MalwareTotal: Exposing Weakness of Real-World Anti-virus Software via a Black-Box Adversarial Attack

ABSTRACT

Robust malware variant detection is crucial for Anti-Virus Software (AVs) in defending terminal security and mitigating individual cyberspace risks. In this paper, we propose MalwareTotal, an reinforcement learning based adversarial framework to examine different architectures of static anti-virus software robustness. Particularly, MalwareTotal models malware variant generation as a Markov Decision Process, then we design a 23-dimensional discrete action space by exploiting ambiguities of the PE format, and establish agent environments under a black-box situation. The proximal policy optimization is utilized to optimize the sequential decision making during variants generation. Our evaluations are conducted on both machine learning based AVs, commercial terminal AVs and an integrated online AV service, results show that mutated malware generated by MalwareTotal achieves a 97.03% average attack success rate over 5 anti-virus software. Besides, MalwareTotal outperforms the SOTA method in widely used benchmarks, our research designs a mutator that draws experiences from trained RL agents to reveal that the real-world static anti-virus software is facing a veritable challenge since they can be bypassed within 2 modifications.

KEYWORDS

malware mutation, black-box attack, proximal policy optimization, malware detection

ACM Reference Format:

. 2023. MalwareTotal: Exposing Weakness of Real-World Anti-virus Software via a Black-Box Adversarial Attack. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Background. Anti-malware is the front-line of cyberspace security, where mutated malware detection has always been a long-term challenge for anti-virus software. According to *SonicWall Cyber Threat Report* [1], malware attacked global users over 2.8 billion times during the first half of 2022, of which 36.5% are mutated malware. Despite of how malicious campaigns are targeting victims, it generally involves a piece of malware breaking into a system to fulfill its goals such as exfiltrating data or attempting to render a system unusable [2]. Particularly, detected malware is generally useless in launching new attacks, therefore automatizing malware generation by recycling detected threats could be an interesting approach for attackers and, an additional constraint for malware protection [3].

Motivation. As the number of malicious software increased in the last years to hundreds of thousands of new malware samples daily [4], security technologies can no longer purely rely on virus signatures anymore, instead, they start using heuristics and machine learning techniques for over a decade [5]. The problem is that exclusively relying on machine learning is hardly recommended given the large number of false negative (FN) detections that may arise [6], due to the fact that machine learning encounters an intrinsic obstacle that an indistinguishable perturbation changes the prediction to an incorrect result, known as adversarial attack [7]. Prior studies [8–12] have mounted such attacks on machine learning-based malware detection models [13–15]. Nonetheless, through our preliminary investigation, we find that most existing methods more or less modify malware functionalities during their generation process, resulting in software crashing, no responding, etc [16]. Raphael et al. [17] achieve a functionality-preserving method by deploying a virtual box to discard non-executable malware variants during the generation, which is feasible but time-consuming. Besides, when detectors are tailored to commercial anti-virus software (AVs), there is no known research that can achieve more than 50% average evasion rate, which is less pressing to real-world malware detection [18].

Challenge. By digging into previous work [19–23], we conduct latent reasons for unsatisfactory evasion ability below: (i) due to the target is limited in open-sourced malware detectors, the design of manipulations is lack of diversity, for example, Luca et al. [24] proposed a set of manipulations for DOS Header of malicious files, as they found that MalConv [25] focuses more on PE Header features. (ii) Some research [26–29] made a conventional hypothesis that the full information of target detectors is known, which is infeasible when facing commercial AVs. (iii) Most of the existing work is designed for one or a single type of malware detector, while commercial AVs are often hybrid in underlying model architecture. (iv) Previous work usually utilizes Genetic Algorithm [30] or Multi-armed Bandit [31] for manipulation optimization, but mutated malware generation requires exploring a giant high-dimensional space [32], which leads to less satisfactory results in evasion ability, query and time efficiency.

Aim and scope. In the light of these challenges, we propose MalwareTotal, an automatic large-scale malware variants generation method, to preserve original functionality during the manipulations, we carefully test manifold PE file modifications and divide them into four categories: header manipulation, section manipulation, overlay manipulation, and instruction manipulation, eventually unearth 23 functionality-preserving manipulations. Inspired by a rookie hacker learning process, we optimize the malware mutation process through proximal policy optimization [33], a steady reinforcement learning algorithm. By recording the RL agent trajectory in mutating malware, we calculate the importance of each

Conference'17, July 2017, Washington, DC, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

manipulation and design a simplified mutator without extra learning. It also allows a better understanding of why mutated malware bypasses AV software, which has positive effects on reducing the false negative rate in malware detection. To the best of our knowledge, this paper presents the largest scale systematic analysis of false negative detections in anti-virus software. The Results indicate that the phenomenon of false negative detections is common from machine learning-based malware detectors to commercial AV software, the former might be due to the high dimensionality and excessive linearity of neural networks [34], and the latter suggests that current AV software may possess explicit vulnerabilities which will be illustrated in Section 5.5.

Contribution. Our findings can benefit both researchers and practitioners. Moreover, our findings can help further explore the role of developers and companies in devising strategies for improving AV products. We also make the data and code used in our study available in a replication repository. In summary, this paper brings the following contributions:

- **A Fully Functionality-preserving Attack.** To our best knowledge, MalwareTotal is the first method that automatically generates mutated malware and fully preserves post-modification functionalities. It includes four types with 23 malware manipulations and utilizes reinforcement learning to optimize the attack process.
- **A Practical Framework.** We propose a proximal policy optimization based malware evolution framework, which is practical and effective in malware variants generation, revealing the risks that static anti-virus software are facing have been underestimated.
- **Learning from a machine.** We design a mutator that draws experience from reinforcement learning trajectory through calculating the importance of different manipulations, which concludes the vulnerabilities of heterogeneous anti-virus software.
- **Efficient Evasion Performance.** We conduct a thorough evaluation of the end-to-end implementation of MalwareTotal, showing it can demonstrate substantially stronger by an attack success rate up to at least 16.55%, while being 1.74 times faster in generating mutated variants, compared to the state-of-the-art method.

Roadmap. The remainder of this paper is organized as follows: Section II describes the preliminaries of the proposed method, while Section III describes the threat model. Section IV illustrates the design of our study, and Section V discusses the study performance on different AV detectors. Section VI discusses the potential mitigation and limitations of the proposed attack. After the discussion of related work in Section VII, Section VIII concludes the paper.

2 PRELIMINARIES

2.1 PE File Format Brief Overview

The Windows portable executable (PE) format [35] defines how executable programs are stored as a file on disk, including items such as dynamic link library (DLL) and executable program for Windows (EXE). To avoid breaking the functionality of malware during mutation, there are members in each structure should not be modified as they will be mapped into memory by the operate system (OS) loader during execution. Here we give a brief introduction about the PE file format as well as members can not be modified:

DOS Header and Stub. The DOS header contains metadata for loading the executable inside a DOS environment, while the DOS Stub is made up of few instructions that will print "This program cannot be run in DOS mode" if executed inside a DOS environment. From the perspective of an application functionality, the only relevant locations contained inside the DOS Header are (i) member *e_magic* i.e. the two-byte magic number MZ, which is the abbreviation 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 considered as corrupted.

PE Header. It is the short form of *IMAGE_NT_HEADER*, and contains the target architectures (conditional modifiable, detailed in Section IV) that can run the program, the size of the header (conditional modifiable) and the attributes of the file.

