MiniMal: Hard-Label Adversarial Attack Against Static Malware Detection with Minimal Perturbation

Chengyi Li[1*], **Zhiyuan Jiang**[1*†], **Yongjun Wang**[1], **Tian Xia**[1], **Yayuan Zhang**[1] and **Yuhang Mao**[2]

[1]College of Computer Science and Technology, National University of Defense Technology, Changsha, China

[2]University of Southern California

{lichengyi, jzy, wangyongjun, xiatian19, zhangyayuan}@nudt.edu.cn, yuhangma@usc.edu

Abstract

Static malware detectors based on machine learning are integral to contemporary antivirus systems, but they are vulnerable to adversarial attacks. While existing research has demonstrated success with adversarial attacks in black-box hard-label scenarios, challenges such as high perturbation rates and incomplete retention of functional integrity remain. To address these issues, we propose a novel black-box hard-label attack method, MiniMal. MiniMal begins with initialized adversarial examples and utilizes binary search and particle swarm optimization algorithms to streamline the perturbation content, significantly reducing the perturbation rate of the adversarial examples. Furthermore, we propose a functionality verification method grounded in file format parsing and control flow graph comparisons to ensure the functional integrity of the adversarial examples. Experimental results indicate that MiniMal achieves an attack success rate of over 98% against three leading machine learning detectors, improving performance by approximately 4.8% to 7.1% compared to state-of-the-art methods. MiniMal reduces perturbation rates to below 40%, making them 9 to 11 times lower than those of previous methods. Additionally, functional verification via Cuckoo Sandbox revealed that the adversarial examples generated by MiniMal retained 100% functional integrity, even with various modifications applied.

1 Introduction

Malware refers to applications utilized by attackers to execute malicious actions, frequently serving as a tool to gain control over victims and inflict damage. Signature-based static detection methods are efficient and low-cost, making them the primary approach for current malware detection [Zhan *et al.*, 2023a]. This method relies on malware signature databases; however, unknown or obfuscated malware often lacks corresponding signatures in these databases, resulting in identification failures that allow attackers to evade detection. In light of the rapid advancements in machine learning (ML) technology, researchers have introduced numerous ML-based methods for malware detection to tackle the vulnerability of static malware signatures being readily circumvented [Anderson and Roth, 2018; Raff *et al.*, 2021b]. These methods extract multi-dimensional features from malware and learn their behavioral patterns, enabling detection without fixed signatures and exhibiting strong generalization capabilities in identifying unknown or obfuscated samples [Ling *et al.*, 2023]. Consequently, an increasing number of security vendors are integrating ML-based malware detection techniques into their security products.

While ML methods offer significant advantages in malware detection, they are inherently susceptible to adversarial example attacks [Goodfellow *et al.*, 2014]. Adversarial examples are inputs generated by intentionally adding subtle perturbations to the data, which can induce neural network models to generate incorrect predictions. Although adversarial example attacks have been extensively researched in computer vision [Cheng *et al.*, 2018; Chen *et al.*, 2023b] and natural language processing [Peng *et al.*, 2023; Zhu *et al.*, 2024], their exploration in malware has only recently begun [Song *et al.*, 2022]. Based on the attacker's capabilities, adversarial example attacks against malware detection can be classified into white-box and black-box attacks [Zhu *et al.*, 2024]. In a **white-box scenario**, the attacker has access to the ML model's internal parameters and gradients. Black-box attacks can be further divided into two categories: (1) **soft-label attacks**, where the attacker can access the model's confidence scores (indicating probabilities of malicious or benign). (2) **hard-label attacks**, where the attacker only receives the model's final predicted label (malicious or benign). In real-world scenarios, white-box attacks and black-box soft label attacks are challenging to execute; attackers typically get the final label result via uploading malware through an API [Zhan *et al.*, 2023b]. Consequently, this paper focuses on investigating adversarial example attacks on malware within black-box hard-label scenarios.

Given that malware typically disseminates in Windows environments as Portable Executable (PE) file format [AV-TEST, 2024b], researchers are prompted to concentrate on attacks targeting this format. To date, a variety of methods for modifying PE files have been proposed, including those based on heuristic algorithms [Zhan *et al.*, 2023b; Demetrio *et al.*, 2021; Yuste *et al.*, 2022], reinforcement learning (RL)

[Anderson *et al.*, 2018; He *et al.*, 2024; Song *et al.*, 2022], and generative adversarial networks (GAN) [Hu and Tan, 2022; Yuan *et al.*, 2020]. For instance, Demetrio et al. [Demetrio *et al.*, 2021]employed genetic algorithms to select appropriate perturbation bytes and inject them into specific sections or the tail of PE files. Additionally, Song et al. [Song *et al.*, 2022]designed a series of actions for modifying malware and used a multi-armed bandit approach to select appropriate actions for generating adversarial examples.

Although current black-box attack methods have successfully generated adversarial examples with effective evasion, several limitations persist (1) **Excessive perturbation**: Existing research attempts to add numerous perturbation bytes to malware to evade detection. However, adding excessive perturbations often makes malware unusable in real-world scenarios. For instance, when attackers exploit file upload vulnerabilities, they are limited by the maximum file size [Group, 2024]; excessive modifications can dramatically increase the size of the malware, resulting in upload failures. Moreover, too many perturbations may render files unexecutable [Song *et al.*, 2022], as modifications to the PE file header beyond a certain threshold—such as exceeding six times the original file size may fail to run properly [He *et al.*, 2024]. (2) **Incomplete functional integrity of adversarial examples**: Researchers aiming to evade detection often apply aggressive modifications to PE files [Etter *et al.*, 2023]. Additionally, during adversarial example generation, functional verification of the PE file is often neglected [Hu and Tan, 2022], or handled manually post hoc using sandbox environments [Song *et al.*, 2022].

