Binary Black-Box Attacks Against Static Malware Detectors with Reinforcement Learning in Discrete Action Spaces

1st Mohammadreza Ebrahimi
Artificial Intelligence Lab
University of Arizona
Tucson, USA
ebrahimi@email.arizona.edu

2nd Jason Pacheco
Department of Computer Science
University of Arizona
Tucson, USA
pachecoj@cs.arizona.edu

3rd Weifeng Li
Department of Management Information Systems
University of Georgia
Athens, USA
weifeng.li@uga.edu

4th James Lee Hu
Artificial Intelligence Lab
University of Arizona
Tucson, USA
jameshu@email.arizona.edu

5th Hsinchun Chen
Artificial Intelligence Lab
University of Arizona
Tucson, Arizona
hchen@eller.arizona.edu

***Abstract*—Recent machine learning- and deep learning-based static malware detectors have shown breakthrough performance in identifying unseen malware variants. As a result, they are increasingly being adopted to lower the cost of dynamic malware analysis and manual signature identification. Despite their success, studies have shown that they can be vulnerable to adversarial malware attacks, in which an adversary modifies a known malware executable subtly to fool the malware detector into recognizing it as a benign file. Recent studies have shown that automatically crafting these adversarial malware variants at scale is beneficial to improve the robustness of malware detectors. For conciseness, we refer to this process as Adversarial Malware example Generation (AMG). Most AMG methods rely on prior knowledge about the architecture or parameters of the detector, which is not often available in practice. Moreover, the majority of these methods are restricted to additive modifications that append contents to the malware executable without modifying its original content. In this study, we offer a novel Reinforcement Learning (RL) method, AMG-VAC, which extends Variational Actor-Critic (VAC) to non-continuous action spaces where modifications are inherently discrete. We evaluate the evasion performance of the proposed AMG-VAC on two reputable machine learning-based malware detectors. While the proposed method outperforms extant non-RL and RL-based AMG methods by statistically significant margins, we show that the obtained evasive action sequences are useful in shedding light on malware detectors' vulnerabilities.**

***Index Terms*—binary black-box attack, adversarial malware generation, static malware detection, reinforcement learning, variational actor-critic, approximate sampling**

I. Introduction

With the recent increase in the scale of malware attacks, machine learning (ML) and deep learning (DL) have been adopted to enable static malware detection based on the features automatically extracted from (parts of or the entire) malware executable [1]. Static malware detectors have gained attention due to being faster and far less resource intensive than dynamic malware analysis [2]. These malware detectors have shown breakthrough performance in detecting unseen malware variants at an unprecedented scale. However, these ML-based malware detectors have shown to be vulnerable to adversarial attacks – functional malware executables meticulously modified by an adversary to fool the detector into recognizing them as benign [3]. This process is known as Adversarial Malware Generation (AMG) [4]. Although AMG is construed as a major threat when conducted by the adversary, it can be helpful to autonomous malware detection when performed on the defender side [5]. AMG can provide an effective way to improve malware detectors by learning from adversarial attacks and identifying the vulnerabilities of malware detectors [6]. Thus, studying the characteristics of adversarial attacks through AMG is a viable defense mechanism [7].

However, existing studies on AMG has three major limitations. First, most AMG methods require prior knowledge about the malware detector architecture, its parameters, or its confidence score [2], [8]–[10]. These assumptions are not in accordance with realistic attack scenarios in which the detector's information is often unknown [11]. Second, existing AMG research mostly focuses on the *additive* modifications, which involves appending content to the empty space at the end of the malware executable [12]. In reality, human adversaries can choose from a broad range of *editing* adversarial actions in modifying malware to evade detection. Third, while the sequential order of adversarial modifications is important to understand the vulnerabilities of malware detectors and detect modified malware, this aspect of AMG has not been ex-

This material is based upon work supported by the National Science Foundation (NSF) under the grants SaTC-1936370 and SFS-1921485. We would like to thank VirusTotal for providing the malware dataset and granting access to the APIs for malware functionality assessment.

tensively studied. To address these limitations, in this study, we propose a new threat model featuring a novel Reinforcement Learning (RL) method, AMG-VAC, to automatically construct realistic malware variants for evading ML- and DL-based static malware detectors. AMG-VAC enables more realistic AMG to help discover vulnerabilities of malware detectors. Drawing upon the state of the art in RL, AMG-VAC extends Variational Actor-Critic (VAC), to effectively emulate evasive malware variants. AMG-VAC applies a set of allowable additive and editing modifications (i.e., actions) on malware executables to generate evasive sequences of actions aiming to maximize the chance of evading malware detectors. The resultant evasive action sequences from AMG-VAC enable further analysis for better understanding the detector's vulnerabilities.

