electronics

MDPI

Review

A Survey on Reinforcement Learning-Driven Adversarial Sample Generation for PE Malware

Yu Tong [1,2], Hao Liang [1,2,*], Hailong Ma [1,2], Shuai Zhang [1,2] and Xiaohan Yang [1,2]

[1] Information Technology Research Institute, Information Engineering University, Zhengzhou 450002, China; tongyu17@mails.ucas.ac.cn (Y.T.); longmanclear@163.com (H.M.); 2012301200229@whu.edu.cn (S.Z.); yangxiaohan@stu.xjtu.edu.cn (X.Y.)
[2] Key Laboratory of Cyberspace Security, Ministry of Education of China, Zhengzhou 450002, China
* Correspondence: lhmailhappy@163.com

Abstract: Malware remains a central tool in cyberattacks, and systematic research into adversarial attack techniques targeting malware is crucial in advancing detection and defense systems that can evolve over time. Although numerous review articles already exist in this area, there is still a lack of comprehensive exploration into emerging artificial intelligence technologies such as reinforcement learning from the attacker's perspective. To address this gap, we propose a foundational reinforcement learning (RL)-based framework for adversarial malware generation and develop a systematic evaluation methodology to dissect the internal mechanisms of generative models across multiple key dimensions, including action space design, state space representation, and reward function construction. Drawing from a comprehensive review and synthesis of the existing literature, we identify several core findings. (1) The scale of the action space directly affects the model training efficiency. Meanwhile, factors such as the action diversity, operation determinism, execution order, and modification ratio indirectly influence the quality of the generated adversarial samples. (2) Comprehensive and sensitive state feature representations can compensate for the information loss caused by binary feedback from real-world detection engines, thereby enhancing both the effectiveness and stability of attacks. (3) A multi-dimensional reward signal effectively mitigates the policy fragility associated with single-metric rewards, improving the agent's adaptability in complex environments. (4) While the current RL frameworks applied to malware generation exhibit diverse architectures, they share a common core: the modeling of discrete action spaces and continuous state spaces. In addition, this work explores future research directions in the area of adversarial malware generation and outlines the open challenges and critical issues faced by defenders in responding to such threats. Our goal is to provide both a theoretical foundation and practical guidance for building more robust and adaptive security detection mechanisms.

Keywords: reinforcement learning; adversarial malware generation; evasion; malware detection; adversarial sample

check for updates

Academic Editor: Aryya Gangopadhyay

Received: 10 May 2025
Revised: 6 June 2025
Accepted: 10 June 2025
Published: 13 June 2025

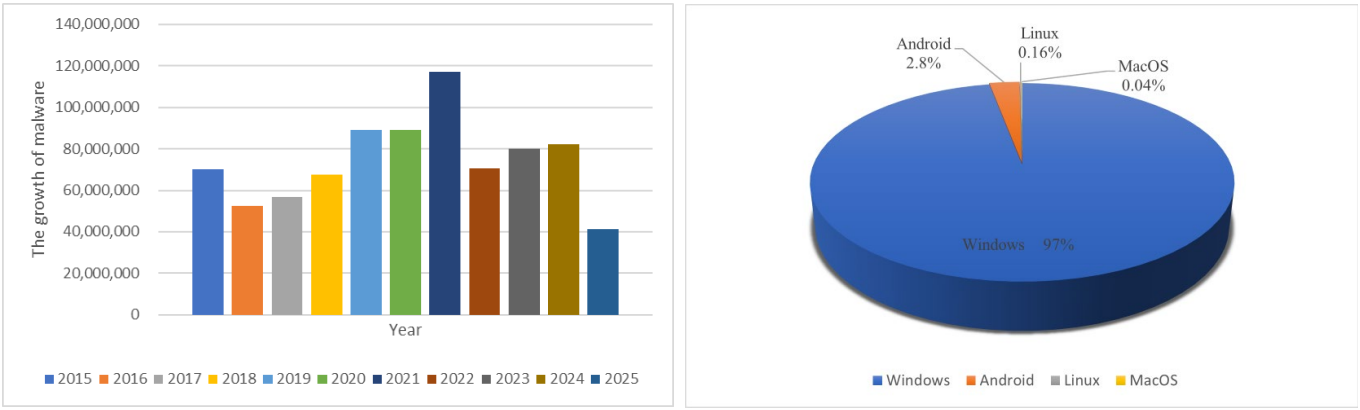
Citation: Tong, Y.; Liang, H.; Ma, H.; Zhang, S.; Yang, X. A Survey on Reinforcement Learning-Driven Adversarial Sample Generation for PE Malware. *Electronics* **2025**, *14*, 2422. https://doi.org/10.3390/electronics14122422

