g., reject or accept an input sample). Moreover, the adversary can only manipulate malicious samples using a blackbox morpher. That is, the adversary has to evade the targeted classifier by morphing malicious samples "in the dark". We present a scoring mechanism that can assign a real-value score which reflects evasion progress to each sample based on the limited information available. Leveraging on such scoring mechanism, we propose an evasion method – EvadeHC– and evaluate it against two PDF malware detectors, namely PDFRATE and Hidost. The experimental evaluation demonstrates that the proposed evasion attacks are effective, attaining 100% evasion rate on the evaluation dataset. Interestingly, EvadeHC outperforms the known classifier evasion techniques that operate based on classification scores output by the classifiers. Although our evaluations are conducted on PDF malware classifiers, the proposed approaches are domain-agnostic and are of wider application to other learning-based systems.

1 INTRODUCTION

Machine learning techniques have witnessed a steady adoption in a wide range of application domains such as image classification [31] or natural language processing [33]. Various learning-based systems have been reported to achieve utmost accuracy, even surpassing human-level performance [15, 26]. Given the assumption that the training datasets are representative, these systems are expected to perform well in reality on operational data.

Learning methods have also lent themselves to security tasks. Numerous innovative applications of machine learning in security contexts, especially for detection of security violations, have been discussed in the literature [19, 22, 28]. However, when learning-based systems are deployed for security applications, their accuracy may be challenged by intentional noise and deviations. Various studies [7, 11, 12] suggest that they are likely susceptible to evasion

through adversarial data manipulation, for the assumption on representativeness of the training datasets no longer holds true in the presence of malicious adversaries. For example, a motivated attacker may be able to morph a malicious instance so that it resembles a benign instance found in the training datasets, evading the targeted classifier.

Several works have studied evasion attacks against learning-based systems, making different assumptions on the amount of knowledge the adversaries have about the system they are attacking. For examples, Šrndic et al. [30] and Sharif et al.[24] studied attack scenarios wherein the adversaries have a high level of knowledge about the internals of the targeted system (e.g., features space and classification algorithm). Xu et al. [34] investigated a more constrained evasion scenario in which the adversaries only have black-box accesses to the targeted detector that outputs a real-value classification score for an input sample. While these evasion attacks are effective in the presence of the required auxiliary information, it has been suggested that a simple preventive measure of hiding the classification score would be sufficient to thwart these attacks [34]. Thus, it remains a question how these techniques operate against truly blackbox systems deployed in reality (e.g., malware detector built-in to email services), for they are unlikely to reveal the real-value scores, but rather expose only their final decisions (e.g., reject or accept a sample).

In this work, we study the problem of classifier evasion in the dark, investigating an evasion scenario that is much more constrained and realistic than those that have been considered in existing works [25, 30, 34]. In particular, we study a very restricted setting in which the adversary does not have any knowledge about the targeted system (e.g., model internals and training dataset). Moreover, the targeted system only reveals its final decisions instead of real-value scores that reflect its internal state. We further assume that the only feasible mean to manipulate the samples is through some given tools that perform "random" morphing.

To this end, we formulate a model that captures the restricted setting described above. Under the proposed model, the adversary is confined to three blackboxes, a binary-output detector (or classifier) that the adversary attempts to evade, a tester that checks whether a sample possesses malicious functionality, and a morpher that transforms the samples. Given a malicious sample, the adversary adaptively queries the blackboxes to search for a morphed sample that evades detection. A sample is said to evade detection if it is perceived as benign by the targeted detector, yet retains its malicious functionality.

We show that under this constrained setting with only blackbox accesses, learning-based systems are still susceptible to evasion attacks. We contend that this is an important result, as it indicates that suggested defensive mechanisms are insufficient to prevent evasion attacks against learning-based systems. We demonstrate

feasibility of effective evasion attacks by describing various evasion approaches, chief among them is an hill-climbing based approached EvadeHC. The main component of EvadeHC is a scoring mechanism that assigns real-value scores to samples based on the binary outputs obtained from the tester and detector. The intuition is to measure the number of morphing steps required to change the detector and tester's decisions, and derive the score based on these values. We believe that this scoring mechanism can be used to relax the assumption on the availability of real-value scores that other settings [25, 34] make.

We evaluate the proposed evasion technique on two well-known PDF malware classifiers, namely PDFRATE [27] and Hidost [29]. The evaluation is conducted on the same dataset that is used in previous work by Xu et al. [34], so as to enable a fair basis for benchmarking. This dataset consists of 500 malicious samples chosen from the Contagio archive [6]. We first compare EvadeHC against a baseline solution which keeps generating random morphed samples until an evading sample is found. Empirical results show that EvadeHC attains 100% evasion rate on the experimental dataset, and outperforms the baseline solution by upto 80 times in term of execution cost. Further, we also experiment on situations wherein the classifiers are hardened (i.e., decreasing false acceptance rate at a cost of increasing false rejection rate). The results strongly demonstrate the robustness of EvadeHC. We also benchmark EvadeHC against a state-of-the-art technique in evading blackbox classifiers, which uses real-value classification scores during evasion [34]. Interestingly, even with only access to binary outputs of the detector, EvadeHC still outperforms the previous work. While this may appear counter-intuitive, the result in fact explicable. We believe the reasons are two folds. First, EvadeHC makes decisions based on information obtained from both the tester and detector, as opposed to previous work relying solely on the classification scores output by the detector. Secondly, values assigned by our scoring mechanism could be a more accurate measure of evasion difficulty in comparison with the real-value classification scores.

Contributions. This paper makes the following contributions:

(1) We give a formulation of classifier evasion in the dark whereby the adversary only has blackbox accesses to the detector, a morpher and a tester. We also give a probabilisitic model HsrMopher to formalise the notion that almost no domain-specific knowledge can be exploited in the evasion process.

(2) We design a scoring function that can assign real-value score reflecting evasion progress to samples, given only binary outcomes obtained from the detector and tester. We believe that this scoring mechanism is useful in extending existing works that necessitate classification scores or other auxiliary information to operate under a more restricted and realistic setting like the one we study.

(3) Leveraging on the scoring function, we propose an effective hill-climbing based evasion attack EvadeHC. This algorithm is generic in a sense that it does not require any knowledge about the target system, even its confidence level in the classification decision.

(4) We conduct experimental evaluation on two popular PDF malware classifiers. The empirical results demonstrate not only the efficiency but also the robustness of EvadeHC. More notably, it is also suggested that the scoring mechanism underlying EvadeHC is more informative than the one that relies solely on classification scores [34].

The rest of the paper is structured as follows. We formulate the problem and discuss its challenges in Section 2 before proposing our evading methods in Section 3. Next, we formalize our proposed approach by presenting probabilistic models in Section 4 and report the experimental evaluation in Section 5. We discuss the implications of our evasion attacks and their mitigation strategies in Section 6. We survey related works in Section 7 before concluding our work in Section 8.

2 PROBLEM FORMULATION

In this section, we formulate the problem of evading classifier in the dark and discuss its challenges. Prior to presenting the formulation, we give a running example to illustrate the problem and its relevant concepts.

2.1 Motivating Scenario

Let us consider an adversary that aims to send a malware to a victim via email. The adversary chooses to embed the malware in a PDF file, for the victim is more willing to open a PDF file than other file formats. Most email service providers would have built-in malware detection mechanisms scanning users' attachments before delivering them. Such malware detection mechanism is usually a classifier that makes decision based on some extracted features, with a classification model trained using existing data. It is reasonable to assume that the adversary does not have any knowledge about the detector that the email service provider employs (e.g., the algorithms and feature space adopted by the classifier, or its training dataset). Nevertheless, the adversary could probe the detector by sending emails with the malicious payload attached, possibly to another account that it owns, and observing the binary responses (accept/reject) from the email server. Further, the adversary could adaptively modify the malicious PDF file to search for a variant that evades detection. However, we assume that the adversary could only probe the email server's detector a limited number of times before it is blacklisted.

The adversary expects that its malicious file will be rejected by the detector. To evade detection, it has access to two tools: a morphing tool that transforms the sample, and a sandboxing tool that tests whether the morphed sample retains its malicious functionalities. For instance, the morphing tool may insert, delete or replace objects in an underlying representation of the file, and the sandboxing tool dynamically detects whether the sample causes the vulnerable PDF reader to make certain unexpected system calls. Due to its insufficient understanding of the underlying mechanism to manipulate the PDF sample, the adversary employs the morphing tool as a blackbox. Furthermore, due to the complexity of the PDF reader, the adversary does not know whether a morphed PDF sample will retain its functionality, and the only way to determine that is to invoke the sandboxing tool.

Given such limitations on the knowledge of both the detector and morphing mechanism, we aim to investigate whether effective attacks are still possible. To capture such constrained capability, our formulation centres on the notion of three blackboxes: a binary-outcome detector $\mathcal{D}$ that the adversary would like to evade, a morpher $\mathcal{M}$ that "randomly" yet consistently morphs the sample, and a tester $\mathcal{T}$ that checks the sample's functionality.

2.2 Tester $\mathcal{T}$, Detector $\mathcal{D}$ and Evasion

The tester $\mathcal{T}$, corresponding to the sandbox in the motivation scenario, declares whether a submitted sample x is *malicious* or *benign*. $\mathcal{T}$ is deterministic in the sense that it will output consistent decisions for the same sample.

The blackbox detector $\mathcal{D}$ also takes a sample x as input, and decides whether to *accept* or to *reject* the sample. $\mathcal{D}$ corresponds to the malware classifier in the motivating scenario. Samples that are rejected by the detector are those that are classified by the detector to be malicious. It is possible that a sample x declared by $\mathcal{T}$ to be malicious is accepted by $\mathcal{D}$. In such case, we say that the sample x *evades* detection. Of course, if the detector is exactly the same as the tester, evasion is not possible. In fact, the main objective of our work is to study the security of detectors with imperfect accuracy. Similar to $\mathcal{T}$, we consider detectors that are deterministic (i.e., their output is always the same for the same input).

We highlight that, in our formulation, the detector's output is binary (e.g., accept/reject), as opposed to many previous works (e.g., [30, 34]) that assume real-value outputs. Such a real-value output can be interpreted as the detector's confidence level in its decision.