The main contributions of this paper are twofold. AMG-VAC offers an automated vulnerability discovery method for both advanced ML-based and DL-based static malware detectors without requiring any prior knowledge about their architecture or parameters. Furthermore, AMG-VAC extends VAC to operate in non-continuous action spaces where discrete sequential modifications on a malware executable can lead to evasive malware variants.

II. Related Work and Background

A. Adversarial Malware Example Generation

Adversarial Malware example Generation (referred to as AMG for brevity), is a specific type of adversarial example generation, an emerging deep learning research area [7], [13]. Adversarial example generation often aims at generating input data that misleads a model into incorrect classifications. A large body of studies on adversarial example generation focuses on image applications. Unlike image applications in which adversarial modifications are continuous (e.g., applying a noise signal), the modifications for malware executables are inherently discrete. Moreover, applying arbitrary modifications (common in adversarial example generation for images) to a malware executable is likely to affect the functionality of the executable. Accordingly, AMG concerns automatically generating such discrete functionality-preserving modifications to evade malware detectors. It is critical to verify malware detectors against AMG and improve their robustness as a viable defense mechanism [7]. This verification goes beyond current evaluation practices such as precision, recall, accuracy, and F_1-score. Overall, while AMG is damaging when utilized by adversaries, it could be beneficial for defenders to gain insights into their vulnerabilities and improve their robustness.

Motivated by the importance of AMG, numerous AMG methods have been proposed in the recent literature [2], [8]–[10], [14]–[20]. A large body of these studies target *white-box* attack scenario in which the attacker has full knowledge about the targeted DL-based static malware detector [2], [8], [21], [22]. These methods often rely on the gradient errors obtained from the malware detector, which are not accessible in real-world attacks. As a result, adversarial malware variants generated by these methods could be unrealistic. Another group of AMG studies target a more realistic *black-box* scenario, in which the adversary only has access to the confidence score produced by the detector or malware features that are important to the detector [9], [10], [14]–[17], [20], [23]. Few studies consider a *binary black-box* scenario in which the only information known to the attacker is whether a generated malware variant is able to evade the detector or not [12], [18], [19], [24]. Binary black-box attack scenario is the most realistic type of AMG due to its minimal reliance on the insider knowledge about the target malware detector. The binary black-box AMG approaches proposed by Dey et al. [12] and Ebrahimi et al. [24] rely on additive actions, which only allow adversaries to append additional content to the executable. However, human adversaries can leverage a number of other types of actions in developing adversarial attacks. For example, editing actions can create a wider range of modifications such as renaming the code sections in a malware executable. Anderson et al. [18] and Fang et al. [19] have shown that deep RL can model the interactions between the adversary and malware detector in order to learn effective editing actions that mislead the detector in binary black-box settings. However, tackling AMG with RL requires handling environments with combinatorially large state spaces (i.e., all possible permutations of editing a section name of a malware executable). These two studies employ mainstream Actor-Critic RL [18] and deep Q-learning [19], which are not specifically designed to handle very large state spaces [25]. To address this issue, and motivated by the benefits of RL in AMG applications, we next review state of the art in RL with high dimensional state spaces.