Optional Header. It is essential for EXEs and DLLs, as it contains the information needed by the OS for loading the binary into memory. Among these fields, the optional header specifies the (i) file alignment (unmodifiable), a constraint on the structure of the executable, as each section in a PE file must start at an offset multiple to that field, and the (ii) size of headers (conditional modifiable) that specifies the amount of bytes that are reserved to headers of the programs. Note that if this member is modified, the post-modification must be a multiple of the file alignment.

Section Table. It exists between the PE file header and sections, which is a set of entries that indicates the characteristics of each section of the program. The unmodifiable members in section table are (i) member *Characteristics*, indicates each section is readable, writable or executable and so on, and (ii) member *SizeOfRawData*, as it is determined during compilation.

Sections. The sections area contains several components of which are important to understand for malware analysis, with the mostly commonly encountered being: (i) *.text*, contains the executable component and is the only section with execute privileges and to contain resident code [36]. (ii) *.rdata*, contains the import and export information. (iii) *.data*, contains the global data required by the program. We consider all sections unmodifiable except unused bytes at the end of each section, known as code caves [8].

2.2 Target Systems

Here, we describe recent modes for static malware detection, the first two of which use convolutional neural networks and Gradient Boosting Decision Tree algorithm respectively. In addition to these two learning-based malware detectors, we also introduce prevalent commercial anti-virus software workflow. Despite most of which are black-box to us, their detection mechanism underlying AV engines are transparent. While they share common design concepts, they differ in their overarching architecture and, as we will see, this difference is the key to their respective robustness to the various mutated malware attacks described in this article.

EMBER A gradient-boosted tree ensemble model provided along with the EMBER dataset [37]. EMBER utilizes 2381 hand-engineered features derived from the LIEF [38] PE parsing library, including header properties, exports, byte-entropy histograms, and strings. With a diverse set of semantically meaningful static features, it provides an excellent baseline in static malware detection and helps demonstrate the gap between the features learned by byte-based convolutional neural networks and those created by experts.

MalConv This model [25], is a convolutional neural network that produces an 8-dimensional embedding for each byte, which is fed to two convolutional layers. The convolutional layer contains 128 filters with a stride and size 500, iterating over non-overlapping windows of 500 bytes each. Then a global max pooling is applied to select the a 128-dimensional feature with the largest-activation. Result is then used as the input to a fully connected layer for malware classification. MalConv provides an end-to-end malware detection method without any artificial feature engineering. Concurrent to our work, we notice that the enhanced model MalConv2 [39] is proposed. Considering that the refinement of MalConv2 mainly improves the detection time efficiency, which is not the main concern of our intention. Thus, we decide not to alter the original MalConv in our experiments.

Commercial AVs. The earliest, simplest, and fastest way of detecting is the signature-based method [40], which is still widely applied today. Antivirus software companies first obtain large-scale malware, then design a signature extraction method by anti-virus experts. A signature refers to data that can uniquely identify the file, for example, Extracting MD5 of the malware can be a simple way or a string that only exists in the file. However, this one-to-one mode is rigorous for storage space and can not match mass malware growth, anti-virus producers such as Cylance [41] and Avast [42] begin to incorporate neural networks in identifying never-before-seen malware.

The robustness of the malware detection model MalConv has been extensively studied by previous work and adversarial attacks have shown success [24, 26]. As pointed out by Suciu et al. [43], the lack of robustness in those models may be strongly tied to its weak notions of spatial locality among the learned features—meaning that the location of the injected adversarial noise does not matter as long as the activation on the noise bytes overwhelms those from the actual binary. Even the GBDT model has been shown to be vulnerable to evasion attacks [44, 45], albeit with more advanced and computationally-intensive attacks. While each of these models has been previously evaluated in an ad hoc manner, we are the first to treat attacks on machine-learning models and commercial AVs in a holistic manner with a unifying framework, and in doing so we uncover a novel mutated malware generation method that applies to both model-based and model-free AVs despite the unique architectural differences between them.

3 THREAT MODEL

Before introducing the proposed method, we formulate the malware variant generation problem from the perspective of the generation goal and examiner's capabilities:

3.1 Generation Goal

Given a malware detector $f(\cdot)$, an input malware $x \in \mathcal{X}$, we create a set of manipulating actions $\mathcal{A}$, the desired goal can be formulated as:

$$\begin{aligned} &\underset{\delta}{\operatorname{argmin}} \mathcal{J}(f(x), f(x+\delta)) \\ &s.t. f(x) \neq f(x+\delta) \\ &\delta \in \mathcal{A}, \mathcal{F}(x) = \mathcal{F}(x+\delta), \end{aligned} \tag{1}$$

where the objective function $\mathcal{J}(\cdot)$ indicates the detected result differences between malware and its variant, $\mathcal{F}(\cdot)$ represents the functionality evaluation function of the given malware, which should remain the same after manipulations, the ultimate goal is to find a minimal action δ that could be single manipulation or sequential manipulations drawn from the created action set $\mathcal{A}$ to bypass the given malware detector.

3.2 Examiner's Capabilities

We focus on robustness examination in the context of static PE malware detection, which generally confines the examiner by manipulating malware in the test phase, where the malware detector training phase is held fixed, and preserving the malicious functionality. Note that the terminology of *learning-based detector* and *academical detector* are used interchangeably, referring to the open-source objects to be examined by different malware variant generation methods.

Without loss of generality, there are three possible examination scenarios: *white-box*, *grey-box*, and *black-box*. In a *white-box* scenario, an examiner has full knowledge about the target system (e.g. the dataset, the feature extraction method, the learning algorithm, etc.), which appears less feasible in the context of anti-virus vendors. In a *black-box* scenario, an examiner has no prior knowledge except the detected label. The *grey-box* scenario resides in between. We consider a broad range of prevalent malware detectors, which are categorized into four categories: (i) academic detectors that are proposed by researchers but have not been publicly applied to the real-world anti-virus product yet. (ii) commercial AVs and (iii) integrated online AV service. For academic detectors, we examine their robustness under a *grey-box* scenario since the label and malicious confidence are both accessible, and we conduct other robustness examinations under a strict black-box scenario.

4 METHODOLOGY

In this section, we present the functionality-preserving malware variants generation framework, namely MalwareTotal (the contrary of the integrated anti-virus tool ViursTotal). To extensively examine the mutated malware detect ability of static malware detection, it is important to address the following challenges:

- Challenge 1: How to mutate outdated malware so that can bypass heterogeneous malware detectors?
- Challenge 2: How to fully preserve the functionalities of mutated malware?
- Challenge 3: How to efficiently inject various content into a PE malware?

To address these challenges, we design MalwareTotal, which incorporates four key modules, as shown in Fig. 1. The Reinforcement Learning environment measures the victim anti-virus software to acquire attack-related parameters including scanning results and malicious confidence. The Reinforcement Learning action space generates specific manipulation that can be injected into the input malware. The Reward module converts a desired malware variant generation into an iterative optimization by reporting a timely reward that specifies the reaction of each manipulation. The reinforcement learning agent coordinates the scanning results of the victim AV and the timely reward with selected manipulation sequence to launch variant generation.

4.1 Functionality-preserving Modification

In dealing with *Challenge 1 & 2*, here we first discuss possible functionality-preserving manipulations in the context of Windows PE file format. In principle, there are more than a hundred modifications that can be applied to the input malware. We categorize them as header, section, overlay and compound manipulations. During the functionality test of possible modifications, we found not all modifications are infeasible due to different latent constraints, as detailed below.