To address the issues mentioned above, we propose a **mini**mal perturbation adversarial **mal**ware examples generation method named **MiniMal**. This method effectively minimizes perturbation by targeting both the actions and content involved. Initially, we implemented a decremental loop for each action, reducing the maximum call count to zero to identify key perturbation actions. Then, we used binary search and particle swarm optimization (PSO) algorithms to filter critical content. Finally, we ensured the functional integrity of the adversarial examples by extracting their format features and control flow graph (CFG) information.

We evaluated MiniMal on three state-of-the-art ML detectors and compared it with other state-of-the-art adversarial attack methods regarding attack success rate, perturbation rate, and functionality. The results show that the success rates of MiniMal attacks on the three detectors are 99.96%, 98.56%, and 99.62%, while the average perturbation rates are 38.91%, 32.18%, and 33.92%. These results consistently outperform existing methods and retain the functional integrity of adversarial examples.

In summary, We summarize our contribution as follows:

- We propose a method for reducing perturbation content using binary search and particle swarm optimization, which eliminates redundant elements by determining the minimal perturbation ratio in multi-dimensional space. This addresses the problem of excessive perturbation in generated adversarial examples.
- We propose a functional verification method based on file format parsing and CFG comparison. This method evaluates the functional consistency of adversarial examples with their originals by extracting file format features and calculating CFG similarity, thus ensuring the functional integrity of the adversarial examples.
- We implemented a prototype system of MiniMal and validated its effectiveness through comprehensive experiments. The results demonstrate that our method achieves over 98% attack success against three advanced ML detection models, outperforming existing methods by approximately 4.8% to 7.1%. The perturbation rate consistently remains below 40%, with the minimum rate being 11 times lower than state-of-the-art methods.

Our source code and experimental data are available at https://github.com/2002lcy0401/MiniMal.

2 Related Work

This section reviews existing work on hard-label attacks. Current hard-label attacks primarily use heuristic algorithms, RL, and GANs to generate adversarial examples.

Heuristic algorithm-based methods. Demetrio et al. [Demetrio *et al.*, 2021] proposed a black-box attack method called GAMMA, which generates adversarial examples by adding benign content produced by a genetic algorithm to the sections or end of a PE file. Wang et al. [Wang *et al.*, 2022] introduced an adversarial example generation method based on a co-evolutionary algorithm. This approach frames the tasks of injecting minimal content and being classified as benign by the target model as two populations that cooperate during co-evolution to minimize the fitness function. Zhan et al. [Zhan *et al.*, 2023b] introduced the concept of adversarial patches with the MalPatch method, which generates universal content via a genetic algorithm and injects it into the end of PE files to create adversarial examples.

Reinforcement Learning-based methods. Anderson et al. [Anderson *et al.*, 2018] proposed gym-malware in a black-box setting, which collects a series of modification techniques and was the first to apply reinforcement learning to find the optimal attack sequence. Song et al. [Song *et al.*, 2022] developed MAB-malware, transforming the adversarial example generation problem into a multi-armed bandit problem and proposing the use of micro-actions instead of macro-actions to minimize the adversarial example. He et al. [He *et al.*, 2024] introduced MalwareTotal, which focuses on evading antivirus software in real-world scenarios while preserving malware functionality.

Generative Adversarial Networks-based methods. Hu et al. [Hu and Tan, 2022] introduced MalGAN, the first to apply the GAN concept to modify API sequences in the malware feature space, generating adversarial examples. Yuan et al. [Yuan *et al.*, 2020] proposed GAPGAN, a method for performing byte-level black-box attacks by generating adversarial payloads using GANs and appending them to the end of PE files. Gibert et al. [Gibert *et al.*, 2023] developed a query-free method for crafting adversarial malware examples and designed a GAN-based framework specifically targeting detectors that use byte, API, and string features.

3 Methodology

3.1 Problem Definition

Let X be a dataset containing n malware samples, and let x_i be a sample in X. Let f be the malware detector that produces the classification result (0 or 1) for x_i and $x_i + \Theta_i$. Θ represents the perturbation applied. Equation 1 defines the objective function for generating adversarial examples:

$$\arg\max_{\Theta} \sum_{i=1}^{n} f(x_i + \Theta_i) \oplus f(x_i) \tag{1}$$

$$\text{s.t.} \quad \Theta_i \leq r * x_i, \quad q \leq Q, \quad \mathcal{F}(x_i + \Theta_i) = \mathcal{F}(x_i)$$

Here, r represents the upper limit of the perturbation rate, which quantifies the maximum allowable perturbation. The symbol $\oplus$ represents the XOR operation. Q denotes the maximum number of queries, indicating the limit on interactions with the target model. $\mathcal{F}$ is the functionality verification function. Our goal is to generate as many fully functional adversarial examples as possible under the constraints of maximum perturbation and query limits.