B. Deep Reinforcement Learning for AMG

RL features a Markov decision process in which an agent iteratively interacts with an environment [26]. Given a state $s_t \in S$, the agent takes an action $a_t \in A$. In response, the environment produces the reward $r(s_t, a_t)$. The ultimate goal of RL is to learn a policy $\pi(a_t|s_t)$ that maximizes the expected reward (i.e., accumulated reward in the long run). Learning the policy is accompanied with estimating the state-action value $Q_\pi(s, a) = E[R|s, a]$. Deep RL has shown breakthrough performance in estimating π and Q using neural networks [27]. Accordingly, learning the behavior of an adversary is equivalent to learning a neural network that parameterizes the policy π. Finding the policy $\pi(a_t|s_t, \theta)$ parametrized by vector θ that maximizes the reward leads to solving an optimization problem given in Eq. 1.

$$J(\theta^*) = \max_\theta \sum_{t=0}^{\infty} E_\tau[r(s_t, a_t)] \tag{1}$$

where $\tau = s_0, a_0, \ldots, s_t, a_t$ denotes the trajectory (sequence of state-actions). As seen in Eq. 1, unlike black-box AMG methods, utilizing the reward signal eliminates the need for the confidence score from the malware detector. Thus, deep RL is naturally suited for modeling the interactions between the adversary and malware detector in AMG. Fig. 1 shows the RL framework for modeling these interactions.

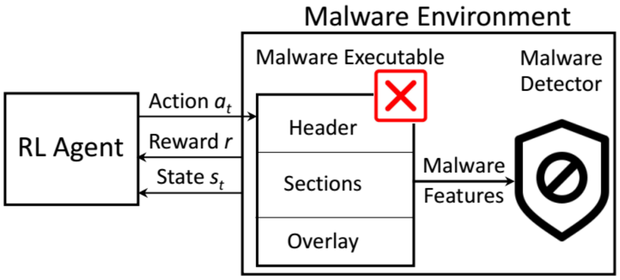

Fig. 1. Illustration of AMG in RL Settings

As shown in Fig. 1, the RL agent aims to mimic the adversary by applying functionality-preserving actions to different parts of a malware executable. At the highest level, a malware executable consists of Header (i.e., metadata), Sections (i.e., executable code and data), and Overlay (i.e., the free space at the end of the file that is often not executed). Static malware detectors extract features from these parts to classify the file either as malicious or benign. The RL agent is rewarded only when the series of applied actions lead to a functional malware variant that evades the detector. The states are modeled as the set of all possible features from the malware executable [28]. To model the adversary's actions with deep RL, two considerations are needed. First, as the attack vector is discrete (e.g., editing malware timestamp), the action space (i.e., set of possible modifications that an adversary can apply to a malware executable) is inherently discrete [2]. Consequently, Anderson et al. [18] and Fang et al. [19] utilize Actor-Critic with Experience Replay (ACER) [29] and Double Deep Q Network (D-DQN) [30] as two recently proposed RL methods suitable for discrete action spaces to conduct AMG. Second, as AMG involves large sequences of bytes, deep RL is required to process a combinatorially large number of states [18] in the environment. The number of states, for instance, amounts to at least 16^{2048} states for a 2KB modification in a hex-coded malware executable. While useful, ACER and DDQN are not specifically designed for very large state spaces. Recently, Fellows et al. [25] have proposed Variational Actor-Critic (VAC) that has yielded the state-of-the-art performance in tasks with very large number of states. VAC results in 33% more cumulative reward over ACER and DDQN on average [25]. However, VAC is not directly applicable to discrete action spaces. Motivated by its success in high-dimensional state spaces, we propose a novel method to extend VAC to discrete action spaces. We next introduce a threat model to conduct AMG and discuss core VAC framework. Finally, we introduce our proposed AMG-VAC to conduct AMG using VAC on discrete action spaces.

III. Method

A. Threat Model

Following [18] and [31], we define the threat model for binary black-box AMG attacks against static malware detectors. Unlike the threat model in [18], our threat model is able to launch attacks against both traditional machine learning- and deep learning-based malware detectors. Our threat model consists of three components:

- `Adversary's Goal`: Automatically generating malware variants that evade static ML- and DL-based detectors.
- `Adversary's Capability`: Applying allowable (i.e., functionality preserving) additive and editing actions on malware binary.
- `Adversary's Knowledge`: The parameters and deep learning architecture of the malware detector are not available to the adversary. Moreover, the adversary does not have access to any real-valued confidence score generated by the detector. The only available information is whether the generated malware variant can evade the detector or not (binary black-box scenario).

To implement this threat model, we first introduce the baseline VAC method and then show how AMG-VAC builds on VAC to accomplish AMG by discretizing the action space using approximate sampling.