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 likewise since it won't execute on systems above win32. Theoretically speaking, DOS Stub can be extended to any length, but we find arbitrarily modify DOS Stub arousing files to crash due to an unannounced upper limit at 6*file alignment, besides, there are 3 members in NT File Header and 5 members in NT Optional Header can theoretically be modified arbitrarily, but we find only 6 of them are conditionally modifiable, where member *MajorLinkerVersion* optional value are 2, 6, 7, 9, 11, 14; *MinorLinkerVersion* optional value are 0, 16, 20, 22, 25; *MajorOperatingSystemVersion* optional value are 4, 5, 6, 10; *MinorOperatingSystemVersion* optional value are 0, 1, 3; *MajorImageVersion* optional value are 0, 1, 5, 6, 10; *MinorImageVersion* optional value are 0, 1, 3. Thus, we conclude following manipulations:

Manipulation 1: Extend the DOS Stub by one file_alignment (at large 6*file_alignment) and inject random bytes.

Manipulation 2: Set the member *CheckSum* in Optional Header to zero.

Manipulation 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.

Manipulation 4 : Set the *IMAGE_DIRECTORY_ENTRY_DEBUG* member in Optional Header to zero.

Manipulation 5: Modifiy the *MajorLinkerVersion*, *MinorLinkerVersion*, *MajorOperatingSystemVersion*, *MinorOperatingSystemVersion*, *MajorImageVersion*, *MinorImageVersion* to aforementioned values respectively.

Manipulation 6 : Zero out the certification in the optional header, if the input malware has a certificate.

Section Manipulations. This portion is related to section adding, shifting or modifying. Some manipulations have been explored in concurrent work (detailed below), our augmentation focuses on two aspects: (i) functionality: We find that previous implementation does not check the upper limit when inserting a section; (ii) inject content: Most manipulations inject random bytes and we refine it by injecting benign content.

Manipulation 7: Rename a section by randomly selecting a section name extracted from benign binaries.

Manipulation 8: Import a library that never been called. This is a prevalent manipulation, since the imported library can be arbitrary, and has been applied in [9, 24, 28, 46]. However, we find former implementation break functionality occasionally due to an unsafe PE parsing library i.e. Lief [38], we rewrite this function with a safer library PeFile [47] to avoid breaking the functionality.

Manipulation 9: Import a library and create a section to call the imported library. Calling a newly imported library requires modifying the relative virtual address (RVA) and extending its size, moreover, the technical implementation can be injecting a library by adding an appropriate entry to the Import Address Table (IAT), as [46] has done. However, this implementation does not consider the case that IAT does not contain enough space to import a new library, which is functionality-breaking. Thus we implement manipulation 8 by creating a section to overwrite IAT and Import Name Table (INT), ensuring that there will be enough space when importing a new library.

Manipulation 10: Create a section and inject content extracted from benign binaries' sections, if there is enough space to insert.

Manipulation 11: Create a section and inject content, similar to manipulation 10, but the content is extracted from benign binaries' strings.

Manipulation 12: Inject bytes in unused space in a section. Note that this is the sole manipulation explored in [8], but instead of injecting random bytes, the content injected in manipulation 12 is extracted from benign binaries' sections.

Manipulation 13: Inject benign content before the first section, to preserve functionality, the injected bytes must be an integral number of file alignment.

Overlay Manipulations. Appending data in overlay theoretical won't break functionality, since overlay will not be reflected to memory. Note that directly overwriting the original overlay will arouse crashing, particularly, when malware is made from the Nullsoft Scriptable Install System (NSIS), or other tools with integrality checking, appended bytes will cause a function alert, which blocks the program from executing, but this can be solved by adding a */NCRC* execution prefix.

Manipulation 14: Append random bytes to input malware overlay.

Manipulation 15: Append benign content extracted from a random benign binary's *.text* section to input malware overlay.

Manipulation 16: Append benign hexadecimal data extracted from a random benign binary to input malware overlay.

Manipulation 17: Append ASCII code converted from strings extracted from a random benign binary.

Compound Manipulations. This portion is related to header, section and overlay ensemble modifications. Considering single manipulation could be query-consuming, we thus conduct several compound manipulations derived from different manipulation granularity. For manipulations whose inject content is discrete, we inject random bytes, and for manipulations whose inject content is continuous and numerous, we inject bytes extracted from benign binaries.

Manipulation 18: Change all modifiable bytes in DOS Header, DOS Stub, Optional Header, and Section Table to random bytes.

Manipulation 19: Alter 58 modifiable bytes in DOS Header to random bytes.

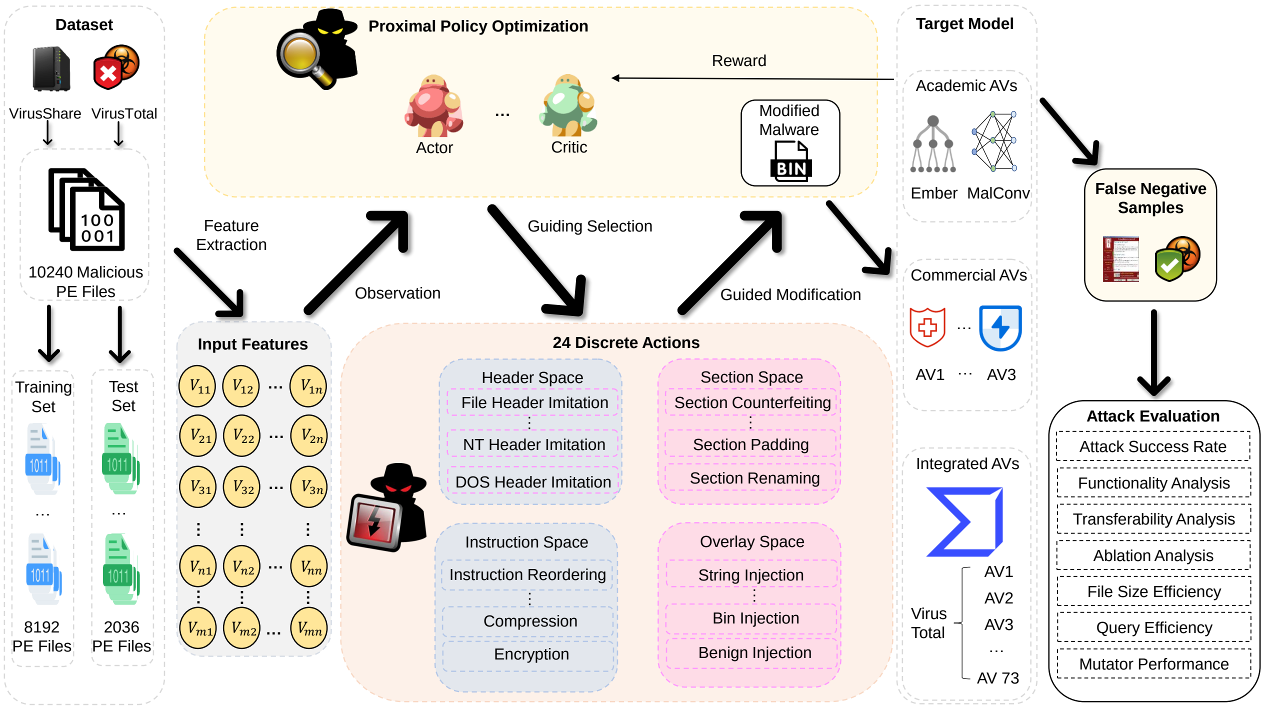

Figure 1: An overview of proposed MalwareTotal, a black-box attack for static PE malware detection.