3.2 Design Overview

Figure 1 outlines the overview of MiniMal: (1) **Initialize Adversarial Examples** : The roulette wheel selection algorithm is employed to iteratively select perturbation actions and contents until the target detector is successfully bypassed. (2) **Optimize Adversarial Examples**: Redundant actions are eliminated through action minimization, followed by the removal of redundant perturbation content using the binary search method and particle swarm optimization algorithm. Differences between the original and adversarial examples are analyzed to update the weights of perturbation actions and contents. (3) **Functionality Verification**: Throughout the generation process, the functionality verification module is invoked repeatedly to ensure the complete functional integrity of the adversarial examples.

3.3 Step 1: Initialize Adversarial Examples

Perturbation Actions and Contents. We review the file perturbation actions reported in previous studies [Anderson *et al.*, 2018; He *et al.*, 2024] and build the perturbation action set used in this paper. As shown in Table 1, the modifications to the files are categorized into four categories: Header, Section, Overlay, and Overall.

We finally selected four actions: **MD**, **AS**, **SA**, and **OA**. The reasons for choosing these actions are as follows:

- They cover all parts of the PE file. Since the detector's features may be derived from any part of the PE file, we require the ability to modify every part of the file.
- These actions are relatively independent and safe, meaning they do not depend on the order in which they are applied, thereby simplifying the process.
- These actions allow for significant perturbation, providing a greater capacity for modifications.

Besides the selected actions, the injected contents are also critical. According to Suciu et al. [Suciu *et al.*, 2019], benign byte features from benign files assist malware in crossing the decision boundary. Therefore, we extract benign segments from benign file samples to construct the content set.

Initialization of Adversarial Examples. After establishing the action and content sets, we need to choose suitable perturbation actions and contents to modify the PE file, thereby generating adversarial examples. We assume that the weaknesses of the target detector can be exploited repeatedly. To this end, we create weight tables for both the action set and the benign content set, which track the frequency of actions and content used in previously generated adversarial examples. The roulette wheel selection algorithm [Lipowski and Lipowska, 2012] is employed to select actions and contents. The selection probability of each individual is proportional to its weight, meaning that individuals with greater weights have a higher chance of being selected. During the initialization process, the weight table serves as the basis for roulette wheel selection, increasing the likelihood of selecting effective actions and contents. By continuously modifying the malware samples, once an adversarial example successfully evades detection, it proceeds to Step 2 for optimization.

3.4 Step 2: Optimize Adversarial Examples

Action Minimizer. We designed an action minimization module to select critical actions. Since the four selected actions are independent and do not influence each other, we implement a decremental loop mechanism for each action A_i, reducing its call count step-by-step from T_i (The maximum number of times A_i can be used) down to 0. Specifically, we begin by removing all calls of A_i. If the adversarial example still evades the target model's detection after removal, this reduced state is retained; otherwise, the loop continues to gradually decrease the call count of A_i. This approach effectively eliminates redundant actions at a coarse-grained level, facilitating the subsequent removal of redundant content.

Binary Deletion. Once the action minimization process is complete and only one action remains, we employ a binary deletion method to refine the redundant perturbation content. We treat the perturbation content as a byte array, which allows us to reframe the optimization problem of the adversarial example as the challenge of removing non-critical segments from this array. Binary deletion serves as an effective strategy for efficiently searching and removing unnecessary perturbation bytes. We begin by dividing the array into left and right halves, attempting to delete each half separately to determine if the adversarial example still evades detection. If successful, the removed segment is deemed redundant and can be discarded; if not, the binary search process continues within that segment, repeating this step until either the maximum query limit is reached or further division is no longer feasible.

Particle Swarm Optimization Deletion. When multiple actions remain after the action minimization, optimizing adversarial examples becomes a combinatorial optimization problem involving multiple perturbation contents. PSO is a population-based optimization algorithm often employed to address such optimization problems [Kennedy and Eberhart, 1995]. The fundamental idea is to transform the optimization problem into a search problem in a multi-dimensional space, where each particle represents a potential solution. Since the

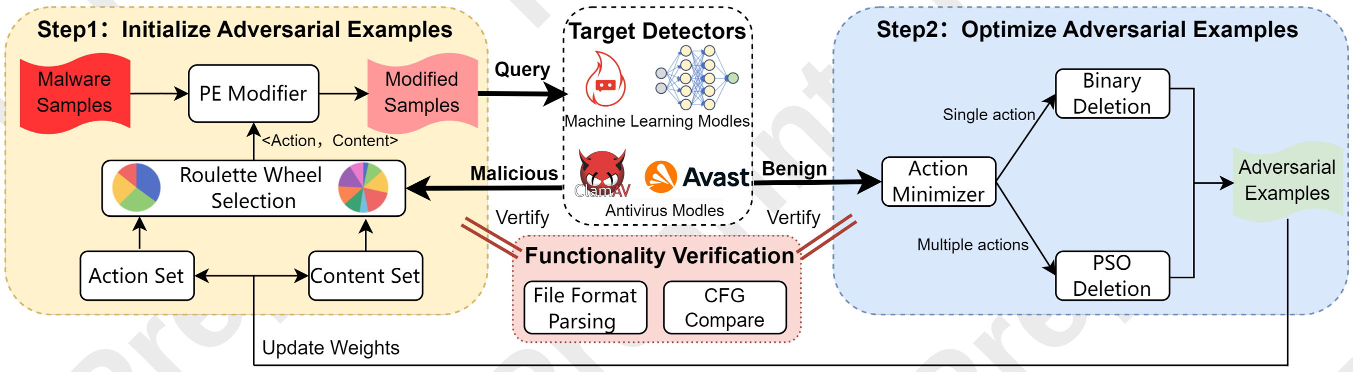


Figure 1: Overview of MiniMal.

