Learning Contextualized Action Representations in Sequential Decision Making for Adversarial Malware Optimization

Reza Ebrahimi, *Senior Member, IEEE*, Jason Pacheco, James Hu, *Member, IEEE*, and Hsinchun Chen, *Fellow, IEEE*

***Abstract*—Deep learning (DL)-based malware detectors have shown promise in swiftly detecting unseen malware without expensive dynamic malware behavior analysis. These detectors have been shown to be susceptible to adversarial malware variants generated from meticulously modifying known malware to mislead detectors into recognizing them as benign. Being able to automatically generate optimized functional adversarial malware variants by defenders is crucial to effective cyber defense and staying ahead of the adversary. Current adversarial malware example generation methods often assume threat models with any of the following four restrictions: (1) requiring access to insider knowledge about malware detectors, (2) an unlimited size of adversarial modifications, (3) an unlimited number of queries to malware detector, and (4) relying on dynamic analysis of malware behavior in a sandbox. Drawing on Actor-Critic Reinforcement Learning (RL), we propose a novel closed-box binary manipulation method for adversarial malware optimization, named Actor-Critic with Contextualized Action Representations (AC-CAR), to generate malware variants without these restrictions. AC-CAR leverages two novel components, a contextualized policy and a neural language model-based RL-augmented top-k sampling method. Unlike current methods, AC-CAR can utilize tens of thousands of actions to augment malware executables for evading DL-based malware detectors. AC-CAR yields an approximately 2-fold performance increase over the current methods on average, while decreasing the payload size to 20 times smaller than leading methods. We show that using the malware variants generated by AC-CAR in an adversarial re-training procedure improves malware detector' robustness against adversarial variants by 29.65% on average.**

***Index Terms*—Actor-critic reinforcement learning, adversarial malware example generation, contextualized action representations.**

Received 10 September 2023; revised 12 June 2024; accepted 5 October 2024. Date of publication 9 October 2024; date of current version 15 May 2025. This work was supported in part by National Science Foundation (NSF) under Grant CNS-1936370 (SaTC CORE) and Grant DGE-1921485 (SFS). *(Corresponding author: Reza Ebrahimi.)*

Reza Ebrahimi is with the School of Information Systems and Management, University of South Florida, Tampa, FL 33620 USA (e-mail: ebrahimim@usf.edu).

Jason Pacheco is with the Department of Computer Science, University of Arizona, Tucson, AZ 85721 USA.

James Hu and Hsinchun Chen are with the Artifical Intelligence Lab, University of Arizona, Tucson, AZ 85721 USA.

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2024.3477272, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2024.3477272

I. Introduction

CYBER attacks cost the U.S. economy $109 billion per year [1]. Malware attacks are the costliest type of these cyber attacks [2]. As an example, the cost of global ransomware damage was estimated to be $20 billion by the end of 2021 [3]. Modern malware detectors are an integral component of cyber defense protecting Information Technology (IT) infrastructure [4]. Despite their usefulness, traditional malware detectors depend on predefined signatures (often determined and vetted by malware analysts) to match identified patterns against known malicious files [5]. Consequently, signature-based methods can be ineffective in detecting unseen variants of malware (e.g., zero-days) since they rely on manually defined rules that cannot keep up with the rapid evolution of malware variants [6]. On the contrary, newly emerged DL-based malware detectors are able to process raw malware executable content without manual feature/signature engineering. These malware detectors can extract salient features (representations) that are crucial for detection from the raw malware content automatically. As a result, in the past few years, successful DL-based malware detectors have emerged [7], [8], [9]. Given their success in the early detection of new malware variants without human involvement, leading cybersecurity providers, including Avast, Endgame, and Symantec, have started adopting DL-based malware detectors in their antivirus products [10], [11]. Despite the success of DL-based malware detectors, they have been shown to be vulnerable to adversarial malware attacks. These attacks feature an adversary that meticulously modifies a malware executable (while preserving its malicious functionality), such that the malware detector recognizes the modified variant as a benign executable [6], [11], [12], [13], [14], [15], [16], [17], [18], [19]. These modified variants are known as adversarial examples (AEs) in adversarial machine learning literature [20]. For brevity, we refer to the task of generating AEs in the malware analytics context as Adversarial Malware example Generation (AMG). The abstract view of the AMG process is shown in Fig. 1, in which a known (i.e., detectable) malware is modified such that it evades a targeted DL-based malware detector.

When employed by defenders, AMG offers a vital mechanism to emulate adversarial malware variants in order to train and improve the robustness of DL-based malware detectors against adversarial malware attacks [21]. However, current AMG

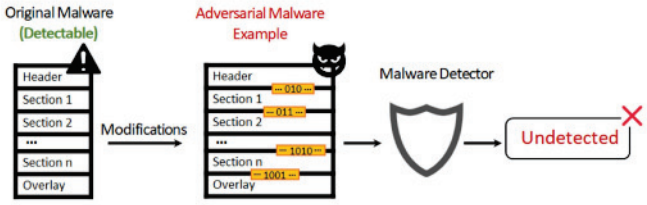


Fig. 1. Abstract View of AMG. The process involves modifying a Windows malware executable by injecting benign-looking content into its file sections such that it evades a targeted malware detector while preserving its malicious functionality.

methods often deploy threat models with any of these three restrictions. (1) They often require insider knowledge (i.e., white- or gray-box access) about malware detectors (e.g., architecture or parameters of the detector model). While this is often justified by the Kerckhoff's principle, the information about the architecture of the malware detector is often unknown to the attacker. Thus, such threat models could lead to focusing on adversarial malware variants that might be remote from real-world adversaries [22], [23]. In real adversarial malware deployment scenarios, outsider threats (e.g., AI-enabled adversaries) are also of major concern for organizations. [22]. (2) Most extant works assume an unlimited size of adversarial modifications (known as payload) and unconstrained number of queries to the malware detector. Allowing an unlimited size of modifications and number of queries can lead to threat models that lack stealth [11]. (3) Some extant work require dynamic analysis of malware behavior in a sandbox. Relying on dynamic analysis is expensive and often requires human interpretation of the malware behavior. Moreover, fewer methods that operationalize closed-box adversarial attacks support a limited number of adversarial actions [5], [24] making them less effective as it could lead to generating limited adversarial variants and larger modification (payload) sizes. Given the crucial role of AMG, our study aims to propose a novel and practical closed-box binary manipulation AMG framework for adversarial malware optimization that alleviates these restrictions without requiring reverse engineering, access to the malware source code, or dynamic behavioral analysis.

Creating adversarial malware examples entails sequential decision-making (SDM), which involves repeatedly choosing an action based on the current input, previously generated output, current state, and the feedback from the malware detector [25]. Deep reinforcement learning (RL) offers breakthrough results in solving SDM problems with large state spaces by representing the space with deep architectures [26]. However, modern RL problems often involve large action spaces in addition to large state spaces [27]. For instance, in AMG, the action space could involve all possible valid combinations of added bytes, a space that grows exponentially (e.g., 2^{100} for injecting 100 bytes to a malware executable). Learning representations of actions has shown to be useful to deal with large action spaces [27]; yet lacking in the RL-based AMG literature. Moreover, the actions are often selected regardless of the context in which they appear [28]. As a concrete example, the occurrence of 'MZ' (with the hexadecimal byte equivalent of '4d5a') at the beginning of the file indicates starting a Microsoft Windows executable, while appearing in other positions could indicate a typical string variable with different semantics. Recent neural language models can help contextualize the action space to attend to the position of bytes. Drawing on Actor-Critic Reinforcement Learning (RL) theory and contextualized neural language models, we propose a novel Actor-Critic with Contextualized Action Representations (AC-CAR) algorithm that operates in large action spaces while attending to the context to automatically generate realistic malware variants without assuming the above four restrictions.

AC-CAR learns context-aware action representations to generate adversarial malware variants that improve the robustness of DL-based malware detectors against unseen adversarial malware variants. To achieve this, AC-CAR uses deep RL to identify sequences of actions required to evade the malware detector and employs neural language models to contextualize action representations in deep RL. AC-CAR can utilize tens of thousands of modification types (actions) to augment malware executables for evading DL-based malware detectors. AC-CAR yields a 2-fold increase in the evasion performance over the state-of-the-art benchmark methods while decreasing the payload size to 0.5KB (20 times smaller than the current leading AMG methods) and limiting the average number of queries to the malware detector to 50. To foster reproducibility, we made AC-CAR implementation publicly available on GitHub at https://github.com/star-ailab/ac-car. To demonstrate practical utility, we use malware variants generated by AC-CAR to improve the robustness of malware detectors against adversarial ransomware attacks. We discuss the practical implications of AC-CAR for cybersecurity organizations and security practitioners.

II. Related Work: Adversarial Malware Generation

DL models have been shown to be vulnerable when an adversary modifies their input data through subtle perturbations. These perturbations result in the creation of AEs that can cause a certain DL model to produce wrong results. As a specific-purpose DL-based model, malware detectors are also vulnerable to AEs. AMG entails automatically generating these AEs by the defender to identify the vulnerabilities of malware detectors and improve their robustness [24]. Based on the amount of information about the malware detector model that is available to the adversary, four threat models can be envisioned for AMG [5], [29]: white-box, gray-box, closed-box, and binary closed-box. The white-box threat model pertains to attacks in which the adversary has full access to the structure and parameters of the malware detector model. In the gray-box threat model, the parameters of the malware detector model are not available, but the adversary has access to the features that affect the detector's output. closed-box (BB) threat model features an adversary that does not have access to model's specifications or features; however, it can observe confidence scores [30]. The confidence score is a real-valued feedback between 0 and 1 from the malware detector, which indicates the probability that the input file is malicious. Lastly, the binary closed-box (BBB) threat model features closed-box attacks that neither assume a priori knowledge (i.e., model parameters, architecture, or features) of the malware detector nor require access to its confidence

TABLE I
SELECTED RECENT AMG RESEARCH WITH CLOSED-BOX AND BINARY CLOSED-BOX THREAT MODELS

Authors	Data Source	Attack Method	Targeted Malware Detector	Threat Model	Model's Input	Payload Size	No. of Queries	Sandbox Use	Action Space Size	Action Context Aware -ness
Abusn-aina et al. [31]	VirusTotal, VirusShare	Benign feature append	DL-based	BB	ME	UNL	UNL	Yes	-	-
Song et al. [32]	VirusTotal	MAB	DL-based	BBB	ME	NR	60	No	13	No
Phan-et al. [36]	VirusTotal, VirusShare	Deep RL	DL-based	BB	ME	NR	80	No	9	-
Demet-rio et al. [13]	VirusTotal, GitHub	GA	DL-based	BBB	ME	~300 KB	30-110	No	-	-
Ebrahi-mi et al. [24]	VirusTotal	Deep RL	GBT	BBB	ME	10 KB	50	No	10	No
Song et al. [11]	VirusTotal	Code randomiza -tion	GBT, Signature-based	BBB	ME	10 KB	50	No	-	-
Kucuk et al. [35]	Rbot	GA	DL-based	BB	API	UNL	UNL	No	-	-
Castro et al. [12]	VirusTotal	RP	Signature-based	BB	EF	NR	UNL	No	-	No
Chen et al. [6]	VirusShare, Malware-benchmark	Enhanced RP	DL-based	BB	ME	20 KB	UNL	No	-	-
Fang et al. [14]	VirusTotal	Deep RL	GBT	BBB	EF	NR	80	No	11	No
Park et al. [19]	Malmig, MMBig	DP	DL-based	BB	ME	NR	UNL	No	-	-
Rosenb-erg et al. [37]	VirusTotal	GAN	DL-based	BBB	API	NR	UNL	Yes	-	-
Suciu et al. [18]	VirusTotal, FireEye	Benign feature append	DL-based	BB	ME	10 KB	UNL	No	-	-
Ander-son et al. [5]	VirusTotal	Deep RL	GBT	BBB	ME	NR	NR	No	10	-
Hu & Tan [38]	Malwr	RNN	DL-based	BB	API	UNL	UNL	Yes	-	-

Note: DP=Dynamic Programming; GA= Genetic Algorithm; GAN=Generative Adversarial Network; GBT=Gradient Boosting Trees; MAB: Multi-arm Bandit; RP= Randomized Perturbations; RL=Reinforcement Learning; RNN=Recurrent Neural Network; SVM=Support Vector Machine; BB=Black-Box; BBB= Binary Black-Box; ME=Malware Executable; EF=Engineered Features; UNL=Unlimited; NR=Not Reported; '-'=Not Applicable

scores. In the binary closed-box threat model, the detector output signifies whether the sample has been classified as malware or benign-ware, based on which the adversary deduces whether or not the attack has been successful. Since the specifications and confidence score of the malware detector model are often unknown in practice, the binary closed-box threat model is the most realistic adversarial attack scenario [7], [11], [31], [32].

