MalwareTotal: Multi-Faceted and Sequence-Aware Bypass Tactics against Static Malware Detection

Shuai He[1,3], Cai Fu[1,3], Hong Hu[2], Jiahe Chen[1,3], Jianqiang Lv[1,3], Shuai Jiang[1,3]
[1]School of Cyber Science and Engineering, Huazhong Science and technology University, China
[2]Pennsylvania State University, United States
[3]Hubei Key Laboratory of Distributed System Security, Huazhong Science and technology University, China

ABSTRACT

Recent methods have demonstrated that machine learning (ML) based static malware detection models are vulnerable to adversarial attacks. However, the generated malware often fails to generalize to production-level anti-malware software (AMS), as they usually involve multiple detection methods. This calls for universal solutions to the problem of malware variants generation. In this work, we demonstrate how the proposed method, MalwareTotal, has allowed malware variants to continue to abound in ML-based, signature-based, and hybrid anti-malware software. Given a malicious binary, we develop sequential bypass tactics that enable malicious behavior to be concealed within multi-faceted manipulations. Through 12 experiments on real-world malware, we demonstrate that an attacker can consistently bypass detection (98.67%, and 100% attack success rate against ML-based methods EMBER and MalConv, respectively; 95.33%, 92.63%, and 98.52% attack success rate against production-level anti-malware software ClamAV, AMS A, and AMS B, respectively) without modifying the malware functionality. We further demonstrate that our approach outperforms state-of-the-art adversarial malware generation techniques both in attack success rate and query consumption (the number of queries to the target model). Moreover, the samples generated by our method have demonstrated transferability in the real-world integrated malware detector, VirusTotal. In addition, we show that common mitigation such as adversarial training on known attacks cannot effectively defend against the proposed attack. Finally, we investigate the value of the generated adversarial examples as a means of hardening victim models through an adversarial training procedure, and demonstrate that the accuracy of the retrained model against generated adversarial examples increases by 88.51 percentage points.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**.

KEYWORDS

Anti-malware software robustness, black-box attacks, binary manipulation

ICSE 2024, April 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04…$15.00
https://doi.org/10.1145/3597503.3639141

1 INTRODUCTION

The robustness of malware variants detection methods is imperative, and not only for software security: it also fundamentally affects the process and quality of software engineering. The relentless expansion of newly discovered malware samples, amounting to hundreds of thousands daily over the past decade, has posed a persistent challenge to the software development community [70]. The heuristics and machine learning (ML) techniques that have been proposed in response to this challenge do show promise [43]. However, relying exclusively on machine learning cannot be recommended [62], because machine learning suffers from an inherent vulnerability to adversarial attacks, where imperceptible input perturbations lead to incorrect predictions [52]. Prior studies [10, 71] have mounted such attacks on ML-based models [35, 53] and made valuable progress. However, our investigation shows their limited effectiveness against real-world anti-virus software (AVs). To our knowledge, no existing malware generation approach has achieved better than a 50% average attack success rate on commercial AVs, which is insufficient for real-world malware detection [59].

The continuous advancement and diversification of malware requires comparable strides in software engineering practices, particularly in static malware detection techniques. An analysis of prior work [8, 12, 33] uncovers four underlying factors that hinder the optimal evasion capability. (i) The design of manipulations lacks diversity. For example, Demetrio et al. [22] manipulate the DOS Header of malicious files, because the victim model MalConv [54] focuses more on Windows Portable Executable (PE) [1] header features. (ii) Recent methods [16, 19, 21] follow the hypothesis that the full information of target detectors is known, which is infeasible for commercial AVs. (iii) Most existing work is designed for one or a single type of malware detector, while commercial AVs are often characterized by hybrid underlying model architecture. (iv) Existing work aims to optimize the benign score of injection content to achieve a minimal injection. This is ideal, and a crucial target in the beginning period of an adversarial attack [46], when excessive modifications can arouse distinguishable changes to the input. We consider this a non-necessary target, as it can be constrained by a query budget with a fixed threshold of injection bytes.

Motivated by the above challenges, we propose MalwareTotal, a sequence-aware adversarial malware generation method. Rather than merely focusing on minimizing the injected content, sequence-aware manipulation emphasizes the identification of the vulnerable sequence that can successfully evade detection. To achieve robust evasion against different anti-virus architectures, we carefully collect and extend manifold *Portable Executable* (PE) file modifications

[1]https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

that have been scattered in the literature, and eventually unearth 23 functionality-preserving manipulations. We have divided them into four categories: header manipulation, section manipulation, overlay manipulation, and instruction manipulation. Leveraging the sequential nature of malware mutations, we optimize the malware mutation process by leveraging proximal policy optimization (PPO) [58], a gradient-based algorithm used to update manipulating policies iteratively. To the best of our knowledge, this paper presents the first black-box malware generation method that can robustly bypass real-world commercial anti-virus software. Results indicate that malware evasion can be common in ML-based, signature-based, and hybrid anti-malware products, highlighting the crucial role of improved software engineering practices in ensuring robustness against evolving malware threats.

Contribution. Our findings can benefit both researchers and practitioners. Moreover, insights derived from our method can potentially guide software generation, especially in the domains of obfuscation [30], security [47], testing [44], and automation [60]. By understanding and simulating malware evasion, we can engineer more resilient and secure systems that can thrive in the face of evolving malware threats. We also make the data and code used in our study available in a replication repository.[2] In summary, this paper makes the following contributions:

- *A Sequence-aware Black-box Adversarial Attack.* We propose a novel bypass tactic by learning the AV-oriented vulnerable sequence through proximal policy optimization. To the best of our knowledge, this is the first method that can robustly bypass real-world commercial anti-virus software with detected malware in a complete black-box scenario (also known as "the miracle"), revealing that the risks faced by static anti-virus software have been underestimated.
- *Efficient Evasion Performance.* We conduct a thorough evaluation of our end-to-end implementation of MalwareTotal, demonstrating its substantially stronger attack success rate of between 16.55% and 83.59%. The ability to generate variants 1.74 times faster than the state-of-the-art method.
- *Effective Robustness Enhancement.* We show the value of MalwareTotal-generated examples: retraining the victim model with these adversarial samples can improve the robustness against MalwareTotal by 88.51 percentage points. Additionally, the retrained model shows more resilience against other black-box adversarial attacks.

2 RELATED WORK

Previous works are significantly different from MalwareTotal, as they design attacks from the perspective of an adversarial attack, which naturally can be categorized into two schemes:

White-box attacks: White-box attacks mostly rely on the full target information, including model gradient, output score, and so on. Kolosnjaji et al. [39] propose the *Padding* method, which manipulates malware through padding embedding values at the PE file overlay guided by the positive direction of the gradient. The experiment is conducted on MalConv detector and achieves a 60% attack success rate. Demetrio et al. [19] propose the *Partial DOS* method, which manipulates the first 58 bytes contained in the DOS header. Additionally, *Header Fields* [3] manipulates the names of each section in *Optional Header*, and *Full DOS* [22] proposes three individual attacks *Full DOS, Extend,* and *Shift*. *Full DOS* manipulates 58 bytes in the DOS header and the whole DOS stub, while *Extend* enlarges the DOS header and injects adversarial content, and *Shift* injects content before the first section.

Black-box attacks: Demetrio et al. [20] propose two attacks. One, *Gamma Section* conducts manipulations through a set of manipulations in section and a packer. The other, the *Gamma Padding* attack, is a combination of *Padding, Partial DOS, Extend,* and *Shift*; this manipulation optimization relies on a genetic algorithm. Similarly, Song et al. [61] propose the *MAB* method, which utilizes a packer as well as a set of manipulations and uses a multi-armed bandit as the optimizer, *MAB* requires 20 servers to conduct manipulations in their experiments. *Aimed* [11] utilizes the same manipulations proposed in *Header Fields*, and utilizes a genetic algorithm to minimize the score of the mutated malware. They also employ a sandbox to discard malware whose functionality is broken.

The novelty of our work lies on the following aspects: (i) unlike other evasion attacks, MalwareTotal focuses on directly optimizing the minimum manipulation sequence instead of optimizing the minimum injection. This is due to the sequential nature of malware mutations. (ii) MalwareTotal is simple and easy to apply in real-world scenarios, since it is a query-efficient evasion attack that only needs hard labels provided by black-box malware detectors.

3 PRELIMINARIES AND THREAT MODEL

In this section, we introduce the portable executable (PE) file format and formulate the problem of generating malware variants. Specifically, we focus on examining robustness in the context of static PE malware detection via a novel problem space attack. Compared to feature space attacks [41, 42], a problem space attack in the malware variant generation manipulates the input data directly, making it easier to retain the functionality in the generation process. The attack is of black-box type and can only manipulate malware in the test phase of a ML anti-malware model, because the architecture of anti-virus software is usually unknown.

3.1 PE File Format Brief Overview

The PE file format defines how executable programs are stored as a file on disk. Here, we provide a brief introduction to the PE file format, particularly the members cannot be modified.

DOS Header and Stub. The DOS header contains metadata for loading the executable inside a DOS environment, while the DOS Stub prints “This program cannot be run in DOS mode” if the file is executed in a DOS environment. The functionality-related members contained inside the DOS Header are (i) member *e_magic*, the two-byte magic number MZ, which are the initials of the designer, and (ii) the member *e_lfanew*, four bytes at offset 0x3c, which is the pointer to the actual header. If one of these two values is altered, the program will be corrupted.