2.3 Morpher $\mathcal{M}$

The morpher $\mathcal{M}$ takes as input a sample x and a random seed s, and outputs another sample $\widetilde{x}$. We call such action a morphing step. The morpher corresponds to the morphing mechanism described in the motivating scenario. The random seed s supplies the randomness required by the morpher, and at the same time enables the morphing mechanism to be replicable (i.e., given the same input x and s, $\mathcal{M}$ consistently output the same morphed sample). We are not concerned with the representation of the random seed s, and for simplicity, treat it as a short binary string.

Starting from a sample x_0, the adversary can make successive calls to $\mathcal{M}$, say with a sequence of random seeds $\mathbf{s} = \langle s_1, s_2, \ldots, s_L \rangle$, to obtain a sequence of samples $\mathbf{x} = \langle x_0, x_1, x_2, \ldots, x_L \rangle$, where $x_i = \mathcal{M}(x_{i-1}, s_i)$. Let us call $(\mathbf{x}, \mathbf{s})$ a *path* with starting point x_0, endpoint x_L, and path length L. When it is clear from the context, we shall omit $\mathbf{s}$ in the notation, and simply refer to the path by $\mathbf{x}$.

The formulation does not dictate how the morphing is to be carried out. The adversary can exploit useful properties of the application domain to manipulate the samples. For instance, the adversary may be able to efficiently find a morphing path connecting two given samples, or know how to merge two morphing paths to attain certain desirable property. Nevertheless, such domain specific properties are not always available in general. In Section 4, we propose a probabilistic model of "random" morphing to capture the restriction that almost no underlying useful properties can be exploited in manipulating the samples.

2.4 Adversary's Goal and Performance Cost

Given a malicious sample x_0, the goal of the adversary is to find a sample that evades detection with minimum cost. We call a sample *evading* if it is accepted by the detector $\mathcal{D}$, but exhibits malicious behaviours as perceived by the tester $\mathcal{T}$. If the given sample x_0 is already evading, then the adversary has trivially met the goal. Otherwise, the adversary can invoke the morpher $\mathcal{M}$ to obtain other samples, and check the samples by issuing queries to $\mathcal{T}$ and $\mathcal{D}$.

Let N_d, N_t, and N_m be the number of queries the adversary sent to $\mathcal{D}, \mathcal{T}$ and $\mathcal{M}$ over the course of the evasion, respectively. We are interested in scenarios where N_d is a dominating component in determining the evasion cost. In the motivating scenario, the detector can only be accessed remotely and the email server (who is the defender) imposes a bound on the number of such accesses.

While the adversary could freely access the tester, its computational cost might be non-trivial. For instance, in the motivating scenario, computationally intensive dynamic analysis is required to check the functionality. In our experiments, each such test takes around 45 seconds on average. Morphing, on the other hands, is less computationally expensive. Hence, it is reasonable to consider an objective function where N_t carries significantly more weight than N_m.

A possible optimisation model is to minimise some cost function (e.g., $N_d + 0.5N_t + 0.01N_m$). Alternatively, we could consider constrained optimisation that imposes a bound on N_d, while minimising a cost function involving the other two measurements (e.g., $N_t + 0.02N_m$). Since cost functions depend on application scenarios, we do not attempt to tune our algorithm to any particular cost function. Instead, we design search strategies with a general objective of minimising N_d, followed by N_t and N_m.

2.5 Flipping Samples and Gap

Let us consider a path $\mathbf{x} = \langle x_0, x_1, \ldots, x_L \rangle$ in which x_0 is malicious and rejected while x_L is benign and accepted. Somewhere along the path, the samples turn from malicious to benign, and from rejected to accepted. Starting from x_0, let us call the first sample that is benign the *malice-flipping sample*, and the first accepted sample the *reject-flipping sample* (see Figure 1). Let $m_{\mathbf{x}}$ and $r_{\mathbf{x}}$ be the respective number of morphing steps from x_0 to reach the malice-flipping sample and reject-flipping sample. That is,

$$m_{\mathbf{x}} = \arg\min_i \{\mathcal{T}(x_i) = benign\},$$
$$r_{\mathbf{x}} = \arg\min_i \{\mathcal{D}(x_i) = accept\}$$

We call $m_{\mathbf{x}}$ the *malice-flipping distance*, and $r_{\mathbf{x}}$ the *reject-flipping distance*. We further define by $g_{\mathbf{x}}$ the gap between $m_{\mathbf{x}}$ and $r_{\mathbf{x}}$:

$$g_{\mathbf{x}} = r_{\mathbf{x}} - m_{\mathbf{x}}.$$

The value of $g_{\mathbf{x}}$ is of particular interest. If $g_{\mathbf{x}} < 0$, then the reject-flipping sample is still malicious, and therefore is an evading sample.

It typically requires special crafting to embed an exploit in a malicious sample. Thus, it is expected that once a sample has turned to being benign, it is unlikely to regain the malicious functionality via random morphing. Similar assumption can be made about samples flipping from being accepted to rejected, for the features that the
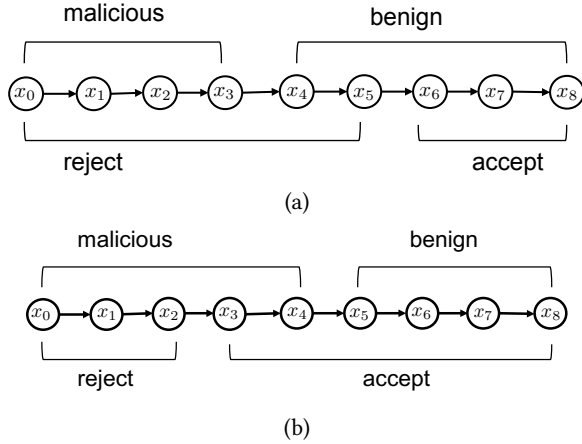
Figure 1: Illustration of flipping samples. (a) The malice-flipping sample is x_4 and the reject-flipping sample is x_6. The malice-flipping distance $m_\mathbf{x} = 4$ and the reject-flipping distance $r_\mathbf{x} = 6$. (b) The malice-flipping sample is x_5 whereas the reject-flipping sample is x_3. The malice-flipping distance $m_\mathbf{x} = 5$ and the reject-flipping distance $r_\mathbf{x} = 3$. Since the gap $(r_\mathbf{x} - m_\mathbf{x})$ is negative, x_3 is evading.

classifiers are trained to identify as grounds for making classification decision are unlikely to be formed via random morphing. Thus, we make an assumption that once a sample has flipped from being malicious (or rejected) to benign (or accepted), it would not be morphed to malicious (or rejected) again.

2.6 Challenges in Evasion in the Dark

There are two main challenges in evasion with the limited blackbox accesses. First, the detector $\mathcal{D}$ only provides a binary output of accept/reject. Hence, from a set of malicious samples, it is not clear which is "closer" to an evading sample. Previous works such as [34] assume that $\mathcal{D}$ returns a real-value confidence level, based on which the evasion is guided. Nevertheless, in practice, a more cautious detector might not reveal any information other than its final decision.

Secondly, the blackbox morpher $\mathcal{M}$ alone does not provide flexibility in manipulating the samples. To illustrate, consider the situation where the adversary has a malicious sample x_0, and another sample x_1 that losses its malicious functionality and being accepted by the detector. Intuitively, one could construct another sample $\hat{x}$ from x_1 and x_0 that has higher chance of evasion. With only blackbox accesses to $\mathcal{M}$, it is challenging to construct such $\hat{x}$. One possible way is to find a sample that lays somewhere in-between a path that starts from x_1 and ends at x_0. However, it is not clear how to efficiently find a morphing path that connects the two given samples. Even if it is possible to connect them, there is still a fundamental question on whether samples along the path have higher chance of evasion.

3 OUR PROPOSED EVASION METHODS

In this section, we present evasion methods that search for evading samples given a malicious sample that is rejected by the detector. For clarity of exposition, we first start with a trivial exhaustive search algorithm SeqRand which serves as the baseline for evaluation. We then optimize SeqRand to reduce its operation costs (i.e., number of queries issued to $\mathcal{D}$ and $\mathcal{T}$), obtaining BiRand. Next, we propose a hill-climbing based algorithm $\mathsf{EvadeHC_0}$ and finally present our main evasion approach EvadeHC that further reduces the number of detector's queries.

3.1 Evasion by Exhaustive Search SeqRand

On a malicious sample x_0 which is rejected by $\mathcal{D}$, SeqRand operates by successively morphing x_0 and checking the morphed samples against $\mathcal{D}$ and $\mathcal{T}$ until an evading sample is found. SeqRand is parameterised by a single threshold L and proceeds as follows:

(1) Choose a sequence of L random seeds $\mathbf{s} = \langle s_1, s_2, \ldots, s_L \rangle$.
(2) For each seed s_i in $\mathbf{s}$, obtain a morphed sample x_i by invoking $\mathcal{M}(x_{i-1}, s_i)$. Check x_i against $\mathcal{D}$ and $\mathcal{T}$, if it is evading, output x_i and halt. Otherwise continue until $\mathbf{s}$ is consumed.
(3) If $\mathbf{s}$ has been consumed and an evading sample has not been found, repeat step 1 to 3 until an evading sample is found.

Let N_r be the number of iterations SeqRand executed when the search halts. It is easy to see that the numbers of queries issued to $\mathcal{D}, \mathcal{T}$ and $\mathcal{M}$ are the same:

$$N_d = N_t = N_m = N_r \cdot L$$

3.2 Evasion by Binary Search BiRand

The baseline approach SeqRand can be improved in various ways. One straight-forward enhancement is to reduce the number of detector queries issued in each iteration (a.k.a. path). In particular, one does not need to invoke $\mathcal{D}$ after every morphing step, but rather should submit only the sample preceding the malice-flipping sample (e.g., x_3 in Figure 1a and x_4 in Figure 1b) to the detector. Further, one can also reduce the number of tester queries in searching for the malice-flipping samples by employing binary search in place of a series of sequential check as in the baseline.