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

As a central instrument of cyberattacks, malware continues to evolve in terms of both its technical sophistication and dissemination efficiency, surpassing the limitations of traditional defensive mechanisms. The scope of its impact has expanded beyond conventional data theft to encompass non-traditional security domains, including the disruption of critical infrastructure, the contamination of supply chains, and its influence on geopolitical

strategies [1–3]. According to recent security reports, ransomware exploiting zero-day vulnerabilities has emerged as a predominant tool in cyberattacks, while the threat landscape continues to evolve with increasingly sophisticated Trojans and backdoor-type malware [4,5]. Figure 1 illustrates the growth trends of malware from January 2015 to March 2025, along with its distribution across various operating systems [6]. During the period of ten years, the annual growth of malware has fluctuated significantly, with lows reaching approximately 800 million new instances per year and peaks surpassing 1.4 billion. In comparison to other operating systems, including Android, Linux, and MacOS, Windows demonstrates the highest infection rate, with 97% of all identified malware cases targeting this platform.



(**a**) (**b**)

Figure 1. (**a**) The growth trends of malware from January 2015 to March 2025; (**b**) the malware distribution across various operating systems.

Static analysis, which does not require the execution of malware samples, is widely adopted by security vendors as a first line of defense against large-scale malware threats due to its efficiency and low resource consumption. However, driven by advances in artificial intelligence, traditional signature-based static detection methods [7] have proven insufficient in coping with the rapid proliferation and increasing sophistication of modern malware. To address these challenges, researchers have introduced machine learning and deep learning architectures to enhance both the efficiency and accuracy of malware detection [8–14]. Recent deep learning-based approaches for malware detection leverage deep neural network architectures to enable automatic feature extraction and classification. Depending on the input data format, these methods can be broadly categorized into three types: byte sequence-based detection methods [15–17], image-based detection techniques [18–20], and hybrid pattern-based approaches [21–23]. However, emerging studies indicate that such deep learning-based detectors remain vulnerable to adversarial samples [24]. Adversaries can manipulate the low-level code of malicious samples while preserving their intended functionality, thereby evading detection by causing misclassification as benign software [25–28].

Due to their strong resemblance to real-world attack scenarios, black-box evasion techniques have attracted extensive research attention across various domains, including genetic algorithms [25,28–30], generative adversarial networks (GANs) [31–33], and reinforcement learning (RL) [34–36], with the goal of improving the ability of attacks to bypass detection systems. Among these, RL has demonstrated particular promise by employing a discrete action selection mechanism to perform the end-to-end generation of adversarial malware directly in the problem space, rather than within the feature space, which significantly enhances the efficiency of adversarial sample generation [28]. Prior studies

have shown that reinforcement learning accounts for approximately 22% of all adversarial attack methodologies, while its adoption rate rises to nearly 50% in black-box settings [37]. Notably, in such black-box environments, RL-based adversarial attacks have achieved an evasion success rate of 50% against commercial antivirus engines.

A systematic investigation of adversarial attack techniques against malware—particularly reinforcement learning-driven approaches to generating adversarial samples for PE malware—offers valuable methodological insights for the development of robust detection frameworks and the continuous enhancement of the detection capabilities. In recent years, as adversarial malware techniques have rapidly evolved and their threats to cybersecurity have grown increasingly severe, this area has garnered significant attention from the academic community, leading to a surge in review articles and comprehensive surveys [38–42]. Overall, existing review studies exhibit significant limitations, with insufficient systematic exploration of emerging artificial intelligence techniques such as reinforcement learning from an attacker's perspective. For instance, Ling et al. [38] classify adversarial malware attacks into black-box and white-box settings and provide an analysis focused on problem space and feature space attacks; however, they offer limited systematic discussion of the application of modern AI-driven methods like reinforcement learning. Similarly, Bulazel et al. [40] present a comprehensive overview of malware generation techniques across PC, mobile, and web platforms, yet they primarily focus on evasion strategies targeting dynamic detection mechanisms, with minimal coverage of static analysis-based scenarios. Meanwhile, Maniriho et al. [42] systematically explored the malware detection technologies based on artificial intelligence methods for PC and mobile environments, but there is a lack of research on malware detection and evasion technologies from the perspective of attackers. In their 2024 survey, Geng et al. [37] presented the first systematic analysis of evasion techniques against malware detection from the attacker's perspective and briefly discussed reinforcement learning-based approaches to generating adversarial samples. However, their work primarily focused on methodological categorization and summarization, with limited in-depth exploration of the underlying mechanisms of the generative models, particularly in terms of reward function design and state–action space modeling. Notably, their study did not address the dynamic interplay between the iterative optimization of adversarial sample generation frameworks and the evolving defensive capabilities of detection systems.

Accordingly, this paper focuses on the domain of PE adversarial malware generation techniques and presents a systematic review of current research on reinforcement learning-based adversarial attacks from the attacker's perspective.