We also note that recent progress has been made in providing generic defenses against adversarial attacks using the control graph flow [33], function calls [34], and system calls [35] obtained from the source code, reverse engineering, or dynamic analysis to provide generic defense architectures against adversarial attacks. While a fruitful, this line of work is orthogonal to adversarial malware optimization which is our focus and thus the scope of our current work.

Consistent with our goal of proposing a more realistic threat model, we examine selected recent studies that support closed-box and binary closed-box AMG. While the selected studies are not meant to be exhaustive, they serve to reflect the latest advancements in AMG. We summarize these studies in seven key dimensions as shown in Table I in reverse chronological order.

These dimensions include data source, attack method, targeted malware detector model, threat model (BB, or BBB), detector model's input type (the entire raw malware executable (ME) versus manually engineered features (EF)), payload

(modification) size of adversarial malware variants in kilobytes (KB), and the number of queries to the malware detector. We note that the payload size and the number of queries are not independent and positively correlate. and For RL-based AMG methods, we additionally examine the action space size and context-awareness (the last two columns in Table I). As shown in Table I, studies that offer closed-box threat models do not require knowing the specifications of the targeted malware detectors [6], [12], [13], [18], [19], [31], [32], [38], [39]. These studies employ a wide range of methods such as genetic algorithms [13], [35], random perturbations [6], [12], dynamic programming [19], and RNN [38]. These methods heavily depend on the confidence score attained from the malware detectors. However, the confidence score is oftentimes internal to the malware detector and is not visible to the adversary in practice. As such, this could restrict their applicability [11], [37]. As opposed to the closed-box threat model, binary closed-box threat models do not require observing the confidence score [5], [14], [37], [40], and thus are closer to real attack scenarios. As observed in Table I, many binary closed-box studies target detectors that require manual feature engineering [12], [14], [39], [40], and hence are not the focus of our study. Within binary closed-box methods that focus on DL-based detectors, methods proposed in [37] and [38] both require API call sequences attained from dynamic analysis of the malware executable in a sandbox, which could be limiting in practice. Another stream of binary closed-box methods train a surrogate model locally by querying the closed-box malware detector [37], [38]. While these methods often require many interactions with the detector to construct a surrogate model, they often require the attacker to know the complete feature space of the detector model [5], [38]. As seen, deep RL-based AMG methods [5], [6], [24] lead to more realistic binary closed-box threat models that tend to offer minimal payload size (i.e., 10KB) and fewer number of queries to the malware detector model (50 queries per input file), even with sparse reward signals. Although deep RL AMG models yield promising results, they often operate on a small set of actions (e.g., 10 actions in [24] and 11 actions in [14]). Unfortunately, RL agents operating in limited action spaces can lack expressive power to solve complicated problems such as AMG [27]. Nevertheless, dealing with high-dimensional action spaces in AMG is an open research area in RL [5], [14]. Furthermore, as seen in Table I, current RL-based AMG approaches often do not support action context awareness that allows them to generate benign-looking content that can evade malware detectors more effectively.

In sum, based on our review of the AMG literature, several domain gaps are identified despite the latest remarkable advancements. Specifically, little work has been done on realistic attacks with minimal insider knowledge of the attack target (i.e., binary closed-box AMG). Additionally, many binary closed-box AMG studies are based on assumptions that allow an unlimited number of queries to the detector, or allow an unlimited payload size of adversarial modifications. While deep RL-based AMG is promising, current deep RL-based methods do not operate in large action spaces and are not context-sensitive, which could lead to a lack of evasiveness. To address the identified gaps, we pose the following research questions:

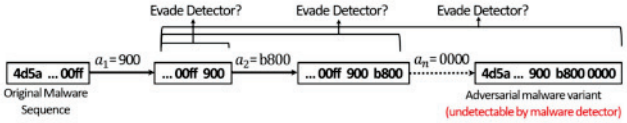


Fig. 2. Illustration of SDM process for AMG.

- How to conduct closed-box AMG with high-dimensional action spaces to improve defending malware detectors against adversarial malware variants?
- How can deep RL operate in context-aware large action spaces to conduct SDM for AMG?

We expect that casting AMG into an SDM problem which can be effectively solved through RL is a viable mechanism for operationalizing an AMG threat model that addresses these questions. In the remainder of this paper, we first describe the foundation of our proposed model as well as its positioning and contribution in Section III. We next introduce the proposed model and elaborate its components in Section IV. Section V presents an overview of the testbed and evaluations. We present the corresponding experiments' results in Section VI. We examine the practical utility of AC-CAR by a case-study and highlight its implications for cybersecurity in Section VII. Lastly, we discuss significant promising future directions for our study in Section VIII.

III. Model Preliminaries

A. Sequential Decision Making (SDM) for AMG

AMG is a specific type of Sequential Decision Making (SDM) - a field of AI, in which an agent repeatedly chooses actions based on the feedback from the environment and the previously taken actions [25], [41]. SDM is characterized by the fact that earlier decisions influence the later available choices. An example of SDM for AMG is illustrated in Fig. 2. As seen, the process starts with an original known malware executable. At each time step, the emulated adversary generates hexadecimal byte sequences (by taking corresponding actions $a_1, a_2, \ldots, a_n$), and injects them into the original malware content such that at a certain time step the generated malware variant can evade the malware detector.

While SDM can be formulated as an optimization problem for which the solution could be approximated by linear programming or heuristic-based methods, RL has been shown to be able to provide an effective framework for SDM with unknown environment specifications [26], [41]. The RL framework for AMG features an agent that interacts with a malware environment over discrete time steps until the agent learns a policy $\pi(a|s)$ (a distribution over actions given the current state), which produces actions that maximize the cumulative reward. The RL agent emulates the adversary by applying functionality-preserving action a_t (e.g., byte injections) to different parts of a malware executable. At a high level, a malware executable consists of a Header (encompasses a malware file's metadata), Sections (include executable code and data), and an Overlay (free spaces in the file that is not executed). Malware detectors extract features from these parts to classify the input file as

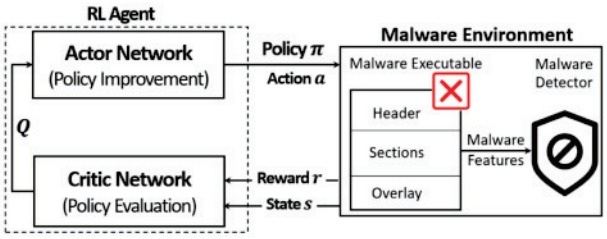


Fig. 3. Abstract View of AMG with Actor-Critic Deep RL. The RL agent includes actor and critic interacting with the malware environment to generate adversarial malware variants.

malicious or benign. At each time step, the agent receives an immediate reward r_t and observes the malware executable state s_t from the environment. The state represents any combination of bytes that forms a functional malware executable file. Once the actions lead to a functional malware variant that can evade the detector, the RL agent receives a positive real-valued reward. We present the state space, action space, and rewards in detail in Section IV-B.

Deep RL has shown promise in solving SDM problems in environments with large state spaces by learning a state representation rather than operating directly on the raw states [26]. Within deep RL models, actor-critic (AC) deep RL have yielded breakthrough results in complex environments [42], [43], [44], [45]. Specifically, recent AC deep RL models have shown to be capable of naturally modeling SDM for emulating the behavior of a variety of agents including real-life agents (e.g., animals) [25]. Accordingly, we next present AC deep RL as a promising approach for modeling SDM problems.

B. Actor-Critic Deep RL and Action Representation Learning

The AC model involves two iterative learning steps: policy improvement and policy evaluation. These steps are associated with two collaborative learning components named actor and critic, respectively. The abstract view of AC deep RL in the AMG context is illustrated in Fig. 3. In the policy improvement step, the actor finds a policy π that is compatible with the current state-action value function $Q(s,a) = \mathbb{E}[R|s,a]$, where R denotes the cumulative reward. In the policy evaluation step, critic estimates the state-action value function consistent with the current policy π.

As shown in Fig. 3, the actor network outputs policy distribution $\pi(a|s)$ estimated by a neural network that accepts states and outputs actions. The critic network outputs $Q(s,a)$ estimated by a neural network that accepts state-action pairs. In addition to generalizing over states by learning a state representation in deep RL, Sharma et al. [46] and Dulac-Arnold et al. [47] have shown that representation over actions is critical for learning in environments with large action spaces. For instance, it is evident that real-life agents decompose their plans into intermediate abstractions rather than the raw low-level control needed for movements [48]. This phenomenon is also known as primitive elements in robotics [49]. However, the methods proposed by Sharma et al. [46] and Dulac-Arnold et al. [47] require providing the action representations a priori. Recently, Zahavy et al. [50] and Chandak et al. [27] have shown that action representations can be learned autonomously. Zahavy et al. [50] propose learning to eliminate inconsequential actions, which enables learning in an environment with 1,000 actions. Chandak et al. [27] offer to learn a low-dimensional representation of salient actions, which enables learning in an environment with 4,096 actions. Their method employs an AC framework with Representation for Actions (AC-RA) that maps actions into a lower dimension space. As Chandak's method is closely related to our proposed framework we briefly describe its underlying mechanism first.

To generalize over large action sets in robotic applications, AC-CAR proposes to decompose the overall policy into two components: an internal policy $\pi_i : \mathcal{S} \times \mathcal{E} \rightarrow [0,1]$ that operates on a low-dimensional action representation space $\mathcal{E} \subseteq \mathbb{R}^d$ and the state space $\mathcal{S}$, and a deterministic embedding-to-action mapping function, $f : \mathcal{E} \rightarrow A$ that maps actions from the low-dimensional action representation space to the actual (high-dimensional) space such that the agent is able to interact seamlessly with the environment. One important aspect of their work is showing that there exist a parameterization of the overall policy that is equivalent to the canonical optimal policy, π^*. They also show that the mapping function f can be learned via a supervised loss $\mathcal{L}(f) = -\mathbb{E}[\ln(\hat{P}(a_t|s_t, s_{t+1}))]$, where $\hat{P}$ is an estimation of $P(a_t|s_t, s_{t+1})$.