Location	Abbr	Name	Description
Header	BC	Break Checksum	Set the checksum member variable to 0.
	RC	Remove Certificate	Clear the certification flag in the optional header.
	MT	Modify Timestamp	Change the PE file's timestamp.
	MD	Modify DOS Header	Fill bytes between the "MZ" file flag and PE header.
	AI	Add Imports	Import new functions into the PE file from a library.
Section	**AS**	Add Section	Add a new section to the PE file.
	RS	Rename Section	Rename the sections of the file.
	SA	Slack Append	Fill bytes in the slack space between sections.
Overlay	**OA**	Overlay Append	Add bytes to the end of the PE file.
Overall	UP	UPX Pack	Pack the PE file using the UPX tool.
	OB	Obfuscation	Obfuscate the PE file.

Table 1: Action set for modifying PE files.

actions we employ are relatively independent, the particle update process in PSO is also conducted independently, making this algorithm particularly suitable for our needs.

Algorithm 1 outlines the complete process of using the PSO algorithm to optimize the perturbation content. First, we treat the perturbation content used by each action A_i as an array C_i. Each particle's value ranges from 0 to 1, representing the usage ratio of the array for each action. For example, if $r_i = 0.3$, this indicates that 30% of the perturbation content C_i is utilized by action A_i. Thus, the problem is further transformed into finding a set of decimal solutions between 0 and 1, which represent the percentage of perturbation content used for each action.

In PSO, the design of the fitness function is crucial, as it guides the search direction of the particles. We use the size of the perturbation as the optimization objective and introduce two penalty terms to ensure that the adversarial example successfully evades the target detector while retaining functional integrity. The specific design is presented in Equation 2.

$$fitness = \sum_{i=1}^{n}(r_i * C_i) + w_1 * f(x') + w_2 * F(x') \quad (2)$$

The $fitness$ function is the objective function. The term $\sum_{i=1}^{n}(r_i * C_i)$ measures the size of the perturbation. w_1 is the penalty term for evading the detector f, and w_2 is the penalty term for the functionality verification function F, where x' represents the adversarial example with added perturbation.

Algorithm 1: PSO Deletion

```
Input: Malware: $X$, Perturbation content list: $C$, Max
       iterations: $n$, Penalty term: $w_1$, $w_2$
Output: Optimized adversarial example: $X_{final}$
1:  begin
2:    Initialize particle swarm $P \leftarrow$ Random values
        between [0,1] for each perturbation in $C$ ;
3:    Initialize best solution $P_{best} \leftarrow 0$ ;
4:    Initialize global best solution $G_{best} \leftarrow 0$ ;
5:    for $i \leftarrow 1$ to $n$ do
6:      foreach particle $p \in P$ do
7:        $P1 \leftarrow \sum (p * \text{Len}(C))$ ;
8:        $X_{temp} \leftarrow Modify(X, p * C)$;
9:        $P2 \leftarrow 0$ if $f(X_{temp})$ == Benign, else $w_1$;
10:       $P3 \leftarrow 0$ if $F(X_{temp})$ == True, else $w_2$;
11:       fitness($p$) $\leftarrow P1+P2+P3$;
12:       if fitness($p$) <fitness($P_{best}$) then
13:         $P_{best} \leftarrow p$;
14:       if fitness($p$) <fitness($G_{best}$) then
15:         $G_{best} \leftarrow p$;
16:     Update each particle's velocity and position;
17:   $X_{final} \leftarrow Modify(X, G_{best} * C)$;
18:   return $X_{final}$ ;
```

3.5 Functionality Verification

Inspired by the work of Tian et al.[Tian *et al.*, 2024], we designed an efficient, automated functional verification method. We begin by extracting the structural features of the PE file to ensure the integrity of its core structure and execution logic. Key features, such as the DOS header, PE signature, section table, and import table, are examined to verify the file's integrity and standard format, preventing execution errors caused by format corruption or inconsistencies.

A control flow graph represents a program's control flow, where nodes correspond to basic blocks, and edges denote potential control paths between them. By comparing the CFGs of the original PE file and the adversarial example, we can assess whether their functionalities are consistent [Yuan *et al.*, 2024]. Once the adversarial example is generated, we use the Jaccard similarity to measure the overlap of nodes and edges between the two CFGs. Based on a predefined threshold, we determine whether the adversarial example retains the same functionality as the original.

4 Experiments

4.1 Experimental Setup

Datasets

For this study, we primarily sourced data from the publicly available Malware Detection PE-Based Dataset [Tuan *et al.*, 2018], which has been widely used in previous work [Zhan *et al.*, 2023b]. It includes five malware types: Locker, Mediyes, Winwebsec, Zbot, and Zeroaccess, as well as 1,000 benign software samples. Additionally, we downloaded other common malware types from the MalwareBazaar website[MalwareBazaar, 2024], including Trojan, Backdoor, and Ransomware. To ensure compatibility with all target detectors, we select 2,642 malware samples and 473 benign samples that are correctly classified by the target detectors.