Manipulation 20: Alter modifiable bytes in DOS Header and DOS Stub to random bytes.

Manipulation 21: Conduct ensemble header manipulations i.e. manipulations 1 to 6.

Manipulation 22: Conduct ensemble sectional manipulations i.e. manipulations 7 to 12.

Manipulation 23: Compress and encryption the input malware, also refers as instruction reordering [12] or code randomization [48]. Utilizing a packer is an intuitive choice in evading anti-virus detection, and has been explored in existing attacks [44, 46, 48, 49]. Besides, [46] has evidenced that sole packing is insufficient to evade detection, nevertheless, the effect of packing on functionality and evasion has not been fully discussed, we demystify this effect as well as partially correct conclusions drawn by prior research in Section V.

4.2 Malware Variant Generation Optimization

To solve *Challenge 3* as well as the problem proposed in equation 1, we model the problem as a Markov Decision Process (MDP) which 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 $\mathcal{P}\ (s_{t+1} \mid s_t, a_t)$ represents the probability that the agent transits from state $s_{(t)}$ to $s_{(t+1)}$ by taking action a_t. (iv) a reward function $R(s_t; a_t)$ denotes the expected reward if the agent takes action a_t at state $s_{(t)}$; (v) a scalar discount factor $\gamma \in [0, 1]$ where 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:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E}[R \mid \pi], \tag{2}$$

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

$$V_\pi(x) = \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid x_0 = x, \pi\right\}, \tag{3}$$

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

$$Q_\pi(x, a) = \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid x_0 = x, a_0 = a, \pi\right\}. \tag{4}$$

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 the 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, and 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 [50], for any differentiable policy $\pi_\theta(s, a)$, the policy gradient can be denoted as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a) Q_{\pi_\theta}(s, a)\right]. \tag{5}$$

By approximating Q_{π_θ} through a deep neural network and taking the action with the highest Q-value, one could maximize cumulative reward and obtain an optimal policy π^* for an agent to generate malware variants, as has been done in [17]. However, this optimization suffers inefficient sampling and over estimation resulting in time-consuming. To address the above limitation, recent research introduces an advantage function, which measures the difference between the Q value for action a in state s and the average value of that state [51], denoted as:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A_{\pi_\theta}(s)\right], \\ A_{\pi_\theta}(s,a) &= Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s).\end{aligned} \tag{6}$$

Through this advantage function, one can acquire an augmentation over the average action taken at the current state during malware variant generation, as has been done in [9]. Nevertheless, recent research indicates that the actor usually experiences enormous variability which influences the performance of the trained agent [52]. Therefore, we introduce Proximal Policy Optimization (PPO) Algorithm in this work, which utilizes importance sampling to achieve asynchronous sampling and adaptive KL-divergence to stabilize the agent training. As discussed in [33], the original form of the PPO objective function is denoted as:

$$\begin{aligned}&\text{maximize}_\theta\ \mathbb{E}_{(a_t,s_t)\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta\text{old}}\left(a_t \mid s_t\right)} A_{\pi_{\theta_{\text{old}}}}\ \left(a_t, s_t\right)\right] \\ &\text{s.t. } \mathbb{E}_{s_t\sim\pi_{\theta_{\text{old}}}}\left[D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}\left(\cdot \mid s_t\right) \| \pi_\theta\left(\cdot \mid s_t\right)\right)\right] \leq \delta\end{aligned} \tag{7}$$

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. $A_{\pi_{\theta_{\text{old}}}}$ represents the advantage function in Equation (6). By solving Equation (7), 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*, cause it is deduced by replacing the KL-constrained with a clipped objective function:

$$\begin{aligned}&\sum_{(s_t,a_t)} \min(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta\text{old}}\left(a_t \mid s_t\right)} A_{\pi_{\theta\text{old}}}(s_t, a_t), \\ &\text{clip}\left(\frac{\pi_\theta\left(\mathrm{a_t} \mid \mathrm{s_t}\right)}{\pi_{\theta_{\text{old}}}\left(\mathrm{a_t} \mid \mathrm{s_t}\right)}, 1-\varepsilon, 1+\varepsilon\right) \mathrm{A}_{\pi_{\theta_{\text{old}}}}\ \left(\mathrm{s_t}, \mathrm{a_t}\right))\end{aligned} \tag{8}$$

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, reward function and environments are required to calculate state and reward transmission. As discussed in [25, 37], we utilize a 2381-dimensional vectorized environment for examining Ember and a 1024×1024 box environment for MalConv. As for commercial AVs, we inherit Ember vectorized environment due to more robust posterior performance. The reward function is divided into ML-Detector and Commercial AVs as well, for the former: (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 terminal hits; (iii) the reward is equal to the difference between the initial malicious score and in-process score. Due to the fact that the malicious score in examining commercial AV is inaccessible, thus the reward function for the latter only contains the first two components.

5 EVALUATION

In this section, we first provide a comprehensive understanding of MalwareTotal where we test the performance of the proposed attack under machine-learning malware detectors and commercial AV products. After that, we showcase a MalwareTotal transfer attack performance on an integrated commercial malware scanner to show the superiority of our method in producing false negative detections. Below, we first present our experiment setup. Then, we discuss the design of our experiment, and evaluate the performance of MalwareTotal by answering the following four research questions.

• **RQ1: Effectiveness analysis.** *How the effective is MalwareTotal against the state-of-the-art learning-based and commercial malware detectors?*

• **RQ2: Mutation efficiency comparisons.** *Compared with other mutated malware generation methods, how is the mutation efficiency of MalwareTotal?*

• **RQ3: Transferability assessment.** *Can mutated malware generated by MalwareTotal also evade other detectors?*

• **RQ4: Explainability Exploration.** *Can we gain valuable experience through evaded malware?*

5.1 Target AVs, Data & Benchmarks

We now illustrate our experimental setup and target state-of-the-art malware detection techniques that are well-cited in academia or provided by premier anti-virus companies:

Detector 1: Ember, a GBDT-based malware detector, is introduced in Section II. We use the trained model in SEC-ML [53], 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.

Detector 2: MalConv, a convolutional malware detect model, While the original MalConv model considers a maximum input file size of 2MB, the model used in our experiments relies on the reproduction in ToucanStrike [54] 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 separate from the standard bytes in the file, with a malicious threshold of 0.5, leading an FPR of 0.035 and a TPR of 0.69.

Detectors 3-5: Commercial AVs, all of which are rewarded top-rated in *The best Windows antivirus software for home users 2022* by AV-Test [55]. We believe this selection stands for an advanced anti-virus detection ability.

Detector 6: VirusTotal [56], an online AV scanner, integrates over 70 AV products for PE malware detection.

In examining machine-learning AVs and commercial AVs, we evaluate the MalwareTotal performance with an attack success rate (ASR), queries, file size, time consumed, and functionality, comparing with broadly existing methods. Here, ASR refers to the ratio of mutated malware that is classified as benign in all mutations. Queries indicate the total amount of calling anti-virus detectors during generation. The file size is the average of post-modification malware size, showing how many bytes are injected during the generation.