However, in both [27] and [50], action representations are learned regardless of the context/position in which they appear. Not being inherently equipped with context-aware actions can make AC-RA less effective in AMG. It is expected that equipping AC with learning low-dimensional contextualized action representations can result in state-of-the-art performance in complex applications with high-dimensional action spaces such as AMG. To this end, we propose to enhance AC-RA to learn contextualized action representations for AMG application domain. Neural language models could provide a viable mechanism to enhance deep RL with contextualized action representations.

C. Contextualized Neural Language Models

Recent neural language models feature architectures that generate sequences resembling the content of a training set by learning contextual embeddings (continuous representations) of the discrete elements in a sequence. These embeddings are referred to as contextual since the representation of a specific token could vary when it appears in different contexts (i.e., surrounding sequence elements) [51]. These neural language models fall into two categories: masked and causal (also known as autoregressive) language models. Masked language models learn to predict a masked token from its surroundings. Renowned recent examples of masked language models include BERT [51] and RoBERTa [52]. On the contrary, causal language models learn to predict the next token based on the previously generated ones, which makes them a natural fit for SDM problems ([53], [54]). As a pioneering causal language model, Generative Pre-trained Transformer (GPT) [55] was first proposed with 117 million parameters, followed by GPT-2 [56] with 1.5 billion parameters, which led to breakthrough results in sequence generation tasks

such as natural language processing [54]. Recently, GPT-3 with 175 billion parameters was introduced as a proprietary tool [54]. Although GPT-3's output is available via API to researchers, its underlying architecture is not available to the public at the time of writing. The context-awareness in GPT and other causal language models is achieved by attending to the position of tokens in a sequence by self-attention [55]. Causal language models, including GPT, operate according to the principle that the probability distribution of a generated sequence given an initial context W^0 can be decomposed into the product of the conditional probabilities of previous elements as given in (1).

$$P\left(w^{1:N}|W^0\right) = \prod_{n=1}^{N} P\left(w^n|w^{1:n-1}, W^0\right) \tag{1}$$

The left-hand side in (1) denotes the probability distribution of a generated sequence given W^0, and $P(w^n|w^{1:n-1}, W^0)$ denotes the conditional probabilities over each token given previous tokens. Despite GPT's promise in sequence generation, two major challenges arise when utilizing GPT in SDM settings to address AMG. First, the evasiveness of the generated sequence depends on the choice of the initial context W^0. Second, GPT is not equipped with an internal mechanism to encourage evading malware detectors. These two challenges inhibit the direct use of GPT for SDM applications such as AMG. In Section IV, we describe how our proposed model addresses these two challenges leveraging RL. We next highlight the contribution of our proposed model to the AMG literature and its positioning against major recent RL methods.

D. Model Positioning and Contribution

From a methodological perspective, the current AC deep RL methods do not support context-awareness in the action space. While GPT could be a viable solution to endow AC with context-awareness, it does not provide a mechanism to learn generating sequences that promote evasiveness. To address this gap, our AC-CAR offers a novel solution to enhance causal neural language models to accomplish detector evasion in adversarial settings. Specifically, AC-CAR learns effective context-aware action representations to automatically generate realistic malware variants at a large scale. These variants could be used to improve the robustness of DL-based malware detectors against unseen adversarial malware variants. In addition to enhancing neural language models for AMG purposes, our AC-CAR model contributes to the AC deep RL literature as the first AC deep RL method designed to operate in large context-aware action spaces. To highlight our contribution to RL, we further position the novelty of our proposed AC-CAR among recent AC deep RL methods in Table II. In addition to the AC family, in Section VI, we evaluate AC-CAR's performance against the leading non-AC deep RL methods.

As highlighted in Table II, AC-CAR is uniquely positioned among the most recent probabilistic variants of AC deep RL methods that only use a generalization over state representations, including Soft Actor-Critic (SAC) [44] and Variational Actor-Critic (VAC) [42]. AC-CAR is also distinguished from the very recent variant of AC, Actor-Critic with Representations for Actions (AC-RA) [27], which uses a generalized representation over actions without accounting for context awareness. Based on this overall positioning against recently proposed leading RL methods, we next describe our proposed AC-CAR model and its underlying components.

TABLE II
POSITIONING OF AC-CAR AMONG STATE-OF-THE-ART AC DEEP RL METHODS

AC Deep RL Model	State Representation Learning	Action Representation Learning	Action Context Awareness
SAC [44]	✓	✗	✗
VAC [42]	✓	✗	✗
AC-RA [27]	✓	✓	✗
AC-CAR (Ours)	✓	✓	✓

IV. PROPOSED MODEL

To address the need for generating realistic adversarial malware variants, we follow three essential steps. In accordance with [7], [17], [40], we first characterize a threat model for automated AMG. Second, based on the determined threat model, we propose our novel AC deep RL algorithm, AC-CAR. Third, we examine the functionality of the adversarial malware variants generated by AC-CAR to ensure that they maintain their intended malicious functionality. We describe each of these steps in the following subsections.

A. Threat Model Identification

Consistent with [17], we characterize a realistic threat model for conducting binary closed-box adversarial malware attacks against DL-based malware detectors by identifying three major adversary properties, including the adversary's goal, knowledge, and capability.

- `Adversary's Goal`: The adversary aims to automatically generate malware variants that are capable of evading a DL-based malware detector via binary manipulation. To this end, the adversary receives an initial set of known malware binaries and modifies them to generate evasive functionality-preserving malware variants. In addition to functionality preservation, restrictions determined in the adversary's capability must be met.
- `Adversary's Knowledge`: The architecture and parameters of the malware detector model are not known to the adversary. Additionally, the adversary does not have access to any insider knowledge, including the real-valued confidence score from the malware detector. The only information observable by the adversary is whether the generated malware variant evades the malware detector (i.e., binary closed-box setting). To maximize practicality, consistent with [31], the adversary does not need reverse engineering, behavioral analysis in a sandbox, or access to the malware source code.
- `Adversary's Capability`: The adversary can apply functionality preserving modifications on malware

binaries. To maintain stealthiness, the average number of queries to the malware detector in order to generate evasive malware variants cannot exceed 50 (the minimum number of queries identified in Table I). Furthermore, the modification size is limited to 500 bytes (20 times less than the proposed AMG methods in [18], [30], as shown in Table I). Following [18], [39], to ensure the functionality preservation of generated malware variants, we focus on additive modifications since they tend to not interfere with malware functionality [30]. These modifications add the generated content to unused segments of the original malware files that will be not executed by the operating system, while leaving the malicious functionality of the original malware intact.

We next propose AC-CAR to operationalize the identified threat model.

B. Actor-Critic With Contextualized Action Representations (AC-CAR)

Operationalizing this threat model in an RL setting requires characterizing the key RL elements, including state, action, policy, reward, and environment:

- `State`: The state space size is the number of all possible executables (any combination of bytes that forms a functional malware executable file) or its representation. Consistent with [57], the state s_t of the malware executable was constructed by a holistic representation encompassing a set of 2,350 features from the raw and parsed malware file. These features include header metadata, section metadata (section name, size, and characteristics), imported and exported libraries metadata, counts of human-readable strings (such as file paths, registry keys, and URLs), byte histogram, and 2D byte-entropy histogram [58]. Accordingly, the states are represented as a set of all possible features representing a malware executable [5].
- `Action`: Modifications of byte sequences in a malware executable resulted from injecting non-executable content into an original malware binary file. Consistent with past research [18], [39], we focus on additive rather than editing modifications.
- `Policy`: A conditional probability distribution $\pi(a_t|s_t)$ that determines the next action given the current state. The policy is denoted by a neural net that receives the the state s_t as the input and determines the next action a_t as the output.
- `Reward`: Under the closed-box attack model assumption, the confidence score of the malware detector is not accessible. As such, the reward magnitude is often a fixed positive value [5]. Thus, following [5], a fixed reward with the magnitude of 10 was awarded when the modified malware file could evade the malware detector and preserve its original malicious functionality.
- `Malware Environment`: The environment encompasses the modified malware variant and the malware detector. The malware environment specification is detailed in Appendix A.1, available online.

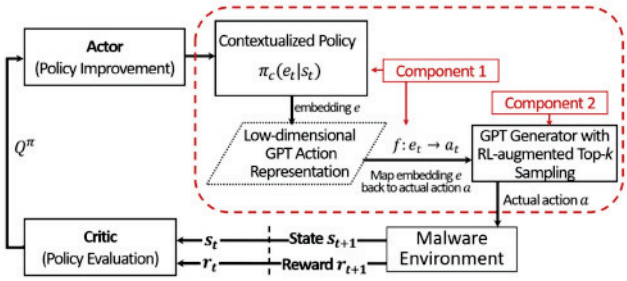


Fig. 4. The abstract view of AC-CAR Architecture. Component 1 includes the actor network operating in a low-dimensional action representation obtained from GPT's embedding space. This embedding is subsequently mapped to the environment's actions using function f. Component 2 leverages these actions as initial context to generate evasive contents using a modified top-k sampling procedure.

Based on the identified threat model, mimicking the patterns of benign executables to inject into the original malware file at each time step, as illustrated in the SDM process in Fig. 2, could be a viable AMG approach. To this end, the injected content must be benign-looking, evasive, and context-sensitive. Additionally, the proposed AMG method needs to operate in a large action space (i.e., all possible combinations of candidate bytes for injection). Building on [27] and the core AC deep RL architecture (shown in Fig. 3), we propose AC-CAR to learn action representations from the contextualized embeddings generated by GPT and subsequently generate benign-looking and evasive malware content accordingly. Fig. 4 shows the abstract view of AC-CAR.

One key aspect of our design is that in dealing with very large action spaces (e.g., 10,000 actions in AMG), we decompose the policy into two components: a contextualized policy $\pi_c(e_t|s_t)$ that operates on a low-dimensional action embedding space and an embedding mapping function f, which maps the representations back to the actual space to allow the interactions with the environment. Accordingly, AC-CAR has two main components. The first component encompasses the contextualized policy along with the mapping function f. The second component is RL-augmented Top-k Sampling, which operates based on the input of function f and aims to encourage the evasiveness of the generated bytes. Each component is described next.