Target Detectors

We evaluated MiniMal using two types of malware detectors: machine learning-based malware detection models and real-world antivirus products. Their characteristics are as follows:

- **Detectors 1-3:** We selected three state-of-the-art ML models that have been frequently used in previous work: MalConv [Raff *et al.*, 2018], Ember [Anderson and Roth, 2018], and MalGCG [Raff *et al.*, 2021b]. MalConv and MalGCG are deep learning models and Ember is based on the LightGBM model. Moreover, we used the pre-trained models provided by the official sources [Anderson, 2019; Raff *et al.*, 2021a].
- **Detectors 4-5:** We selected ClamAV and Avast. ClamAV is an antivirus engine that detects files based on an up-to-date malware signature database [ClamAV, 2024]. Avast ranked first in AV-Test's Best Windows Antivirus Software for Home Users [AV-TEST, 2024a]. Due to their powerful detection capabilities, they have been widely studied in both industry and academia.

Evaluation Metrics

We evaluate MiniMal's performance using four metrics: Attack Success Rate (ASR), Query Budget (Q), Perturbation Rate (Pert), and Functionality. ASR indicates the percentage of original malware samples where adversarial examples successfully evade the target detector. Query Budget specifies the maximum number of interactions with the target model permitted during adversarial example generation. The Perturbation Rate measures the degree of modifications applied to the adversarial example relative to the original sample. Functionality evaluates whether the generated adversarial examples remain executable and retain their malicious behavior.

Baseline

We compare MiniMal against several state-of-the-art methods for generating adversarial malware examples: BIA [Suciu *et al.*, 2019], GAMMA [Demetrio *et al.*, 2021], and MAB [Song *et al.*, 2022], which represent the most advanced techniques in hard-label scenarios.

Although recent years have seen the proposal of several new methods, such as MalwareTotal [He *et al.*, 2024], AMGmal [Zhan *et al.*, 2023c], PSP-Mal [Zhan *et al.*, 2023a], MalAder [Chen *et al.*, 2023a], and MalGuise [Ling *et al.*, 2024], they are not suitable for the comparison presented in this paper. For instance, AMGmal generates adversarial samples in a white-box scenario; MalwareTotal and PSP-Mal train RL models based on known features used by the detector, whereas our scenario assumes no prior knowledge of the features used by the target detector; MalGuise focuses on bypassing detectors based on CFG features, and MalAder focuses on evading detectors relying on API Call sequence features, while we focus on evading static detectors.

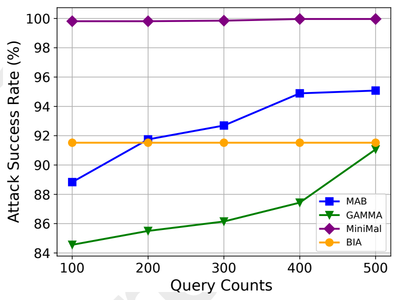
Implementation Detail

We developed the prototype implementation of MiniMal using Python. All experiments were conducted on a computer equipped with an NVIDIA GeForce RTX 4070 and a Linux server featuring an AMD EPYC 9654 96-core processor.

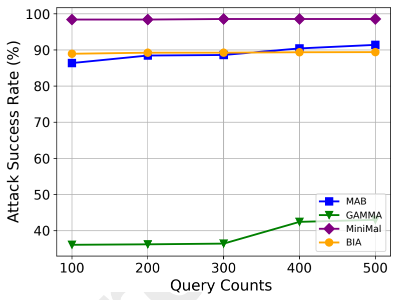
4.2 Comparison With State-of-the-Arts

To ensure a fair comparison, we used the original settings from the official source code for baseline, with all comparisons conducted on the same test set. Moreover, to reduce the impact of randomness, we run each method three times and take the average of the results. Following prior work [He *et al.*, 2024], we set the query budget to 500 and the maximum perturbation rate to 1000% for each method to fully utilize their performance.

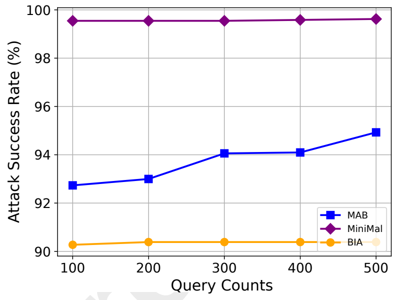
As shown in Table 2, MiniMal outperforms existing methods in terms of attack success rate and perturbation rate across the three machine learning-based models. MiniMal achieved attack success rates of 99.96%, 98.56%, and 99.62% on the three detectors, with all rates above 98%, representing an improvement of about 4.8% to 7.1% over the best existing methods. Additionally, We found that the average perturbation rates of adversarial examples generated by MiniMal were 38.91%, 32.18%, and 33.92%, all below 40%, and reduced by 9 to 11 times compared to state-of-the-art methods. From Table 2, we observe that all attack methods are least effective against the Ember model. This may be because Ember utilizes diverse PE file features, providing it with greater robustness compared to detectors that rely on byte features.



(a) ASR against MalConv.

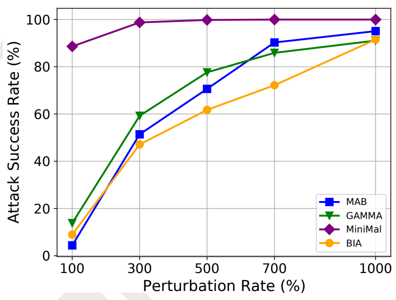


(b) ASR against Ember.