The algorithm BiRand, on an input x_0 which is malicious and rejected by $\mathcal{D}$, carries out the following:

(1) Choose a sequence of L random seeds $\mathbf{s}$, and invoke $\mathcal{M}$ successively with s (starting from x_0) to obtain a path $\mathbf{x}$ of length L.
(2) Find the malice-flipping sample $\tilde{x}$ in the path using binary-search with the tester $\mathcal{T}$. Let $\hat{x}$ be the immediate predecessor of $\tilde{x}$ on the path (i.e., $\hat{x}$ is malicious).
(3) If $\hat{x}$ is accepted by the detector $\mathcal{D}$, output $\hat{x}$ and halt; otherwise repeat step 1 to 3 until an evading sample is found.

Similar to SeqRand, BiRand is also parameterised by a single threshold L. The parameter L is set to be sufficiently large such that with high probability, L morphing steps would turn x_0 from malicious to benign. In our experiments, L ranges from 50 to 80.

Number of Queries. We note that the number of paths BiRand traverses in finding evading samples is the same as the number of iterations taken by SeqRand, hence they incur the same

number of morphing steps. On the other hand, BiRand requires less queries to $\mathcal{D}$ and $\mathcal{T}$. In particular, determining the malice-flipping sample on each path (step 2) incurs at most $\lceil \log_2 L \rceil$ queries to $\mathcal{T}$, and there is only one query to the detector $\mathcal{D}$ per each path (step 3). In sum, we have:

$$N_d = N_r, \quad N_t \le N_r \lceil \log_2 L \rceil, \quad N_m = N_r \cdot L$$

Note that the determination of the flipping samples is designed to minimise the number of queries to $\mathcal{D}$. In the scenario where N_t imposes a larger cost than N_d, the algorithm can be modified so that the binary searches are conducted on $\mathcal{D}$, finding reject-flipping samples. It then tests the candidates against $\mathcal{T}$, incurring only one query to $\mathcal{T}$ per each path.

Robustness of Binary Search. The binary search significantly reduces N_t compared to the linear scan. As discussed earlier, we make an implicit assumption that once a sample has become benign (or accepted), it is highly unlikely that $\mathcal{M}$ will morph it to malicious (or rejected) again. Nevertheless, if such event happens, the binary-search may return a wrong flipping sample. Fortunately, the effect of such failure is confined to the path in question, and does not affect other paths.

We can further improve the efficiency of the evasion by incorporating information obtained from $\mathcal{D}$ and $\mathcal{T}$ in a heuristic which "guides" the evasion. To this end, we propose a hill-climbing algorithm, dubbed EvadeHC$_0$.

3.3 Evasion by Hill-Climbing EvadeHC$_0$

EvadeHC$_0$ is parameterised by two pre-determined integers q_1, q_2, a pre-determined real-value *jump factor* $0 < \Delta < 1$ and a scoring function. The algorithm proceeds in iterations, maintaining a set C of q_2 candidates over the iterations (in the first iteration, C contains the given malicious samples x_0). In each iteration, EvadeHC$_0$ first generates q_1 random paths with starting points originating from C. If any of the paths contains an evading sample, the algorithm outputs it and halts. Otherwise, a new candidate, determined by the jump factor Δ, is chosen from each path, and a real-value score is computed for each new candidate using a scoring function which takes as inputs the flipping distances of the path. Finally, q_2 candidates with the highest score are retained for the next iteration.

Details of each iteration are as follow:

(1) For each sample x in C, generate $\lfloor q_1/|S| \rfloor$ random paths with x as starting point, obtaining approximately q_1 paths. Let P be the set of generated paths. Reset C to empty set.
(2) For each path $\mathbf{x} \in P$, perform the following:
   (a) Find the malice-flipping distance $m_{\mathbf{x}}$ using binary search.
   (b) Find the reject-flipping distance $r_{\mathbf{x}}$ using binary search. If $m_{\mathbf{x}} > r_{\mathbf{x}}$, output the reject-flipping sample and halt (i.e., an evading sample is found).
   (c) Let $\widehat{x}$ be the sample along $\mathbf{x}$ at a distance of $\lfloor \Delta m_{\mathbf{x}} \rfloor$ morphing steps from the starting point. Compute the score $v = \text{score}(m_{\mathbf{x}}, r_{\mathbf{x}})$, and associate the sample $\widehat{x}$ with the score v. Insert $\widehat{x}$ into C.
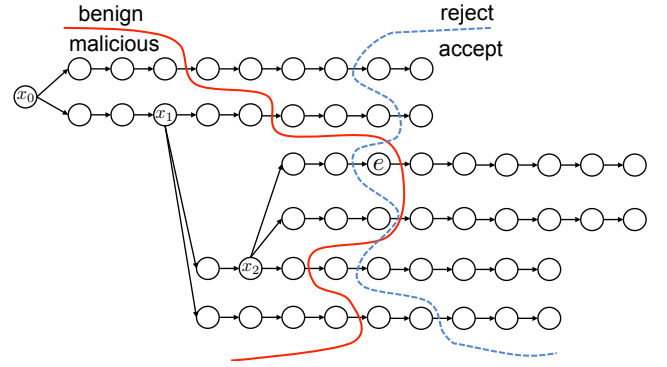(3) Keep q_2 samples with the largest scores in C, and discard the rest .

Figure 2: Illustrating three iterations of Hill-Climbing. The parameters (q_1, q_2, Δ) are $(2, 1, 0.5)$. Hence, C only contain one starting sample. The sample x_0, x_1 and x_2 are the starting sample chosen for the first, second and third iteration respectively. The sample e is evading.

Scoring function. A crucial component is the scoring function score. Two possible choices of the scoring function are based on difference or relative ratio of the two flipping distances:

$$\text{score}_1(m_{\mathbf{x}}, r_{\mathbf{x}}) = m_{\mathbf{x}} - r_{\mathbf{x}}, \quad \text{score}_2(m_{\mathbf{x}}, r_{\mathbf{x}}) = m_{\mathbf{x}}/r_{\mathbf{x}}.$$

Finally, we present our main evasion approach, called EvadeHC. EvadeHC is built on top of EvadeHC$_0$ described earlier. The approach employs a branch-and-bound technique to further reduce the number of required detector queries.

3.4 Enhanced Evasion by Hill-Climbing EvadeHC

Recall that in EvadeHC$_0$, a complete binary search is carried out on each path (Step 2(b)) to find a reject-flipping distance $r_{\mathbf{x}}$. Thus, it requires q_1 binary searches with the detector $\mathcal{D}$ to find the best q_2 candidates in each iteration. EvadeHC, on the other hands, performs a *single* binary search to find a score $\hat{v}$ such that exactly q_2 candidates have scores greater than $\hat{v}$.

Specifically, the binary search takes in a set of paths P as input. We assume that the malice-flipping distance of each path in P is already determined. Let $\hat{v}$ be the target score, and $\hat{P}$ be the target set of paths that attains scores greater than $\hat{v}$. EvadeHC searches for $\hat{v}$ and $\hat{P}$ in a single binary search, while maintaining a set of candidate paths C which is initially set to be P.

We now describe a step in the binary search that determines whether the target score $\hat{v}$ is greater than a testing value v_0.

(1) For each path $\mathbf{x}$ in C, carry out the following:
   (a) From the scoring function score and the known malice-flipping distance $m_{\mathbf{x}}$, find the smallest reject-flipping distance $r'_{\mathbf{x}}$ such that the score to be associated with $\mathbf{x}$ is greater than v_0. That is, find the smallest $r'_{\mathbf{x}}$ such that $\text{score}(m_{\mathbf{x}} - r'_{\mathbf{x}}) > v_0$.
   (b) Check whether this path attains a score larger than v_0. This is achieved by querying $\mathcal{D}$, with the sample at distance $r'_{\mathbf{x}}$ from the starting point. If the sample is accepted, then the score is greater than v_0.

(2) If the total number of paths attaining score greater than v_0 is more than q_2, then $\hat{S}$ must be larger than v_0. Discard all paths whose score is lower than v_0, and recursively search on the upper half of v_0.

(3) Otherwise, $\hat{S}$ must be smaller than v_0, and all paths with score higher than v_0 must be in $\hat{P}$. Remove them from C and recursively search on the lower half of v_0 for the remaining paths.

The applicability of this reduction depends on the choice of scoring function. It is applicable as long as the scoring function is decreasing with respect to the reject-flipping distance, which is the case for our choices of scoring functions $\texttt{score}_1$ and $\texttt{score}_2$.

Similar to $\texttt{BiRand}$, the binary search is designed to reduce N_d. If queries to the tester are more expensive than the detector's queries, we can swap and apply the binary search on the malice-flipping distances, and thus reducing N_t instead.

Parameters q_1, q_2, Δ. Figure 2 illustrates the searching process. Intuitively, a larger q_1 would improve accuracy of going toward the right direction while a larger q_2 would increase the robustness by recovering from local minimum, and a larger jump factor Δ could reduce the number of iterations in good cases. Nevertheless, excessively large parameters might not increase the effectiveness of the search but rather incurs unnecessarily expensive performance cost. We examine in further details the effect of these parameters on the performance of $\texttt{EvadeHC}$ in our experiments (Section 5).

4 PROBABILISTIC MODELS

In this section, we formalise the intuition behind the design of the proposed $\texttt{EvadeHC}$. Without loss of generality, for searching to make sense, each sample should have a score-value for the underlying "state". We first discuss a mechanism to assign a real-value state (which is also the score) to a sample based on the binary outcomes of $\mathcal{T}$ and $\mathcal{D}$ ("accept" versus "reject" and "malicious" versus "benign"). Next, we propose a hidden state model to capture the notion that the morphing process is seemingly random.

4.1 States Representation

Given a sample x, we have 4 possible states from the binary outcomes of $\mathcal{T}$ and $\mathcal{D}$. A searching strategy generally needs to select the "best" candidate from a given set of samples, and this 4-state representation alone would not provide meaningful information for selection.

Our main idea is to, ideally, assign the probability that a random path (generated by the morpher) starting from x has a reject-flipping sample that is malicious[1].

Evading Probability as the state. Given a starting point x, let M_x be the random variable of the malice-flipping distance on random path. Recall that the random paths arise from sequences of random seeds. Likewise, let R_x and G_x be the random variable of the reject-flipping distance and the gap respectively. It is not necessary that M_x and R_x are independent. Indeed, one would expect that they are highly positively correlated, since the detector attempts to detect

malicious functionality. We shall revisit this later in Section 5. At this points, readers can refer to Figure 11 for an intuition. The figure depicts flipping distances of 500 morphed samples originating from the same malware (Pearson correlation coefficient is 0.34).