1) Component 1: Contextualized Policy $\pi_c(E_t|s_t)$ and Embedding Mapping Function f.: In line with the gaps identified in our review, Component 1 aims to 1) add contextualization to core AC by incorporating a contextualized neural language model and 2) alleviate the described context initialization issue in GPT for generating evasive content while operating in a large action space. As seen in Fig. 4, the actor network in AC-CAR operates on a low-dimensional action space (embedding) obtained from training a contextualized GPT language model on benign executables (dashed diamond box). Unlike basic AC, in which the actor network outputs an action per given state (i.e., $\pi(a|s)$), the actor network in AC-CAR yields a contextualized policy $\pi_c(e|s)$ that accepts state s and outputs embedding e. However, the output embeddings need to be mapped back to actual actions to enable interaction with the malware environment. To map

embeddings into actions, Chandak et al. [27] establish the following relationship between the action-embedding conditional probability $P(a_t|e_t = e)$ and embedding conditional probability $P(e_t = e|s_t, s_{t+1})$ using the standard Markov property assumption and the total law of probability over the embedding space ϵ as given in (2),

$$P\left(a_t|s_t, s_{t+1}\right) = \int_{\epsilon} P\left(a_t|e_t = e\right) P\left(e_t = e|s_t, s_{t+1}\right)\, de, \tag{2}$$

in which $P(a_t|s_t, s_{t+1})$ decomposes under the Markovian assumption that given an embedding e_t, a_t is independent of the current state s_t and the next state s_{t+1}. It is shown that by using supervised learning, the action-embedding conditional $P(a_t|e_t = e)$ can be approximated by a deterministic function $f(a_t|e_t = e)$ that receives embeddings and maps them to actual actions responsible for the current state transition. For brevity, we denote this deterministic estimator by $f(e_t)$. While we focus on the implication of this approximation approach to devise a novel AC deep RL method for the AMG application domain, further details of this approximation and corresponding proofs can be found in [27]. The approximated $f(e_t)$ yields the candidate action (i.e., token) that is likely to contribute to the evasiveness of the generated content. Having the approximated $f(e_t)$, the GPT's generated content probability from (1) can be expressed in (3),

$$P\left(w_t^{1:N}|f\left(e_t\right)\right) = \prod_{n=1}^{N} P\left(w_t^n|w_t^{1:n-1}, f\left(e_t\right)\right), \tag{3}$$

in which the deterministic function $f(e_t)$ replaces the initial context W^0 in (1) for causal language models. (3) is crucial in addressing the first problem of applying GPT for AMG applications (i.e., dependency on the choice of the initial context). (3) offers a solution to incorporate evasive actions suggested by the AC model into GPT as the initial context for content generation.

2) Component 2: GPT Generator With RL-Augmented Top-K Sampling: AC-CAR's Component 2 aims to increase the evasiveness of GPT by incorporating the candidate actions obtained from the AC deep RL agent into GPT. To this end, we propose modifying the *decoding* mechanism of the GPT causal language model to incorporate these candidate actions. The decoding mechanism in causal language models is responsible for selecting the generated output from a set of high-probability candidate tokens [59]. Two effective decoding mechanisms in the causal language model literature are greedy sampling and top-k sampling [59]. Greedy sampling selects the next token w^n, which has the highest probability from conditional probability distribution $w^n \sim P(w|w^{1:n-1})$. On the contrary, in top-k sampling the next word is selected from the k most probable tokens forming a candidate set V_k. The probability mass is redistributed among the candidates in V_k such that $\sum_{w \in V_k} P(w|w^{1:n-1})$ encompasses a large portion of the probability mass (e.g., 0.9). Top-k sampling has yielded high-quality sequences in complex natural language processing tasks such as story generation [59]. To generate evasive content, we present an effective method to incorporate the AC output action (obtained from mapping function f) into GPT's top-k sampling.

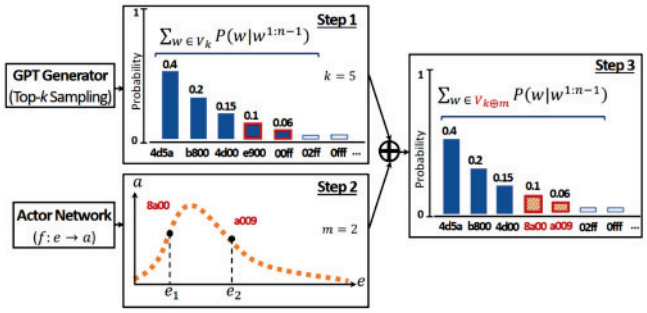


Fig. 5. The 3-step Procedure in RL-augmented Top-k Sampling. Basic Top-k sampling with $k = 5$ yields the most probable 5 tokens as GPT's output in Step 1. Step 2 samples $m = 2$ tokens from the action distribution learned by the actor network from component 1. In Step 3, these m samples replace the lowest probability tokens from GPT's output to augment the result of Top-k Sampling in Step 1. The tokens from the actor network are denoted with diagonal stripes in Step 3.

Fig. 5 shows the 3-step procedure of our proposed RL-augmented Top-k Sampling in AC-CAR in each time step to achieve this goal:

- `Step 1`: To obtain the k most probable candidate tokens, conduct basic top-k sampling as proposed in [59] (e.g., with $k = 5$) in the GPT generator. These tokens denote a set of benign-looking additions, which are considered to modify the original malware samples with.
- `Step 2`: To enhance evasiveness, we need to augment the candidate tokens from Step 1 with the tokens obtained from the actor network. To this end, we draw m embeddings (e.g., e_1 and e_2 for $m = 2$) from the actor network distribution ($m < k$) and compute the corresponding actions using function f obtained from AC-CAR's Component 1.
- `Step 3`: Finally, we augment the actions from top-k sampling with m actions obtained from Step 2 such that the probability mass of the entire set stays the same. The $\oplus$ operator ensures the probability mass $\sum_{w \in V_{k \oplus m}} P(w|w^{1:n-1})$ remains the same during this augmentation. As shown in Step 3 in Fig. 5, one simple but effective way to achieve this is by replacing m actions with the lowest probabilities from the distribution in Step 1 (i.e., '$e900'$ and '$00f'$) with the candidate actions from actor network in Step 2 (i.e., '$8a00'$ and '$a900'$), denoted in diagonal stripes in Step 3 of Fig. 5.

As a result, the proposed GPT generator with RL-augmented Top-k Sampling incorporates effective actions from AC deep RL into GPT generation mechanism to enhance GPT with evasiveness. As such, the second challenge of using GPT in the AMG application domain (i.e., lack of a mechanism for generating evasive content) is addressed. During this process, the enhanced GPT generator learns to produce benign-looking content to inject into a malware executable such that an evasive malware variant is constructed. The AC-CAR learning procedure, given in Algorithm 1, outlines how the two proposed components of AC-CAR operate to achieve both benign-looking and evasive content.

As shown in Algorithm 1, action representations are initialized with embeddings obtained from GPT. At each time step,

Algorithm 1: AC-CAR Learning Procedure.

Input: A set of benign executables, a set of initial malware executables, a malware environment, generated content length at each time step N, maximum number of episodes n_h, and the length of the episode n_t
Output: Learned contextualized policy network π_c, Action embedding mapping function f
Initialize action representations with the embeddings from training a causal language model (e.g., GPT) on benign executables.
Initialize f by interacting with the malware environment and using **Equation 2**.
for *episode* $h = 0, 1, ..., n_h$ **do**
  Initialize the malware environment with a random sample from initial malware executables.
  for *time step* $t = 0, 1, ..., n_t$ **do**
    Sample action embedding, $e_t = e$ from the initial contextualized policy π_c
    Apply the mapping function f on the embedding e_t to obtain the actual action $a_t = f(e_t)$
    Use action a_t as the initial context in $P(w^{1:N}|f(a_t|e_t = e))$ from **Equation 3**.
    for $i = 0, 1, ..., N$ **do**
      Generate content from GPT using RL-augmented Top-k Sampling with the initial context (Section 4.2.2).
    end
    Receive the next state S_{t+1} and immediate reward R_t.
    Update gradients of policy network π_c and critic network.
    Update gradients of f based on the supervised loss in [27].
  end
end
return π_c and f.

an action representation e_t is sampled from the contextualized policy π_c, and then is mapped to the actual action space using the mapping function f . The obtained action is used as the initial context in (3). RL-augmented Top-k Sampling is used for subsequent content generation based on the initial context. AC-CAR's methodological novelty is two-fold. Its first novelty lies in enhancing AC-based deep RL by adding context-awareness in high-dimensional action environments through learning contextualized representations over actions. The second novelty lies in extending causal language models by adding evasion awareness to the generated actions in GPT. Since content generation with AC-CAR amounts to an effective attack that is agnostic to the architecture of the targeted malware detector, AC-CAR can serve as a valuable tool for effective binary closed-box AMG. Following [27], AC-CAR's policy network was implemented by a simple neural network with one hidden layer. The policy network was AC-CAR is able to operate on an action space with a size of 10,000 actions (i.e., the size of the vocabulary in GPT embeddings). This is almost 1,000 times larger than current action sizes found in the AMG literature [5], [14], [24]. For reproducibility, AC-CAR's model specification, including the architecture and (hyper) parameter settings, are given in Appendix A.2, available online.

C. Malware Functionality Preservation

Following Castro et al. [12], we adopted VirusTotal's API to assess the functionality of the malware variants generated by AC-CAR. The API provides a holistic malware fingerprint, including a malware behavior report obtained from static and dynamic analysis of the malware variant. The report includes information such as network behavior (e.g., DNS requests, outgoing URLs) and file access patterns. Using this API, we compared the behavior reports for evasive variants (after adversarial modification) and original malware samples (before adversarial modification). Consistent with prior work in adversarial malware example generation [5], [14], [32], the payloads generated by AC-CAR where injected to the empty sections of the executable (e.g., file overlay and empty spaces between sections) that does not affect the logic of execution. This mechanism ensures that the addresses in the header sections (DOS header, PE header, and Section Table) and the rest of the executable remain unaltered, which is necessary to preserve functionality. Nevertheless, we verified that the key parts of the VirusTotal's report stayed unchanged after modification, indicating that the generated malware samples are fully functional. All included malware samples generated by AC-CAR in our dataset were checked and confirmed to be functional after modification.

V. Evaluation Overview

To systematically evaluate the performance of the proposed model against alternative solutions as well as the state-of-the-art AMG methods, we performed a series of benchmark evaluations on a dataset of real malware samples based on well-established evaluation metrics. Additionally, we conducted a case study to demonstrate the practical value and implications for cybersecurity researchers. We present an overview of our testbed construction, benchmark experiments, and ablation analysis before presenting the experiment design and results.

A. Testbed Construction

Consistent with [6], [32], [60], to construct a malware testbed, we created a repository with 31,188 (14.69 GB) recent malware and benign executable files collected from VirusTotal and Microsoft Windows 10. Our testbed includes 5,403 (3.18 GB) malware executables used for training AC-CAR and conducting benchmark evaluations. We also formed an evaluation set of 2,730 (1.06 GB) unseen malware executables as seeds to generate adversarial malware variants for measuring the improvement of the robustness of malware detectors in our case study. Dividing the training and evaluation sets based on the recency of the malware files is a common practice in malware analysis literature [60], [61], [62], which is often preferred over cross-validation that ignores the chronological order of malware files. The training set encompasses two subsets of executable files with a total volume of 8.22 GB: a malware training set and a benign training set. The malware training set serves as the initial seed to generate evasive malware variants. Table III shows the breakdown of the malware executables in the malware training set by malware type (adware, botnet, ransomware, rootkit, spyware, and virus).