NT Header. This member is the short form of *IMAGE_NT_HEADER*, where NT stands for new technology. It contains target architectures, the size of the header, and the attributes of the file. The optional header (positioned after the NT_Header) specifies (i) the file alignment, a constraint on the structure of the executable,

[2]https://github.com/Optimus-He/MalwareTotal

as each section must start at an offset multiple to that field, and (ii) the size of headers, which specifies the number of bytes that are reserved to headers of the programs. Note that if it is modified, the post-modification must be a multiple of the file alignment.

Section. A section is a logical division of the file that contains specific types of information. Each section in a PE file has its own characteristics and serves a particular purpose: (i) the *.text section*, is the only section with execute privileges and contains resident code [24], and (ii) the *.data section* contains initialized data, such as global or static variables that have predefined values.

Overlay. This is a separate section that exists outside of other defined sections in the file. The overlay is usually used to store additional data or code.

3.2 Threat Model

We now formulate the malware variant generation problem. Let $f(\cdot)$ be a malware detector, $x \in \mathcal{X}$ be a malware sample, and $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ be a finite set of manipulating actions, where n is the number of actions. We aim to find a minimal manipulation sequence $\pi^* \subseteq \pi$ that can bypass the malware detector. This process should satisfy the following constraints: (i) the malware behavior function $\mathcal{F}(\cdot)$ should remain unchanged; and (ii) both query consumption q and evasion cost should not exceed the upper bounds Q and α. Formally, we seek a minimal evasive manipulation sequence:

$$\begin{aligned} &|\pi^*| = \min_{\pi \subseteq \mathcal{A}} |\pi| \\ &s.t. f(x) \neq f(x+\pi),\ \mathcal{F}(x) = \mathcal{F}(x+\pi) \\ &q \leq Q,\ c(x+\pi, x) \leq \alpha, \end{aligned} \tag{1}$$

where $|\pi|$ is the cardinality of π. Moreover, the evasion cost function $c(\cdot)$ is used to calculate the relatively increased file size of adversarial malware x^* after applying π, and is computed through:

$$c(x+\pi, x) = \frac{|x^*| - |x|}{|x|}. \tag{2}$$

Suppose $\psi : \mathcal{X} \to Z \subset \mathbb{R}^n$ is the feature extraction function of a malware detector that maps a malware sample x to a feature vector $z \in Z$ and $g_y(\psi(x))$ shows the prediction score in classifying x to class y, where $y = 1$ shows x is a malware sample and vice versa. In this study, we determine the minimal manipulation sequence π^* presented in Eq. (1) by computing a minimal sequence that maximizes the benign prediction score of the mutated malware:

$$\begin{aligned} &\pi^* = \operatorname*{argmax}_{\pi \subseteq \mathcal{A}} g_{y=0}(\psi(x+\pi)) \\ &s.t. f(x) \neq f(x+\pi), \mathcal{F}(x) = \mathcal{F}(x+\pi) \\ &q \leq Q,\ c(x+\pi, x) \leq \alpha. \end{aligned} \tag{3}$$

4 METHODOLOGY

This paper proposes MalwareTotal (as opposed to the integrated anti-virus tool VirusTotal[64]), a black-box attack that leverages the bytes that do not affect malware functionality. This approach generates sequential bypass tactics that embed malicious behaviors within multi-faceted manipulations, as detailed in Fig. 1. MalwareTotal takes two steps to find malware variants that are classified to be benign by anti-malware software. First, it builds a comprehensive manipulation set; second, it searches the vulnerable manipulation sequence, leveraging proximal policy optimization.

Table 1: Implemented novel and state-of-the-art manipulations within MalwareTotal

Category	Manipulation	Description	Refs
Header Manipulation	1	Extend the DOS Stub by one file_alignment (at large 6*file_alignment) and inject random bytes.	[19, 22, 38]
	2	Set the member *CheckSum* to zero.	[11, 22]
	3	Change the member *TimeDateStamp* in File Header to a *int* value by random selection that can be zero or any datetime after 1970-1-1.	[67]
	4	Set the *IMAGE_DIRECTORY_ENTRY_DEBUG* member to zero.	[2, 3]
	5	Modifiy the *MajorLinkerVersion*, *MinorLinkerVersion*, *MajorOperatingSystemVersion*, *MinorOperatingSystemVersion*, *MajorImageVersion*, *MinorImageVersion* respectively.	This work
	6	Zero out the certification in the Optional Header.	[11, 61]
Section Manipulation	7	Rename a section by randomly selecting a section name extracted from benign binaries.	[3, 22]
	8	Import a library that has never been called.	[20, 34]
	9	Counterfeit the IAT and INT in the *Data Directory*. Copy the original Import Address Table (IAT) and Import Name Table (INT) to a position at the end of the final section, but before the overlay.	This work
	10	Create a section and inject content extracted from benign binaries' sections.	[2]
	11	Create a section and inject content, the injected content is extracted from benign binaries' strings.	[61, 67]
	12	Inject bytes in unused space in a section. Bytes injected are extracted from benign PE's sections.	[20, 71]
	13	Inject benign content before the first section.	[22]
Overlay Manipulation	14	Append random bytes to the input overlay.	[2, 11]
	15	Append benign content extracted from a random benign binary's section to input malware overlay.	[20, 41]
	16	Append benign hexadecimal data extracted from a random benign binary to input malware overlay.	[3]
	17	Append ASCII code converted from strings extracted from a random benign binary.	[27]
Compound Manipulation	18	Change all modifiable bytes in DOS Header, DOS Stub, Optional Header, and Section Table to random bytes.	[38]
	19	Alter 58 modifiable bytes in DOS Header to random bytes.	[19]
	20	Alter modifiable bytes in DOS Header and DOS Stub to random bytes.	[22]
	21	Conduct ensemble header manipulations (i.e. manipulations 1 to 6).	This work
	22	Conduct ensemble sectional manipulations (i.e. manipulations 7 to 13).	This work
	23	Compress and encrypt the input, also refers as instruction reordering or code randomization.	[2, 3, 20, 45, 61, 65, 67]

4.1 Multi-faceted Functionality-preserving Modification

In this section, we discuss MalwareTotal's manipulations within the context of the Windows PE file format. In principle, there are more than a hundred modifications that can be applied to the input malware. We categorize these into four groups: header, section, overlay, and compound manipulations. During the functionality-preserving test of possible modifications, we found that not all modifications are feasible due to various inherent constraints. We now illustrate the reasons, for each of the PE file structures.

Header Manipulations. In a 64-byte-long DOS Header, all bytes are modifiable except *e_magic* and *e_lfanew*, and all bytes in *DOS Stub* are modifiable since it won't execute on systems above win32. Theoretically, *DOS Stub* can be extended indefinitely. However, in practice, arbitrary modification of the *DOS Stub* can cause the program to crash due to an unannounced upper limit at six times the *file alignment*. Three members in the *NT File Header* and five in the *NT Optional Header* could theoretically be modified arbitrarily.

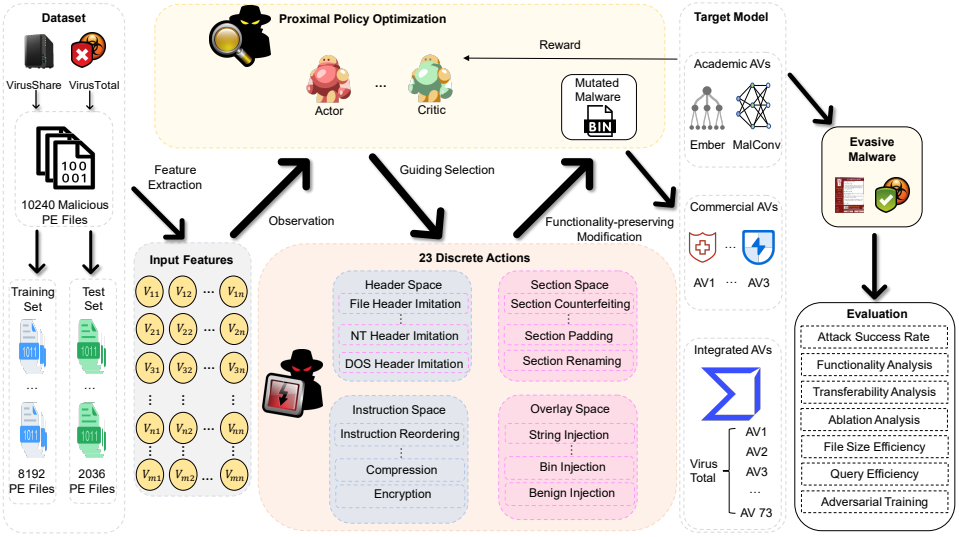

Figure 1: The overview of the proposed method MalwareTotal, a black-box attack for static PE malware detection.

However, our findings indicate that only six of these are conditionally modifiable, where member *MajorLinkerVersion* optional values are 2, 6, 7, 9, 11, 14; *MinorLinkerVersion* optional values are 0, 16, 20, 22, 25; *MajorOperatingSystemVersion* optional values are 4, 5, 6, 10; *MinorOperatingSystemVersion* optional values are 0, 1, 3; *MajorImageVersion* optional values are 0, 1, 5, 6, 10; and *MinorImageVersion* optional values are 0, 1, 3. Assigning any other value to the members will trigger an abnormal termination of the program.