1. We summarize the advantages of RL in the context of adversarial malware generation and propose a foundational framework and an evaluation system for RL-driven malware generation techniques.
2. Furthermore, we provide a comprehensive comparative analysis of existing methodologies across multiple critical dimensions, including the design of action spaces, representation of state spaces, construction of reward functions, and selection of RL architectures.
3. Finally, drawing upon recent advancements in the field, we reflect on and discuss the major challenges and future research directions for PE malware generation technologies.

The structure of this paper is organized as follows. Section 1 introduces the research background of adversarial malware generation techniques. Section 2 formulates the research questions and presents the search methodology employed to identify the relevant literature. Section 3 provides an overview of the PE file format and fundamental concepts of RL, along with a discussion of RL's advantages in the context of adversarial malware gen-

eration. Section 4 proposes a foundational framework for RL-based adversarial malware generation techniques. Section 5 offers an in-depth analysis of existing RL-driven adversarial malware generation approaches. Section 6 introduces a comprehensive evaluation system tailored to RL-based adversarial malware generation methodologies. In Section 7, we revisit and address the research questions formulated in Section 2 based on the findings from the reviewed literature. Section 8 explores potential future research directions in the field of malware generation technologies. Finally, Section 9 concludes the paper with a summary of the main insights and contributions.

2. Methodology

This section outlines the systematic methodology adopted in this survey. The process was structured around two core stages: first, the formulation of clearly defined research questions; second, the design and implementation of a rigorous and reproducible search strategy aimed at identifying all relevant literature within the scope of the study.

2.1. Research Question

This study aimed to address the following research questions:

RQ1: How does the design of the action space influence the quality of adversarial sample generation? What principles should be followed to ensure an effective and semantically meaningful action space design?

RQ2: How can a reinforcement learning environment module be constructed to interact with anti-malware engines in real-world scenarios?

RQ3: Within the context of generating adversarial malware samples, which reinforcement learning architectures demonstrate applicability and effectiveness?

2.2. Search Strategy

To conduct a comprehensive review of RL-driven adversarial malware generation techniques, we performed a systematic literature search across major academic databases including Springer, IEEE Xplore, Elsevier, and ACM Digital Library, as well as third-party search platforms such as Google Scholar and ResearchGate. The search was guided by a set of keywords including “reinforcement learning”, “malware”, “evasion”, “Windows”, and “adversarial attack”. A thorough analysis was then carried out on the selected body of work.

Our investigation revealed that the majority of the relevant studies were published after 2017, with earlier works primarily focusing on traditional machine learning approaches for malware evasion. Since 2017, there has been growing interest in applying RL techniques to the domain of adversarial malware generation, shifting the research focus toward understanding the adaptability and effectiveness of RL-based methods in this context.

This study focuses on publications from 2017 to 2025. As summarized in Table 1, we adopted a two-stage approach: we first conducted extensive preliminary reading to identify representative papers, followed by an in-depth manual analysis to ensure a comprehensive understanding of each paper’s core contributions and technical insights. The “Validation” column denotes whether it was confirmed that the generated adversarial malware samples retained their functional consistency. The “Content” column describes the nature of the content injected into the malware, which could be either random, derived from benign samples, or deterministically chosen.

Table 1. Summary of state-of-the-art RL-based adversarial malware generation methods.

Method	Year	Target Model	RL Method	State Space	Action Space	Reward	Validation	Content
gym-malware [34]	2017	GDBT	DQN	2350d	10	evasion	-	random
gym-plus [43]	2018	LightGBM [44]	Double DQN, Sarsa	2350d	16	-	-	random
DQEAF [35]	2019	GBDT	DDQN	513d	4	evasion, query number	✓	random
gym-malware-mini [45]	2020	GDBT	DQN,A2C	2350d	10	evasion	-	deterministic
RLAttackNet [36]	2020	DeepDetectNet	Double DQN	2478d	218	evasion, query number	✓	deterministic
AMG-VAC [46]	2021	LightGBM, MalConv [15]	VAC	2350d, raw bytes	10	-	✓	random
AIMED-RL [47]	2021	LightGBM	DiDDQN	2351d	10	evasion, query number, similarity	✓	random
AMG-IRL [48]	2021	360engine	IRL	-	4	IRL	✓	random
Gibert [49]	2022	CNN classifiers	Double DQN	-	1	evasion	-	deterministic
mab-malware [50]	2022	LightGBM, MalConv, 3 AVS	multi-armed-bandit	2350d, raw bytes	8	evasion	✓	random, benign
MERLIN [51]	2022	LightGBM/ MalConv/AV	DQN policy Gradient	-	15	evasion	-	random
Malinfo [52]	2022	VirusTotal [53]	DP, TD	-	Obfusmal, Stealmal, Hollow-mal	evasion	-	deterministic
SRL [54]	2022	GNN detector	CFG	-	injecting semantic NOPs into the CFG	evasion	-	deterministic
PSPmal [55]	2023	LightGBM	PER/TS	2381d	10	evasion	✓	deterministic
OBFU-mal [56]	2023	LightGBM, MalConv, AV	DQN	2350d, raw bytes	12	evasion intrinsic reward	-	-
Enhancing RLAMG [57]	2024	LightGBM, MalConv, FireEyeNet [58]	PPO	2350d, raw bytes	5	evasion, intrinsic reward	✓	deterministic
RLAEG [59]	2024	LightGBM, MalConv, MalConv-GCT [16]	SAC	636d	12	evasion	✓	-
GAME-RL [60]	2025	API Call Sequence-Based Black-Box Model	-	-	insert API calls	evasion, query number	✓	-