TABLE III
BREAKDOWN OF THE MALWARE EXECUTABLES IN THE MALWARE TRAINING SET

Malware Type	Description	Examples	Malware Training Set (Number of files, size)
Adware	Shows unwanted ads and forces internet traffic to websites	eldorado, razy, gator	1,947 (1.5 GB)
Botnet	Establishes a network of bots connected through the internet and IoT devices	virut, salicode, sality	526 (0.15 GB)
Ransomware	Encrypts data and files, restricting access and usage until decrypted by malware authors	vtflooder, msil, bitman	900 (0.45 GB)
Rootkit	Grants admin privilege to remotely control the victim system	onjar, dqqd, shipup	731 (0.51 GB)
Spyware	Allows malware authors to steal personal information covertly	mikey, qqpass, scar	640 (0.38 GB)
Virus	Corrupts files on the host system	nimda, shodi, hematite	659 (0.19 GB)
Total	-	-	**5,403 (3.18 GB)**

To obtain the malware training set, we compiled a real collection of malware samples by obtaining an academic license from VirusTotal. This set includes 5,403 recent malware executables collected from 2017 to 2019in six categories, including adware, botnet, ransomware, rootkit, spyware, and virus. Consistent with the data collection approach in [6], [9], to obtain the benign executable dataset, we collected 13,554 benign executables from a clean installation of Microsoft Windows 10 on a regular office desktop computer. The benign training set has a total volume of 5.04 GB from which random subsets are utilized for training the causal neural language model (see Section VI-A and Appendix A.2 for details, available online). In accordance with extant malware analytics studies, we converted the executables' content to hexadecimal byte format [5], [6], [9]. In this format, each byte is represented by its hexadecimal equivalent ranging from 0 to 15. Every four hexadecimal values are grouped together to form a word (e.g., '4d5a'). Words are separated with a delimiter (e.g., blank space). To avoid excessively long sequences leading to inefficient training, we limited the maximum length of a sentence to a 2-KB sequence using the empty zero spaces in the file as a natural delimiter.

B. Experiment Setup

We identified two renowned DL-based malware detectors as our attack target: MalConv [9] and GBDT [7]. These detectors were selected by consulting with malware analysts in National Cyber-Forensics and Training Alliance (NCFTA), being utilized in renowned malware evasion competitions, accessibility (i.e., open-source versus proprietary), and the number of citations they received in the cybersecurity malware analytics community. MalConv and GBDT were featured as benchmark malware detector architectures in a malware evasion competition hosted by Endgame and Microsoft [63]. These highly cited malware detector models are made available to the cybersecurity research community through public GitHub repositories (https://github.com/endgameinc/malware_evasion_competition) and their characteristics are as follows.

- MalConv is an effective DL-based malware detectors for identifying unseen malware variants, developed through a collaboration between the Laboratory for Physical Sciences (LPS) and NVIDIA. The model incorporates a deep convolutional neural network architecture trained on approximately half a million malware binaries and achieves an area under the Receiver Operating Characteristic (ROC) curve of 98.5% on an unseen test set.
- GBDT is a machine learning-based Gradient Boosted Decision Tree model implemented with a tree induction algorithm called Light Gradient Boosting Machine (GBM). GBDT has been trained on Ember dataset [57], and was used by Endgame to obtain efficient and accurate malware detection [13].

To evaluate AC-CAR's performance against the state-of-the-art benchmark AMG methods, two well-established performance metrics were used: *evasion rate* and *cumulative reward*. Evasion rate is a widely used performance metric in the AMG literature [5], [7]. The evasion rate of an AMG method against a given malware detector is defined as $Evasion\,Rate = \frac{|E|}{N}$, where E represents the set of evasive malware variants obtained from the AMG method and N denotes the total number of original malware samples provided as input to the AMG method. This metric determines the effectiveness of a given method in evading a malware detector. The cumulative reward achieved by the RL agent is another well-established performance metric widely used for RL-based methods [27], [42].

We designed four sets of experiments to systematically evaluate the proposed AC-CAR.

Experiment 1 is designed to compare AC-CAR's performance with the state-of-the-art benchmark methods across six different malware types based on evasion rate and cumulative reward. This experiment also assesses AC-CAR's generalizability to variable payload sizes and shows the effect of contextualization in isolation. Experiment 2 evaluates the sensitivity of AC-CAR's performance to the quality of learning the embedding mapping function f in Algorithm 1. In Experiment 3 we conduct an ablation analysis on AC-CARs internal components (i.e., the

TABLE IV
BENCHMARK AMG METHODS FOR PERFORMANCE EVALUATION

Benchmark Category	Model	Description	Reference (venue)
Non-RL AMG	Benign Feature Append (BFA)	Brute-force injection of part of a benign file into malware sample	[18] (IEEE S&P)
	BFA + GPT	Injection of content generated by language models into malware sample	[30] (AAAI RSEML)
	GAMMA	A genetic algorithm (GA)-based black-box AMG	[13] (IEEE TIFS)
	EvadeHC	A meta-heuristic approach based on hill climbing that operates in black-box settings	[64] (ACM SIGSAC)
	Surrogate RNN	A generative RNN model trained against a surrogate black-box malware detector	[38] (AAAI)
	MAB-Malware	Uses multi-arm bandit to generate adversarial malware samples	[32] (ASIA-CCS)
AC RL-based AMG	Policy Gradient	Classic Policy Gradient RL	[26] (NeurIPS)
	Soft Actor-Critic (SAC)	AC with entropy regularization	[44] (ICLR)
	Variational AC (VAC)	Adapts AC to high-dimensional states with variational inference	[42] (NeurIPS)
	AC with Representations for Actions (AC-RA)	Maps actions into a low-dimensional space by a 1-d neural net	[27] (ICML)

contextualized policy and RL-augmented Top-k Sampling). Finally, Experiment 4 provides the runtime analysis of AC-CAR in Appendix B, available online.

For benchmark evaluations, we identified ten leading AMG methods, including six renowned non-RL and four AC RL-based binary closed-box benchmark AMG methods. Table IV summarizes the benchmark methods, their brief description, and corresponding references for each method. Non-RL AMG benchmark methods include Benign Feature Append (BFA), BFA+GPT, GAMMA, EvadeHC, Surrogate RNN, and MAB-Malware. BFA serves as a common baseline method that operates by injecting sections from benign files to the end of a malware sample until evasion occurs [12], [30]. BFA+GPT is a variant of BFA inspired by the recent AMG method proposed in [30] that achieves more effective malware variants by extending BFA with benign-looking content generated by a language model. Genetic Adversarial Machine learning Malware Attack (GAMMA) is a recent efficient genetic algorithm (GA)-based approach to generate closed-box attacks [13]. EvadeHC is a successful metaheuristic AMG approach that features a powerful hill climbing method for closed-box AMG [64]. Surrogate RNN offers a generative RNN that generates benign-looking malware samples by approximating the closed-box malware detector model [38]. MAB-Malware is an effective binary manipulation method based on multi-arm bandit [32].

Within AC RL-based AMG benchmark methods, Policy Gradient is a widely adopted model serving as the foundation of AC RL, which leverages the classic policy gradient theorem [26], [65]. Soft Actor-Critic (SAC) [44] is a renowned probabilistic variant of AC that has yielded breakthrough performance in many RL tasks by enhancing the core AC with an entropy regularization framework. As a successor of SAC, Variational AC (VAC) [42] features a probabilistic AC framework that uses variational inference to extend the applicability of AC to state spaces with higher dimensions. Finally, AC with Representations for Actions (AC-RA) [27] is the closest AC deep RL method to our proposed AC-CAR, which offers a mechanism to map high-dimensional action spaces into low-dimensional ones without contextualization of the action space. Following [14], to gain insight into each specific malware type, we conducted separate benchmark experiments on each malware type. Utilizing the functionality assessment procedure mentioned in Section IV-C, we checked the functionality of all generated malware variants to ensure they retain their functionality after modification.

VI. RESULTS

A. Benchmark Evaluations

We compared AC-CAR performance against the identified state-of-the-art AMG methods across the six determined malware types against MalConv and GBDT as the attack target. The evasion rate of each benchmark method per specific malware type is shown in Table V. A higher evasion rate indicates a better AMG method. The highest performance for each malware type is shown in bold face. To ensure independence from the benign samples used for the causal language model training, AC-CAR experiments were repeated three times, each with a GPT trained on different non-overlapping sets of samples selected based on a specific random seed (shown as Seeds 1-3in Table V). shown To encourage emulating minimal and inconspicuous malware variants, the maximum number of modifications was fixed to 0.5KB (20 times less than white-box attack in [16]). Additionally, the average number of queries to the malware detector model was limited to 50 queries [11], [30]. To ensure that the performance difference between AC-CAR and competing benchmark methods is statistically significant, we ran each experiment five times and conducted a paired t-test between AC-CAR and benchmark methods performances. P-values obtained from the t-test are denoted by asterisks in Table V, and are significant at 0.05:* and 0.01:**. For GBDT, benchmark methods with 0% evasion rates across all malware types are not shown in the table. The last

TABLE V
COMPARING THE EVASION RATE OF THE PROPOSED AC-CAR AGAINST IDENTIFIED AMG BENCHMARKS ACROSS SIX MALWARE TYPES

	MalConv						
AMG Method	Adware (%)	Botnet (%)	Ransomware (%)	Rootkit (%)	Spyware (%)	Virus (%)	Avg. (%)
BFA	0.92**	1.90**	1.20**	0.96**	3.28**	2.28**	1.76
BFA+GPT	0.98**	2.85**	1.40**	1.09**	2.03**	3.18**	1.92
EvadeHC	3.62**	5.98**	1.85**	8.10**	2.21**	9.71**	5.25
Surrogate RNN	4.69**	4.56**	7.48*	4.50**	10.50**	16.25**	7.99
GAMMA	12.63*	13.27**	14.11*	7.39**	9.55**	18.97**	12.65
Policy Gradient	2.67**	5.51**	2.33**	7.11**	2.81**	3.18**	3.94
SAC	2.13**	6.80**	5.27**	3.24**	4.16**	7.21**	4.80
VAC	2.82**	10.27**	9.15*	4.24**	10.31**	13.66**	8.41
AC-RA	9.71*	23.90**	11.11*	9.70**	30.62**	42.51*	21.26
MAB-Malware	12.58*	34.59**	13.33*	30.50**	33.33**	35.18*	26.59
AC-CAR (Seed 1)	14.02	48.29	18.22	54.71	70.31	60.84	44.39
AC-CAR (Seed 2)	**14.62**	**50.99**	**20.32**	**56.83**	71.00	**61.70**	**45.91**
AC-CAR (Seed 3)	13.20	45.93	18.11	51.71	**71.69**	61.30	43.76
	GBDT						
AMG Method	Adware (%)	Botnet (%)	Ransomware (%)	Rootkit (%)	Spyware (%)	Virus (%)	Avg. (%)
EvadeHC	3.48***	2.90***	7.79***	2.14***	4.00***	9.65***	4.99
Surrogate RNN	3.21***	4.00***	4.95***	5.99***	4.90***	8.25***	5.22
Policy Gradient	2.70***	1.11***	3.71***	5.10***	2.80***	3.37***	3.13
SAC	3.67**	5.10**	5.13**	4.50**	4.40**	3.10**	4.32
VAC	4.05***	8.70*	8.16**	6.00***	4.54***	10.72**	7.03
AC-RA	8.48*	10.69*	11.20	8.20**	9.71**	22.50*	11.80
MAB-Malware	9.07*	10.41*	7.68	10.43**	18.32**	16.65*	12.09
AC-CAR (Seed 1)	**12.80**	25.76	13.16	**29.80**	**27.61**	34.18	**23.89**
AC-CAR (Seed 2)	12.10	**26.35**	**15.28**	27.57	26.23	**35.08**	23.76
AC-CAR (Seed 3)	**13.17**	23.76	12.10	28.70	26.24	33.22	23.86

Note: Benchmark methods that are not shown in the table yielded 0% evasion rate across all malware types.
Asterisks denote statistical significance with P-values significant at 0.05:*, 0.01:**.

column shows the average evasion rate across all six malware types.