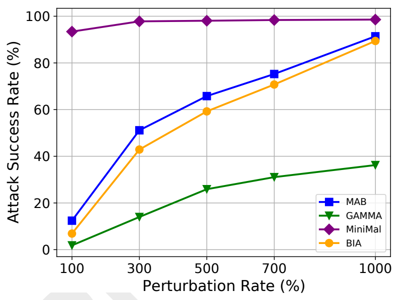


(c) ASR against MalGCG.

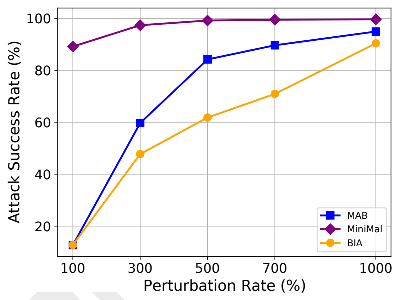
Figure 2: Comparison of the attack success rates by increasing the limited query.



(a) ASR against MalConv.



(b) ASR against Ember.



(c) ASR against MalGCG.

Figure 3: Comparison of the attack success rates by increasing the limited perturbation rate.

	MalConv		Ember		MalGCG	
Methods	ASR ↑	Pert ↓	ASR ↑	Pert ↓	ASR ↑	Pert ↓
MAB	95.08%	386.44%	91.41%	376.03%	94.93%	327.50%
GAMMA	91.07%	282.90%	43.01%	371.58%	*	*
BIA	91.52%	403.23%	89.40%	416.38%	90.39%	389.47%
MiniMal	**99.96%**	**38.91%**	**98.56%**	**32.18%**	**99.62%**	**33.92%**

Table 2: The attack success rate (ASR,%) and perturbation rate (Pert,%) of different hard-label attack algorithms on three models.

To further evaluate the effectiveness of various attack algorithms under different constraints, we tested with the number of queries ranging from 100 to 500 and maximum perturbation limits from 100% to 1000%. As shown in Figure 2, MiniMal maintains an attack success rate above 98% across different query constraints, outperforming the baseline methods. Figure 3 illustrates the attack success rates of various algorithms under different perturbation constraints. Notably, MiniMal achieves over 80% success even under a strict 100% perturbation limit, whereas baseline methods average below 15%. This highlights the significant advantage of our approach under low perturbation conditions.

4.3 Functionality Preservation

We tested the functional integrity of adversarial examples generated by the MiniMal. We used the Cuckoo Sandbox [Cuckoo, 2024], following the previous method [Zhan *et al.*, 2023b], to further verify the effectiveness of our approach. We selected 100 adversarial examples for testing. We submitted the samples to an online sandbox platform for dynamic execution and extracted their API sequences. Using histogram analysis, we checked for significant changes in API calls. The results showed that the API sequences of the adversarial examples closely matched those of the original samples, confirming that the adversarial behavior remained intact. This validates the robustness of our functionality verification.

4.4 Ablation Study

Effect of the Roulette Wheel Selection. During the initialization phase (Step 1), the roulette wheel selection (RWS) was used to select actions and contents iteratively. To examine whether this improves attack success, we included a control group that used random selection (RS). We randomly selected 200 samples and applied the top fifty action-content pairs with the highest weights from the roulette wheel selection algorithm, as well as fifty randomly chosen pairs, to evaluate their attack performance. From Figure 4, we observe that our method achieved attack success rates of 77.02%, 76.76%, and 89.52% under single modification conditions, significantly outperforming the random selection method, which recorded success rates of 42.58%, 14.96%, and 41.58%. Therefore, we infer that the roulette wheel selection method can effectively select beneficial perturbation actions and contents, and reusing these actions helps increase the ASR.

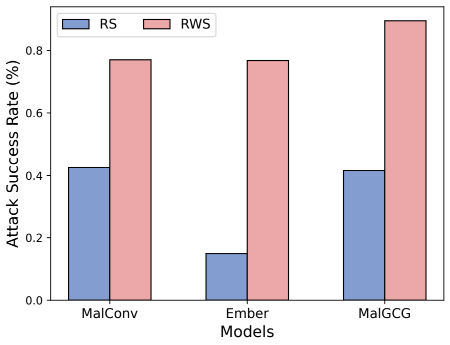


Figure 4: Comparison of the ASR between the top 50 highest-weighted action-content pairs and randomly selected pairs.

Effect of Action Minimization. To evaluate Action Minimization (AM), we created a control group (No AM) by excluding this module and measured the average action count on three malware detectors, both with and without AM. As shown in Table 3, without action minimization, the average action count ranges from 1.3 to 1.5. With action minimization introduced, the average action count decreased by approximately 0.3. Although the reduction is modest, this module effectively reduces the influence of non-critical actions on weight table updates.

Methods	MalConv	Ember	MalGCG
No AM	1.3028	1.2945	1.4484
AM	1.0386	1.0465	1.0752

Table 3: Average number of actions used before and after AM.

Effect of Binary Deletion and PSO Deletion. To evaluate their effectiveness, we recorded the file sizes of the original samples (Original), and adversarial examples after initialization (Step 1), and after optimization (Step 2). Figure 5 shows that after optimizing the adversarial examples, the average file sizes are 344.08 KB and 337.93 KB. The perturbation added in Step 1 averaged 953.04 KB, whereas, after optimization, this was reduced to just 76.53 KB on average. Thus, we conclude that binary deletion and PSO deletion significantly reduced the perturbation size.