B. Preliminary: Variational Actor-Critic (VAC)

VAC builds upon Actor-Critic (AC) model [32]. AC has two main iterative steps carried out by two complementary components called *actor* and *critic*:

1) Policy improvement, in which the *actor* finds a policy π that is compatible with current action value function Q. The *actor* is characterized by a neural network that accepts states and outputs actions to estimate the policy distribution $\pi_\theta(a|s)$
2) Policy evaluation, in which the *critic* estimates the action value function consistent with the current policy π. The *critic* is characterized by a neural network that accepts state-action pair and outputs the expected value of state-action pairs to estimate $Q_w(s,a)$.

Derived from Eq. 1, this iterative process translates to AC's objective given in Eq. 2:

$$J(\theta) = E_\tau[\sum_{t=0}^{\infty} \log \pi_\theta(a_t|s_t) Q_w(s_t,a_t)] \tag{2}$$

in which, $\pi_\theta(a_t|s_t)$ is learned by the *actor* network, while $Q_w(s_t,a_t)$ is learned by the *critic* network. In their seminal work, Fellow et al. [25] show that AC's learning objective is solved more effectively via variational inference and propose Variational AC (VAC). Variational inference approximates the action posterior $\pi_\theta(a_t|s_t)$ with a tractable family of distributions suitable for large number of states. However, VAC's policy is assumed to have a continuous distribution, and thus is not directly applicable to discrete action spaces required in AMG [25].

C. The Proposed Variational Actor-Critic for Discrete Adversarial Malware Generation (AMG-VAC)

Our proposed AMG-VAC aims to emulate the adversary's evasion behavior, given a set of discrete actions that do not change the functionality of malware executable. To this end, AMG-VAC discretizes continuous actions from the VAC's actor network to accommodate for discrete additive and editing actions via an approximate sampling operator. The overview of AMG-VAC and the approximate sampling is depicted in Fig. 2.

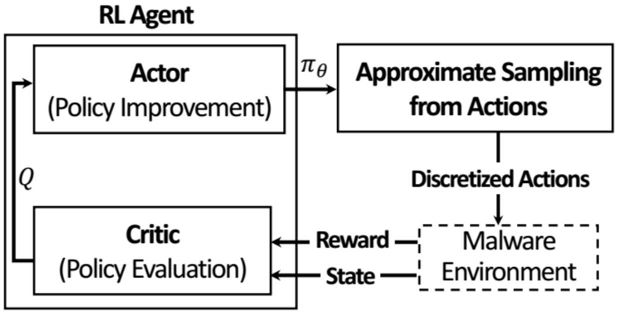

Fig. 2. Abstract view of the proposed AMG-VAC

The RL agent consists of actor and critic network explained earlier. The components of the malware environment were depicted in Fig. 1. The approximate sampling component outputs approximated actions given the policy π_θ. We next characterize the action space and the procedure of approximate sampling in AMG-VAC.

1) Action Space: Following Anderson et al. [18] and Fang et al. [19], we identified ten functionality-preserving actions including five additive and five editing actions shown in Table I. Additive and editing actions are denoted by A and E, respectively.

TABLE I
DISCRETE ACTION SPACE FOR AMG-VAC

Action Name	Description	Type
Add Import (AI)	Adds a library or function to the import table	A
Add Section (AS)	Adds a new section to executables	A
Break Checksum (BC)	Sets file's checksum	E
Change Timestamp (TS)	Sets timestamp	E
Overlay Append (OA)	Appends Bytes to the end of PE file	A
Remove Debug (RD)	Unlinks debug section from header	A
Remove Signature (RS)	Unlinks digital signature from certificate table	A
Section Rename (SR)	Change sections' name in malware executable	E
UPX Compression (UC)	Compress malware executables	E
UPX Decompression (UD)	Decompress malware executables	E

2) Approximate Sampling of Actions with Concrete Distribution: As mentioned above, the actor network in VAC performs policy improvement and outputs actions to interact with the environment. The policy improvement translates into estimating $\pi_\theta(a|s)$ via the actor neural network (shown in Fig. 2). The actor network learns a distribution over actions. Learning this action distribution in AMG involves sampling from discrete actions described in Table I. As the sampling operation is not differentiable, it inhibits gradient propagation in stochastic gradient ascent – required for learning the neural network parameters [33]. That is, the discrete stochastic action sampling precludes the gradient flow needed for policy improvement in the actor network. Maddison et al. [34] showed that discrete samples can be approximated via using a differentiable *Concrete* distribution. Concrete distribution, also known as Gumbel-Softmax distribution, is a well-established differentiable distribution in statistical machine learning [33]. Inspired by Maddison et al. [34], we propose to approximate the discrete action distribution over discrete AMG modifications (shown in Table I) by *Concrete* distribution.