As shown in Table V, the proposed AC-CAR attained the highest evasion rate across all six malware types and with an average evasion rate of 45.91% for MalConv and 23.89% for GBDT. Learning contextualized action representations by the proposed AC-CAR yields an approximately 2-fold performance increase over the best-performing method, MAB-Malware [32] as well as the highest-performing RL counterpart, AC-RA, which does not support contextualized action representations (26.59% for Mab-Malware and 21.26% for AC-RA versus 45.91% for AC-CAR against MalConv). Same performance improvement is observed in the GBDT detector, in which AC-CAR outperforms both MAB-Malware and AC-RA (12.09% for MAB-Malware and 11.80% for AC-CAR versus 23.89% for AC-CAR). It is also observed that learning action representations with AC-RA yields an approximately 2-fold performance increase on average over VAC, which does not use action representations (21.26% for AC-RA versus 8.41% for VAC on MalConv, and 11.80% versus 7.03% on GBDT in Table V). This indicates the positive effect of learning a low-dimensional action representation. Also, as shown in Table V, merely using a causal language model (GPT) to generate benign-looking content does not yield high evasion rates (only 1.92% average evasion rate for BFA+GPT on MalConv and 0% on GBDT). Overall, AC-CAR outperforms all benchmark methods on average across all malware categories for both MalConv and GBDT. The considerable performance gain in our AC-CAR demonstrates that AC-CAR is effective in generating realistic adversarial malware variants. Also, the results show that all different malware samples selected for training AC-CAR's causal language model (shown with Seed 1, 2, and 3in Table IV) lead to significantly outperforming competing benchmark methods (including MAB-Malware). This suggests that AC-CAR is not drastically sensitive to the selection of malware samples for contextualization.

It is also observed from Table V that spyware and virus have the highest evasion rates of 71.69% and 61.70%, respectively. The high evasion rate for these malware types suggests that malware detector models could be more susceptible to AMG modifications of spyware and virus. This could be attributed to the characteristics of both spyware and virus which often contain small-sized malicious executables that are embedded in benign programs to camouflage as benign executables. As such, their content may ostensibly look like non-malicious content, increasing the chance of evading DL-based malware detectors. Conversely, the evasion rate on adware and ransomware (14.62% and 20.32%, respectively) are lower than other malware types. Specifically, ransomware executables have sections dedicated to data encryption routines, which could be uniquely distinguished with DL-based classifiers. These distinctive characteristics can render adversarial modifications less evasive.

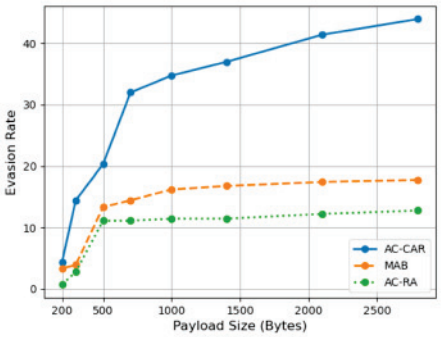


Fig. 6. Comparison of the Evasion Rate for Varied Payload Sizes.

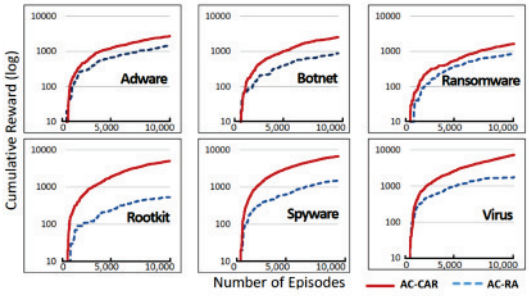


Fig. 7. Comparison of Cumulative Rewards of AC-RA and the Proposed AC-CAR Across all Six Malware Types.

1) Generalizability to Variable Payload Sizes: To further assess the generalizability of AC-CAR with respect to the payload size, we compare its performance with both the second and third best-performing counterparts from Table V, MAB-Malware and AC-RA. Fig. 6 shows the evasion rates when the payload size increases from 200 bytes to 2,800 bytes (equivalent to approximately 0.2% to 5% with respect to the average size of original malware executables) for ransomware.

As shown in Fig. 6, while the evasion rates for all methods increases as the payloadsize increases, AC-CAR clearly outperforms both MAB-Malware and AC-RA across all payload sizes, indicating that the effectiveness of AC-CAR in comparison with its counterparts is independent of the payload size. Further, it is observed that not only AC-CAR outperforms in small payload sizes, but also the performance gap between AC-CAR and MA-Malware grows larger as the payload size increases showing that AC-CAR gains more advantage with higher payload sizes compared to MAB-Malware and AC-CAR.

While this experiment is carried out to clearly show that AC-CAR performance is generalizable to variable payload sizes, we also note that in the AMG literature (e.g., [16], [32]), to maintain the stealthiness of the the adversarial variant, lower but effective payload sizes are often preferred.

2) Isolating the Effect of Contextualization on Rewards: To examine the effect of contextualized action representations, following the evaluations conducted in [27], [42], we compared the accumulated rewards of AC-CAR and its non-contextual counterpart, AC-RA (which does not learn contextualized action representations), during 10,000 episodes. Fig. 7 shows the results for each malware type. The vertical axis shows the cumulative reward in a logarithmic scale. The horizontal axis denotes the episode's number.

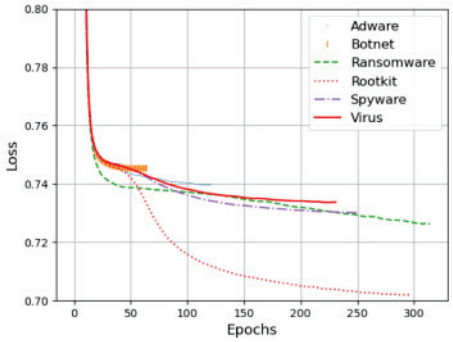


Fig. 8. Convergence of the Supervised Learning of the Mapping Function f in Algorithm 1.

Achieving a higher cumulative reward indicates more evasive malware variants generated. As shown in Fig. 7, the proposed AC-CAR accumulates a significantly higher reward than AC-RA for all malware types. Specifically, the proposed AC-CAR increases the accumulated rewards by almost an order of magnitude in Rootkit and Spyware. Overall, the AC-CAR's results confirm its effectiveness in generating realistic evasive adversarial malware variants against DL-based malware detectors without requiring prior knowledge of the attacked malware detector, with limited payload size, and with a limited number of queries to the detector.

B. Generalizability of the Embedding Mapping Function f

As noted in Section IV-B1, the embedding mapping function f maps the low-dimensional action representations back to the actual space to allow interactions with the environment (See (2)). Following [27], we characterize this function by a simple neural network and show that supervised learning can be used to approximate it. The network consists of an input layer with a size of low-dimensional actions and one optional hidden layer. Here we show how the losses of this supervised learning approach in Algorithm 1 converges and stabilizes for different malware categories using the same method in [27]. Consistent with their work, we empirically determine convergence if the changes in loss is less than an infinitesimally small $\epsilon = 1e{-}9$. Fig. 8 shows the results of the widely-used cross-entropy loss during learning in Algorithm 1 for each malware category.

As shown in Fig. 8, for all malware types the supervised learning loss for the initial mapping function f stabilizes in at most 300 epochs ranging from approximately 0.8 to 0.74 (for Botnet) and to 0.71 (for rootkit). Convergence in less than 300 epochs (5 seconds across each malware category on average) with a simple neural network suggests that this component of AC-CAR could be generalizable to different malware types.

C. Ablation Analysis of AC-CAR

To gauge the performance contribution of AC-CAR's components, including the contextualization of the action space and the proposed RL-augmented Top-k Sampling, AC-CAR's

TABLE VI
ABLATION ANALYSIS OF THE ISOLATED EFFECT OF OTHER SAMPLING ALTERNATIVES, TOP-k SAMPLING, AND OUR PROPOSED RL-AUGMENTED TOP-k SAMPLING IN AC-CAR

	Malware Type						
Method	Adware (%)	Botnet (%)	Ransomware (%)	Rootkit (%)	Spyware (%)	Virus (%)	Average (%)
GloVe Embeddings	1.23**	3.04**	2.22**	3.42**	2.50**	3.03**	2.57
Greedy Sampling	5.70**	8.37**	8.88**	5.48**	14.69**	22.91**	11.01
Top-k Sampling (k=2)	7.70**	15.97**	12.33**	7.25**	25.31**	32.32**	16.81
Top-k Sampling (k=4)	6.93**	17.49*	15.22**	7.93**	25.93**	41.42*	0.00
Top-k Sampling (k=5)	6.57**	17.68*	13.89**	7.25**	28.44**	36.87*	18.45
Top-k Sampling (k=10)	6.06**	10.65*	11.33**	6.02**	19.69**	35.50**	14.88
Top-k Sampling (k=16)	5.96**	9.32**	7.44**	6.29**	20.16**	37.33*	14.42
Top-k Sampling (k=40)	5.81**	18.06*	8.67**	10.94**	15.16**	38.84*	16.25
AC-CAR (Ours)	**14.62**	**50.99**	**20.32**	**56.83**	71.00	**61.70**	**45.91**

Note: Asterisks denote statistical significance with P-values significant at 0.05:*, 0.01:**.

performance was compared against alternative models. To measure the contribution of the contextualization of the action space, we replaced the GPT embeddings with a widely-used non-contextualized embedding known as Global Vectors (GloVe) [66]. To measure the contribution of the RL-augmented Top-k Sampling, we replaced the proposed sampling with two alternative encoding mechanisms: greedy sampling and top-k sampling (with $k \in \{2, 4, 5, 10, 16, 40\}$). We selected these values of k based on the best-performing values reported for GPT's top-k sampling in [59], [67], [68], [69]. We report the evasion rate of each of these alternative designs for each malware type against MalConv in Table VI. The statistical significance of the performance of AC-CAR was compared against other alternatives (P-values that are denoted by asterisks in Table VI are significant at 0.05:* and 0.01:**).

Comparing the performance of Glove with the proposed AC-CAR in Table VI suggests that the proposed contextualized representations are essential for generating evasive and benign-looking content. Additionally, it is observed that both Greedy and top-k sampling procedures lag behind the proposed RL-augmented Top-K Sampling method. As expected, incorporating RL in the GPT decoding process is crucial in generating evasive content. We next show the practical benefit of these results and the applicability of the adversarial malware variants generated by AC-CAR in improving the robustness of DL-based malware detectors in our case study.