Section Manipulations. This portion involves operations such as appending, shifting, and filling sections. Our augmentation concentrates on two aspects. First, with respect to functionality, we noticed that the previous implementations do not incorporate an upper limit check when inserting a section. Second, while most manipulations inject random bytes, we were inspired by Zhou et al. [72] to refine this process by injecting benign content.

Overlay Manipulations. Appending data to the overlay should not disrupt functionality, because the overlay will not be mapped to memory. However, it should be noted that directly overwriting the original overlay can lead to crashes, particularly when malware is assembled from the Nullsoft Scriptable Install System (NSIS) or other tools with integrity checking. In such cases, appended bytes can trigger an alert, obstructing the program's execution. We overcome this obstacle by appending an */NCRC* (No Cyclic Redundancy Check) execution prefix.

Compound Manipulations. This portion is related to a collection of modifications in headers, sections, and overlays. Considering that single manipulation could be query-consuming, we thus design several compound manipulations derived from different degrees of granularity. For manipulations whose positions of injected bytes are discrete (*Manipulation* 21), we inject random bytes, and for manipulations whose positions of injected bytes are continuous (*Manipulation* 22), we inject bytes extracted from benign binaries.

4.2 Malware Variant Generation Optimization

To solve the problem proposed in Eq. (1), we model the problem as a Markov Decision Process (MDP) that contains five components: (i) a finite set of states S, where each state $s_t (s_t \in S)$ refers to the state of the agent at the timestep t; (ii) a discrete action set $\mathcal{A}$, where each action $a_t (a_t \in \mathcal{A})$ represents the action of an agent at the timestep t; (iii) the transition probability $\mathcal{P}(s_{t+1} \mid s_t, a_t)$, which represents the probability that the agent transits from state $s_{(t)}$ to $s_{(t+1)}$ by taking action a_t; (iv) a reward function $R(s_t; a_t)$ that denotes the expected reward if the agent takes action a_t at state $s_{(t)}$; and (v) a scalar discount factor $\gamma \in [0, 1]$ in which lower values place more emphasis on immediate rewards.

With the MDP above, the ultimate goal of this model is to train an agent to find an optimal policy that could maximize the expectation of the total rewards over a sequence of manipulations selected, denoted as:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E}[R \mid \pi], \tag{4}$$

mathematically, this could be accomplished by maximizing *state-value function* $V\pi(s)$, defined as:

$$V_\pi(x) = \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid x_0 = x, \pi\right\}, \tag{5}$$

or it could be solved by *action-value function*, defined as:

$$Q_\pi(x, a) = \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid x_0 = x, a_0 = a, \pi\right\}. \tag{6}$$

The *state-value function* describes how good a state is for an agent to be in, and the *action-value function* indicates how good it is for an agent to conduct the action a while being in state s. To learn a policy network for an agent, the policy gradient algorithm defines an objective function $J(\theta) = \mathbb{E}_{s_0, a_0, \ldots \sim \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r^{(t)}\right]$, which is the expectation of the entire discounted rewards. In order to obtain parameters θ, the policy gradient iteratively applies stochastic gradient-ascend to find a local maximum in $J(\theta)$. On the basis of [40], for any differentiable policy $\pi_\theta(s, a)$, the policy gradient can be denoted as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a) Q_{\pi_\theta}(s, a)\right]. \tag{7}$$

By approximating Q_{π_θ} through a deep neural network and taking the action with the highest Q-value, it is possible to maximize cumulative reward and obtain an optimal policy π^* for an agent to generate malware variants, as has been demonstrated in [11]. However, this optimization approach has two deficiencies, inefficient sampling methods and overestimation of parameters and thus requires a large amount of computation time. To address the above limitations, recent research introduces an advantage function, that measures the difference between the Q value for action a in state s and the average value of that state [28], denoted as:

$$\begin{aligned} &\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A_{\pi_\theta}(s)\right], \\ &A_{\pi_\theta}(s,a) = Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s). \end{aligned} \tag{8}$$

Through this advantage function, an augmentation over the average action taken at the current state can be acquired during malware variant generation, as has been shown in [2]. Nevertheless, recent research indicates that the actor usually experiences enormous variability, which influences the performance of the trained agent [68]. Therefore, we introduce the Proximal Policy Optimization (PPO) Algorithm in this work, which utilizes importance sampling to achieve asynchronous sampling and adaptive Kullback-Leibler divergence (KL-divergence) [1] to stabilize the agent training. As discussed in [58], the PPO objective function is denoted as:

$$\begin{aligned} &\text{maximize}_\theta\, \mathbb{E}_{(a_t,s_t)\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta\,(a_t \mid s_t)}{\pi_{\theta\text{old}}\,(a_t \mid s_t)} A_{\pi_{\theta_{\text{old}}}}\,(a_t,s_t)\right] \\ &\text{s.t. } \mathbb{E}_{s_t\sim\pi_{\theta_{\text{old}}}}\left[D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}\,(\cdot \mid s_t)\,\|\pi_\theta\,(\cdot \mid s_t)\right)\right] \le \delta \end{aligned} \tag{9}$$

where $\pi_{\theta old}$ is the old policy obtained from off-policy sampled distribution. $D_{KL}(\pi_{\theta old}||\pi_\theta)$ refers to the KL-divergence between distribution $\pi_{\theta old}$ and π_θ, while $A_{\pi_{\theta\text{old}}}$ represents the advantage function in Eq. 8. By solving Eq. 9, the new policy π_θ can be obtained. Note that in actual implementation, PPO does not require an additional neural network to approximate *action-value function*, because it is deduced by replacing the KL-divergence with a clipped objective function:

$$\begin{aligned} &\sum_{(s_t,a_t)} \min\Big(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta\text{old}}\,(a_t \mid s_t)} A_{\pi_{\theta\text{old}}}(s_t,a_t), \\ &\text{clip}\left(\frac{\pi_\theta\,(\mathrm{a_t} \mid \mathrm{s_t})}{\pi_{\theta_{\text{old}}}\,(\mathrm{a_t} \mid \mathrm{s_t})}, 1-\varepsilon, 1+\varepsilon\right) \mathrm{A}_{\pi_{\theta\text{old}}}\,(\mathrm{s_t},\mathrm{a_t})\Big) \end{aligned} \tag{10}$$

where $clip(\cdot, 1-\epsilon, 1+\epsilon)$ denotes clipping the output to the range of $[1-\epsilon, 1+\epsilon]$, in case an excessive variance between $\pi_{\theta\text{old}}$ and π_θ.

Furthermore, the reward function and environments are critical in calculating state transitions and reward propagation. As discussed in [4, 54], we utilize a 2381-dimensional vectorized environment for examining EMBER and a 1024×1024 box environment for MalConv. For production-level AVs, we inherit an EMBER vectorized environment due to more robust posterior performance. The reward function contains the following values: (i) the reward equals β when the malware variant bypasses the target detector; (ii) the reward is zero if the malware variant does not evade when the query budget is reached; and (iii) the reward is equal to the difference between the initial malicious score and in-process score.

5 EVALUATION

In this section, we provide a comprehensive explication of MalwareTotal, testing the performance of the proposed attack under ML-based, signature-based, and hybrid anti-malware software. We examine the functionality of adversarial malware generated by our method through a sandbox, then we compare the performance of MalwareTotal with 10 of the latest methods. We then showcase a MalwareTotal transfer attack performance on an integrated commercial malware scanner to show the superiority of our method in producing adversarial malware. We first present our experiment setup. Then, we discuss the design of our experiments, and finally evaluate the performance of MalwareTotal by answering the following research questions:

- **RQ1: Effectiveness Analysis.** *How successful and minimal are the generated variants against ML-based, signature-based, and hybrid anti-malware software?*
- **RQ2: Mutation Efficiency Comparisons.** *Compared with other methods, how efficient is the mutation of MalwareTotal?*
- **RQ3: Ablation Study.** *To what extent do sequence-aware optimization and multi-faceted manipulations independently contribute to the MalwareTotal's bypass capability?*
- **RQ4: Transferability Assessment.** *Can mutated malware generated by MalwareTotal also evade other anti-virus software?*
- **RQ5: Robustness Enhancement.** *Can we utilize MalwareTotal-generated samples to enhance the robustness of the model?*

5.1 Target AVs, Data, and Benchmarks

In this section, we illustrate our experimental setup and identify the target state-of-the-art malware detection techniques that we employed for comparison, all of which are well-cited in academia or provided by premier anti-virus companies.

Detection Tools. We select six malware detection tools to evaluate MalwareTotal: two prevalent ML-based research tools, EMBER[4] and Malconv[54], one signature-based tool, *ClamAV*[3], two real-world commercial anti-virus products, anonymized as AMS A and AMS B, and the online scanner VirusTotal[64]. The first five AVs are employed to measure the effectiveness and efficiency of MalwareTotal. VirusTotal is used to assess the generated malware transferability. In subsequent discussions and depictions, EMBER and MalConv are referred to as ML-based AVs, while the ClamAV, AMS A, and AMS B are denoted as production-level AVs, because the ML-based tools provide soft labels (probability scores) in their scan results, whereas the production-level AVs only output hard labels of "malware" or "benign". Our evaluation uses the version of the three production-level tools as of January 4th, 2023. Their features are presented as follows.