For a sample x, let us assign the probability $\Pr(G_x < 0)$ as its state. Recall that a path with negative gap implies that its reject-flipping sample is malicious and thus is an evading sample. In other words, the state of x is the probability that a random path leads to an evading sample. Now, suppose that the adversary has to pick one of two candidates x and y to continue the search, then the candidate with the larger state-value gives higher chance of finding an evading sample. This comparison provides a way for a hill-climbing algorithm to select candidates in each round.

Expected flipping distances as the state. A main drawback of explicitly taking $\Pr(G_x < 0)$ as x's state is the high cost in determining the probabilities during the search process. Although one may estimate the distribution of G_x by sampling multiple random paths, such accurate estimation would require extremely high number of queries, which in turn offsets the gain offered by efficient searching strategies. Alternatively, we can take the expected reject-flipping and malice-flipping distances as the state. Compare to the distribution G_x, the expected distances can be accurately estimated by drawing fewer random paths. In the proposed $\texttt{EvadeHC}$, the measured flipping distances can be viewed as the estimates of the expected distances. From the expected distances, we can derive the expected gap. Intuitively, the smaller the expected gap is, the higher the probability $\Pr(G_x < 0)$ would be. Although the expected flipping distances do not provide sufficient information for us to estimate the probability, we can still employ the expected gap for comparison of probabilities. This motivates the choices of the two scoring functions $\texttt{score}_1$ and $\texttt{score}_2$ proposed in Section 3.3.

4.2 Hidden-State Random Morpher

We consider an abstract model $\texttt{HsrMorpher}$ for analysis. Under $\texttt{HsrMorpher}$, the morpher operates in the same way as the Random Oracle [17] employed in the studies of hash functions does, producing truly random outcomes for unique query, and being consistent with repeated queries. Based on $\texttt{HsrMorpher}$, we give a condition whereby $\texttt{EvadeHC}$ performs poorly and cannot outperform $\texttt{BiRand}$.

Randomly morphed sample. Let us call a sample *seen*, if it has been queried against $\mathcal{T}, \mathcal{D}$ or $\mathcal{M}$, or has been an output of $\mathcal{M}$. In other words, a sample is seen if it has been processed by one of the blackboxes. A sample is *unseen* otherwise. On a query consisting of a sample x and a random seed s, $\mathcal{M}$ first checks whether the query (x, s) has been issued before. If that is not the case, $\mathcal{M}$ randomly and uniformly chooses an unseen sample to be the morphed sample[2]. Otherwise, the previously chosen sample will be outputted. In additional, $\mathcal{M}$ also decides, in a random way, the hidden state of the morphed sample, which is to be described next.

Randomly reducing hidden values. Under $\texttt{HsrMorpher}$, each sample x is associated with a hidden state, represented by two real values (a, b). There is a particular sample x_o with state $(1, 1)$ set to be seen initially, and it is also known to the adversary. Upon receiving

---

[1]An alternative choice is to consider the probability that a random path has an evading sample. However, this choice is not suitable, since an arbitrary long path is likely to have an evading sample.

[2]If the pool of unseen samples is empty, $\mathcal{M}$ simply halts and does not generate a morphed sample.

a query (x, s), $\mathcal{M}$ outputs a morphed sample $\widetilde{x}$ as described in the previous paragraph. If x is unseen, then the morpher sets the state of x to $(0, 0)$ and remembers that. Next, the morpher selects two real values (α, β) from a random source $\mathcal{R}$ and sets the hidden state of $\widetilde{x}$ as

$$(a - \alpha, b - \beta)$$

where (a, b) is the hidden state of x.

The tester $\mathcal{T}$, when given a query x, decides the outcome based on the hidden state. If x is unseen, then $\mathcal{T}$ sets the hidden state to $(0, 0)$. Suppose the hidden state of x is (a, b), then $\mathcal{T}$ outputs *malicious* iff $a > 0$. Likewise, for the detector $\mathcal{D}$, it outputs *reject* iff $b > 0$, and sets the state of x to be $(0, 0)$ if it is unseen.

Note that the only parameter in the HsrMorpher model is the random source $\mathcal{R}$.

4.3 Analysis of EvadeHC on HsrMorpher

HsrMorpher behaves randomly, and is arguably the worst situation the adversary can be in, as the adversary is unable to exploit domain knowledge to manipulating the samples. For instance, given two samples x_0 and x_1, it is not clear how to find a sequence of random seeds that morph x_0 to x_1.

Ineffectiveness of EvadeHC. To understand the performance of EvadeHC, let us give a particular random source $\mathcal{R}$ for HsrMorpher that renders EvadeHC ineffective. Consider a random source $\mathcal{R}$ where α and β are discrete and only take on value 0 or 1. Thus, all possible hidden states are (a, b) where a, b are integers not greater than 1. We can view the samples seen during an execution of EvadeHC as a tree. It is not difficult to show that the conditional probability $\Pr(G_x > 0 \mid state\ of\ x\ is\ (1, 1))$ is the same for all x in this tree. Thus, in every iteration, all candidates have the same probabilities and it is irrelevant which is chosen for the next iteration.

Random Source $\mathcal{R}$. The model HsrMorpher assumes that the reduction of the hidden values is independent of the sample x. To validate that real-life classifiers exhibit such property, in our empirical studies, we treat their internal classification score assigned to a sample as the sample's hidden state. The empirical results demonstrate that the distributions of the reduction are similar over a few selected samples.

Remarks. One can visualise the search for evading sample as a race in reducing the hidden state (a, b). Statistically, a is expected to drop faster than b. The evasion's goal is to find a random path that goes against all odds with b reduces to 0 before a does. Essentially, EvadeHC achieves this in the following way: It generates and instantiates a few random paths. The path with the smallest gap arguably has b reducing faster than average. Hence, EvadeHC chooses a point along this path as the starting point for the next iteration. Figure 8 depicts this process based on an actual trace obtained in our experimentation.

5 EVALUATION

To study the feasibility and effectiveness of our proposed approaches, we conduct experimental evaluation on two well-known PDF malware classifier, namely PDFRate [27] and Hidost [29]. We

defer overview of PDF malwares and the two classifiers to Appendix A. We first describe our experimental setups and then reports the results.

5.1 Experimental Setups

Morpher $\mathcal{M}$. We employ a simple morphing mechanism $\mathcal{M}$ that performs three basic operations which either insert, delete or replace an object in the original PDF file. In order to perform these operations, the morpher $\mathcal{M}$ needs to parse the PDF file into a tree-like structure, to perform the insertion, deletion or replacement on such a structure and finally to repack the modified structure into a variant PDF[3]. We employ a modified version of pdfrw [5] available at [3] for parsing and repacking.

For insertion and replacement, an external object placed into the original file is drawn from a benign PDF file. We collect hundred benign PDF files of size less than 1MB via a Google search to feed into $\mathcal{M}$ as a source of benign objects. They are all accepted by the targeted detectors and confirmed by the tester $\mathcal{T}$ as not bearing any malicious behaviours.

Tester $\mathcal{T}$. The tester $\mathcal{T}$ is implemented using a malware analysis system called Cuckoo sandbox [1]. The sandbox opens a submitted PDF file in a virtual machine and captures its behaviours, especially the network APIs and their parameters. The PDF file is considered malicious if its behaviours involve particular HTTP URL requests and APIs traces. We follow previous work [34] in selecting the HTTP URL requests and APIs traces as reliable signature in detecting malicious behaviours of the submitted PDF files.

Dataset. We conduct the experimental studies using a dataset which consists of 500 malicious PDF files that had been selected in previous work [34]. These samples are drawn from the Contagio [6] dataset, satisfying the following three conditions. First, they exhibit malicious behaviours observed by the tester. Secondly, they have to be compatible with the morphing mechanism (i.e. can be correctly repacked by pdfrw). And lastly, they are all rejected by both targeted detectors. We shall refer to these as malware seeds hereafter.

Targeted detectors. The targeted detectors in our experiments are based on the two PDF malware classifiers, namely PDFRate and Hidost. We retrain the classifiers following the open-source implementations made available by previous works [29, 30]. The training data of each classifier consists of $5,000$ benign and $5,000$ malicious PDF files. These files are randomly selected from a subset of the Contagio dataset which excludes the 500 malware seeds that are already picked for the experimental dataset described above. This is to account for a restriction that the adversary does not have access to the targeted classifier's training data. In addition, we make small changes to the classifiers such that the final outputs are binary decisions, rather than real-value scores.

---

[3]To avoid redundant parsing of the same PDF file in multiple steps, $\mathcal{M}$ caches the modified tree structure (or the original tree structure of the malicious PDF file), and directly modifies it without having to parse the PDF file in each step.
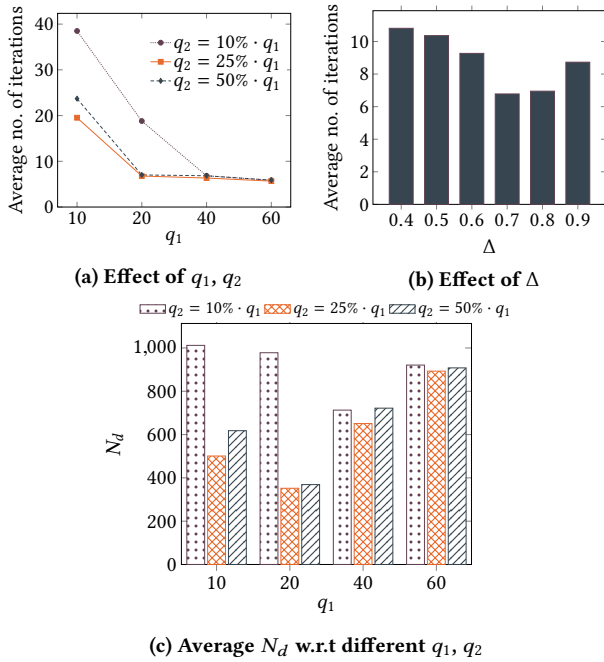
(a) Effect of q_1, q_2

(b) Effect of Δ

(c) Average N_d w.r.t different q_1, q_2

Figure 3: Effect of q_1, q_2 and Δ on performance of EvadeHC

Scoring function. The score function adopted in the experimental studies is $\mathrm{score}_1(m, r) = m - r$, where m is the malice-flipping distance and r is the reject-flipping distance.