VII. CASE STUDY: PRACTICAL UTILITY AND IMPLICATIONS

As a proof-of-value in improving the malware detector's robustness against unseen adversarial malware variants, we demonstrate that the malware variants generated by AC-CAR are useful in reducing the evasion rate of MalConv across all identified malware types through a case study. To this end, we conducted the most common defense mechanism against adversarial malware variants, known as adversarial re-training [70]. Adversarial re-training in the malware context begins with generating evasive malware variants by an AMG method such as AC-CAR. Subsequently, these generated adversarial malware variants are used to augment the original training set of a specific malware detector. Next, the malware detector model is re-trained on the augmented training set. In classic re-training methods, the re-trained model is tested against adversarial malware variants that could not be detected originally to verify the performance improvement in detecting previously undetected malware executables [37]. To help assess the generalizability of our robustification, we considered a more challenging setting in which we computed the robustness of the new malware detector against another effective attack, VAC, a variant of the attack proposed in [5].

Following [37], [70] and according to the described adversarial re-training process, we generated 455 malware variants by our proposed AC-CAR and re-trained MalConv in the presence of these adversarial variants. We then measured the malware detection performance of the re-trained MalConv on an evaluation set of 2,730 malware variants (455 malware samples for each of the six malware types) generated by the attack method proposed in [5]. Consistent with [5], the performance of the re-trained malware detector model against adversarial malware variants is compared to its original performance using the evasion rate. A lower evasion rate indicates a higher robustness against adversarially generated malware variants. To ensure that the adversarial re-training reduces the evasion rate (i.e., false negatives) without increasing false positives, we vary the number of generated malware files used in adversarial re-training while monitoring true negatives at each stage. Fig. 9 shows the results for the ransomware type.

As shown in Fig. 9, the original MalConv cannot detect 9.15% of the adversarial malware variants generated by VAC (i.e., 9.15% evasion rate). As the number of adversarial ransomware variants generated by AC-CAR increases in the training process, the evasion rate decreases. However, once the number of generated adversarial variants passes 347, the false positive rate starts increasing, indicating that excessive adversarial re-training damages the detector model's accuracy and utility. The corresponding points are shown with a cross sign in Fig. 9. Stopping the robustification is necessary to ensure that increasing the adversarial robustness does not undermine the practical utility of the malware detector by a high false alarm rate [7]. Thus, we stopped robustification after re-training with 75% of

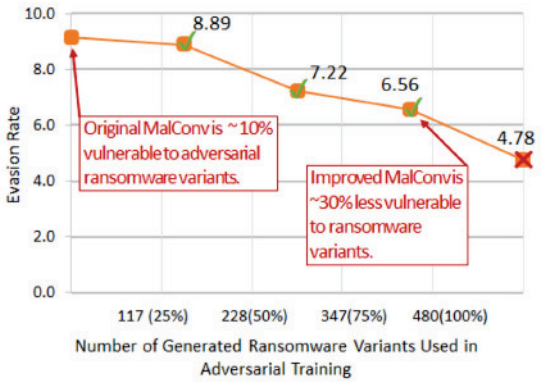


Fig. 9. Changes in the Evasion Rate During Adversarial Re-training with AC-CAR. A lower evasion rate indicates a higher robustness against adversarially generated malware variants.

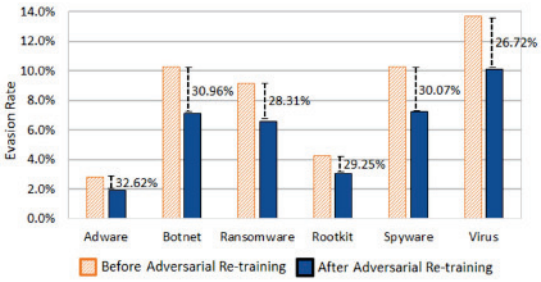


Fig. 10. Evasion Rate Before (Strip Pattern) and After (Solid Pattern) Adversarial Re-training with Adversarially Generated Malware Variants Obtained from AC-CAR.

adversarial variants (shown with a check mark in Fig. 9). As a result, ransomware variants generated by AC-CAR contribute to a 30% reduction in ransomware evasion rate (9.15% versus 6.56%) without damaging the model's accuracy. Fig. 10 depicts the results of conducting the described adversarial re-training process to reduce the evasion rate across all six malware types using the adversarial variants generated by AC-CAR.

As shown in Fig. 10, the evasion rate decreases after the adversarial re-training of the malware detector. As a result, the re-trained detector model is more resistant against the adversarial malware variants generated by the AMG method proposed in [5]. On average, MalConv's robustness was increased by 29.65% across all malware types. Specifically, ransomware was associated with a 28.31% reduction of evasion rate as shown in Fig. 10. Given that the global cost of ransomware damage was estimated to be $20 billion (USD) in 2021 [3], reducing the undetected adversarial ransomware attacks could be an effective step towards automated cyber defense, which potentially translates to preventing a sizable loss in the cybersecurity industry.

A. Discussion

While our work focused on binary manipulations on malware executable without reverse engineering and requiring access to the malware source code, we acknowledge that, when possible, it is also beneficial to consider other types of sophisticated actions that modify the control flow of the execution in a way that preserves functionality. Promising work in this direction is presented in [35], where simple basic blocks are added to the original control flow without affecting functionality, or the adversarial source code attack presented in [34]. While devising functionality-preserving actions that can drastically modify the control flow is still a challenging and open problem, we note that the AC-CAR's action set can be augmented to incorporate such actions in order to build adversaries that utilize the malware source code or function calls achieved from reverse engineering.

Finally, as a corollary to our case study results, we note that while common defense mechanisms such as adversarial re-training are crucial to improving adversarial robustness, they cannot fully eliminate the vulnerability to adversarial malware variants. Thus, significant future research is needed on novel generic defense architectures against these attacks. Promising studies have started to investigate this important direction through processing control flow graph and function calls [33], [34].

VIII. Conclusion and Future Directions

Automated adversarial malware optimization is crucial for constantly and rapidly improving the robustness of DL-based malware detectors. Extant approaches to this end rely on assumptions, including access to insider knowledge about the malware detector, unlimited size of adversarial modifications, and unlimited queries to the malware detector. In this study, we presented a novel closed-box binary manipulation method that enables generating adversarial malware variants to improve the robustness of DL-based malware detectors without relying on these assumptions.

Apart from contributing to the AMG literature, to our knowledge, the proposed AC-CAR algorithm features the first method that enables learning contextualized action representations in the deep RL literature. AC-CAR also advances causal language modeling by offering a way to incorporate evasiveness for adversarial applications.

Through rigorous evaluations, we showed that AC-CAR generates evasive and benign-looking byte sequences that lead to the state-of-the-art performance in the AMG literature. In our experiments and case study, we investigated the generalizability of AC-CAR to different malware detectors, payload sizes, and to new malware samples for both MalConv and GBDT. Furthermore, our case study demonstrated AC-CAR's practical value in enhancing robustness against adversarial malware variants.

Further research is needed to enhance the adversarial modifications to support editing actions that can rewrite parts of the malware executable while preserving its functionality. Applying these modifications is challenging since they tend to affect the *semantics* of the executable and yield corrupt (i.e., non-executable) malware variants. Recent advances in de-compilation tools and reverse engineering methods contribute to this stream of research for achieving more powerful AMG methods. Another promising future direction is designing explainable attack methods that may slightly sacrifice performance to gain more practical utility.

Acknowledgment

The authors thank VirusTotal for providing the academic license to access the malware dataset and APIs for malware

functionality assessment. We would also like to thank National Cyber-Forensics and Training Alliance (NCFTA) for their constructive feedback. We are grateful to Yash Chandak from the University of Massachusetts for the helpful discussion and sharing the implementation of their work.

REFERENCES

[1] Executive Office of the President, "The cost of malicious cyber activity to the U.S. economy," The Council of Economic Advisers, Tech. Rep., Feb. 2018. [Online]. Available: https://www.hsdl.org/?view&did=808776

[2] K. Bissell, R. M. LaSalle, and P. D. Cin, "Ninth annual cost of cybercrime study: Unlocking the value of improved cybersecurity protection," Accenture and Ponemon, Mar. 2019. [Online]. Available: https://iapp.org/media/pdf/resource_center/accenture_cost_of_cybercrime_study_2019.pdf

[3] S. Morgan, "Global ransomware damage costs," *Cybercrime Mag.*, 2019. [Online]. Available: https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-20-billion-usd-by-2021/

[4] W. Tounsi and H. Rais, "A survey on technical threat intelligence in the age of sophisticated cyber attacks," *Comput. Secur.*, vol. 72, pp. 212–233, 2018.

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

[6] B. Chen, Z. Ren, C. Yu, I. Hussain, and J. Liu, "Adversarial examples for CNN-Based malware detectors," *IEEE Access*, vol. 7, pp. 54360–54371, 2019.

[7] W. Fleshman, E. Raff, J. Sylvester, S. Forsyth, and M. McLean, "Non-negative networks against adversarial attacks," *Stat*, vol. 1050, p. 15, 2018.

[8] M. Krčál, O. Švec, M. Bálek, and O. Jašek, "Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Lables," in *Proc. Int. Conf. Learn. Representations*, Vancouver, Canada, 2018. [Online]. Available: https://openreview.net/forum?id=HkHrmM1P

[9] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole EXE," in *Proc. Workshops 32nd AAAI Conf. Artif. Intell.*, Jun. 2018.

[10] Bloomberg, "Symantec unveils industry's first neural network to protect critical infrastructure from cyber warfare," 2018. [Online]. Available: https://www.bloomberg.com/press-releases/2018-12-05/symantec-unveils-industry-s-first-neural-network-to-protect-critical-infrastructure-from-cyber-warfare

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

[12] R. Castro, C. Schmitt, and G. D. Rodosek, "ARMED: How automatic malware modifications can evade static detection?," in *Proc. Int. Conf. Inf. Manage.*, 2019, pp. 20–27.

[13] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Functionality-preserving black-box optimization of adversarial windows malware," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3469–3478, 2021.

[14] 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. 48867–48879, 2019.

[15] D. Han et al., "Practical traffic-space adversarial attacks on learning-based NIDSs," 2020, *arXiv:2005.07519*.

[16] B. Kolosnjaji et al., "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *Proc. 26th Eur. Signal Process. Conf.*, 2018, pp. 533–537.

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

[18] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *Proc. IEEE Secur. Privacy Workshops*, 2019, pp. 8–14.

[19] D. Park, H. Khan, and B. Yener, "Creating adversarial malware examples using code insertion," 2019, *arXiv: 1904.04802*. [Online]. Available: http://arxiv.org/abs/1904.04802

[20] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://iclr.cc/Conferences/2018/Schedule?showEvent=67

[21] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," *Commun. ACM*, vol. 61, no. 7, pp. 56–66, 2018.

[22] F. Ahmed, P. Vaishnavi, K. Eykholt, and A. Rahmati, "Ares: A system-oriented wargame framework for adversarial ML," in *Proc. IEEE Secur. Privacy Workshops*, 2022, pp. 73–79.

[23] G. Apruzzese, H. S. Anderson, S. Dambra, D. Freeman, F. Pierazzi, and K. Roundy, ""Real attackers don't compute gradients": Bridging the gap between adversarial ML research and practice," in *Proc. IEEE Conf. Secure Trustworthy Mach. Learn.*, 2023, pp. 339–364.