Let π_k denote the probability assigned to each action a_k by the policy (i.e., actor) network. Approximate sampling from discrete actions with *Concrete* distribution entails a few sequential steps. First, the probability of the sampled discrete action from the actor network is projected to a logarithmic scale for numerical stability [34]. Second, independent and identically distributed (i.i.d.) random variables are sampled from Gumbel distribution G_k. Third, the obtained sample from Gumbel distribution is combined with the discrete action probabilities via component-wise addition and a softmax operator. The softmax operator is parameterized by temperature τ that could be hand-tuned. Following this process leads to a *Concrete* distribution over the outputs of the actor network, which is expressed via Eq. 3.

$$a_k = \frac{\exp((\log \pi_k + G_k)/\tau)}{\sum_{i=1}^{n} \exp((\log \pi_i + G_k)/\tau)} \tag{3}$$

where the obtained output (i.e., action sample a_k) is differentiable and thus can be used in learning the parameters of the actor network via gradient ascent. Such an approximate sampling process enables AMG-VAC to learn policies on discrete action spaces seen in modifications on malware executables. The discrete samples are approximated via using a differentiable *Concrete* distribution over the attack vectors after receiving the discrete actions from the actor network. The novelty of AMG-VAC lies in extending VAC via approximate sampling to handle discrete action environments in AMG applications while still benefiting from stochastic gradient ascent.

IV. EVALUATION

We conducted a series of evaluation experiments to rigorously assess AMG-VAC on emulating adversarial malware files in comparison with baseline AMG methods.

A. Research Testbed

Following the malware testbed construction in [18], [19], our research testbed comprised approximately 19,650 (15.38 GB) recent Windows malware executables from VirusTotal – a renowned aggregator of emerging malware from multiple contributors worldwide [18], [35]. The date of these malware executables ranges from 2017 to 2019. About 3,429 (1.68 GB) of the malware executables were dedicated for training the AMG-VAC. The testbed included five types of common malware files, including botnet, ransomware, rootkit, spyware, and virus. The distribution of these types is given in Table II.

TABLE II
DISTRIBUTION OF MALWARE TYPES IN OUR TESTBED

Malware Type	**Train Size**
Botnet	526 (151.2 MB)
Ransomware	900 (454.2 MB)
Rootkit	731 (511.1 MB)
Spyware	640 (377.2 MB)
Virus	659 (186.3 MB)
Total	3,429 (1.68 GB)

B. Experimental Setup

The performance of AMG-VAC was evaluated in comparison with the state-of-the-art AMG methods. In our experiment, all AMG methods were trained on the 1.68 GB of training malware testbed. The generated adversarial malware variants of these trained AMG methods were tested against two renowned malware detectors: EMBER and MalConv. Endgame Malware Benchmark for Research (EMBER) is a well-established malware detector developed by Endgame Inc. EMBER leverages gradient boosting and gradient-based sampling for malware detection [18], [36]. MalConv is a premier deep learning-based malware detector, developed by the Laboratory of Physical Sciences [1]. MalConv uses a large convolutional neural network (CNN), which was trained on over a million malware executables.

We adopted the evasion rate metric as our performance evaluation criterion. Evasion rate is a widely used metric for measuring the AMG performance [18], [24], [37]. The evasion rate of an AMG method against a given malware detector is defined as follows:

$$Evasion\ Rate = \frac{|E \cap F|}{N} \tag{4}$$

Where E is the set of generated adversarial malware variants capable of evading the malware detector, F is the set of generated adversarial malware variants that are functional, and N is the total number of adversarial malware variants generated by the AMG method. A higher evasion rate suggests that the AMG method is capable of evading the malware detector more effectively.