3. Preliminaries

3.1. Portable Executable Format

The Portable Executable (PE) file format [61] is a binary file structure developed by Microsoft for Windows NT and its subsequent operating systems, based on the Common Object File Format (COFF). It is primarily used to store executable code, data, and resources for loading and execution by the operating system. The PE format is widely adopted across the Windows ecosystem and supports various file types, including but not limited to .exe, .dll, .ocx, .sys, .cpl, .mui, .scr, and .acm. As illustrated in Figure 2, the core structure of a PE file consists of three main components: the header information, the section table, and the unmapped data.

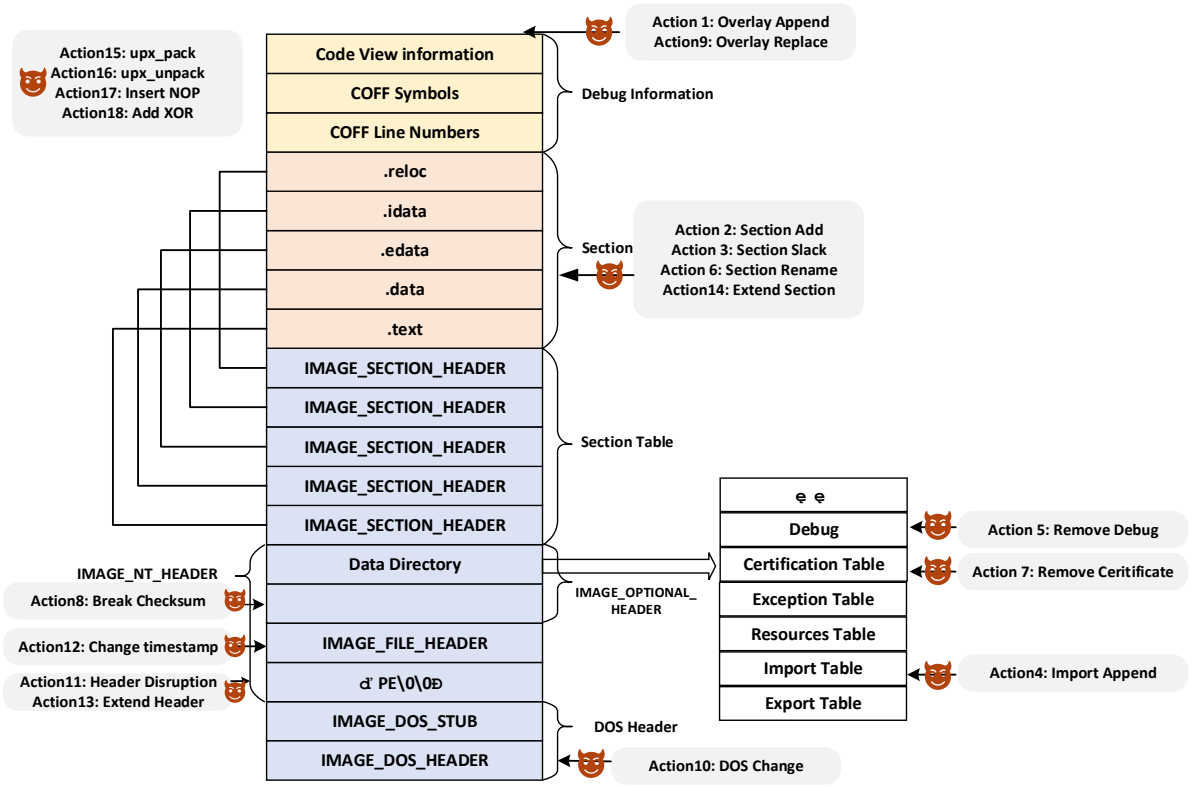


Figure 2. PE file format and the locations perturbed by different actions.