We investigate prevalent datasets in malware detection including Sorel-20M [57], MS BIG 2015 [58], most of which only release preprocessed features. Since the functionality evaluation is essential for mutated malware, we mainly adopt two malware datasets: VirusShare [59] and VirusTotal Academic share [60], which provide the original malware samples, we randomly collect 8192 PE malware for training and 2036 for testing. To guarantee the quality of samples, all collections are identified as malware both by EMBER and MalConv.

We implement MalwareTotal using PyTorch 1.8.2 and stable-baselines3 [61], in particular, to manipulate PE files, we rely on PEfile [47], which is based on python 3.6.2. We run all our experiments on a windows server configured with a 2.8 GHz Intel i5-8400 processor, 1×8G NVIDIA 1080 GPU and 32G memory.

5.2 RQ1: Effectiveness analysis

In our first experiment, we measure the attack performance of Ember [37] and MalConv [25] detectors since they both are learning-based and have been widely applied in recent research. We train our agent with 8192 malware, with each malware having at most 3 episodes to reset the mutation environment, in each episode, the agent has up to 30 steps, i.e. maximal 30 manipulations per sample. Due to arbitrarily encrypting or compressing the malware during mutation will break the functionality, we thus conduct *Manipulation 23* as the 31st, i.e. the final action during mutation, the optimization terminates when the last sample in the training set is traversed. Then we test and report the performance with the test set 2036 malware. Table 1 illustrates the performance of MalwareTotal when facing the first two detectors (i.e. EMBER and MalConv). As we can see, MalwareTotal achieves an effective evasion performance with 98.67% and 100% ASR against EMBER and MalConv respectively. Note that the average file size of the test set is 315.8KB, and MalwareTotal increases average of 124.8 and 159.5 KB to create a mutated malware that can bypass EMBER and MalConv respectively. To find out the optimization ability of proposed MalwareTotal, we conduct ablation experiments that replace the *Proximal Policy Optimization* with *Trust Region Policy Optimization* [62], *Advantage Actor Critic* [63], *Deep Q-learning* [64], but use the same MalwareTotal 23 actions (namely MT-TRPO, MT-A2C, MT-DQN respectively). We also utilize a random agent which selects random actions during mutation (i.e. no optimizer, namely MT-Random). Results show that MalwareTotal achieves the best ASR both in EMBER and MalConv. Although TRPO achieves slightly better file size and queries in bypassing EMBER, it performs weaker in ASR and takes more than 10 hours to create mutations, while our method only takes less than 6 hours.

In experiment 2, we evaluate the performance of MalwareTotal against commercial AVs, This is obviously a harder task due to: (i) we can no longer gain prediction confidence since commercial AVs only return binary scan results to users, this scenario is also known as *Hard-label Attack* [46, 48]. (ii) commercial AVs contain a huge malware signature cloud and determine with multi-module detection technologies. In this experiment, we train and test MalwareTotal with the same data used in experiment 1, besides, we use the 2381-dimensional Ember environment as the substitute commercial AV environment due to a better performance than the MalConv environment in our fundamental performance test. In addition, due to the optimization of MalwareTotal relies on the iterative interaction with target AVs, we use the premier edition of *Detector 4-5*, due to it is the only edition that provides the *CMD Scan Mode*, we set the *Scan sensitivity* to medium in balance of scan ability and time efficiency. *Detector 3* directly provides the *CMD Scan Mode*, and we scan mutations with the default setting. Results are reported in Table 2. In our expectation, there should be a decline in ASR and a rapid growth of queries. However, MalwareTotal achieves 95.33%, 92.63%, and 98.52% ASR respectively, meaning that with an experienced agent, tested AVs can be bypassed with an average of 156.26KB injected content with 21 average manipulations at most.

To ensure that MalwareTotal fully preserves malicious functionality, we conduct experiment 3, firstly, we ensure all original malware is runnable and malicious through Cuckoo Sandbox [65] with a 32-bit Window 7 virtual system, then we test the functionality of individual manipulations, we randomly select 50 malware in our test set, and iteratively modify each malware from manipulating once to manipulating 30 times to simulate the most extreme situation in actual mutation, resulting in 33000 mutated malware, and test their functionalities with Cuckoo, the result shows that all malware behaviors are malicious. Moreover, to demonstrate overlapped manipulations do not harm malware functionality, we randomly select 200 mutated malware that has evaded from each detector (i.e. 1000 malware in total) and validate their functionalities with Cuckoo. Again, all tested samples are runnable and show malicious functions in the sandbox. Through the two functionality validations above, we believe it is safe to say that MalwareTotal is functionality-preserving in mutating malware.

Answer to RQ 1: MalwareTotal achieves 100% ASR against *Detector 1*, 98.67% ASR against *Detector 2*, 95.33%, 92.63%, 98.52% ASR respectively against *Detector 3-5*.

5.3 RQ2: Mutation efficiency comparisons

In experiment 3, we compare MalwareTotal on learning-based detector (i.e. *Detector 1-2*) with latest work including *Gamma* [46], *MAB* [48], *Full-DOS* [24], and so on. For *Gamma MAB* and *Aimed* [17], we use the code released by the official, for other methods, our reproduction relies on an integrated PE adversaries framework Toucanstrike [54], and we follow the original setting in each paper. All comparisons are conducted on the same test set used in MalwareTotal. Since most compared methods are heuristic, we repeatedly run each experiment 3 times and report the average result in Table 3. We can see from the result that MalwareTotal outperforms compared method in ASR both in evading EMBER and MalConv, Although *MAB* consumed fewer queries and less time than us, MalwareTotal has huge advantages in ASR and file size. As for MalConv, MalwareTotal achieves 100% ASR with the fewest time consumed and queries. In experiment 4 and further experiments, considering commercial AVs consume extreme time in scanning progress, we select the *Gamma Section* as a comparison, which is well-performed both in *Detector 1* and *Detector 2*. The detectors are replaced with commercial AVs, and mutated malware is iterative generated utilizing *Gamma Section*. Unfortunately, as we can see from Table 2, *Gamma Section* achieves only 36.11%, 17.33%, and 14.93% ASR respectively. By contrast, MalwareTotal bypasses three commercial AVs with an

Table 1: Effective of Malwaretotal Against Learning-Based Malware Detectors.

Methods	EMBER				MalConv			
	ASR(%)	Queries	File Size	Time	ASR(%)	Queries	File Size	Time
MalwareTotal	**98.67%**	14028	440.6KB	5h48mins	**100**	**4459**	475.3KB	**1h1min**
MT-TRPO	96.22	**11680**	**427.3KB**	10h37mins	**100**	7239	473.63KB	1h50mins
MT-A2C	89.29	21106	577.6KB	**2h17mins**	97.24	15363	**443.7KB**	2h17mins
MT-DQN	59.38	24129	577.6KB	13h46mins	59.38	37838	494.9KB	60h21mins
MT-Mutator	87.28	18387	400.9KB	11h48mins	99.8	7187	483.44KB	1h57mins
MT-Random	62.13	38827	605.20KB	13h22mins	93.66	42267	527.1KB	11h16mins

Table 2: Comparisons Between Individual Attack Strategies and MalwareTotal Against Commercial Anti-virus Software.