Detector 1: EMBER, a gradient boosting decision tree based malware detector, has been chosen due to its extensive usage in the field of adversarial malware generation research [13, 20, 56, 66]. We use the trained model in SEC-ML [18], with a malicious threshold of 0.8336, corresponding to a False Positive Rate (FPR) of 0.039 and a True Positive Rate (TPR) of 0.95. This choice of malicious threshold follows the original author's suggestion in [4], ensuring that our results are commensurate with previous research.

Detector 2: MalConv, a convolutional malware detection model, has become an industry standard due to its capability to analyze entire executable files for potential threats [5]. The original MalConv model considers a maximum input file size of 2MB, however

[3] https://www.clamav.net/

the model used in our experiments relies on the reproduction in ToucanStrike [17], which is trained on the EMBER dataset with a maximum input file size of 1 MB. Files exceeding the maximum allowable size are truncated, while shorter files are padded using a special padding token separated from the standard bytes in the file. Utilizing a malicious threshold of 0.5 derived from the original MalConv model [54] and subsequent replications [20, 36, 61], we recorded a False Positive Rate (FPR) of 0.035 and a True Positive Rate (TPR) of 0.69.

Detectors 3: We select ClamAV, a signature-based detection technique, due to its widespread acceptance and use in the industry as a reliable malware detection tool. It relies on a continuously updated database of known malware signatures for file scanning and identification, providing a comprehensive library of potential threats. ClamAV offers two types of rules in the virus database, MD5 file checksums and embedded virus signatures, we decided to utilize both of them in our experiments to ensure a robust and thorough verification process.

Detectors 4-5: AMS A and AMS B are both rewarded top-rated in *The Best Windows Antivirus Software for Home Users 2022* by AV-Test [6]. While the two commercial anti-malware products do not make the details of their techniques public, following assumptions from earlier works [72], we assume they apply hybrid detection techniques. We use the version of the local installation and we believe these selections are both representative of software with an advanced anti-virus detection capability.

Detector 6: VirusTotal is an online AV scanner that integrates over 70 AV products for PE malware detection.

Metrics. We evaluate MalwareTotal performance and compare it with the performance of ten of the latest methods, using five key metrics: **Attack Success Rate** (ASR), **Query Budget** (Q), **Evasion Cost** (α), **Time Consumption** (T), and **Functionality**. Here, ASR refers to the percentage of malware samples in the test set that, after undergoing mutation, are classified as benign. The query budget indicates the total number of calling anti-virus detectors during generation. Note that a query is performed after a manipulation is applied to the malware. The evasion cost is the average of post-modification malware size, showing how many bytes are injected during the generation. We define the **functionality** via a binary metric, which measures whether the generated adversarial example is both executable and malicious.

Baselines. In order to evaluate the efficacy of MalwareTotal, we compare it with ten recent methods in adversarial malware generation: five heuristic or genetic algorithm-based methods, *MAB* [61], *Padding* [22], *Aimed* [11], *Gamma Section* (GS), and *Gamma Padding* (GP) [20], and five gradient-based methods, *Full Dos* (FD) [22], *Partial Dos* (PD) [19], *Header Fields* (HF) [3], *Shift* [22], and *Extend* [22]. Nine of these methods were selected because they are highlighted in a comprehensive review of adversarial malware generation [22]; the tenth, the *MAB* method, is included due to its novel viewpoints, which were presented after the publication of that review. For detailed descriptions of each method, please refer to the §2 related work.

Dataset. Many existing malware detection datasets like Sorel-20M [31] and MS BIG 2015 [37] only provide preprocessed features (e.g., header information of the file, imported and exported functions) instead of the original PE files. However, evaluating the functionality of mutated malware requires access to the original samples. To address this need, we have chosen to utilize the VirusShare [63] and VirusTotal Academic Share [14] datasets, both of which contain the complete PE files.

We construct the experimental dataset as follows. First, we collect a composite set of malware samples from the first three zip files in VirusShare and all samples from VirusTotal. Then, we select samples that are unanimously identified as malicious by the first five AVs, resulting in a collection of 46,229 malware samples. Based on the data scale of [71], which uses a test set comprised of 2,036 samples, we randomly select 2,036 samples from our refined composite dataset. Finally, upholding the conventional machine learning practice of an 8:2 train-to-test ratio, we randomly choose another 8,192 samples from the filtered set to form our training set.

Implementation. We implement MalwareTotal using PyTorch 1.8.2 and stable-baselines3 [55]. To manipulate PE files, we rely on PEfile [9], which is based on Python 3.6.2. We run all our experiments on a Windows server configured with a 2.9 GHz Intel Xeon 6226R CPU, 2×48G NVIDIA RTX A6000 GPU, and 256GB memory.

5.2 RQ1: Effectiveness Analysis

We start our experiments by exploring the limitations on evasion cost α and query budget Q. We evaluate the performance of MalwareTotal by varying evasion costs (α) between 0.5 and 6. Then, the query budget is set, grounded in three particular empirical observations. First, to ensure a breadth of manipulations, the length of the manipulation sequence is preset to 30, accommodating at least one instance of each manipulation along with some repetition. Second, during initial training stages, the model incessantly selected *Manipulation 23* rather than devising an effective sequence. Therefore, we extended the sequence length to 31. The additional position mandates conducting *Manipulation 23*, while the preceding 30 are reserved for learning an optimized sequence from *Manipulation 1-22*. Third, owing to the environmental dynamics, multiple episodes enable the model to acquire experience across varied instances [32]. Therefore, because each malware sample afforded 3 episodes (restart opportunities) and each sequence up to 31 perturbations, our resulting query budget is set to 3*(30+1) = 93. Then we randomly select 200 malware samples (since 24 independent experiments are required) from the test set to explore the relation between α and evasion capacity. With a varying maximum evasion cost α, we compare the ASR, the actual evasion cost (α'), queries (query consumption), and time consumption, as shown in Table 2. The ASR increases linearly and achieves optimal ASR when α reaches 4.5 and 5.5 respectively against EMBER and MalConv. Thus, we set α to 5 in all follow-up experiments. Notably, the average of actual evasion cost α' is far less than α in both EMBER and MalConv, indicating that only edge cases require MalwareTotal to inject a relatively large number of bytes into malware.

We then evaluate the effect of different query budgets on the ASR with comparison methods, by unifying the upper query amount for each malware sample from 10 to 500 and examining the attack success rate with 2,036 malware samples. The results, which are presented in Fig. 2, indicate that, the attack success rate of most

Table 2: Malwaretotal performance under different evasion cost α

EMBER-α	ASR(%)	Queries	Time	EMBER-α'	MalConv-α	ASR(%)	Queries	Time	MalConv-α'
0.5	79.00	**1120**	**10mins31s**	**0.15**	0.5	54.00	**363**	14mins54s	0.40
1	84.50	2026	17mins5s	0.21	1	85.00	398	15min15s	0.39
1.5	89.00	2620	21mins26s	0.20	1.5	93.50	415	14mins27s	0.42
2	94.50	2689	21mins33s	0.25	2	98.50	420	14mins20s	0.44
2.5	96.00	2777	21mins53s	0.24	2.5	98.50	428	15mins12s	0.48
3	96.00	2803	22mins22s	0.22	3	99.00	440	15min54s	0.49
3.5	96.00	3098	24mins11s	0.24	3.5	99.00	441	15mins42s	0.46
4	96.50	3105	29mins16s	0.24	4	99.00	415	13mins42s	0.44
4.5	**98.50**	2970	28mins1s	0.25	4.5	99.50	410	13mins22s	0.46
5	**98.50**	3049	28mins54s	0.24	5	99.50	477	17mins52s	0.49
5.5	**98.50**	3098	31mins8s	0.24	5.5	**100.00**	448	17mins2s	0.46
6	**98.50**	3031	30mins38s	0.24	6	**100.00**	492	19mins35s	0.52

compared methods gradually increases and reaches optimal ASR at 100 queries per sample in both detectors. By contrast, our method achieves an optimal ASR at the very beginning of each comparison.

Further, we measure the performance of MalwareTotal and the ten comparison methods on EMBER [4] and MalConv [54] detectors, since they both are ML-based and have been widely applied in recent research. We train our method with 8192 malware samples, with the query budget Q=93 and evasion cost c=5. The optimization terminates when the last sample in the training set is traversed. Then we test and report the performance with the test set of 2,036 malware samples. Table 3 illustrates the performance of MalwareTotal with respect to the first two detectors (i.e., EMBER and MalConv). As can be seen from Table 3, MalwareTotal achieves an effective evasion performance, with 98.67% and 100% ASR against EMBER and MalConv, respectively. Note that the average file size of each malware sample in the test set is 315.8KB, and MalwareTotal increases only an average of 124.8 and 159.5 KB, respectively, to create adversarial malware. This demonstrates that MalwareTotal can robustly bypass ML-based detectors.

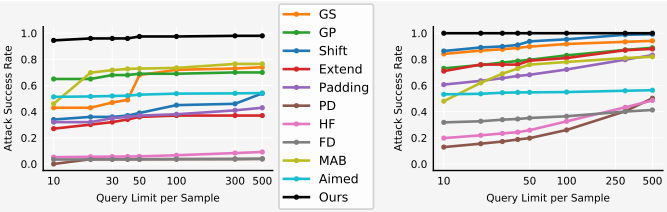

Figure 2: Comparison of Query-Limit Bypassing Results against EMBER (left) and MalConv (right) Detectors.