Machine and other parameters. All experiments are conducted on Ubuntu 14.04 commodity machines equipped with Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz processors and 64GB of physical memory[4]. The evasion process can be run in parallel, as there are no dependencies between different executions. In our experiments, we dedicate one machine to run $\mathcal{D}$ and $\mathcal{M}$, and nine others to run the tester $\mathcal{T}$. Each of these nine machines deploys 24 virtual machines simultaneously, and is capable of performing approximately 2,000 tester queries per hour.

We conducted five sets of experiments. The first set of experiments examines how the parameters q_1, q_2 and Δ effect the performance of EvadeHC. The second and third experiment sets evaluate the effectiveness of the proposed approaches in evading PDFRATE and Hidost detectors, respectively. The efficiency of the evasion is determined by number of blackboxes queries and the overall running time. In the forth set of experiments, we consider a hypothetical setting where the detectors are hardened to make evasion more difficult and benchmark our approaches against the closely related work which relies on classification scores output by the detectors to guide the evasion [34]. We emulate the hardening of detectors by reducing their default mailware classification thresholds: below 0.5 for PDFRATE and below 0 for Hidost. The last set of experiments validates the HsrMorpher model we discussed earlier in Section 4. Unless otherwise stated, the reported results are average values obtained over 10 instances.

---
[4]NCL Testbed. https://ncl.sg/testbedInformation
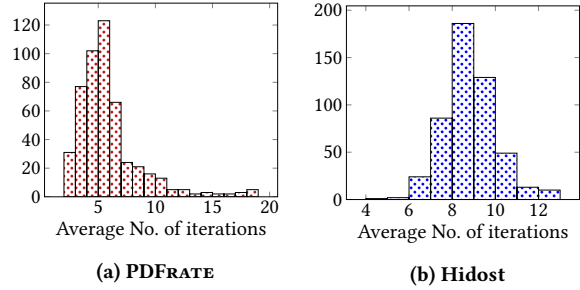

(a) PDFRATE

(b) Hidost

Figure 4: Histogram of average numbers of iterations EvadeHC needs in finding evading samples against the two detectors.

5.2 Effect of Parameter Settings on EvadeHC

We first examine how the choice of parameters q_1, q_2 and Δ effects the performance of EvadeHC. We run EvadeHC with different parameter settings on 100 malware seeds randomly selected from our experimental dataset against PDFRATE detector, varying q_1 from 10 to 60, and setting q_2 proportional to q_1 with a percentage ranging from 10% to 50%. In addition, we vary the jump factor Δ from 0.4 to 0.9. The average numbers of iterations and amounts of detector queries required to find evading samples for each of the 100 malware seeds are reported in Figure 3.

Figure 3a indicates that increasing q_1 will lessen the number of iterations. This is consistent with an intuition we discussed earlier, for larger q_1 would improve a likelihood that the algorithm is going toward the right direction. $q2$, on the other hands, has a more subtle effect on the number of iterations. In particular, a too small value may lead the search to a local minimum, hindering the search progress, while a too large value may accidentally bring "bad" candidates onto the next iterations. A similar trend is also observed on Δ (Figure 3b). When it is increased from 0.4 to 0.7, we witness a reduction in number of iterations. However, when it reaches 0.9, the algorithm tends to loop through more iterations in order to find evading samples. The reason is that too small Δ limits progress the search can make in each iteration, while unnecessarily large Δ would increase the likelihood of samples in the next iterations being rejected.

It is worth noting that the number of detector queries required to find evading samples depends not only on the number of iterations, but also the value of q_1. Figure 3c shows average number of detector queries with respect to different settings of q_1 and q_2. While larger q_1 leads to smaller number of iterations, it may result in larger number of detector queries per each iterations, and thus larger queries overall (e.g., with $q_2 = 25\% \cdot q_1$, increasing q_1 from 20 to 60 leads to an increase in N_d). From the result of this experiment set, we choose $q_1 = 20$, $q_2 = 5$ and $\Delta = 0.75$ as the parameter setting for EvadeHC through all other sets of experiments.

5.3 Evading PDFRATE Detector

The second experiment set focuses on evading PDFRATE detector. All of our methods achieve 100% evasion rates (i.e., found evading samples for all 500 malware seeds in our dataset). While the baseline SeqRand and BiRand traverse 1051 random paths on average to find
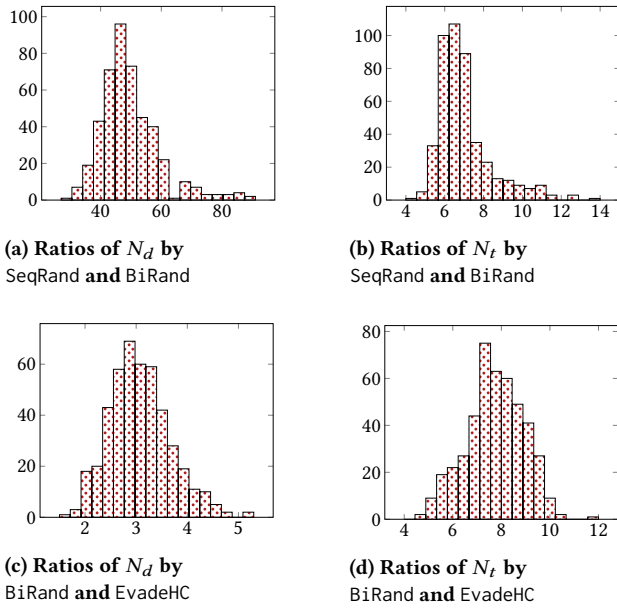
(a) Ratios of N_d by SeqRand **and** BiRand

(b) Ratios of N_t by SeqRand **and** BiRand

(c) Ratios of N_d by BiRand **and** EvadeHC

(d) Ratios of N_t by BiRand **and** EvadeHC

Figure 5: Performance comparison of SeqRand, BiRand **and** EvadeHC **in evading PDFRATE.**

(a) Ratios of N_d by SeqRand **and** BiRand

(b) Ratios of N_t by SeqRand **and** BiRand

(c) Ratios of N_d by BiRand **and** EvadeHC

(d) Ratios of N_t by BiRand **and** EvadeHC

Figure 6: Performance comparison of SeqRand, BiRand **and** EvadeHC **in evading Hidost.**

an evading sample, EvadeHC only needs from three to six iterations (each iteration involves assessing 20 random paths). Nevertheless, there are a few exceptions which require EvadeHC to take up to 19 iterations to find evading samples (Figure 4a).

We first compare the performance of BiRand to the baseline SeqRand. Figure 5a shows the ratios of their N_d, and Figure 5b plots that of N_t, clearly demonstrating the effectiveness of the binary search mechanism employed in BiRand. Compared to the baseline, BiRand could reduce N_d by upto 94 times.

Next, we benchmark EvadeHC against BiRand. Figure 5c report the ratios between their numbers of detector queries, and Figure 5d depicts ratios of their tester queries. BiRand typically demands from 2 to 4 times more detector queries, and 6 to 10 times more tester queries than EvadeHC does. More details in numbers of black-boxes queries required by EvadeHC and BiRand are reported in Appendix B.

Compared to the baseline SeqRand, EvadeHC demands upto 450 times fewer detector queries and 148 times fewer tester queries, which translates to two orders of magnitude faster running times. We report running times of different approaches in Section 5.5.

5.4 Evading Hidost Detector

In the third set of experiments, we evaluate our proposed approaches against Hidost detector. They also achieve 100% evasion rate for the 500 malware seeds in the dataset. For 95% of the seeds, EvadeHC found the evading samples within 11 iterations, while the other seeds need up to 13 iterations (Figure 4b). BiRand and SeqRand, on the other hand, have to traverse approximately 1553 paths on average to find an evading sample.
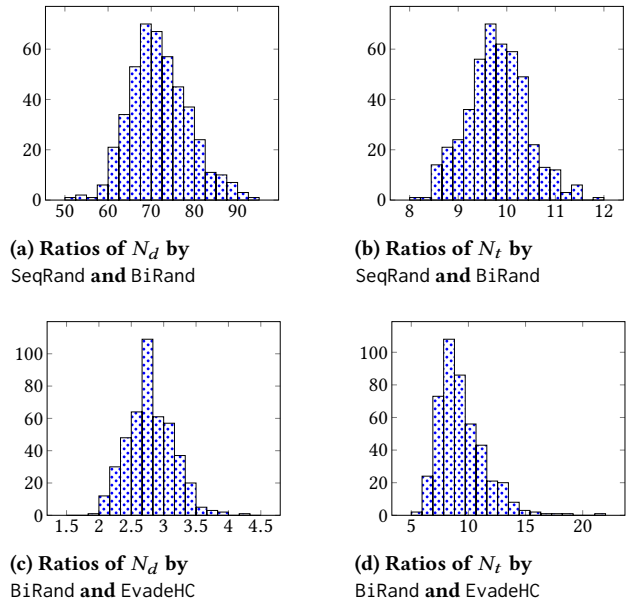
We report performance of SeqRand in comparison with BiRand in Figure 6a and 6b, showing that BiRand is capable of outperforming SeqRand by upto 92 and 12 times in term of detector and tester queries, respectively.

Further, we also benchmark BiRand against EvadeHC by showing the ratios between their N_d (Figures 6c) and N_t (Figure 6d). The results show the superiority of EvadeHC over BiRand. In particular, for most malware seeds, EvadeHC requires two to three times fewer detector queries and seven to ten times fewer tester queries compared to BiRand. More details in numbers of blackboxes queries required by the two approaches are reported in Appendix C.

When benchmarked against the baseline solution, EvadeHC attains upto two orders of magnitude lower execution cost, both in term of number of queries to the blackboxes and the overall running time.

5.5 Execution Cost

Figure 7 reports average running time of different approaches in evading the two detectors. As discussed earlier, EvadeHC and BiRand are order of magnitude more efficient than the baseline solution SeqRand. In particular, to find an evading sample against Hidost, SeqRand takes on average 6.7 hours, while BiRand and EvadeHC need only 40 and 5 minutes, respectively.