The header information defines the memory loading rules and functional characteristics of a PE file through a multi-layered nested structure. It primarily consists of three components: the DOS header, the NT headers, and the section table. The DOS header is a 64-byte structure located at the beginning of the file, originally designed to ensure backward compatibility with the Microsoft Disk Operating System (MS-DOS) [62]. Following the DOS header is the NT header, which contains three key elements: the PE signature, the file header, and the optional header. The PE signature resides at the start of the NT header and is used by the operating system during program loading to validate the file's integrity and format. The file header is a fixed-length structure of 20 bytes that contains basic metadata such as the target CPU architecture, the number of sections, and a timestamp indicating the file creation time. The optional header plays a central role in defining the runtime behavior and memory management of the executable. It includes critical control fields such as the entry point address, subsystem type, image base address, and more. Additionally, it contains the data directory table, which provides predefined entries for quick access to specific data structures within the file. Finally, the section table describes the metadata of all individual sections present in the PE file, including their names, sizes, file offsets, virtual

addresses, and access permissions. This table serves as a mapping guide for the Windows loader when preparing the executable for execution.

PE files organize different functional modules into distinct sections based on their types, such as code, data, and resources, enabling a high degree of extensibility. This modular structure allows developers to add custom sections according to specific requirements. Each section is responsible for storing a particular type of code or data and is associated with its own set of access permissions. The combination of the data directory, section table, and individual sections enables efficient data organization and retrieval within the file.

Unmapped data refers to the physical content of a PE file that is read by the loader but not mapped into virtual memory during the loading process.

3.2. Reinforcement Learning

Reinforcement learning (RL) achieves goal optimization through continuous interaction between an agent and its environment. As illustrated in Figure 3a, the interaction mechanism operates as follows: in each time step, the agent first observes the current state of the environment and then computes an action based on its policy. This action is subsequently applied to the environment. Upon receiving the action, the environment provides two types of feedback: an immediate reward signal and the resulting next state. At the beginning of the subsequent time step, the agent uses this new state to compute the next action, thereby forming a closed-loop iterative process commonly referred to as the state–action–reward–new state cycle. This loop continues until a termination condition is met or the predefined objective is achieved.

The entire process can be formalized as a Markov Decision Process (MDP), as illustrated in Figure 3b. An MDP is defined by the tuple $\langle S, A, R, P \rangle$, where

- S denotes the set of all possible states;
- A represents the set of available actions, with the agent selecting an action a_t based on its policy $\pi(a_t|s_t)$ in state s_t;
- R is the reward function, which defines the immediate reward received after transitioning from state s_t to state s_{t+1} as a result of taking action a_t;
- P is the state transition probability function, representing the likelihood of transitioning to state s_{t+1} given the current state s_t and action a_t. This transition depends only on the current state and action and not on any previous states or actions, thereby satisfying the Markov property.

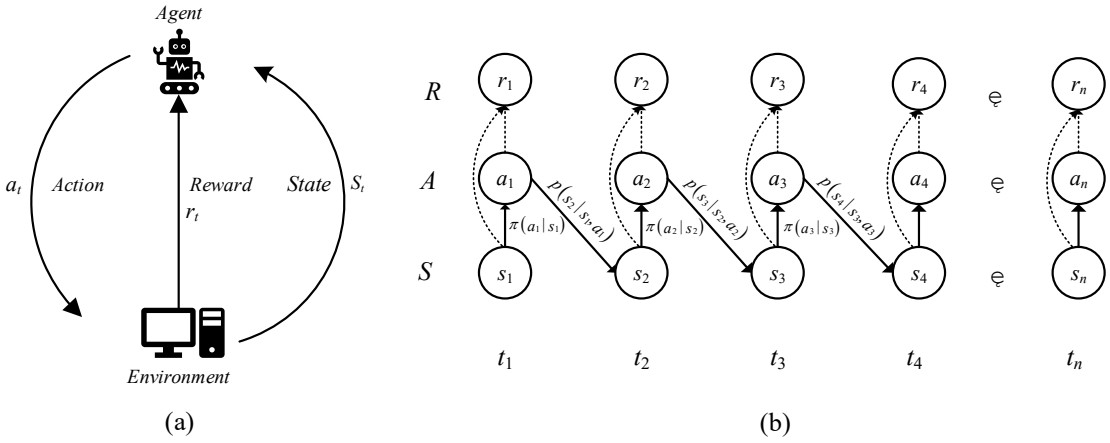


Figure 3. (**a**) Reinforcement learning process; (**b**) representation of Markov Decision Process.

3.3. Advantages of RL in Adversarial Malware Generation

In real-world attack scenarios, the black-box nature of anti-malware engines presents a complete lack of transparency to attackers, revealing only a binary classification outcome (i.e., malicious or benign) as feedback. This severely limits the attacker's ability to obtain intermediate feature interpretations or gradient propagation paths that are typically exploited in white-box settings.

Moreover, unlike adversarial example generation in the image domain, where small pixel-level perturbations are often sufficient to deceive classifiers [24,63], the generation of adversarial malware samples faces multiple critical constraints. First, modifications must strictly adhere to the structural integrity requirements of executable file formats. Second, the resulting sample must remain effective in evading detection models. Third, the original malicious functionality must be preserved without degradation. As a result, crafting adversarial malware samples poses significantly greater practical challenges compared to traditional adversarial attacks in other domains.