Table 3: Ablation results of optimization algorithms against ML-based malware detectors, where q and t stand for query and time consumption, α' stands for the evasion cost.

Methods	EMBER				MalConv			
	ASR(%)	q	α'	t	ASR(%)	q	α'	t
MalwareTotal	**98.67**	14028	0.40	5.8h	**100**	**4459**	0.51	**1.02h**
MT-TRPO	96.22	**11680**	**0.35**	10.62h	**100**	7239	0.50	1.83h
MT-A2C	89.29	21106	0.77	**2.28h**	97.24	15363	**0.41**	2.3h
MT-DQN	59.38	24129	1.26	13.77h	59.38	37838	0.57	60.35h
MT-Random	62.13	38827	0.92	13.37h	93.66	42267	0.67	11.27h

Next, we evaluate the performance of MalwareTotal against production-level anti-virus (AV) software. This is a more challenging task for two reasons: (i) In production-level AVs, the prediction score is no longer available to users. Instead, AVs only provide binary scan results, which is known as the *Hard-label Attack* [20, 61]; and (ii) production-level AVs contain a large malware signature cloud and usually employ multi-module detection technologies.

In this experiment, we use the 2381-dimensional EMBER environment to overcome the absence of the prediction score, as it performed better in our fundamental performance test. As MalwareTotal needs iterative interactions with target AVs, we use the locally installed premier edition of AMS A and AMS B, which provide the Command-Line (CMD) Scan Mode. While CMD Scan Mode is not a standard requirement across all antivirus software, it significantly simplifies the automated collection of scan results during our testing process. In addition, we set the *Scan sensitivity* to medium to balance scan ability and time efficiency. For ClamAV, we scan mutated malware with the default setting,[4] which we believe offers the most unbiased, real-world representation of how these antivirus products operate out-of-the-box. The results are reported in Table 4. MalwareTotal achieves ASRs of 95.33%, 92.63%, and 98.52%, an average of 156.26KB of injected content, and up to 21 average manipulations, respectively. This indicates that with an experienced agent, production-level AVs can be robustly bypassed with tactics generated by MalwareTotal.

To demonstrate that MalwareTotal fully preserves malicious functionality, we validate the functionality of mutated malware as follows. We first ensure that the original malware is executable and malicious using Cuckoo Sandbox [26] on a 32-bit Windows 7 virtual system. Specifically, we inspect whether the *executable* field is reported as *Ture* and then record the content of signatures in a JSON file. Then, we test the functionality of individual manipulations by randomly selecting 50 malware samples from our test set and iteratively modifying each sample from one manipulation to up to 30 manipulations to simulate the most extreme mutation scenario. This results in 33,000 mutated malware samples, which are tested for functionalities with the sandbox by determining whether all *signatures* of the original samples are presented within the *signatures* of the corresponding mutated samples. The results show that all the mutated malware retained their malicious behaviors.

Then we demonstrate that overlapped manipulations do not affect malware functionality. Since it takes minutes to verify the functionality of a mutated sample, we randomly selected 200 mutated malware samples that have evaded ML-based detectors (i.e., 400 samples per method, 4,400 samples in total) and validated their functionalities using Cuckoo Sandbox. As shown in Table 5, all tested samples were executable and exhibited malicious functions in the sandbox. Based on the results of the two functionality validations, we believe that MalwareTotal is capable of preserving the functionality of mutated malware.

[4] https://docs.clamav.net/manual/Usage/Scanning.html#clamscan

Answer to RQ 1: MalwareTotal achieved an ASR of 100% against MalConv, 98.67% against EMBER, and 95.33% against ClamAV, 92.63% against AMS A, and 98.52% against AMS B. This indicates that MalwareTotal is capable of generating robust tactics that can consistently bypass ML-based, signature-based, and commercial AVs.

5.3 RQ2: Bypass Efficiency Comparisons

We now evaluate the bypass efficiency of MalwareTotal. We compare MalwareTotal against ML-based detectors (i.e., *Detectors 1-2*) with ten recent works. For *Gamma Section*, *Gamma Padding*, *MAB*, and *Aimed*, we use the code released by the official source. For other methods, our reproduction relies on an integrated framework, *ToucanStrike* [17]. We follow the original settings presented in each paper. All comparisons are conducted on the same test set. To obtain maximum performance in ASR, we set the evasion cost $\alpha = 5$, and query budget $Q = 500$ for each method. Note that most compared methods are heuristic. We repeatedly run each experiment three times and report the average result in Table 5.

We can see from the results that MalwareTotal outperforms compared methods in ASR in evading both EMBER and MalConv. Although *MAB* consumes fewer queries and less time than the proposed method, MalwareTotal has huge advantages in ASR and evasion cost. As for MalConv, MalwareTotal achieves 100% ASR with the lowest time and query consumption. Considering that production-level AVs consume extreme amounts of time in scanning progress, in further experiments, we select to use only the *Gamma Section* as a comparison in further experiments, because it performs well in both EMBER and MalConv. We replace the detectors with ClamAV, AMS A, and AMS B, and iteratively generate mutated malware utilizing *Gamma Section*. As shown in Table 4, *Gamma Section* achieves only 36.11%, 17.33%, and 14.93% ASR, respectively. By contrast, MalwareTotal bypasses the three production-level AVs with 95.33%, 92.63%, and 98.52% ASR, respectively. Moreover, as the scanning process of production-level AVs usually takes anywhere from seconds to minutes, an effective optimization can be a highly beneficial feature of a model. As is reported in Table 4, MalwareTotal not only consumes 68.7, 63.1, and 108.7 hours less than *Gamma Section*, while injecting less content in evading AMS A and AMS B.

Answer to RQ2: The baseline methods perform well in bypassing ML-based detectors but are ineffective in bypassing commercial anti-virus software. In contrast, MalwareTotal achieves the highest ASR with the lowest time and queries consumed when bypassing different malware detectors.

5.4 RQ3: Ablation Study

Manipulation diversity ablation study. To validate the fundamental hypothesis that multi-faceted manipulations enhance bypass capability, we conduct an ablation study by excluding each type of manipulation from the action set and analyzing the ensuing impact on performance. In the first experiment, we exclude ten manipulations (*Manipulation* 1-6 and 18-21) associated with header manipulation, denoted as Header Manipulation Removed (HM-R). Similarly, experiments excluding section manipulations (*Manipulation* 7-13 and 22) are designated as Section Manipulation Removed (SM-R). Experiments removing overlay manipulations (*Manipulation* 14-17) are termed Overlay Manipulation Removed (OM-R). Additionally, the experiment eliminating instruction manipulation (*Manipulation* 23) is labeled as Instruction Manipulation Removed (IM-R). We select IM-R as the fourth experiment instead of eliminating compound manipulations to investigate whether packing methods can solely conceal malicious functionality.

Next, we evaluated the bypass capability by applying only one type of manipulation, i.e., header, section, overlay, or instruction manipulation, referred to as Header Manipulation Only (HM-O), Section Manipulation Only (SM-O), Overlay Manipulation Only (OM-O), and Instruction Manipulation Only (IM-O). We revisit the performance of MalwareTotal using the same 200 malware samples as in RQ1. We compute the average attack success rate (Avg-ASR) by averaging the ASRs of each method across *Detectors* 1 to 5. Additionally, three metrics are utilized: average Q (Avg-Q), average α (Avg-α), and average time consumption (Avg-**T**). To be specific, we first calculate the mean query consumption, evasion cost, and time for each generated evasion malware. This is achieved by dividing the sum of these metrics by the number of evasive malware generated. Then, we obtain Avg-Q, Avg-α, and Avg-**T** by averaging the computed metrics across AV1 to AMS B. The ablation results are reported in Table 6.

Without loss of generality, removing any type of manipulation leads to a decrease in ASR and an increase in query budget, evasion cost, or time consumption. On the other hand, although different detectors show fragility to different types of manipulation, relying solely on a single type of manipulation cannot achieve satisfactory evasion performance. In conclusion, our experiments demonstrate that the diversity of manipulations is essential to reinforcing bypass capability and relying solely on a single type of manipulation is not sufficient to achieve satisfactory evasion performance.

Sequence-aware bypassing ablation study. To demonstrate the effectiveness of sequence-aware manipulations, we randomly select 200 malware samples that have bypassed each detector. Through recording the bypass sequence during mutation, we randomly switch the manipulation order and re-manipulate the original malware with the switched manipulation sequence. Then we use each detector to examine whether the re-manipulated malware can again bypass detection. The attack success rate decrease for the re-manipulated malware decreased 47.0%, 10.2%, 99.0%, 99.0%, and 97.0% against each anti-malware software. The decrease in attack success rate when bypassing EMBER suggests that executing each manipulation in the proper order plays a significant role in bypass tactics. The results in MalConv can be explained by MalConv's sensitivity to all kinds of manipulation. In addition, the significant disparity in ASR reduction between ML-based (47.0%, 10.2%) and production-level (99.0%, 99.0%, 97.0%) anti-malware software indicates production-level AVs are much more sequence-sensitive than ML-based methods. This could explain why the comparison methods fail to generalize to production-level AVs, as they focus on searching the minimal injected bytes rather than the vulnerable sequences. Therefore, we conclude that the proposed sequence-aware manipulation is the key to bypassing production-level AVs.