Methods	Commercial AV1				Commercial AV2				Commercial AV3			
	ASR	Queries	File Size	Time	ASR	Queries	File Size	Time	ASR	Queries	File Size	Time
MalwareTotal	**95.33**	**36491**	466.70KB	**45.9h**	**92.63**	**42756**	**472.06KB**	**52.2h**	**98.52**	**7240**	**341.5KB**	**24.0h**
MT-Mutator	95.09	38012	464.40KB	70.7h	84.43	49190	472.98KB	54.7h	87.33	10997	472.98KB	38.8h
Gamma	36.11	61839	**337.92KB**	114.6h	17.33	61057	478.64KB	115.3h	14.93	53244	488.42KB	132.7h
MT-Random	40.57	51680	513.73KB	88.4h	23.87	61027	498.02KB	115.2h	66.10	59316	498.02KB	147.8h

extremely high ASR, Besides, as the scanning process of commercial AVs usually takes seconds to minutes, an effective optimization becomes prominent, as is reported in Table 2, MalwareTotal not only takes 68.7, 63.1 and 108.7 hours less than *Gamma Section*, but also injects less content in evading AV2 and AV3.

Answer to RQ2: MalwareTotal achieves the highest ASR in evading all detectors, and consumes the fewest time and queries in evading all commercial AVs and MalConv.

5.4 RQ3: Transferability assessment

Due to the fact that adversarial examples show a strong transferability in computer version (CV) fields [66]. It is worthy to investigate whether mutated malware transfers among malware detectors, considering that many AVs do not claim to have deployed learning-based detection technology. We thus conduct experiment 5, utilizing 5000 malware generated by MalwareTotal and Gamma respectively. Each 500 malware has evaded an individual detector, for example, 500 malware generated by MalwareTotal which has evaded from *Detector 1* is denoted as MT-D1, 500 malware generated by *Gamma* which has evaded from *Detector 2* is denoted as Gamma-D2. Then mutated malware is scanned by VirusTotal to examine the transferability. There are 73 anti-virus services integrated in VirusTotal, in our experiments, 4 of which fail to return almost all results, therefore are excluded from our experiment. We report the overall and segmented performance in Table 4, where the first row presents the ASR calculated through the ratio of malware that is identified as benign by 69 online AV scanners of all scanned results. The rest rows illustrate the quantity of AVs that are bypassed with the corresponding percentage. The results indicate that: (i) mutated malware evaded from learning-based detectors shows a weaker transferability against heterogeneous real-world AVs compared with mutated malware evaded from commercial AVs. As the ASR of MT-D1 and MT-D2 is much lower than MT-D3 to D5. (ii) MalwareTotal has a better ability in exposing commercial AV vulnerabilities. When results are tailored to commercial AVs mutations, MalwareTotal bypasses 73 AVs in total with 90% ASR (i.e.the sum of the third row in Table 4), while *Gamma* only bypasses 33 AVs. (iii) known to date, MalwareTotal is the first method that achieves an average ASR above 50% (especially 63.31% in MT-D3), revealing that the risks in the robustness of static malware detection have been overlooked.

To further demonstrate the fragility of commercial AVs in detecting MalwareTotal mutations is susceptible and extensive, we report the ASR against 16 (out of 31) AVs that are rewarded top-rated in *The best Windows antivirus software for home users* by AV-Test in 2022. Again, we emphasize that our work does not aim to compare differences in AV product capabilities, but to evaluate the robustness of AV products as a whole, thus all AV products are anonymized. The results are reported in Table 5, among 16 awarded AV products, only 2 of them are robust to both MalwareTotal and *Gamma* mutations (i.e. AV 10 and AV 12). Note that the first three AV products are same as AVs evaluated in experiment 2. Results show that the first two AVs perform no resilience to transfer attacks as they classified all mutations as benign. The ASR of the third AV product decays, we assume the reason is the same as the work [67] discussed that some anti-virus software provides different versions between the local scanner and online scanner to VirusTotal, usually the online scanner has a stronger ability in detecting malware.

Answer to RQ3: Transferability of mutated malware is extensive in static malware detection whether the architecture of AV products is learning-based or not, as up to 63.31% tested detections in VirusTotal is vulnerable to MalwareTotal mutations transfer attack.

5.5 RQ4: Explainability Exploration

To find out how and why mutated malware bypasses each malware detector, we conduct experiment 6, designing a mutator that directly learns from reinforcement learning agent trajectories. By recording the sequence of manipulations during each mutation, we calculate the sequential frequency of each manipulation through malware

Table 3: Comparisons Between Individual Attack Strategies and MalwareTotal Against Learning-Based Malware Detectors.

Methods	EMBER				MalConv			
	ASR	Queries	File Size	Time	ASR	Queries	File Size	Time
MalwareTotal	**98.67%**	14028	440.6KB	5h48mins	**100**	**4459**	475.3KB	**1h1min**
Full DOS	3.97%	22968	1.18MB	**3h22mins**	47.84	38737	364.0KB	39h1min
Partial DOS	3.88%	20805	349.7KB	3h26mins	42.17	41907	335.0KB	17h45mins
Extend	3.98%	24367	321.4KB	5h45mins	59.87	34666	347.9KB	164h22mins
Shift	4.22%	25144	327.3KB	4h10mins	33.35	30789	274.0KB	75h22mins
Padding	4.03%	22968	**263.7KB**	3h40mins	20.68	30251	**271.8KB**	105h25mins
Header Fields	47.79%	19363	361.9KB	128h59mins	36.39	38984	306.0KB	30h44mins
GAMMA Section	74.26%	26738	490.9KB	6h25mins	83.45%	32295	474.0KB	5h12mins
GAMMA Padding	64.00%	26176	482.8KB	5h28mins	52.85%	28489	436KB	4h11mins
MAB	69.50%	**10466**	2.967MB	5h30mins	78.05%	8671	2.31MB	4h28mins
Aimed	47.79%	73642	560.5KB	19h13mins	67.58	73624	555.4KB	12h3mins

Table 4: 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≥90%	10	9	**29**	**29**	15	14	12	12	11	10
AVs ≥80%	14	12	**32**	30	24	15	14	15	14	14
AVs ≥70%	16	15	34	32	**36**	18	16	17	20	17
AVs ≥ 60%	19	17	36	34	**40**	20	19	19	23	18
AVs ≥50%	21	20	**41**	39	**41**	25	23	22	26	26

Table 5: 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%
AV4	19.09%	15.06%	95.83%	90.18%	**97.87%**	51.06%	46.94%	41.67%	78.35%	72.63%
AV5	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%
AV11	18.41%	26.06%	98.00%	74.79%	**100.00%**	17.00%	12.00%	28.57%	37.37%	34.00%
AV12	2.57%	8.16%	4.17%	9.65%	**18.75%**	5.21%	4.17%	6.25%	1.51%	3.71%
AV13	8.46%	4.52%	96.00%	65.83%	**100.00%**	4.00%	2.04%	8.33%	13.26%	17.53%
AV14	12.64%	8.75%	**100.00%**	75.00%	**100.00%**	16.00%	6.00%	24.00%	38.00%	32.65%
AV15	24.12%	15.65%	**100.00%**	75.63%	**100.00%**	24.74%	10.42%	45.65%	42.27%	40.63%
AV16	36.31%	40.16%	**100.00%**	97.50%	**100.00%**	69.00%	52.00%	68.75%	85.00%	80.00%

mutation sequences that have bypassed each detector as the positive effect, as well as the negative effect (i.e. sequential frequency of each manipulation from malware that episodically fails during generation), then we add positive and negative effect and obtain the episodic importance of each manipulation to a given detector. Then we construct mutators that behave following the episodic importance, to validate whether malware generated by mutators can still bypass each detector. The second row in Table 2 illustrates the results, all mutators achieve a decayed ASR of about 10% except MalConv, with the consequence of increasing queries and running time. Indicating that (i) the episodic importance is effective in bypassing each detector since ASR achieves 87.28%, 99.8%, 95.09%, 84.43% and 87.33% respectively. (ii) different malware detectors are vulnerable to different manipulations, for EMBER, the first three manipulations utilized most frequently are manipulation 15, 5, 22. As for MalConv, frequent manipulations become manipulation 16, 18, 1, which matches the conclusion that MalConv is sensitive to features in the PE header and can be bypassed by appending bytes in overlay [26]. For commercial AVs, *Detector 3* is vulnerable to manipulation 15, 4, 21 well *Detector 4* and *5* is fragile to manipulation 22, 12, 9 and 9, 22, 2 respectively. (iii) training an experienced agent is necessary cause it not only learns the vulnerability of each malware detector but also learns which manipulation to take when facing different malware due to an average 6.19% decay in ASR and an increment in file size and queries.