To address the aforementioned challenges, researchers have increasingly turned to RL algorithms to develop more adaptive frameworks for the generation of adversarial malware samples. RL offers several distinct advantages in this context:

- RL enables agents to learn optimal evasion strategies through limited interaction with real-world anti-malware engines, without requiring explicit knowledge of the underlying detection model architecture;
- RL enables the end-to-end modification of malware in the problem space, directly generating adversarial malware samples rather than manipulating features in the feature space;
- RL enables agents to perform a series of discrete operations on malware, modifying it without undermining its original functionality.

4. Framework for RL-Driven Adversarial Malware Generation

Since RL techniques were introduced into the field of adversarial malware generation, the foci of related research have also varied due to differences in attack objectives, technical approaches, and application scenarios. As illustrated in Figure 4, this section provides a systematic overview of the progress in the field by presenting a comprehensive review and analysis of its developmental trajectory. The goal is to distill a general architectural framework for RL-based adversarial malware generation. This synthesis aims to establish a clear evolutionary roadmap and theoretical foundation for future research in adversarial malware engineering.

In the early stages of research on RL-driven adversarial malware generation, most studies focused primarily on demonstrating the feasibility and effectiveness of RL techniques in this domain. These pioneering efforts typically employed models such as LightGBM [16] and MalConv [12] as target detectors, with a strong emphasis on investigating how key components influence the evasion rates of generated adversarial samples. LightGBM is a tree-based classifier model that detects malware by extracting 2351-dimensional feature vectors from each sample. MalConv is a malware detection model that learns knowledge directly from the raw bytes of malware samples via a convolutional neural network.

As the field has matured, recent research has shifted toward exploring malware generation under more constrained or realistic black-box settings. Increasingly, studies are adopting real-world anti-malware engines as target detection systems. In addition to the evasion performance, current work places greater emphasis on other critical factors, including model transferability across different detection systems and the functional consistency of modified samples. This evolution reflects a growing focus on practical applicability and robustness in adversarial malware engineering.

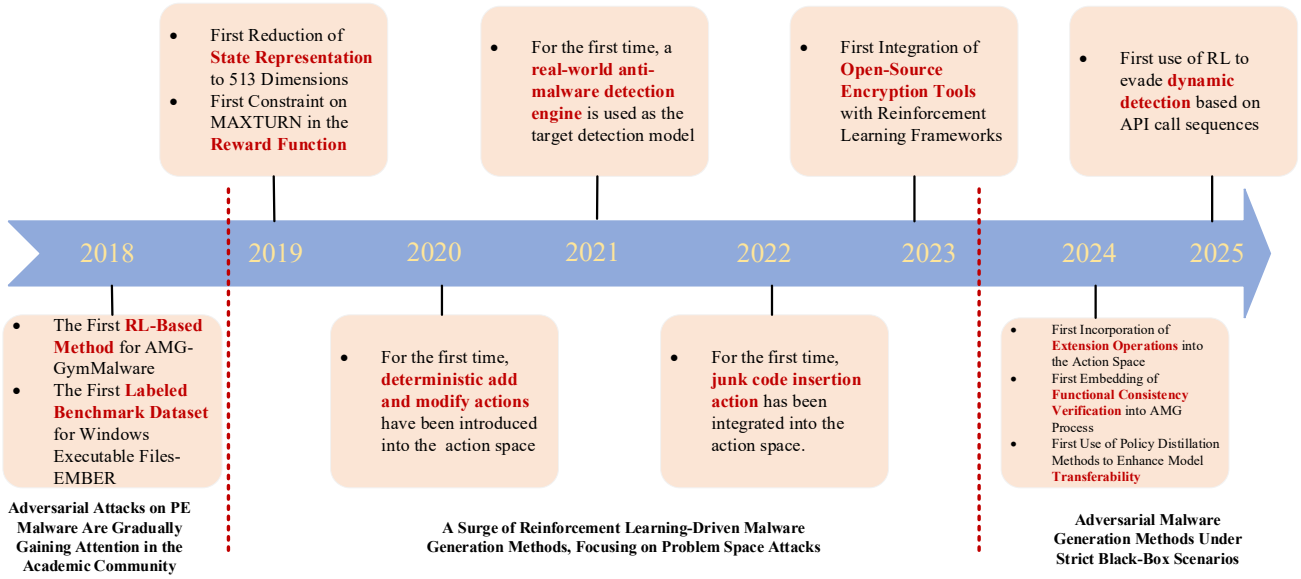


Figure 4. The development trajectory of RL techniques in the field of adversarial malware generation.

In general, PE malware generation methods based on reinforcement learning can be formalized as a closed-loop framework consisting of three sequential stages: modification, query, and feedback. The detailed workflow of this process is illustrated in Figure 5.