Table 4: Comparisons between individual attack strategies and MalwareTotal against commercial anti-virus software.

Methods	ClamAV				AMS A				AMS B			
	ASR	Queries	Evasion Cost	Time	ASR	Queries	Evasion Cost	Time	ASR	Queries	Evasion Cost	Time
MalwareTotal	**95.33**	**36491**	0.48	**45.9h**	**92.63**	**42756**	**0.49**	**52.2h**	**98.52**	**7240**	**0.08**	**24.0h**
Gamma Section	36.11	61839	**0.18**	114.6h	17.33	61057	0.52	115.3h	14.93	53244	0.55	132.7h
MT-Random	40.57	51680	0.63	88.4h	23.87	61027	0.37	115.2h	66.10	59316	0.58	147.8h

Table 5: Comparisons between individual tactics and MalwareTotal against ML-based malware detectors.

Methods	EMBER				Functionality	MalConv			
	ASR(%)	Queries	Evasion Cost	Time		ASR(%)	Queries	Evasion Cost	Time
MalwareTotal	**98.67**	**14028**	0.40	5h48mins	**100%**	**100**	**4459**	0.51	**1h1min**
Full DOS	3.98	22968	2.83	**3h22mins**	98.50%	41.29	38737	0.15	39h1min
Partial DOS	3.88	20805	0.94	3h26mins	27.07%	50.26	41907	**0.12**	17h45mins
Extend	37.03	24367	3.60	5h45mins	72.00%	88.00	34666	3.81	164h22mins
Shift	54.00	25144	3.90	16h18mins	97.00%	99.22	30789	3.78	75h22mins
Padding	43.00	22968	3.86	12h19mins	100%	83.24	30251	3.73	105h25mins
Header Fields	9.18	19363	**0.21**	128h59mins	100%	48.58	38984	0.27	30h44mins
GAMMA Section	74.26	26738	0.63	6h25mins	95.00%	94.15	32295	0.50	5h12mins
GAMMA Padding	70.00	26176	0.58	5h28mins	99.00%	88.85	28489	0.33	4h11mins
MAB	76.52	24286	3.96	5h30mins	92.00%	82.00	22203	3.56	4h28mins
Aimed	54.25	73642	0.77	19h13mins	100%	56.40	73624	0.76	12h3mins

Table 6: Action ablation results of MalwareTotal against all malware detectors

Method	Avg-ASR	Avg-Queries	Avg-Evasion-Cost	Avg-Time (mins)
MalwareTotal	**96.20**	**17.62**	**0.0026**	**2.51**
HM-R	90.70	24.15	0.0030	3.70
SM-R	81.70	61.81	0.0042	5.52
OM-R	71.50	126.31	0.0061	4.91
IM-R	63.30	125.56	0.0047	15.54
HM-O	18.50	262.84	0.0070	20.02
SM-O	26.80	543.58	0.0201	21.29
OM-O	49.50	254.87	0.0076	18.09
IM-O	24.60	6.56	0.0133	1.20

Optimizer ablation study. To determine the optimization ability of MalwareTotal, we replace the *Proximal Policy Optimization* with three other gradient-based optimization algorithms, *Trust Region Policy Optimization* (TRPO) [57], *Advantage Actor Critic* (A2C) [49], *Deep Q-learning* (DQN) [50], but use the same MalwareTotal 23 actions (namely MT-TRPO, MT-A2C, MT-DQN). We also utilize an agent which selects random actions during mutation (namely MT-Random), to assess the performance under a no-optimization scenario. Results are reported in Table 3, we show that MalwareTotal achieves the highest ASR both in EMBER and MalConv. Although TRPO achieves less evasion cost and query consumption in bypassing EMBER, it performs weaker in ASR and consumes more than 10 hours, while our method only takes 5.8 hours.

Answer to RQ 3: The design of sequence-aware tactics and multi-faceted manipulation are both crucial in efficiently bypassing malware detectors, as the ASR drops rapidly in the absence of either one.

5.5 RQ4: Transferability Assessment

We consider two kinds of malware transferability. One of which is malware evolving evasion, which is caused by unseen signatures or imperfect detection rules. The other is adversarial transferability [23], caused by fragile ML models. Considering that many AVs do not claim to have deployed ML-based detection technology, it is worth investigating whether mutated malware transfers among different malware detectors. This investigation is motivated by the observation that adversarial examples demonstrate strong transferability across models in computer vision (CV) fields [23]. We thus utilize 5000 malware samples generated by MalwareTotal and *Gamma Section*. Each set of 500 malware samples have evaded an individual detector, for example, the 500 malware samples generated by MalwareTotal evaded *Detector 1* are denoted as MT-D1, while the 500 malware samples generated by *Gamma Section* evaded *Detector 2* are denoted as Gamma-D2. Mutated malware is then scanned by VirusTotal so that the transferability can be examined. There are 73 anti-virus services integrated in VirusTotal, 4 of which fail to return almost all results, therefore are excluded.

We report the overall and segmented performance in Table 7, where we first present the ASR calculated through the ratio of malware that is identified as benign by 69 online AV scanners. The remaining data illustrates the quantity of AVs that are bypassed with the corresponding percentage. The results indicate that: (i) mutated malware evading ML-based detectors shows a weaker transferability against heterogeneous production-level AVs compared with mutated malware evading commercial AVs, as the ASR of MT-D1 and MT-D2 is much lower than MT-D3 to D5; (ii) MalwareTotal is better than comparison methods at exposing commercial AV vulnerabilities. When pitted against commercial AVs, MalwareTotal bypasses 73 AVs in total with 90% ASR (i.e.the sum of the third row in Table 7), while *Gamma Section* only bypasses 33 AVs; and (iii) to date, MalwareTotal is the first method that achieves an average ASR above 50% (notably 63.31% in MT-D3), revealing that the risks in the robustness of static malware detection have been overlooked.

To further demonstrate the transferability of generated samples, we report the ASR against 3 production-level AVs in RQ1 (online version) and 7 AVs that are rewarded top-rated in *The best Windows antivirus software for home users* by AV-Test in 2022. The results are reported in Table 8: among ten AV productions, only one is robust to both MalwareTotal and *Gamma Section* (i.e., AV 10). Results show that the first two AVs show no resilience to transfer attacks, as they classified all mutated malware as benign. The ASR of the third AV decays; we assume this is because the anti-virus software provides VirusTotal with different versions between the local scanner and online scanner,and usually the online scanner has a stronger ability than the local installation[73].

Answer to RQ4: Transferability of mutated malware is extensive in static malware detection whether the architecture of AV products is ML-based or not, as up to 63.31% of the tested detections in VirusTotal are vulnerable to a MalwareTotal transfer attack.

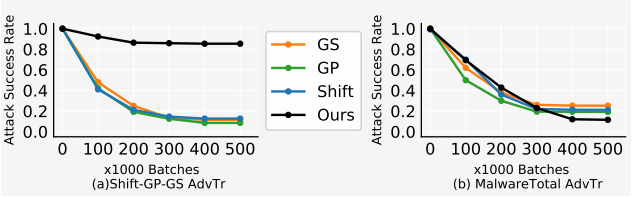

Figure 3: Comparing ASR for the adversarially trained model with a varying number of adversarial examples.

5.6 RQ5: Robustness Enhancement

In this research question, we explore the effectiveness of using adversarial training (ATr) [36, 69] as a defense against attacks. We leverage MalwareTotal to generate malware variants for the MalConv model. If the victim model predicts wrongly on an original input, we skip this example. For other comparison methods, we select three that perform better in RQ2: *Shift*, *Gamma Padding* (GP), and *Gamma Section* (GS). We expand the training set to 10,000 samples from VirusShare to ensure that every method can generate an equal number of adversarial samples. These examples are then augmented into the original training set and form the adversarial training set. We fine-tune the victim model on the adversarial training set into two different versions. The first model is trained on 10,000 MalwareTotal-generated samples and the second model is trained with 10,000 uniformly mixed samples generated by three other methods (Shift, GP, and GS).

After adversarial training, we obtain two models: *MalConv-MalwareTotal-ATr* and *MalConv-Shift-GP-GS-ATr*. Fig. 3 shows the prediction accuracy on malware variants generated in RQ2 for these new models. Note that the original victim models (which are not hardened by adversarial retraining) predict all these examples incorrectly (i.e., they have an accuracy of 0%). We can observe that all the adversarially trained models perform much better than the original ones. The average performance improvement is consistently observed when applying adversarial training to examples generated exclusively by MalwareTotal and other methods. *MalConv-Shift-GP-GS-ATr* improves accuracy against *Shift*, *Gamma Padding*, and *Gamma Section* by 87.33%, 91.50% and 88.51%, respectively. However, the model adversarially trained on examples generated by three other attacks cannot effectively detect the MalwareTotal attack, as MalwareTotal can achieve an 85.41% attack success rate on the retrained model. Conversely, *MalConv-MalwareTotal-ATr* improves accuracy against MalwareTotal by 88.51% and improves accuracy on the other three attacks (78.88%, 80.89%, and 74.81% on *Shift, GP,* and *GS*). We believe this is because the three attacks manipulate a partial file structure, while MalwareTotal obtains a comprehensive manipulation set on all file structures.

> *Answer to RQ5:* The adversarial examples generated by MalwareTotal are valuable in improving the robustness of the victim model. Adversarially training the victim model with MalwareTotal-generated adversarial examples can significantly improve the robustness of MalConv when detecting met and unseen adversarial attacks.