AMG-VAC was compared against three state-of-the-art black-box and binary black-box AMG methods: Benign Feature Append (BFA) [9], [17], Double Deep Q-Network (DDQN) [19], [30], and Actor-Critic with Experience Replay (ACER) [18], [29]. BFA is a widely-adopted non-RL method that appends parts of benign files to the end of malware executables to generate evasive malware variants. BFA is a black-box attack method and has access to the confidence score of the malware detector. Therefore, BFA benefits from more *insider* information than binary black-box benchmark methods (i.e., DDQN, ACER, and AMG-VAC). Double Deep Q-Network (DDQN) is a well-established RL-based binary black-box method that leverages two neural networks for better estimation of the action-value function. Actor-Critic with Experience Replay (ACER) is an effective RL-based binary black-box method that yields the malware variant by applying variance reduction techniques to reduce the instability of learned policies in the baseline AC model [29]. Consistent with [18] and [19], all experiments were implemented using a modified version of the OpenAI Gym environment with the same parameter settings and neural network sizes.

1) Functionality Preservation: The functionality of all generated adversarial malware variants were checked using the academic license of VirusTotal API. The API provides a malware behavior report, including static and dynamic analysis of the malware file. The report reflects the network behavior of the malware file, file access patterns, etc. We compared the behavior reports of the modified malware variants with the unmodified malware files to ensure that the behavior stays the same after modification. With this process, we assured that the modified malware files can be executed on Windows operating system while maintaining their malicious behavior.

C. Results

Table III summarizes the benchmark evaluation results, comparing AMG-VAC's performance against two renowned ML- and DL-based malware detectors (EMBER and MalConv) across five malware types. We identified two major findings from these results. First, while RL-based AMG methods (DDQN and ACER) did not use the confidence score from the malware detector for generating malware variants, the malware variants generated by these methods were more effective (e.g., average of 28.44% and 37.18% for EMBER) than those generated by the confidence score-based BFA (e.g., average of 3.90% for EMBER). This finding highlights the general effectiveness of RL in AMG. Second, and more importantly, AMG-VAC outperformed the state-of-the-art baseline methods across all malware types and on both malware detectors with statistically significant margins as measured by paired t-test.

Overall, while BFA had the lowest evasion rate across all malware categories, RL-based AMG methods had significantly higher evasion rates for all malware types. On average, AMG-VAC's evasion rate across all five malware types and against both malware detectors was considerably higher than all baseline methods with 51.67% for EMBER, and 44.01% for MalConv. Specifically, AMG-VAC outperformed the second best-performing method (ACER) by 14% (51.67% vs. 37.18%) for EMBER and by 9% (44.01% vs. 35.04%) for MalConv on average. These results indicate that enhancing VAC to operate on discrete action spaces via our proposed AMG-VAC yielded considerably stronger adversarial malware variants that are capable of evading malware detectors.

D. Discussion

To gain further insight into the vulnerabilities of malware detectors, we qualitatively analyzed AMG-VAC's output by

TABLE III
COMPARING AMG-VAC'S PERFORMANCE (EVASION RATE) AGAINST TWO RENOWNED MALWARE DETECTORS ACROSS FIVE MALWARE TYPES

Malware Detector	**Method**	**Botnet**	**Ransomware**	**Rootkit**	**Spyware**	**Virus**	**Average**
EMBER	BFA	3.02%	4.44%	4.73%	5.34%	6.16%	3.90%
	DDQN	23.00%	44.33%	39.12%	19.80%	27.77%	28.44%
	ACER	30.99%	60.11%	27.51%	26.87%	62.82%	37.18%
	AMG-VAC (Ours)	**48.29%***	**65.22%***	**61.15%***	**29.53%***	**82.40%***	**51.67%***
MalConv	BFA	3.90%	4.42%	2.96%	3.28%	5.56%	3.76%
	DDQN	6.08%	11.89%	16.83%	27.50%	30.50%	16.63%
	ACER	37.07%	25.33%	29.41%	56.09%	44.76%	35.04%
	AMG-VAC (Ours)	**44.68%***	**26.89%***	**50.48%***	**65.31%***	**48.41%***	**44.01%***

Best performances are highlighted in boldface fonts. Asterisks denote that P-values evaluated by paired t-test are significant at 0.05.