4.5 Real-world Performance

We evaluated MiniMal's effectiveness on real-world commercial antivirus software. We randomly selected 200 samples

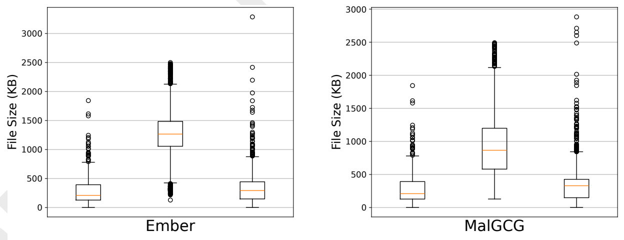


Figure 5: Comparison of file size at different stages.

and set a maximum of 100 queries. We also included the UPX [UPX, 2024] method, a well-known tool for compressing PE files that is widely used by attackers in real-world scenarios.

The results, shown in Table 4, indicate that MiniMal achieved a 79% success rate against ClamAV and a 35% success rate against Avast, outperforming the baseline methods. UPX achieved success rates of 37% and 29.5%, respectively. We infer that the widespread use of UPX for packing malware has led to many UPX-packed signatures being incorporated into ClamAV's virus databases. MiniMal demonstrates effective performance against ClamAV, as introducing perturbations in various regions of the PE file influences the computation of the malware signature. However, MiniMal performed poorly against Avast. Although Avast has not disclosed the specifics of its virus engine architecture, we suspect that Avast incorporates dynamic behavioral analysis of malware in memory, which diminishes the effectiveness of our static modification approach.

	BIA		UPX	MiniMal	
Models	ASR ↑	Pert ↓	ASR ↑	ASR ↑	Pert ↓
ClamAV	15%	2537.77%	37%	**79%**	**3.77%**
Avast	3.5%	232.71%	29.5%	**35%**	**25.27%**

Table 4: The attack success rate results on Antivirus.

5 Conclusion

This paper introduces MiniMal, a hard-label adversarial attack method specifically designed for static malware detectors. Initially, we employ a roulette wheel algorithm to systematically select actions and content for generating adversarial examples. We then apply action minimization, along with binary search and particle swarm optimization algorithms, to effectively reduce the perturbation rate of these samples. Furthermore, we ensure the functional integrity of adversarial examples through file format parsing and CFG comparison. We conducted a series of experiments to demonstrate the effectiveness of our approach. The results indicate that MiniMal achieves an attack success rate exceeding 98% against three ML-based detection models, with a perturbation rate below 40%, surpassing existing hard-label attack methods. Notably, MiniMal achieves over 80% of attack success even under low perturbation rate limits. Finally, We validated that the adversarial examples generated by MiniMal retain their malicious functionality through testing with Cuckoo Sandbox.

Contribution Statement

Chengyi Li and Zhiyuan Jiang made equal contributions to this study and are designated as co-first authors (*Authors contributed equally). Zhiyuan Jiang serving as the corresponding author, are responsible for all communications related to this manuscript (†Corresponding author).

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful suggestions on our paper. This work is supported by the National Natural Science Foundation of China (62402509) and the Natural Science Foundation of Hunan Province (No. 2025JJ40053).

References

[Anderson and Roth, 2018] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[Anderson *et al.*, 2018] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, 2018.

[Anderson, 2019] Hyrum Anderson. Machine learning static evasion competition., 2019.

[AV-TEST, 2024a] AV-TEST. The best windows antivirus software for home users, 2024.

[AV-TEST, 2024b] AV-TEST. Distribution of malware and pua by operating system, 2024.

[Chen *et al.*, 2023a] Xiaohui Chen, Lei Cui, Hui Wen, Zhi Li, Hongsong Zhu, Zhiyu Hao, and Limin Sun. Malader: Decision-based black-box attack against api sequence based malware detectors. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 165–178. IEEE, 2023.

[Chen *et al.*, 2023b] Zihan Chen, Ziyue Wang, Jun-Jie Huang, Wentao Zhao, Xiao Liu, and Dejian Guan. Imperceptible adversarial attack via invertible neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 414–424, 2023.

[Cheng *et al.*, 2018] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. *arXiv preprint arXiv:1807.04457*, 2018.

[ClamAV, 2024] ClamAV. Clamav is an open-source antivirus engine for detecting trojans, viruses, malware other malicious threats., aug 2024.

[Cuckoo, 2024] Cuckoo. Cuckoo sandbox., 2024.

[Demetrio *et al.*, 2021] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security*, 16:3469–3478, 2021.

[Etter *et al.*, 2023] Brian Etter, James Lee Hu, Mohammadreza Ebrahimi, Weifeng Li, Xin Li, and Hsinchun Chen. Evading deep learning-based malware detectors via obfuscation: A deep reinforcement learning approach. In *2023 IEEE International Conference on Data Mining (ICDM)*, pages 101–109. IEEE, 2023.

[Gibert *et al.*, 2023] Daniel Gibert, Jordi Planes, Quan Le, and Giulio Zizzo. A wolf in sheep's clothing: Query-free evasion attacks against machine learning-based malware detectors with generative adversarial networks. In *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 415–426. IEEE, 2023.

[Goodfellow *et al.*, 2014] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[Group, 2024] The PHP Group. Php: Description of core php.ini directives., 2024.

[He *et al.*, 2024] Shuai He, Cai Fu, Hong Hu, Jiahe Chen, Jianqiang Lv, and Shuai Jiang. Malwaretotal: Multi-faceted and sequence-aware bypass tactics against static malware detection. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–12, 2024.