Answer to RQ4: Mutators are effective in bypassing detectors, with an average 6.19% decay in ASR and an increment in file size and queries. Besides, different detectors are vulnerable to different manipulations.

6 DISCUSSION

6.1 Obfuscation

It could be doubted that traditional obfuscation methods can simply hide malicious functionality. On the one hand, [68] has shown that machine learning-based PE malware detection techniques are able to detect malware packed with known packers. On the other hand, the explainability exploration shows that frequently used manipulations do not contain packing, besides, we conduct another ablation experiment on *Detector 1* that removes the packing manipulation, MalwareTotal achieves 96.49% ASR, which is close to the 98.67% ASR performance with packing technique, i.e. although our manipulations contain a packer, packing is not the core factor that MalwareTotal bypasses varieties of anti-virus software. Moreover, the novel mutated malware generation method MalwareTotal in this work demonstrates the feasibility of the "producing statically undetectable PE malware as a service" scenario. As far as we know, the process of malware mutation does not leave any artifact and, the high ASR of MalwareTotal in each detector indicates that the AV-oriented attack has become reality with little esoteric knowledge required. At last, although manipulations 1-22 and packing differ in affecting PE file components. In the code transplantation domain [69], MalwareTotal can be seen as a tool for developing new forms of obfuscation.

6.2 Potential Mitigation

Our study exploits the vulnerabilities of malware detection techniques both in machine learning algorithms and commercial AVs. Through manipulating the PE file, MalwareTotal ultimately affects commercial AV software decisions. In this subsection, we provide several potential defense mechanisms by increasing the difficulty of launching our attacks: (i) Adding extra semantic comparing techniques in static malware detection, such as binary similarity detection techniques [70], which have achieved a significant accuracy with a valid time-efficiency in recent research. Due to the functionality-preserving constraint of MalwareTotal, modifications hardly change the embedding in semantic representation. (ii) Adversarial Training [71], adversarial robustness has been a long-term hotspot to machine learning services, and adversarial training is an effective method to known attack through retraining the model with adversarial examples (i.e. mutated malware in our case) added into the training set. Once the MalwareTotal is published, utilizing adversarial training will help decrease the false negative detection (i.e. mutated malware that is wrongly detected as benign) in their learning-based detection modules. To prevent the potential damage that may arouse to AV producers and Windows users, we have uploaded our mutated malware to many anti-virus software producers could including WinDefender [72], ESET NOD [73], Avast [42], VirusTotal [56] and so on.

6.3 Limitations

Our method still has the following limitations at present. First, we have successfully attacked static anti-virus software but have not succeeded on dynamic anti-virus software with virtual technology yet. 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 WMI API [74]. We remain it as the future work. Second, though we have addressed the functionality issue for physical attacks to some extent by carefully writing and testing the individual manipulations, it is still challenging to fully prevent a commercial packer from breaking malware functionalities especially when the original malware has already been packed by an unknown packer. We have contacted a commercial packer producer and helped fixing a few cases that broke malware functionality during MalwareTotal functionality experiments, but we believe the duty of ensuring the commercial packer functional completeness is outside the scope of our work.

7 RELATED WORK

Preceding work is significantly different from MalwareTotal, as it designs attacks from the perspective of adversarial attack, which naturally can be categorized into three schemes:

White-box attack: As introduced in Section III, white-box attacks mostly rely on the full target information including model gradient, output score and so on. Particularly, Kolosnjaji et.al [75] manipulates malware through *padding* embedding value at the overlay of PE file guided by the positive direction of the gradient, the experiment is conducted on MalConv detector and achieves 60% attack success rate with an average of 10000 bytes modified per malware. Later, [27] enhanced the padding manipulations by padding in section unused space and overlay, and replaces the optimizer with an iterative variant of the fast gradient sign method (FGSM) [76], a simple but effective optimization firstly proposed in image adversarial attack. [26] manipulates 58 bytes in a DOS header (from MZ to the offset to PE header), guided by an introduced integrated gradients method, known as *Partial DOS*.

White & black-box attack: Header Fields [28] and *Full DOS* [24] can be applied both in white-box or black-box scenarios, where white-box optimization relies on the gradient and black-box optimization is guided by a genetic algorithm. Particularly, *Header Fields* manipulates names of each section in Optional Header, and *Full DOS* proposes three individual attacks namely *Full DOS, Extend, Shift*, where *Full DOS* manipulates 58 bytes in the DOS header and the whole DOS stub, *Extend* enlarges the DOS header and injects adversarial content, *Shift* injects content before the first section.

Black-box attack: In this scenario, Demetrio et. al [46] proposes two individual attacks, one of which conducts manipulations through a set of manipulations in section and a packer, they name this method *Gamma Section*, the other attack is a combination of *padding, partial DOS, extend, shift*, their manipulation optimization relies on a genetic algorithm. Similarly, [48] also utilizes a packer as well as a set of manipulations and uses a multi-armed bandit (MAB) as the optimizer, which requires 20 servers to conduct manipulations in their experiments. *Aimed* [17] minimizes the malware score with the same manipulations used in *Header Fields*, but *Aimed* utilizes a genetic algorithm to optimize the chain of manipulations, then they use a sandbox to discard malware whose functionality is broken.

Without loss of generality, Manipulations in previous work focus only on a partial PE file structure (i.e. DOS Header, Section or Overlay), but our work manipulates malware in a whole view of PE file structure. Besides, the perceptibility of the above methods relies on either the model gradient or the score of malware predicted by a detector, which is less feasible when facing commercial AVs, as we detailed in section I.

8 CONCLUSION

Unlike adversarial attacks in the computer version domain, where evading ML-based classifiers is in itself a meaningful and eloquent result, this is not sufficient for robustness examination in malware detection. Considering that malware variants continue to be one of the most pressing security issues that users face today, it is crucial to explore how malware variants bypass heterogeneous anti-virus software. Thus we propose MalwareTotal, a malware variant generation framework that includes 23 functionality-preserving malware manipulations whose perturbation sequences are guided by proximal policy optimization. We show MalwareTotal outperforms the existing state-of-the-art approaches while decreasing the bypassing cost, generalizing across different malware detectors, and exploring realistic operational settings.

We envision this work will enable researchers and practitioners alike to leverage those malware manipulations to build rejection strategies to improve their static malware detection software. To this end, we release our implementation of MalwareTotal, making evaluation available to the community for the first time.