6 LIMITATIONS AND THREATS TO VALIDITY

Limitations. Our method has two main limitations. First, we have successfully attacked static anti-virus software, but have not yet succeeded on dynamic anti-virus software. We assume that our attacks can also be applied to bypass dynamic AVs by incorporating the anti-debug manipulations, e.g., checking hard disk temperature through the *Windows Management Instrumentation* (WMI) API [51]. Secondly, our manipulation tactics target PE format malware samples. This specificity arises from our manipulations, designed for PE file structures, rendering other formats immune.

Threats to validity. There are two main threats to the validity:

The generalizability of our experimental results. The effectiveness and generalization ability of our method are based on two assumptions: the representativeness of the samples and the completeness of the detection rules. To mitigate the first threat, we carefully design the experimental datasets, metrics, and baselines. For the datasets, we follow previous study [20] and select two representative datasets. The two datasets are collected from real-world malware and cover popular malware families (127 families contain more than 20 samples in our experiments). For the metrics, we select five widely used metrics, including the attack success rate, query budget, evasion cost, time, and functionality, all of which are widely used in existing work. To mitigate the second threat, we include a mix of detection tools in our experiments. including two ML-based malware detection models, three product-level anti-malware software and an online malware scanner. Furthermore, we run each approach three times and report the average results.

The implementation of models. It is widely known that deep learning models are sensitive to implementation details, including hyperparameters and network architectures. Our approach employs a widely adopted version of the proximal policy optimization. Due to the high training cost, we set hyperparameters (i.e., learning rate, batch size, and clip range) to the same values as those in previous studies [68]. Thus, there may be room to tune the hyperparameters of our approach to realize greater improvements. Additionally, when learning vulnerable sequences, the attack success rate varies with predefined maximum manipulation size and the number of accumulated manipulations. To mitigate those effects, we conduct experiments §5.2 to find the proper manipulation size.

7 ETHICAL IMPLICATIONS

In this work, our aim is not to facilitate malicious activities, but to utilize malware manipulations to develop defensive strategies to enhance the robustness of static anti-virus software. The generated malware variants serve as a critical step in ethical hacking, closely mirroring practices such as penetration testing. We expect this research to raise no ethical concerns because: (1) real-world malware detection consists of an ensemble of detection mechanisms, where malware static analysis often serves as the first line of defense [29]. An adversary who launches an attack needs to evade each of these components, including dynamic analysis, which infers maliciousness based on API call sequences [15] or I/O behaviors [72]; and (2) we provide mitigations for MalwareTotal. By retraining the model on MalwareTotal-generated samples, the model exhibits improved robustness in detecting unseen adversarial samples and attacks. With respect to commercial anti-virus software, we have shared

Table 7: ASR comparisons between individual attack strategies and malwareTotal against virustotal.

Methods	MalwareTotal					Gamma Section				
	MT-D1	MT-D2	MT-D3	MT-D4	MT-D5	Gamma-D1	Gamma-D2	Gamma-D3	Gamma-D4	Gamma-D5
Avg ASR	35.67	34.38	**63.31**	59.46	57.53	36.88	33.49	38.80	45.07	43.81
AVs (ASR≥90%)	10	9	**29**	**29**	15	14	12	12	11	10
AVs (ASR≥80%)	14	12	**32**	30	24	15	14	15	14	14
AVs (ASR≥70%)	16	15	34	32	**36**	18	16	17	20	17
AVs (ASR≥60%)	19	17	36	34	**40**	20	19	19	23	18
AVs (ASR≥50%)	21	20	**41**	39	**41**	25	23	22	26	26

Table 8: Comprasion results of transfer attack success rate against integrated malware detector virustotal

	MT-D1	MT-D2	MT-D3	MT-D4	MT-D5	Gamma-D1	Gamma-D2	Gamma-D3	Gamma-D4	Gamma-D5
AV1	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
AV2	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
AV3	63.38%	59.23%	**84.00%**	79.83%	79.59%	56.00%	56.00%	58.33%	71.43%	58.59%
AMS A	19.09%	15.06%	95.83%	90.18%	**97.87%**	51.06%	46.94%	41.67%	78.35%	72.63%
AMS B	11.96%	9.70%	56.00%	**76.86%**	58.00%	24.00%	20.00%	34.00%	34.00%	31.00%
AV6	18.30%	22.22%	**98.00%**	84.30%	**98.00%**	47.47%	40.00%	56.25%	70.41%	76.77%
AV7	11.99%	7.43%	**90.00%**	80.17%	**90.00%**	12.00%	6.00%	2.60%	47.00%	31.00%
AV8	40.82%	37.90%	**100.00%**	86.84%	95.45%	35.71%	32.65%	**100.00%**	74.23%	72.45%
AV9	10.36%	5.91%	**72.00%**	64.10%	67.35%	20.00%	10.00%	18.75%	43.88%	30.30%
AV10	0.19%	0.31%	0.00%	1.72%	0.00%	2.02%	0.00%	0.00%	**5.21%**	3.09%

our mutated malware with AV producers, including WinDefender [48], ESET [25], Avast [7], and others.

8 CONCLUSION

Strengthening the resilience of software systems against malware is a pressing need in software engineering. It is expected that tools such as MalwareTotal will aid in achieving this objective as part of the continual improvement efforts in software development practices. To this end, we propose a new approach that carefully collects, categorizes, and extends a range of PE file modifications that have been scattered in the literature, and explore mechanisms to systematically combine them in order to explore the space of successful malware obfuscation techniques. Our approach significantly outperforms previous approaches at avoiding detection by state-of-the-art ML-based and commercial malware detectors. We then use this approach to train ML-based malware detection tools, significantly improving their effectiveness. We believe that this work will enable future research to improve malware detectors, increasing their effectiveness against a range of obfuscation strategies. The application of MalwareTotal reflects a significant step for software engineers in the fight against evolving malware tactics, integrating the detection seamlessly into secure software development pipelines.

ACKNOWLEDGEMENT

We thank all the anonymous reviewers for their constructive comments. This work is supported by the National Natural Science Foundation of China (No. 62072200), the National Key R&D Plan of China (No. 2023YFB3106402), the CCF-NSFOCUS 'Kunpeng' Research Fund (No. CCF-NSFOCUS202202)

REFERENCES

[1] 2004. The Kullback-Leibler divergence rate between Markov sources. *IEEE Transactions on Information Theory* 50, 5 (2004), 917–921.
[2] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to evade static PE machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917* (2018).
[3] Hyrum S Anderson, Anant Kharkar, Bobby Filar, and Phil Roth. 2017. Evading machine learning malware detection. *Black Hat* (2017).
[4] Hyrum S Anderson and Phil Roth. 2018. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637* (2018).
[5] Kshitiz Aryal, Maanak Gupta, and Mahmoud Abdelsalam. 2022. A Survey on Adversarial Attacks for Malware Analysis. arXiv:2111.08223 [cs.CR]
[6] AV-TEST. 2022. *The best Windows antivirus software for home users.* https://www.av-test.org/en/antivirus/home-windows/
[7] Avast. 2023. *Avast Premium Security.* https://www.avast.com/index#pc
[8] Justin Burr. 2022. *Improving Adversarial Attacks Against MalConv.* Ph. D. Dissertation. Dakota State University, Madison, SD, USA. Advisor(s) Xu, Shengjie.
[9] E. Carrera. 2017. *PEfile.* https://github.com/erocarrera/pefile
[10] Raphael Labaca Castro, Battista Biggio, and Gabi Dreo Rodosek. 2019. Poster: Attacking malware classifiers by crafting gradient-attacks that preserve functionality. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2565–2567.
[11] Raphael Labaca Castro, Sebastian Franz, and Gabi Dreo Rodosek. 2021. AIMED-RL: Exploring Adversarial Malware Examples with Reinforcement Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 37–52.
[12] Raphael Labaca Castro, Corinna Schmitt, and Gabi Dreo Rodosek. 2019. Armed: How automatic malware modifications can evade static detection?. In *2019 5th International Conference on Information Management (ICIM)*. IEEE, 20–27.
[13] Raphael Labaca Castro, Corinna Schmitt, and Gabi Dreo Rodosek. 2019. Poster: training GANs to generate adversarial examples against malware classification. *IEEE Secur. Priv* (2019).
[14] Chronicle. 2023. *VirusTotal Academic Share.* https://support.virustotal.com/hc/en-us/articles/360001387057-VirusTotal-Intelligence-Introduction
[15] Lei Cui, Jiancong Cui, Yuede Ji, Zhiyu Hao, Lun Li, and Zhenquan Ding. 2023. API2Vec: Learning Representations of API Sequences for Malware Detection *(ISSTA 2023)*. Association for Computing Machinery, New York, NY, USA, 261–273. https://doi.org/10.1145/3597926.3598054
[16] Adeilson Antonio da Silva and Mauricio Pamplona Segundo. 2022. On deceiving malware classification with section injection. *preprint arXiv:2208.06092* (2022).
[17] L. Demetrio. 2022. *Toucanstrike.* https://github.com/pralab/toucanstrike
[18] Luca Demetrio and Battista Biggio. 2021. Secml-malware: Pentesting Windows malware classifiers with adversarial EXEmples in Python. *arXiv preprint arXiv:2104.12848* (2021).
[19] Luca Demetrio, Battista Biggio, Lagorio Giovanni, Fabio Roli, Armando Alessandro, et al. 2019. Explaining vulnerabilities of deep learning to adversarial malware binaries. In *CEUR WORKSHOP PROCEEDINGS*, Vol. 2315.
[20] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2021. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security* 16 (2021), 3469–3478.
[21] Luca Demetrio, Battista Biggio, and Fabio Roli. 2022. Practical Attacks on Machine Learning: A Case Study on Adversarial Windows Malware. *IEEE Security & Privacy* 20, 5 (2022), 77–85.
[22] Luca Demetrio, Scott E Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. 2021. Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM Transactions on Privacy and Security (TOPS)* 24, 4 (2021), 1–31.
[23] Ambra Demontis, Marco Melis, Maura Pintor, Jagielski Matthew, Battista Biggio, Oprea Alina, Nita-Rotaru Cristina, Fabio Roli, et al. 2019. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX security symposium*. USENIX Association, 321–338.
[24] Drakcybe. 2022. *A dive into the PE file format.* Retrieved January 19, 2023 from https://darkcybe.github.io/posts/Windows_PE_File_Format/
[25] ESET. 2022. *ESET Smart Security Premium.* https://www.eset.com/us/
[26] Stichting Cuckoo Foundation. 2014. *Cuckoo Sandbox.* https://cuckoosandbox.org/