[Hu and Tan, 2022] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on gan. In *International Conference on Data Mining and Big Data*, pages 409–423. Springer, 2022.

[Kennedy and Eberhart, 1995] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

[Ling *et al.*, 2023] Xiang Ling, Lingfei Wu, Jiangyu Zhang, Zhenqing Qu, Wei Deng, Xiang Chen, Yaguan Qian, Chunming Wu, Shouling Ji, Tianyue Luo, et al. Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art. *Computers & Security*, 128:103134, 2023.

[Ling *et al.*, 2024] Xiang Ling, Zhiyu Wu, Bin Wang, Wei Deng, Jingzheng Wu, Shouling Ji, Tianyue Luo, and Yanjun Wu. A wolf in sheep's clothing: practical black-box adversarial attacks for evading learning-based windows malware detection in the wild. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 7393–7410, 2024.

[Lipowski and Lipowska, 2012] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

[MalwareBazaar, 2024] MalwareBazaar. Malwarebazaar is a project from abuse.ch with the goal of sharing malware samples with the infosec community, av vendors and threat intelligence providers., 2024.

[Peng *et al.*, 2023] Hao Peng, Shixin Guo, Dandan Zhao, Xuhong Zhang, Jianmin Han, Shouling Ji, Xing Yang, and

Ming Zhong. Textcheater: A query-efficient textual adversarial attack in the hard-label setting. *IEEE Transactions on Dependable and Secure Computing*, 2023.

[Raff *et al.*, 2018] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. Malware detection by eating a whole exe. In *Workshops at the thirty-second AAAI conference on artificial intelligence*, 2018.

[Raff *et al.*, 2021a] Edward Raff, William Fleshman, Richard Zak, Hyrum Anderson, Bobby Filar, and Mark Mclean. Classifying Sequences of Extreme Length with Constant Memory Applied to Malware Detection. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021.

[Raff *et al.*, 2021b] Edward Raff, William Fleshman, Richard Zak, Hyrum S Anderson, Bobby Filar, and Mark McLean. Classifying sequences of extreme length with constant memory applied to malware detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9386–9394, 2021.

[Song *et al.*, 2022] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. Mabmalware: A reinforcement learning framework for blackbox generation of adversarial malware. In *Proceedings of the 2022 ACM on Asia conference on computer and communications security*, pages 990–1003, 2022.

[Suciu *et al.*, 2019] Octavian Suciu, Scott E Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 8–14. IEEE, 2019.

[Tian *et al.*, 2024] Buwei Tian, Junyong Jiang, Zichen He, Xin Yuan, Lu Dong, and Changyin Sun. Functionality-verification attack framework based on reinforcement learning against static malware detectors. *IEEE Transactions on Information Forensics and Security*, 2024.

[Tuan *et al.*, 2018] Anh Pham Tuan, An Tran Hung Phuong, Nguyen Vu Thanh, and Toan Nguyen Van. Malware Detection PE-Based Analysis Using Deep Learning Algorithm Dataset. 6 2018.

[UPX, 2024] UPX. Upx is a free, secure, portable, extendable, high-performance executable packer for several executable formats., 2024.

[Wang *et al.*, 2022] Fangwei Wang, Yuanyuan Lu, Qingru Li, Changguang Wang, and Yonglei Bai. A coevolutionary algorithm-based malware adversarial sample generation method. In *2022 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8. IEEE, 2022.

[Yuan *et al.*, 2020] Junkun Yuan, Shaofang Zhou, Lanfen Lin, Feng Wang, and Jia Cui. Black-box adversarial attacks against deep learning based malware binaries detection with gan. In *ECAI 2020*, pages 2536–2542. IOS Press, 2020.

[Yuan *et al.*, 2024] He Yuan, Xue Bo, Zhang Lina, and Lu Chengyang. A control flow graph optimization method for enhancing fuzz testing. *IEEE Access*, 2024.

[Yuste *et al.*, 2022] Javier Yuste, Eduardo G Pardo, and Juan Tapiador. Optimization of code caves in malware binaries to evade machine learning detectors. *Computers & Security*, 116:102643, 2022.

[Zhan *et al.*, 2023a] Dazhi Zhan, Wei Bai, Xin Liu, Yue Hu, Lei Zhang, Shize Guo, and Zhisong Pan. Psp-mal: Evading malware detection via prioritized experience-based reinforcement learning with shapley prior. In *Proceedings of the 39th Annual Computer Security Applications Conference*, pages 580–593, 2023.

[Zhan *et al.*, 2023b] Dazhi Zhan, Yexin Duan, Yue Hu, Weili Li, Shize Guo, and Zhisong Pan. Malpatch: Evading dnn-based malware detection with adversarial patches. *IEEE Transactions on Information Forensics and Security*, 19:1183–1198, 2023.

[Zhan *et al.*, 2023c] Dazhi Zhan, Yexin Duan, Yue Hu, Lujia Yin, Zhisong Pan, and Shize Guo. Amgmal: Adaptive mask-guided adversarial attack against malware detection with minimal perturbation. *Computers & Security*, 127:103103, 2023.

[Zhu *et al.*, 2024] Hai Zhu, Qingyang Zhao, Weiwei Shang, Yuren Wu, and Kai Liu. Limeattack: Local explainable method for textual hard-label adversarial attack. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19759–19767, 2024.