TABLE IV
MOST EVASIVE ACTION SEQUENCES PRODUCED BY AMG-VAC FOR GENERATING ADVERSARIAL MALWARE ATTACKS

Malware Detector	**Most Evasive Action Sequences**
EMBER	Append Import → Break Checksum → **Section Rename** → **Section Rename**
	Change Timestamp → **Add Import** → Change Timestamp → **Compression**
	Remove Signature → Overlay Append → **Compression** → **Section Rename**
MalConv	**Compression** → Remove Debug → Overlay Append → Overlay Append →Section Rename
	Add Import → Overlay Append → Change Timestamp
	Remove Debug → Remove Signature → **Add Import** → **Compression**

examining its action sequences that led to evasive malware variants. To this end, we obtained the most frequent actions leading to the creation of evasive malware variants against the EMBER and MalConv malware detectors. Table IV shows the top three most frequent action sequences for each malware detector.

For EMBER, two editing actions (i.e., Compression and Section Rename; boldfaced in Table IV) were the most frequent actions. Compression and Section Rename appeared in 16% and 14% of the evasive sequences, respectively. For MalConv, while Compression was also the most frequent action in evasive attacks with 39% occurrence, Import Append (boldfaced in Table IV) was the second most frequent additive action with 12% occurrence. In sum, the sequences generated by AMG-VAC provided three useful observations about the examined malware detectors. First, the Compression action affected both ML- and DL-based malware detectors as the most effective action in generating malware variants. Second, EMBER was more vulnerable to editing actions (e.g., Section Rename). This is expected since EMBER's decisions are mainly based on the features extracted from the executable's metadata, which is not modified by additive actions (e.g., Add Import). Third, unlike EMBER, MalConv was more vulnerable to additive actions as MalConv is based on automated representation learning from the whole malware executable. Such observations from examining AMG-VAC's sequences can lead to better adversarial attack mitigation for static deep learning-based malware detectors.

V. CONCLUSION

It is vital to defend malware detectors against evolving adversaries who can generate adversarial attacks at scale. This calls for automated adversarial malware generation (AMG) at the defender side. To emulate adversarial malware attacks, we propose AMG-VAC, a novel RL method designed specifically to support discrete modifications of malware executables in AMG tasks. Through rigorous evaluation, we show that AMG-VAC outperforms extant RL-based and non-RL-based AMG methods. AMG-VAC contributes to deep learning research community by offering a novel approach to extending the state-of-the-art RL framework to AMG. Furthermore, AMG-VAC is an effective and explainable AMG technique that contributes to the malware analysis research community. A promising future direction could be a rigorous procedure for using the adversarial malware variants generated by AMG-VAC to enhance the robustness of DL-based malware detectors against adversarial attacks.

REFERENCES

[1] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[2] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Adversarial examples on discrete sequences for beating whole-binary malware detection," *arXiv preprint arXiv:1802.04528*, pp. 490–510, 2018.

[3] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019, publisher: IEEE.

[4] G. Apruzzese, M. Colajanni, L. Ferretti, and M. Marchetti, "Addressing Adversarial Attacks Against Security Systems Based on Machine Learning," in *2019 11th International Conference on Cyber Conflict (CyCon)*, vol. 900. IEEE, 2019, pp. 1–18.

[5] R. Goosen, A. Rontojannis, S. Deutscher, J. Rogg, W. Bohmayr, and D. Mkrtchian, "Artificial Intelligence Is a Threat to Cybersecurity. It's Also a Solution," Boston Consulting Group (BCG), Tech. Rep., 2018. [Online]. Available: https://image-src.bcg.com/Images/BCG-Artificial-Intelligence-Is-a-Threat-to-Cyber-Security-Its-Also-a-Solution-Nov-2018_tcm9-207468.pdf

[6] W. E. Zhang, Q. Z. Sheng, A. Alhazmi, and C. Li, "Adversarial Attacks on Deep Learning Models in Natural Language Processing: A Survey," 2019.

[7] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," *Communications of the ACM*, vol. 61, no. 7, 2018.

[8] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 533–537.

[9] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 8–14.

[10] D. Park, H. Khan, and B. Yener, "Generation & evaluation of adversarial examples for malware obfuscation," in *18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019, pp. 1283–1290.