[27] Daniel Gibert, Jordi Planes, Quan Le, and Giulio Zizzo. 2023. 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 (EuroSPW)*. 415–426.

[28] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. 2004. Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. *Journal of Machine Learning Research* 5, 9 (2004).

[29] Jingcai Guo, Song Guo, Shiheng Ma, Yuxia Sun, and Yuanyuan Xu. 2023. Conservative Novelty Synthesizing Network for Malware Recognition in an Open-Set Scenario. *IEEE Trans. Neural Networks Learn. Syst.* 34, 2 (2023), 662–676. https://doi.org/10.1109/TNNLS.2021.3099122

[30] Mahmoud Hammad, Joshua Garcia, and Sam Malek. 2018. A large-scale empirical study on the effects of code obfuscations on Android apps and anti-malware products. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 421–431. https://doi.org/10.1145/3180155.3180228

[31] Richard Harang and Ethan M Rudd. 2020. SOREL-20M: A large scale benchmark dataset for malicious PE detection. *arXiv preprint arXiv:2012.07634* (2020).

[32] Bryan Horling and Victor Lesser. 2004. A Survey of Multi-Agent Organizational Paradigms. 19, 4 (dec 2004), 281–316. https://doi.org/10.1017/S0269888905000317

[33] James Lee Hu, Mohammadreza Ebrahimi, and Hsinchun Chen. 2021. Single-Shot Black-Box Adversarial Attacks Against Malware Detectors: A Causal Language Model Approach. In *2021 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 1–6.

[34] Weiwei Hu and Ying Tan. 2017. Generating adversarial malware examples for black-box attacks based on GAN. *arXiv preprint arXiv:1702.05983* (2017).

[35] Aparna Sunil Kale, Vinay Pandya, Fabio Di Troia, and Mark Stamp. 2022. Malware classification with Word2Vec, HMM2Vec, BERT, and ELMo. *Journal of Computer Virology and Hacking Techniques* (2022), 1–16.

[36] Lucas Keane, Pai Samruddhi, Lin Weiran, Bauer Lujo, Reiter Michael K., and Sharif Mahmood. 2023. Adversarial Training for Raw-Binary Malware Classifiers. In *32th USENIX Security Symposium (USENIX Security 23)*.

[37] Temesguen Messay Kebede, Ouboti Djaneye-Boundjou, Barath Narayanan Narayanan, Anca Ralescu, and David Kapp. 2017. Classification of malware programs using autoencoders based deep learning architecture and its application to the microsoft malware classification challenge (big 2015) dataset. In *2017 IEEE National Aerospace and Electronics Conference (NAECON)*. IEEE, 70–75.

[38] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. In *26th European Signal Processing Conference, EUSIPCO 2018, Roma, Italy, September 3-7, 2018*. IEEE, 533–537. https://doi.org/10.23919/EUSIPCO.2018.8553214

[39] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. 2016. Deep learning for classification of malware system call sequences. In *Australasian joint conference on artificial intelligence*. Springer, 137–149.

[40] Vijay Konda and John Tsitsiklis. 1999. Actor-critic algorithms. *Advances in neural information processing systems* 12 (1999).

[41] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. 2018. Adversarial examples on discrete sequences for beating whole-binary malware detection. *arXiv preprint arXiv:1802.04528* (2018), 490–510.

[42] Deqiang Li and Qianmu Li. 2020. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3886–3900.

[43] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. 2021. Arms race in adversarial malware detection: A survey. *ACM Computing Surveys (CSUR)* 55, 1 (2021), 1–35.

[44] Yannan Li, Jingbo Wang, and Chao Wang. 2023. Systematic Testing of the Data-Poisoning Robustness of KNN *(ISSTA 2023)*. Association for Computing Machinery, New York, NY, USA, 1207–1218.

[45] Keane Lucas, Mahmood Sharif, Lujo Bauer, Michael K Reiter, and Saurabh Shintre. 2021. Malware Makeover: breaking ML-based static analysis by modifying executable bytes. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 744–758.

[46] Zhinus Marzi, Soorya Gopalakrishnan, Upamanyu Madhow, and Ramtin Pedarsani. 2018. Sparsity-based defense against adversarial attacks on linear classifiers. In *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 31–35.

[47] Francesco Mercaldo, Corrado Aaron Visaggio, Gerardo Canfora, and Aniello Cimitile. 2016. Mobile malware detection in the real world. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, Laura K. Dillon, Willem Visser, and Laurie A. Williams (Eds.). ACM, 744–746. https://doi.org/10.1145/2889160.2892656

[48] MicroSoft. 2022. *Windefender*. https://www.microsoft.com/en-us/windows/comprehensive-security

[49] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[51] Kris Oosthoek and Christian Doerr. 2019. Sok: Att&ck techniques and trends in windows malware. In *International Conference on Security and Privacy in Communication Systems*. Springer, 406–425.

[52] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 506–519.

[53] Daniel Park and Bülent Yener. 2020. A survey on practical adversarial examples for malware classifiers. In *Reversing and Offensive-oriented Trends Symposium*. 23–35.

[54] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. 2018. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

[55] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research* (2021).

[56] Ishai Rosenberg, Shai Meir, Jonathan Berrebi, Ilay Gordon, Guillaume Sicard, and Eli Omid David. 2020. Generating end-to-end adversarial examples for malware classifiers using explainability. In *2020 international joint conference on neural networks (IJCNN)*. IEEE, 1–10.

[57] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. PMLR, 1889–1897.

[58] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *preprint arXiv:1707.06347* (2017).

[59] James Scott. 2017. Signature based malware detection is dead. *Institute for Critical Infrastructure Technology* (2017).

[60] Adriana Sejfia and Max Schäfer. 2022. Practical Automated Detection of Malicious Npm Packages. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1681–1692.

[61] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. 2022. MAB-Malware: A Reinforcement Learning Framework for Blackbox Generation of Adversarial Malware. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 990–1003.

[62] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*.

[63] VirusShare. 2023. *VirusShare*. https://virusshare.com/

[64] VirusTotal. 2023. *VirusTotal*. https://www.virustotal.com

[65] Jialai Wang, Wenjie Qu, Yi Rong, Han Qiu, Qi Li, Zongpeng Li, and Chao Zhang. 2023. MPass: Bypassing Learning-based Static Malware Detectors. In *60th ACM/IEEE Design Automation Conference, DAC 2023, San Francisco, CA, USA, July 9-13, 2023*. IEEE, 1–6. https://doi.org/10.1109/DAC56929.2023.10247858

[66] Shaohua Wang, Yong Fang, Yijia Xu, and Yaxian Wang. 2022. Black-Box Adversarial Windows Malware Generation via United Puppet-based Dropper and Genetic Algorithm. In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 653–662.

[67] Wei Wang, Ruoxi Sun, Tian Dong, Shaofeng Li, Minhui Xue, Gareth Tyson, and Haojin Zhu. 2021. Exposing Weaknesses of Malware Detectors with Explainability-Guided Evasion Attacks. *arXiv preprint arXiv:2111.10085* (2021).

[68] Xian Wu, Wenbo Guo, Hua Wei, and Xinyu Xing. 2021. Adversarial policy training against deep reinforcement learning. In *30th USENIX Security Symposium (USENIX Security 21)*. 1883–1900.

[69] Zhou Yang, Jieke Shi, Junda He, and David Lo. 2022. Natural attack for pre-trained models of code. In *Proceedings of the 44th International Conference on Software Engineering*. 1482–1493.

[70] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. 2017. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)* 50, 3 (2017), 1–40.

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

[72] Chijin Zhou, Lihua Guo, Yiwei Hou, Zhenya Ma, Quan Zhang, Mingzhe Wang, Zhe Liu, and Yu Jiang. 2023. Limits of I/O Based Ransomware Detection: An Imitation Based Attack. In *2023 IEEE Symposium on Security and Privacy (SP)*. 2584–2601. https://doi.org/10.1109/SP46215.2023.10179372

[73] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *29th USENIX Security Symposium (USENIX Security 20)*. 2361–2378.