It can also be seen that evading Hidost is often more expensive than evading PDFRATE. It would be interesting to investigate how the construct of Hidost provides resilience against the attacks.

We remark that BiRand and EvadeHC are designed with a premise that N_d is a dominating component in determining the evasion cost, thus minimising the number of detector queries, rather than tester queries which are computationally expensive. In situations where N_t is the dominating component (e.g., applications for which
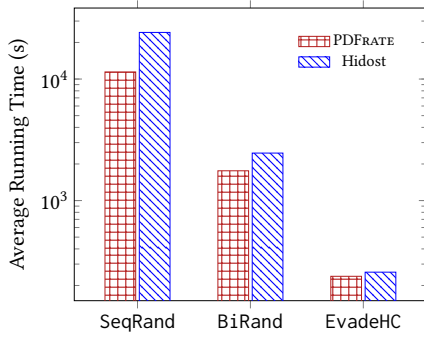
Figure 7: Average running time (in seconds) required to find an evading sample of different approaches. The results are taken over 5000 **evasions,** 10 **for each of the malware seeds in our dataset**
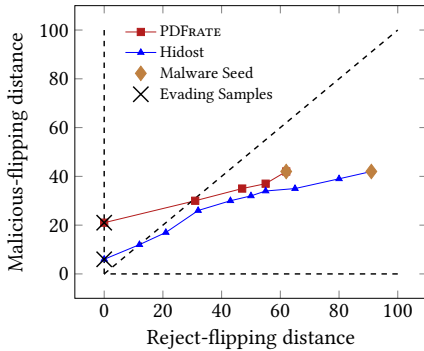

Figure 8: Typical evading traces by EvadeHC **against the two targeted detectors. A diagonal dashed line represents samples whose flipping distances are equal. Samples chosen in successive iterations usually move "closer" to the line.**

computational cost is the main constrain), one can easily derive a variant of EvadeHC that applies the single binary search with $\mathcal{T}$ instead of $\mathcal{D}$. The performance of such algorithm is arguably similar to that of our current implementation, except for N_d and N_t whose values would be swapped. There is no impact on the accuracy, since the modification is purely on the algorithmic aspect.

5.6 Evading Trace Analysis

To gain a better insight on how the hill-climbing heuristic works in EvadeHC, we examine the flipping distances of samples generated along typical evading traces. An evading trace is a collection of samples that EvadeHC successively generates in finding an evading sample for a malware seed. Such a trace starts with a malware seed, and ends with an evading sample. Samples generated in the first iteration would likely have reject-flipping distance larger than malice-flipping distance, otherwise evading samples have been found. EvadeHC would select promising samples from one iteration and continue to morph them in the next iteration so that they are eventually accepted by the detector before losing their malicious functionality.
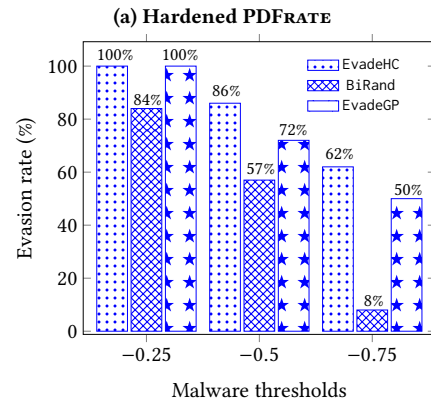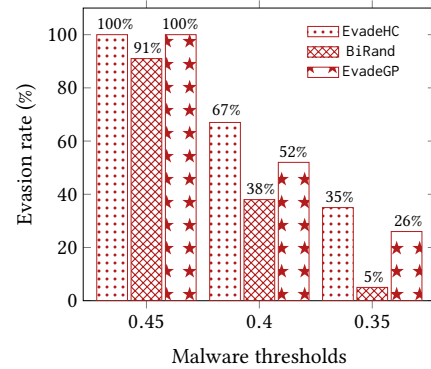

(a) Hardened PDFRATE

(b) Hardened Hidost

Figure 9: Evasion rates of different approaches (displayed on top of the bars) against hardened detectors.

Figure 8 depicts typical evading traces that lead to evading samples against PDFRATE and Hidost detectors. The diagonal line in the figure represents points where malice-flipping distance and reject-flipping distance are equal, the vertical line represents points at which the detector's decision changes from reject to accept, and the horizontal line represents points where the malicious functionality of the samples is lost. Intuitively, the malware seeds' representations in term of malice-flipping distance and reject-flipping distance often lie below the diagonal line. EvadeHC's goal is to morph them in such a way that they move past the diagonal line, reaching the vertical line (i.e., being accepted) before crossing the horizontal line (i.e., losing the malicious functionality). It can also be seen from the figure moving the samples generated against PDFRATE pass the diagonal is easier compared to those that are evaluated against Hidost, suggesting that PDFRATE is more evadable.

5.7 Robustness against Hardened Detectors

In the forth experiment set, we investigate scenarios where the detectors are hardened to render evasions more difficult. We emulate hardened PDFRATE by lowering its malware threshold to 0.35 [5], and hardened Hidost's to −0.75. Furthermore, we bound the maximum number of detector queries that could be issued in finding

---

[5] Authors of PDFRATE [27] reported that adjusting the malware threshold from 0.2 to 0.8 has negligible effect on accuracy.

an evading sample to $2,500$. If an evading sample can not be found after the predefined number of detector queries, we treat the seed as non-evadable.

In addition, we benchmark the two approaches `BiRand` and `EvadeHC` against a technique by Xu et al. [34], which we shall refer to as `EvadeGP`. We stress that while `EvadeGP` relies on real-value classification scores assigned to the samples to guide the search, our approaches do not assume such auxiliary information. We operate `EvadeGP` under a similar set of parameters reported in [34]: population size is 48, mutation rate is 0.1 and fitness stop threshold is 0.0. We bound the number of generations that `EvadeGP` traverses to 50 (instead of 20 as in [34]), effectively limiting the number of detector queries incurred in each evasion to $2,500$. We treat each malware seed independently, and do not make use of mutation traces that work for one malware in searching for evading samples of others. This is to account for an extremely restricted setting in which the adversary starts out an evasion with only one specially crafted malware seed[6].

Figure 9 reports evasion rates of `BiRand` and `EvadeHC` in comparison with that of `EvadeGP` against the hardened detectors. For PDFRATE detector with malware threshold set to 0.45, both `EvadeHC` and `EvadeGP` attained 100% evasion rate on our dataset, while `BiRand` only achieves 91%. When the malware threshold is further lowered to 0.4 and 0.35, the evasion rate of `EvadeHC` decreases to 68% and 35%, respectively, while `EvadeGP` and `BiRand` witness more significant drops in their evasion rates, attaining evasion rate of as low as 5% for `BiRand` and 26% for `EvadeGP`. Similarly, when the malware threshold of Hidost is reduced to -0.75, `BiRand` and `EvadeGP` could only find evading samples for 8% and 50% of the malware seeds respectively, while `EvadeHC` still attains an evasion rate of 62% (Figure 9b).

While it may come as a surprise that `EvadeHC` attains better evasion rate than `EvadeGP` even though it only assumes binary output instead of classification score as `EvadeGP` does, we contend that this result is in fact explicable. We believe the reasons are two folds. First, `EvadeHC`'s scoring mechanism is capable of incorporating information obtained from both the tester and detector and thus is arguably more informative than `EvadeGP`'s scoring mechanism which mainly relies on the classification score output by the detector. Secondly, classification scores are not a reliable representation of the samples' maliciousness, evidenced by the fact that an adversary can trick the classifiers to assign "benign scores" to evading samples.

5.8 Validating the Hidden-state Morpher Model

To justify the proposition on states representation of a sample discussed earlier in Section 4, we generate 200 different morphed samples from the same malware seed, and measure their average flipping distances (the reject-flipping distance is measured against Hidost). The average flipping distances of each sample are computed from 20 different random paths originating from it (the sequences of random seeds are the same for all samples), and plotted in Figure 11. The figure also shows a certain correlation between the
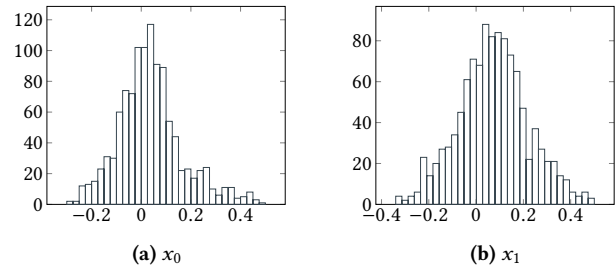
---
[6]We note that our approaches can trivially support the use of trace replay feature suggested in [34].

Figure 10: Effect of morphing on PDFRATE score. The histograms reported changes in classification scores after 5 morphing steps.
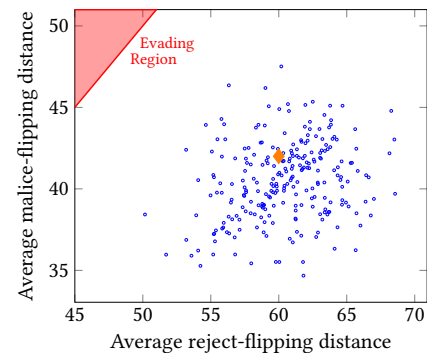

Figure 11: Average flipping distances (with Hidost as the detector) of 200 **morphed samples originating from the same malware seed (depicted by the diamond). The malice-flipping distance of the chosen malware seed is** 42, **and its reject-flipping distance is** 60. **The pearson correlation coefficient of the distances is 0.34.**

malice and reject flipping distances. Indeed, the pearson correlation coefficient of the distances is 0.34, confirming our proposition earlier in Section 4 that they are positively correlated.

The proposed model `HsrMorpher` assumes that the reduction of the hidden values is independent and identical for all the samples. To validate that real-life classifiers exhibit such property, we consider PDFRATE and treat the internal classification score assigned to a sample as it's hidden state. We pick a sample x_0 and create another sample x_1 by applying a sequence of five morphing steps on x_0. We then generate $1,000$ random paths of length five from x_0. For each of the random path, we record the difference between classification scores of the last sample and x_0. Figure 10a shows the histogram of the recorded score differences. Similar experiment is carried out on x_1 and the histogram is shown in Figure 10b. The resemblance of the two histograms gives evidences to suggest that `HsrMoprher` is appropriate.