REFERENCES

[1] SonicWall. Sonicwall's cyber threat report, 2022.

[2] M Satheesh Kumar, Jalel Ben-Othman, and KG Srinivasagan. An investigation on wannacry ransomware and its detection. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2018.

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

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

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

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

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

[8] 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.

[9] Zhiyang Fang, Junfeng Wang, Jiaxuan Geng, Yingjie Zhou, and Xuan Kan. A3cmal: Generating adversarial samples to force targeted misclassification by reinforcement learning. *Applied Soft Computing*, 109:107505, 2021.

[10] Weiwei Hu and Ying Tan. Black-box attacks against rnn based malware detection algorithms. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[11] Raphael Labaca-Castro, Battista Biggio, and Gabi Dreo Rodosek. Poster: Attacking malware classifiers by crafting gradient-attacks that preserve functionality. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2565–2567, 2019.

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

[13] Fangwei Wang, Guofang Chai, Qingru Li, and Changguang Wang. An efficient deep unsupervised domain adaptation for unknown malware detection. *Symmetry*, 14(2):296, 2022.

[14] Aparna Sunil Kale, Vinay Pandya, Fabio Di Troia, and Mark Stamp. Malware classification with word2vec, hmm2vec, bert, and elmo. *Journal of Computer Virology and Hacking Techniques*, pages 1–16, 2022.

[15] Nureni Ayofe Azeez, Oluwanifise Ebunoluwa Odufuwa, Sanjay Misra, Jonathan Oluranti, and Robertas Damaševičius. Windows pe malware detection using ensemble learning. In *Informatics*, volume 8, page 10. MDPI, 2021.

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

[17] Raphael Labaca-Castro, Sebastian Franz, and Gabi Dreo Rodosek. Aimed-rl: Exploring adversarial malware examples with reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 37–52. Springer, 2021.

[18] James Scott. Signature based malware detection is dead. *Institute for Critical Infrastructure Technology*, 2017.

[19] Raphael Labaca Castro, Corinna Schmitt, and Gabi Dreo Rodosek. Armed: How automatic malware modifications can evade static detection? In *2019 5th International Conference on Information Management (ICIM)*, pages 20–27. IEEE, 2019.

[20] Yanchen Qiao, Weizhe Zhang, Zhicheng Tian, Laurence T Yang, Yang Liu, and Mamoun Alazab. Adversarial malware sample generation method based on the prototype of deep learning detector. *Computers & Security*, page 102762, 2022.

[21] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983*, 2017.

[22] Justin Burr. Improving adversarial attacks against malconv. 2022.

[23] James Lee Hu, Mohammadreza Ebrahimi, and Hsinchun Chen. 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)*, pages 1–6. IEEE, 2021.

[24] Luca Demetrio, Scott E Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. 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):1–31, 2021.

[25] 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.

[26] Luca Demetrio, Battista Biggio, Lagorio Giovanni, Fabio Roli, Armando Alessandro, et al. Explaining vulnerabilities of deep learning to adversarial malware binaries. In *CEUR WORKSHOP PROCEEDINGS*, volume 2315, 2019.

[27] Adeilson Antonio da Silva and Mauricio Pamplona Segundo. On deceiving malware classification with section injection. *arXiv preprint arXiv:2208.06092*, 2022.

[28] Hyrum S Anderson, Anant Kharkar, Bobby Filar, and Phil Roth. Evading machine learning malware detection. *black Hat*, 2017, 2017.

[29] Luca Demetrio, Battista Biggio, and Fabio Roli. Practical attacks on machine learning: A case study on adversarial windows malware. *IEEE Security & Privacy*, 20(5):77–85, 2022.

[30] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[31] Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448. Springer, 2005.

[32] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE symposium on security and privacy (SP)*, pages 1332–1349. IEEE, 2020.

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

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

[35] Microsoft. Pe format, 2022.
[36] Drakcybe. A dive into the pe file format, 2022.
[37] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.
[38] Romain Thomas. Lief, 2019.
[39] 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.
[40] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao, and Brian Chavez. Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15–pp. IEEE, 2006.
[41] Cylance. Cylance blackberry cybersecurity, 2019.
[42] Avast. Avast premium security, 1988.
[43] 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.
[44] Fangtian Zhong, Pengfei Hu, Guoming Zhang, Hong Li, and Xiuzhen Cheng. Reinforcement learning based adversarial malware example generation against black-box detectors. *Computers & Security*, 121:102869, 2022.
[45] 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.
[46] 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.
[47] Ero Carrera. Pefile, 2017.
[48] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. 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*, pages 990–1003, 2022.
[49] Wei Wang, Ruoxi Sun, Tian Dong, Shaofeng Li, Minhui Xue, Gareth Tyson, and Haojin Zhu. Exposing weaknesses of malware detectors with explainability-guided evasion attacks. *arXiv preprint arXiv:2111.10085*, 2021.
[50] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.
[51] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.
[52] Xian Wu, Wenbo Guo, Hua Wei, and Xinyu Xing. Adversarial policy training against deep reinforcement learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1883–1900, 2021.
[53] Luca Demetrio and Battista Biggio. Secml-malware: Pentesting windows malware classifiers with adversarial exemples in python. *arXiv preprint arXiv:2104.12848*, 2021.
[54] Luca Demetrio. Toucanstrike, 2022.
[55] AV-TEST. The best windows antivirus software for home users, 2022.
[56] VirusTotal. Virustotal, 2023.
[57] Richard Harang and Ethan M Rudd. Sorel-20m: A large scale benchmark dataset for malicious pe detection. *arXiv preprint arXiv:2012.07634*, 2020.
[58] Temesguen Messay Kebede, Ouboti Djaneye-Boundjou, Barath Narayanan Narayanan, Anca Ralescu, and David Kapp. 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)*, pages 70–75. IEEE, 2017.
[59] VirusShare. Virusshare, 2023.
[60] VirusTotal. Virustotal academic share, 2023.
[61] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.
[62] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.
[63] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.
[64] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
[65] Stichting Cuckoo Foundation. Cuckoo sandbox, 2014.
[66] Ambra Demontis, Marco Melis, Maura Pintor, Jagielski Matthew, Battista Biggio, Oprea Alina, Nita-Rotaru Cristina, Fabio Roli, et al. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX security symposium*, pages 321–338. USENIX Association, 2019.
[67] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. Measuring and modeling the label dynamics of online {Anti-Malware} engines. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2361–2378, 2020.
[68] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. When malware is packin'heat; limits of machine learning classifiers based on static analysis features. In *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.
[69] Aurore Fass, Michael Backes, and Ben Stock. Hidenoseek: Camouflaging malicious javascript in benign asts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1899–1913, 2019.
[70] Irfan Ul Haq and Juan Caballero. A survey of binary code similarity. *ACM Computing Surveys (CSUR)*, 54(3):1–38, 2021.
[71] Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 76–82. IEEE, 2018.
[72] MicroSoft. Windefender, 2022.
[73] ESET. Eset smart security premium, 2022.
[74] Kris Oosthoek and Christian Doerr. Sok: Att&ck techniques and trends in windows malware. In *International Conference on Security and Privacy in Communication Systems*, pages 406–425. Springer, 2019.
[75] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learning for classification of malware system call sequences. In *Australasian joint conference on artificial intelligence*, pages 137–149. Springer, 2016.
[76] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.