[24] M. Ebrahimi, J. Pacheco, W. Li, J. L. Hu, and H. Chen, "Binary black-box attacks against static malware detectors with reinforcement learning in discrete action spaces," in *Proc. IEEE Secur. Privacy Workshops*, 2021, pp. 85–91.

[25] S. Shuvaev, S. Starosta, D. Kvitsiani, A. Kepecs, and A. Koulakov, "R-learning in actor-critic model offers a biologically relevant mechanism for sequential decision-making," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 18872–18882.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[27] Y. Chandak, G. Theocharous, J. Kostas, S. Jordan, and P. Thomas, "Learning action representations for reinforcement learning," *Proc. Mach. Learn. Res.*, Long Beach, California, USA, PMLR, Jun. 2019, pp. 941–950.

[28] Y. Keneshloo, T. Shi, N. Ramakrishnan, and C. K. Reddy, "Deep reinforcement learning for sequence-to-sequence models," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 7, pp. 2469–2489, Jul. 2020.

[29] S. Qiu, Q. Liu, S. Zhou, and C. Wu, "Review of artificial intelligence adversarial attack and defense technologies," *Appl. Sci.*, vol. 9, no. 5, pp. 909–938, 2019.

[30] 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 *Proc. AAAI Workshop Robust, Secure, Efficient Mach. Learn.*, 2021.

[31] A. Abusnaina, Y. Wang, S. Arora, K. Wang, M. Christodorescu, and D. Mohaisen, "Burning the adversarial bridges: Robust windows malware detection against binary-level mutations," 2023, *arXiv:2310.03285*.

[32] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, and H. Yin, "MAB-malware: A reinforcement learning framework for blackbox generation of adversarial malware," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2022, pp. 990–1003.

[33] A. Abusnaina et al., "DL-FHMC: Deep learning-based fine-grained hierarchical learning approach for robust malware classification," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 5, pp. 3432–3447, Sep./Oct. 2022.

[34] O. Kargarnovin, A. M. Sadeghzadeh, and R. Jalili, "MAL2GCN: A robust malware detection approach using deep graph convolutional networks with non-negative weights," *J. Comput. Virol. Hacking Techn.*, vol. 20, no. 1, pp. 95–111, 2024.

[35] Y. Kucuk and G. Yan, "Deceiving portable executable malware classifiers into targeted misclassification with practical adversarial examples," in *Proc. 10th ACM Conf. Data Appl. Secur. Privacy*, 2020, pp. 341–352.

[36] T. D. Phan et al., "Leveraging reinforcement learning and generative adversarial networks to craft mutants of windows malware against black-box malware detectors," in *Proc. 11th Int. Symp. Inf. Commun. Technol.*, 2022, pp. 31–38.

[37] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Generic black-box end-to-end attack against state of the art API call-based malware classifiers," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*, Springer, 2019, pp. 490–510.

[38] W. Hu and Y. Tan, "Black-box attacks against RNN based malware detection algorithms," in *Proc. AAAI Conf. Artif. Intell.*, Palo Alto, CA,USA, 2018, pp. 245–255.

[39] Y. Li, Y. Wang, Y. Wang, L. Ke, and Y.-A. Tan, "A feature-vector generative adversarial network for evading PDF malware classifiers," *Inf. Sci.*, vol. 523, pp. 38–48, 2020.

[40] S. Dey, A. Kumar, M. Sawarkar, P. K. Singh, and S. Nandi, "EvadePDF: Towards evading machine learning based PDF malware classifiers," in *Proc. Int. Conf. Secur. Privacy*, Springer, 2019, pp. 140–150.

[41] M. L. Littman, "Algorithms for sequential decision making," PhD Thesis, Dept. Comput. Sci., Brown Univ., Providence, RI, 1996.

[42] M. Fellows, A. Mahajan, T. G. Rudner, and S. Whiteson, "VIREL: A variational inference framework for reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 7120–7134.

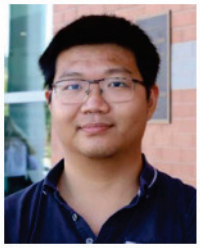
[43] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. 35th Int. Conf. Mach. Learn.*, Stockholmsmässan, Stockholm Sweden, PMLR, 2018, pp. 1587–1596.

[44] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1856–1865.

[45] Z. Wang et al., "Sample efficient actor-critic with experience replay," in *Proc. Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=HyM25Mqel
[46] S. Sharma, A. Suresh, R. Ramesh, and B. Ravindran, "Learning to factor policies and action-value functions: Factored action space representations for deep reinforcement learning," 2017, *arXiv:1705.07269*.
[47] G. Dulac-Arnold et al., "Deep reinforcement learning in large discrete action spaces," 2016, *arXiv:1512.07679*.
[48] J. Jing, E. C. Cropper, I. Hurwitz, and K. R. Weiss, "The construction of movement with behavior-specific and behavior-independent modules," *J. Neurosci.*, vol. 24, no. 28, pp. 6315–6325, 2004.
[49] J. Merel et al., "Neural probabilistic motor primitives for humanoid control," in *Proc. Int. Conf. Learn. Representations*, 2018.
[50] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," in *Proc. Neural Inf. Process. Syst.*, Montreal, Canada, 2018, pp. 3566–3577.
[51] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[52] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.
[53] Z. Yang et al., "XLNet: Generalized autoregressive pretraining for language understanding," in *Proc. Adv. Neural Inf. Process. Syst.*, Curran Associates, Inc., 2019, pp. 5753–5763. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf
[54] T. B. Brown et al., "Language models are few-shot learners," 2020, *arXiv:2005.14165*.
[55] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://openai.com/blog/language-unsupervised
[56] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. [Online]. Available: https://openai.com/blog/better-language-models
[57] H. S. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," 2018, *arXiv: 1804.04637*.
[58] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *10th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, IEEE, 2015, pp. 11–20.
[59] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical neural story generation," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, Melbourne, Australia, 2018, pp. 889–898. [Online]. Available: https://www.aclweb.org/anthology/P18-1082
[60] A. Kyadige, E. M. Rudd, and K. Berlin, "Learning from context: A multi-view deep learning architecture for malware detection," in *Proc. IEEE Secur. Privacy Workshop*, 2020, pp. 1–7. [Online]. Available: https://ai.sophos.com/presentations/learning-from-context-a-multi-view-deep-learning-architecture-for-malware-detection/
[61] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "{TESSERACT }: Eliminating experimental bias in malware classification across space and time," in *Proc. 28th {USENIX} Secur. Symp.*, 2019, pp. 729–746.
[62] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Are your training datasets yet relevant?," in *Proc. Eng. Secure Softw. Syst.*, Cham, Springer International Publishing, 2015, pp. 51–67.
[63] H. Anderson, "Malware evasion competition," 2019. [Online]. Available: https://www.elastic.co/blog/machine-learning-static-evasion-competition
[64] H. Dang, Y. Huang, and E.-C. Chang, "Evading classifiers by morphing in the dark," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 119–133.
[65] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, MIT Press, 1999, pp. 1057–1063. [Online]. Available: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf
[66] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. Empirical Methods Natural Lang. Process.*, 2015, pp. 1532–1543.
[67] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity," 2021, *arXiv:2104.08786*.
[68] T. Huang, J. Chu, and F. Wei, "Unsupervised prompt learning for vision-language models," 2022, *arXiv:2204.03649*.
[69] Z. Wang, A. W. Yu, O. Firat, and Y. Cao, "Towards zero-label language learning," 2021, *arXiv:2109.09193*.
[70] N. Papernot and P. D. McDaniel, "Extending defensive distillation," 2017, *arXiv: 1705.05264*. [Online]. Available: http://arxiv.org/abs/1705.05264

Reza Ebrahimi (Senior Member, IEEE) received the master's degree in computer science from Concordia University, Canada, in 2016, and the PhD degree from the University of Arizona. He is an assistant professor and the founder of Star-AI Lab with the School of Information Systems and Management at the University of South Florida. He was a research associate with the Artificial Intelligence (AI) Lab, in 2021. His dissertation on AI-enabled cybersecurity analytics won the ACM SIGMIS best doctoral dissertation award, in 2021. His research focuses on adversarial machine learning for AI-enabled secure and trustworthy cyberspace. Reza has published more than 30 articles in peer reviewed security venues including NeurIPS, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Transactions on Dependable and Secure Computing*, IEEE S&PW, *Applied Artificial Intelligence*, AAAIW, IEEE ICDMW, *Digital Forensics*, and *IEEE Intelligence and Security Informatics*. He has been serving as a program chair and program committee member in IEEE ICDM Workshop on Machine Learning for Cybersecurity (MLC) and IEEE S&P Workshop on Deep Learning and Security (DLS).

Jason Pacheco received the PhD degree in computer science from Brown University with Erik Sudderth. He is an assistant professor with the Department of Computer Science, University of Arizona in Tucson Arizona. He received the Air Force Office of Scientific Research (AFOSR) Young Investigator Program (YIP) award for Robust Maximum Entropy Planning, Learning, and Control in Uncertain Environments. He also held a postdoc position with MIT CSAIL with John Fisher III. His work has been published in top tier venues such as IEEE/CVF CVPR, NeurIPS, ICML, IEEE S&PW and AISTAT. His research interest lies in statistical machine learning, sequential decision making, and probabilistic graphical models to be used in a variety of domains including computer vision, cybersecurity, computational biology, and computational neuroscience.



James Hu (Member, IEEE) received the BS degree in management information systems from the University of Arizona. He is currently working toward the PhD degree with the Artificial Intelligence (AI) Lab, University of Arizona. His research interests are in the application of adversarial machine learning to cybersecurity domains including static malware analysis, adversarial malware generation, safety of foundation AI models, and attacking and jailbreaking of large language models (e.g., ChatGPT). His work on adversarial malware generation has been published in IEEE ISI and IEEE ICDM Workshops.

Hsinchun Chen (Fellow, IEEE) received the BS degree from the National Chiao-Tung University, Taiwan, the MBA degree from the State University of New York at Buffalo, and the PhD degree in information systems from New York University. He is a regents professor with the University of Arizona. He is also a fellow of ACM, AAAS, and AIS. He received the IEEE Computer Society Technical Achievement Award, in 2006, the IEEE Big Data Security Pioneer Award, and the Extraordinary Faculty Award, in 2022. He founded the Artificial Intelligence (AI) Lab at The University of Arizona, in 1989, which has received $60M+ research funding from NSF, NIH, NLM, DOD, DOJ, CIA, DHS, and other agencies (100+ grants, 50+ from NSF, as PI). He has served as editor-in-chief, senior editor or associate editor of IEEE/ACM journals (*IEEE Information Systems*, *IEEE Transactions on Systems, Man, and Cybernetics*, *ACM Transactions on Management Information Systems*, *ACM Transactions on Information Systems*). His COPLINK system, which has been quoted as a national model for public safety information sharing and analysis, has been adopted in more than 550 law enforcement and intelligence agencies in 20 states. He is the director of the UA AZSecure Cybersecurity Program, which has been supported by NSF SFS, SaTC, and CICI programs and has received CAE-CD/CAE-R cybersecurity designations from NSA/DHS.