We further study the validity of the `HsrMorpher` model by running `BiRand` and `EvadeHC` under the abstract model. Recall that `HsrMorpher` is parameterised by the random source S that dictates how the two values (α, β) are chosen. From previous set of experiments, we observe that a typical malice-flipping distance is 42, and
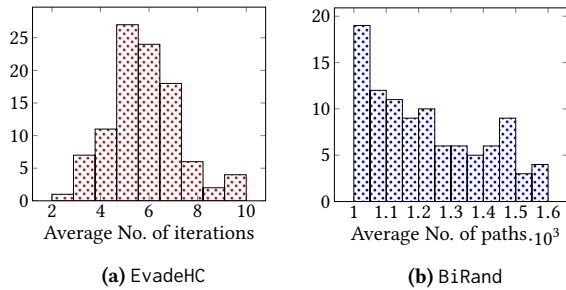
(a) EvadeHC

(b) BiRand

Figure 12: Histogram of average number of iterations and paths that EvadeHC and BiRand needs in finding evading samples under the abstract model.

reject-flipping distance is 60 (Figure 11). A more detailed analysis of the empirical data suggests that, with respect to the three blackboxes deployed in our experiments, α would follow normal distribution $\mathcal{N}(0.024, 0.01)$ and β follows $\mathcal{N}(0.017, 0.011)$. The evasion starts with a malicious sample x_o with state $(1, 1)$ and succeeds if it can find an evading sample e with state $(a, 0)$ for some $a > 0$. We repeat the experiment for 100 times.

Due to space constraint, we only show the average number of iterations (or paths) that the two approaches traverse in searching for evading samples under the HsrMorpher model (Figure 12). The experimental results confirm the consistency of their behaviors under HsrMorpher and the empirical studies. This strongly indicates that the abstract model can indeed serve as a basis to study and analyze the proposed evasion mechanism.

6 DISCUSSION

In this section, we discuss potential mitigation strategies and implications suggested by our evasion attacks.

6.1 Existing Defensive Mechanism

Most existing works in evading classifiers rely on real-value classification scores to guide the evasion [21, 34]. As such, it has been suggested that hiding the classification scores or adding random noise to the scores may protect the classifier [8]. Another proposed defensive mechanism is to use a multi-classifier system whose classification score is randomly picked from different models trained with disjoint feature sets [10]. Our results show that it is still feasible to evade learning-based classifiers without any classification score, rendering the above-mentioned defensive mechanisms ineffective.

6.2 Potential Mitigation Strategies

Hardening the Malware Threshold. Previous work [27, 34] suggested that the classification performance (i.e., false accept/false reject) is typically robust against the choice of the malware threshold, for original samples that are not adversarially manipulated would have classification scores close to either of the extremes. For example, the authors of PDFRATE reported that adjusting the malware threshold from 0.2 to 0.8 has negligible effect on accuracy, because most samples would be assigned scores very close to either 0 or 1 [27]. On the other hand, our experimental study (Section 5.7) suggests that even a slight change of the malware threshold could

have significant impact on the evasion difficulties. Thus, one potential mitigation strategy would be to set the threshold to be more "restrictive". This is worth investigating in future works.

Randomization. Another potential mitigation strategy is to embed into the classifiers a certain level of randomness. In particular, for samples whose classification scores are very close to the threshold, the classifiers can flip its classification decision with some probability. Given a proposition that most samples would have classification scores distant from the threshold, the above-mentioned flipping mechanism would not have significant impact on the classification accuracy. Similar to the previous mitigation strategy, more study is needed to conclude an effect of this countermeasure.

Identifying Evading Attempts/Samples. In most cases, evading samples are found after hundreds of queries to the detector $\mathcal{D}$. In addition, the samples queried against $\mathcal{D}$ over the course of the evasion would resemble one another, to a certain extent. Thus, one possible method to detect evading attempts/samples is to have $\mathcal{D}$ remember samples that have been queried and apply some similarity/clustering-based techniques to identify evading queries/samples. In other words, $\mathcal{D}$ has to remain stateful.

6.3 Evasion for Defense

Ironically, an effective evasion algorithm can also be used to build more secure classifiers that are robust against evasion attacks. Indeed, previous works have suggested that one can enhance the robustness of a learning based model by injecting adversarial examples throughout its training [14, 21]. In particular, those morphed samples that retain the desired malicious activity, especially the ones that are misclassified by the learning based systems, can be included in the trainning dataset for the next step of training. We remark that by formulating the morphing process as random but repeatable, our approaches are capable of generating adversarial examples with great diversity, further enhancing effectiveness of the adversarial training.

7 RELATED WORK

Evasion attacks against detectors have been studied in numerous contexts, ranging from watermark [20] to spammers detection [35]. Šrndic et al. [30] proposed a taxonomy of evasion scenarios based on the knowledge about the targeted detector's components that are available to the adversary. These components include *training datasets* and *classification algorithms* together with their parameters, as well as a *feature set* that the classifier has learnt. Intuitively, detailed knowledge of the detector significantly benefits the adversary in conducting evasion attacks.

Various attacks against learning-based classifiers [9, 24, 30] assume that the adversary has high level of knowledge about the target system's internals (e.g., feature sets and classification algorithms), and could directly manipulate the malicious seeds in the feature space. Our work, on the other hand, investigates a restricted adversarial model which does not assume any of such knowledge.

Papernot et al. [21] considered blackbox evasion attacks against machine learning. Their techniques involve reverse-engineering

the targeted model. In particular, the adversary has to train a substituting model that behaves somewhat similar to the targeted system, then search for evading samples against such substituting model. They show that those evading samples are also misclassified by the targeted system. Clearly, this approach heavily relies on the transferability of the adversarial samples. Our solutions, on the other hand, directly search for evading samples against the targeted classifier without necessitating training a substituting model or making any assumption on the transferability of samples. If we view the reverse-engineering approach as a method determining "global" parameters of the classifier, then in contrast, our approach attains efficiency by searching within "localised" neighbourhood of a given malicious sample.

A line of works on adversarial learning [25, 32] also assume blackbox accesses to the targeted systems, but are different from ours in their adversarial goals. While Tramèr et al. [32] attempted to extract exact value of each model parameter and Shokri et al. [25] were interested in inferring if a data record was in the targeted model's training dataset, our focus is on deceiving the target system into misclassifying evading samples.

The most closely related to our proposed technique is the study by Xu et al. [34]. The main difference distinguishing our technique from the previous work lies in the assumption on the availability of classification scores output by the targeted detector. While previous work relies on such scores to make decisions, EvadeHC assumes binary outputs from the detector, and combines such limited information with binary output of the tester to guide the evasion.

8 CONCLUSION

We have proposed a restricted adversary model for classifier evasion with only blackbox accesses. Such restricted setting allows us to investigate generic evasion attacks that utilise limited information on the domain knowledge and the classifier's specifics (e.g. model internals, and training dataset). On the one hand, this model provides the defender with a baseline of the adversary's capability. On the other hand, the adversary could attain more effective evasion attacks by incorporating domain knowledge into the generic attacks.

We have described EvadeHC, a generic hill-climbing based approach for evading binary-outcome detector. Our technique assumes minimal knowledge about both the detector and the data manipulation mechanism. The effectiveness of the proposed approach has been demonstrated by experimental studies conducted against two PDF malware classifiers. While the experiments are performed on malware classifiers, the proposed technique and its security implications may also be of wider application to other learning-based systems.

ACKNOWLEDGEMENTS

REFERENCES

[1] Claudio Guarnieri, Alessandro Tanasi, Jurriaan Bremer, and Mark Schloesser. Cuckoo Sandbox: A Malware Analysis System. . http://www.cuckoosandbox.org/.
[2] CVE Details. Adobe Acrobat Reader: Vulnerability Statistics. http://www.cvedetails.com/product/497/.
[3] Modified pdfrw. https://github.com/mzweilin/pdfrw.
[4] Nedim Šrndić and Pavel Laskov. Mimicus: A Library for Adversarial Classifier Evasion. https://github.com/srndic/mimicus.
[5] Patrick Maupin. PDFRW: A Pure Python Library That Reads and Writes PDFs. https://github.com/pmaupin/pdfrw.
[6] Stephan Chenette. Malicious Documents Archive for Signature Testing and Research - Contagio Malware Dump. http://contagiodump.blogspot.sg/2010/08/.
[7] Marco Barreno, Blaine Nelson, Anthony D Joseph, and JD Tygar. 2010. The security of machine learning. *Machine Learning* (2010).
[8] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. 2006. Can machine learning be secure?. In *ASIACCS*.
[9] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *ECML-PKDD*.
[10] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2009. Multiple classifier systems for adversarial classification tasks. In *MCS*.
[11] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389* (2012).
[12] Michael Brückner, Christian Kanzow, and Tobias Scheffer. 2012. Static prediction games for adversarial learning problems. *Journal of Machine Learning Research* (2012).
[13] Marco Cova, Christopher Kruegel, and Giovanni Vigna. 2010. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *WWW*.
[14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).
[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*.
[16] Adobe Systems Incorporated. 2006. PDF Reference, Sixth edition, version 1.23. (2006).
[17] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to modern cryptography*. CRC Press.
[18] Pavel Laskov and Nedim Šrndić. 2011. Static detection of malicious JavaScript-bearing PDF documents. In *ACSAC*.
[19] Kyumin Lee, James Caverlee, and Steve Webb. 2010. Uncovering social spammers: social honeypots+ machine learning. In *SIGIR*.
[20] Qiming Li and Ee-Chien Chang. 2002. Security of public watermarking schemes for binary sequences. In *Information Hiding*.
[21] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *ASIACCS*.
[22] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security* (2011).
[23] Karthik Selvaraj and Nino Fred Gutierres. The rise of PDF malware.
[24] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. 2016. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *CCS*.
[25] Reza Shokri, Marco Stronati, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *IEEE S&P*.
[26] David Silver et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* (2016).
[27] Charles Smutz and Angelos Stavrou. 2012. Malicious PDF detection using metadata and structural features. In *ACSAC*.
[28] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE S&P*.
[29] Nedim Šrndić and Pavel Laskov. 2013. Detection of malicious pdf files based on hierarchical document structure. In *NDSS*.
[30] Nedim Šrndić and Pavel Laskov. 2014. Practical evasion of a learning-based classifier: A case study. In *IEEE S&P*.
[31] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*.
[32] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *USENIX Security*.
[33] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *NIPS*.
[34] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically evading classifiers. In *NDSS*.
[35] Chao Yang, Robert Harkreader, and Guofei Gu. 2013. Empirical evaluation and new design for fighting evolving Twitter spammers. *IEEE TIFS* (2013).