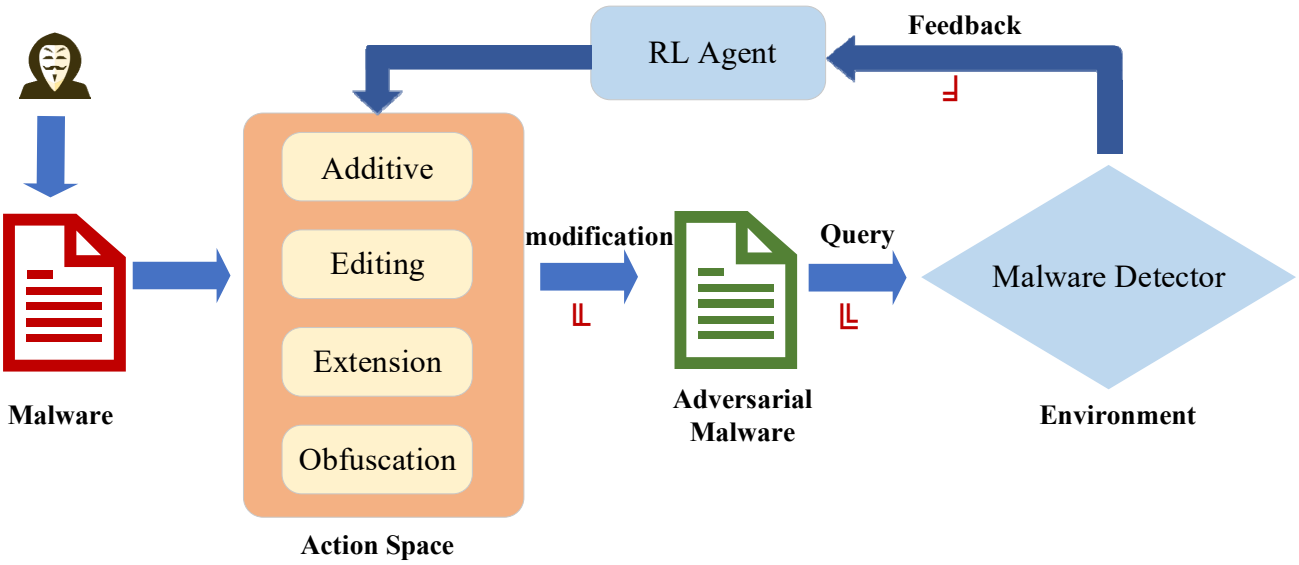


Figure 5. The framework for reinforcement learning-driven adversarial malware generation.

First of all, the agent performs a series of operations on the original malware that do not affect its original functions, such as addition, editing, extension, or obfuscation, to generate adversarial samples capable of evading detection. Subsequently, the generated adversarial samples are submitted to the target detector for interactive queries. Finally, the target detector returns the detection results as environmental feedback. The agent adjusts its strategy based on this feedback information, thereby forming a closed-loop control process of "modification–query–feedback".

This loop continues until either the predefined maximum number of queries is reached or the adversarial sample successfully bypasses the detector. Within this framework,

- The set of permissible operations that the agent can perform constitutes its action space;

- The environmental states perceived by the agent during interaction constitute the state space representation;
- The reward signal derived from the detector's feedback guides the agent's policy updates and forms the basis of the reward function.

Thus far, researchers have primarily focused on the three core elements of reinforcement learning, namely state space representation, action space design, and reward function construction. By establishing a closed-loop iterative process among the state, action, and reward, these approaches aim to guide the agent toward optimal decision making and identify the most effective action sequences. The overarching objective of the framework is to enhance both the effectiveness and evasion success rate of generated adversarial samples, while also improving their robustness and practical applicability in real-world environments.

In this paper, we provide a systematic review and in-depth analysis of the design methodologies for the action space, state space, and reward function. Our goal is to offer both a theoretical foundation and practical guidance for future research in this rapidly evolving field.

5. Methods of RL-Driven Adversarial Malware Generation

5.1. Action Space Design

During each interaction between the agent and the environment, the agent selects the next action from the defined action space based on its policy function in response to the received environmental feedback. In the field of adversarial malware generation, in addition to designing an appropriate action space size to avoid increasing the training complexity and computational cost of the model, it is also necessary to screen out suitable actions to ensure that the executability and malice of the malware are not damaged after the execution of the actions.

The first RL-based framework for adversarial malware generation, gym-malware [34], defines a set of ten actions that modify the malware sample without altering its functionality or file format, including operations such as renaming existing sections, adding new sections, and modifying header checksums.

Subsequently, several studies extended upon Anderson et al.'s foundational work. As summarized in Table 1, methods such as gym-plus [43], MERLIN [51], OBFU-mal [56], and RLAEG [59] expanded the action space to include more than ten operations, with gym-plus defining a total of sixteen distinct actions. However, given that larger action spaces can significantly increase the difficulty of policy learning, other approaches opted for reduced action sets to improve the training efficiency.