[11] W. Hu and Y. Tan, "Black-box attacks against rnn based malware detection algorithms," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] S. Dey, A. Kumar, M. Sawarkar, P. K. Singh, and S. Nandi, "EvadePDF: Towards Evading Machine Learning Based PDF Malware Classifiers," in *International Conference on Security & Privacy*. Springer, 2019, pp. 140–150.

[13] J. Monteiro, I. Albuquerque, Z. Akhtar, and T. H. Falk, "Generalizable Adversarial Examples Detection Based on Bi-model Decision Mismatch," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 2839–2844.

[14] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Efficient black-box optimization of adversarial windows malware with constrained manipulations," *arXiv preprint arXiv:2003.13526*, 2020.

[15] R. L. Castro, B. Biggio, and G. Dreo Rodosek, "Attacking malware classifiers by crafting gradient-attacks that preserve functionality," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2565–2567.

[16] R. L. Castro, C. Schmitt, and G. D. Rodosek, "Armed: How automatic malware modifications can evade static detection?" in *2019 5th International Conference on Information Management (ICIM)*. IEEE, 2019, pp. 20–27.

[17] B. Chen, Z. Ren, C. Yu, I. Hussain, and J. Liu, "Adversarial examples for cnn-based malware detectors," *IEEE Access*, vol. 7, pp. 54 360–54 371, 2019.

[18] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to evade static pe machine learning malware models via reinforcement learning," *arXiv preprint arXiv:1801.08917*, 2018.

[19] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou, and H. Huang, "Evading anti-malware engines with deep reinforcement learning," *IEEE Access*, vol. 7, pp. 48 867–48 879, 2019, publisher: IEEE.

[20] M. Shahpasand, L. Hamey, D. Vatsalan, and M. Xue, "Adversarial attacks on mobile malware detection," in *2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*. IEEE, 2019, pp. 17–20.

[21] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. O'Reilly, "Adversarial deep learning for robust detection of binary encoded malware," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 76–82.

[22] L. Chen, Y. Ye, and T. Bourlai, "Adversarial machine learning in malware detection: Arms race between evasion attack and defense," in *2017 European Intelligence and Security Informatics Conference (EISIC)*. IEEE, 2017, pp. 99–106.

[23] M. Sharif, K. Lucas, L. Bauer, M. K. Reiter, and S. Shintre, "Optimization-guided binary diversification to mislead neural networks for malware detection," *arXiv preprint arXiv:1912.09064*, 2019.

[24] M. Ebrahimi, N. Zhang, J. Hu, M. T. Raza, and H. Chen, "Binary Black-box Evasion Attacks Against Deep Learning-based Static Malware Detectors with Adversarial Byte-Level Language Model," in *AAAI workshop on Robust, Secure, and Efficient machine Learning (RSEML)*. AAAI, 2021. [Online]. Available: arXiv preprint arXiv:2012.07994

[25] M. Fellows, A. Mahajan, T. G. Rudner, and S. Whiteson, "Virel: A variational inference framework for reinforcement learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 7120–7134, reporter: Advances in Neural Information Processing Systems.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[27] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, and others, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

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

[29] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample Efficient Actor-Critic with Experience Replay," 2017, meeting Name: International Conference on Learning Representations (ICLR).

[30] H. v. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press, 2016, pp. 2094–2100.

[31] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[32] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1587–1596, reporter: Proceedings of the 35th International Conference on Machine Learning.

[33] E. Jang, S. Gu, and B. Poole, "Categorical Reparameterization with Gumbel-Softmax," 2017, meeting Name: International Conference on Learning Representations (ICLR).

[34] C. J. Maddison, A. Mnih, and Y. W. Teh, "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables," 2017, meeting Name: International Conference on Learning Representations (ICLR).

[35] A. Kyadige, E. Rudd, and K. Berlin, "Learning from Context: A Multi-View Deep Learning Architecture for Malware Detection," in *3rd Deep Learning and Security Workshop*. IEEE, May 2020. [Online]. Available: https://ai.sophos.com/presentations/learning-from-context-a-multi-view-deep-learning-architecture-for-malware-detection/

[36] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, pp. 3146–3154, 2017.

[37] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, and H. Yin, "Automatic generation of adversarial examples for interpreting malware classifiers," *arXiv preprint arXiv:2003.03100*, 2020.