Header	%PDF-1.5
Body	1 0 obj <</Count 1/Kids[7 0 R]/Type/Pages >>endobj · · · 21 0 obj null endobj
CRT	xref 0 22 0000000002 65535 f · · · · · · 0000000394 00000 n
Trailer	trailer · · · · · · startxref 1637 %%EOF

Figure 13: An example of a PDF file structure

A   PDF MALWARE CLASSIFIERS

In this section, we give an overview of the Portable Document Format (PDF) and PDF malwares, and briefly introduce the two detectors, namely PDFRate [27] and Hidost [29].

A.1   PDF

The Portable Document Format (PDF) is a standardised file format that is designed to decouple the document presentation from the underlying environment platform (e.g., application software, hardware or even OSes), thus enabling consistent presentation of the document across different platforms [16]. A typical PDF file consists of four parts. The first part – *header* – contains the magic number and the version of the format. The second part – *body* – incorporates a set of PDF objects comprising the content of the file. The third part – *cross-reference table* (CRT) – is an index table that gives the byte offset of the objects in the body. The last part – *trailer* – includes references to the CRT and other special objects such as the root subject. We depict an example of a PDF file in Figure 13.

The header, CRT and trailer are introduced with %PDF, xref and trailer keywords, respectively. Objects in the body of the file may be either *direct* (those that are embedded in another object) or *indirect*. The indirect objects are numbered with a pair of integer identifiers and defined between two keywords obj and endobj. Objects have eight basic types which are Booleans (representing *true* or *false*), numbers, strings (containing 8-bit characters), names, arrays (ordered collections of objects), dictionaries (sets of objects indexed by names), streams (containing large amounts of data, which can be compressed) and the null objects. It is worth mentioning that there are some dictionaries associated with special meanings, such as those whose type is JavaScript (containing executable JavaScript code).

A.2   PDF Malwares

Due to its popularity, PDF files have been extensively exploited to deliver malware. In addition, given an increasing number of vulnerabilities in Acrobat readers reported recently [2], the threat of PDF malwares is clearly relevant. The malicious payploads embedded in the PDFs are often contained in JavaScript objects or other objects that could exploit vulnerabilities of a particular PDF reader in use.

A.3   Detectors

Various PDF malware detectors have been proposed in the literatures. A line of works [13, 18] targeted JavaScript code embedded in the malicious PDFs (or PDF malware). They first extract a JavaScript code from the PDF, and either dynamically or statically analyse such a code to assess the maliciousness of the PDF. Nevertheless, these works would fail to detect PDF malware wherein the malicious payloads are not embedded in JavaScript code or the code itself is hidden [23]. Hence, recent works have taken another approach, relying on the structural features of the PDFs, to perform static detection. Structural feature-based detectors perform detection based on an assumption that there exist differences between internal object structures of benign and malicious PDF files.

Our experimental evaluations are conducted against two state-of-the-art structural feature-based detectors, namely PDFRate [27] and Hidost [29]. These two systems are reported to have very high detection rate on their testing datasets, and are also studied in previous works on evading detection [34]. It is worth mentioning that the outputs of these two detectors are real-value scores which are to be compared against thresholds to derive detection results. In our experiments, we make small changes to their original implementations so that they return binary outputs.

PDFRate. PDFRate is an ensemble classifier consisting of a large number of classification trees. Each classification tree is trained using a random subset of the training data and based on an independent subset of features. These features include object keywords, PDF metadata such as author or creation data of the PDF file, and several properties of objects such as lengths or positions. At the time of classification, each tree outputs a binary decision indicating its prediction on the maliciousness of the input. The output of the ensemble classifier is the fraction of trees that consider the input "malicious" (a real-value score raning from 0 to 1). The default cutoff value is set at 0.5.

PDFRate was trained using $5,000$ benign and $5,000$ malicious PDF files randomly selected from the Contagio dataset [6]. The classifier consists of $1,000$ classification trees each of which covers 43 features. The total number of features covered by all the classification trees is 202 (but only 135 are documented in PDFRate's documentation). An open-source implementation of PDFRate, namely Mimicus [4], is by Šrndić et al. [30]. We utilise this implementation in our experiments.

Hidost. Hidost is a support vector machine (SVM) classifier [29]. SVM aims to fit a hyperplane (which can be expressed using a small number of support vectors) to training data in such a manner that data points of both classes are separated with the largest possible margin. Hidost works by first mapping a data point (representing a submitted PDF file) to an indefinite dimensional space using radial
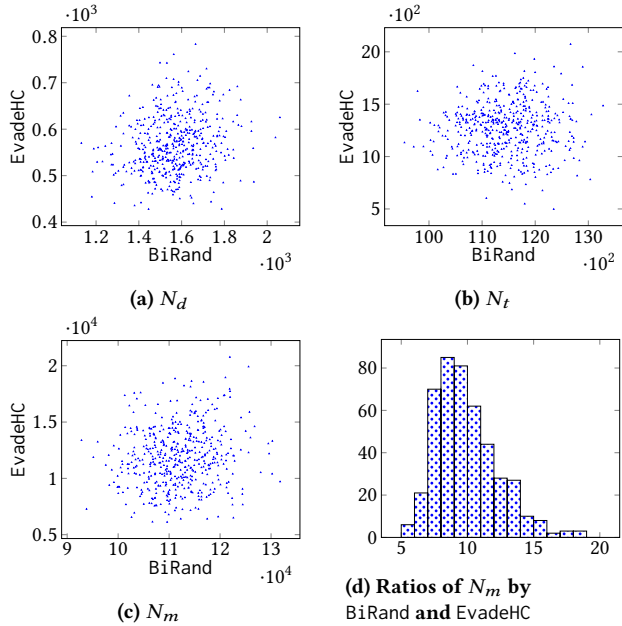
Figure 15: Average number of blackbox queries required in evading Hidost
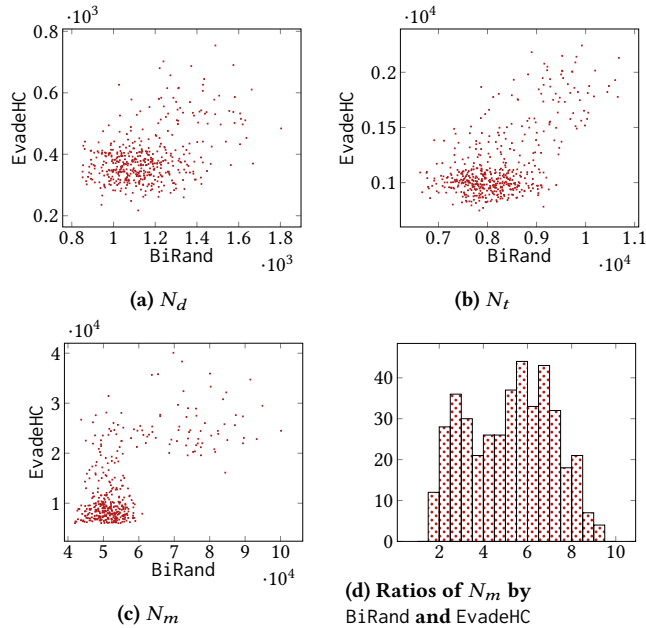

Figure 14: Average number of blackbox queries required in evading PDFRATE

basis function, and reports the distance between the data point and the hyperplane as a measurement of its maliciousness. If the distance is positive, the PDF file is considered malicious; otherwise, the file is flagged as benign.

Hidost was trained using $5,000$ benign and $5,000$ malicious PDF files. Hidost operates based on $6,087$ classification features, which are structural paths of objects. These structural paths are selected from a set of $658,763$ PDF files (including both benign and malicious instances) based on their popularity (i.e., each of them appeared in at least $1,000$ files). We use the implementation made available by the author of Hidost in our experiments [29].

B NUMBER OF BLACKBOX QUERIES IN EVADING PDFRATE

The number of blackbox queries that BiRand and EvadeHC required in evading PDFRATE are reported in Figures 14a, 14b, 14c. We do not report these metrics of the baseline, for they are all equal to the number of morphing steps that BiRand incurs.

For a majority of the malware seeds, EvadeHC needs at most 494 queries, while BiRand requires approximately 1500. The remaining seeds are more difficult to evade, for which EvadeHC and BiRand need up to 786 and 1837 queries, respectively. Similarly, EvadeHC also requires fewer tester queries and less morphing efforts than BiRand (Figures 14b and 14c). In particular, for the majority of the malware seeds, EvadeHC can find an evading sample with less than $1,237$ tester queries and $21,707$ morphing steps, while BiRand consumes up to $9,100$ and $63,288$ morphing steps.

To get an insight why some seeds necessitated much more effort in finding evading samples, we check their classification scores given by PDFRATE. It comes as no surprise that their classification scores are higher than the rest of the malware seeds. In order words, it is harder to find evading samples for these seeds because the detector perceived their maliciousness more clearly.

The histogram of the ratios between the numbers of morphing steps required by the two approaches is depicted in Figure 14d. EvadeHC requires as low as one tenth morphing efforts compared to BiRand in order to find an evading sample.

C NUMBER OF BLACKBOX QUERIES IN EVADING HIDOST

We report the amounts of blackbox queries EvadeHC and BiRand incur in evading Hidost in Figure 15a, 15b and 15c. Overall, EvadeHC outperforms BiRand with respect to all three metrics N_d, N_t and N_m. In particular, EvadeHC's requires as few as 427 detector queries, while BiRand needs as least $1,131$ queries to find an evading sample. Regarding the tester, EvadeHC requires no more than 2073 queries, while BiRand requires $11,500$ queries on average. The similar trend is also observed on morphing effort, wherein EvadeHC requires approximately 10 times fewer morphing steps compared to BiRand (Figure 15d).