In this paper, we present the systematic categorization of existing action space designs, grouping them into four major classes, additive, editing, extension, and obfuscation, as detailed in Table 2. A total of 18 specific actions are covered, and the specific operation positions of each action in the PE file are marked in Figure 2. The "Adoption rate" column specifies the proportion of the reviewed literature that employs this specific action during the generation of adversarial malware samples.

Additive: An analysis of the surveyed literature reveals that addition-based operations represent the most frequently adopted category among the four major action types. As shown in Figure 2, these operations predominantly target structurally flexible regions of the PE file, including the section gaps, overlay, and import table area. When substantial payload insertion is required, attackers may also opt to introduce new sections to accommodate the modifications. By leveraging the inherent structural characteristics of PE files, such additions can effectively alter key feature representations, such as byte histograms, entropy profiles, and string content, without compromising the original malicious functionality.

Table 2. Details of the actions, including the name, type, description, and methods that adopt the action.

Number	Name	Type	Description	Methods	Adoption Rate
1	Overlay Append	Additive	Append bytes to the end of the file	[34–36,43,45–48,50,55–57,59]	86.7%
2	Section Add	Additive	Add a new section	[34,43,45,47,50,55–57,59]	60.0%
3	Section Slack	Additive	Append bytes to extra spaces at the ends of sections	[34–36,43,45–48,50,55,57,59]	80.0%
4	Import Append	Additive	Add a function to the import address table	[34–36,43,45–48,55–57]	73.3%
5	Remove Debug	Additive	Unlink the debug section form the header	[34,36,43,45–47,50,55,56]	60.0%
6	Section Rename	Editing	Manipulate existing section names	[34,36,43,45–47,50,55,56,59]	66.7%
7	Remove Certificate	Editing	Remove the signed certificate	[34–36,43,45–48,50,56]	66.7%
8	Break Checksum	Editing	Modify the header checksum	[34,43,45–47,50,55,56]	53.3%
9	Overlay Replace	Editing	Replace bytes at the end of the file	[43,59]	13.3%
10	Dos Change	Editing	Reset the DOS header except the magic number and PE header offset	[43,57,59]	20%
11	Header Disrupt	Editing	Reset portions of the PE header	[43,55,56,59]	26.7%
12	Change Timestamp	Editing	Reset the timestamp	[43,46,55,56,59]	33.3%
13	Extend Header	Extension	Fill in the strings by increasing the PE header offset to empty the new space	[43,59]	13.3%
14	Extend Section	Extension	Fill in the bytes by increasing the offset of each section to empty the new space	[43,59]	13.3%
15	Upx Pack	Obfuscation	Pack with UPX tools	[34,43,45–47,56,59]	46.7%
16	Upx Unpack	Obfuscation	Unpack with UPX tools	[34,43,45–47,59]	40.0%
17	Insert NOP	Obfuscation	Insert a NOP instruction	[49,54]	13.3%
18	Add XOR	Obfuscation	Appy an XOR encryption loop	[56]	6.70%

An empirical analysis indicates that over 80% of the reviewed studies employ the action overlay append and section slack as primary evasion techniques. Specifically, the overlay is not referenced during memory mapping by the loader; it does not participate in execution and thus provides a safe space for data embedding. The section gap, on the other hand, arises due to alignment constraints imposed by the FileAlignment field in the IMAGE_OPTIONAL_HEADER. Sections stored on disk must be aligned to multiples of this value, often resulting in unused space between sections filled with null bytes (0×00). Since these gaps are not loaded into memory during execution, they offer an ideal location for the introduction of perturbations without altering the runtime behavior.

In addition to the location of padding within a PE file, the content of the inserted data plays a critical role in determining the evasion effectiveness of adversarial malware samples. Existing approaches to padding content can be broadly categorized into three types: random padding, benign injection, and deterministic padding. Random padding involves inserting arbitrary byte sequences into structurally flexible regions such as section gaps, the overlay, or the import table area. Benign injection, inspired by biological virus behavior, enhances random padding by embedding byte fragments extracted from real-world benign executables into the padding regions. This approach aims to mimic legitimate code patterns and improve the evasion performance. Deterministic padding refines the modification process by defining fixed transformations. Unlike random or sample-based methods, deterministic padding avoids stochasticity in the perturbation process, thereby improving both the learning stability and generation efficiency. Moreover, recent work [57] has explored the use of GANs to synthesize realistic benign-like byte sequences that are subsequently used as padding content, offering a middle ground between randomness and determinism. Overall, the three categories exhibit significant differences in evasion efficacy, with the following empirical ranking: deterministic padding > benign injection > random padding. This hierarchy arises due to several key limitations inherent in random padding: the stochastic nature of the inserted content causes the environment to provide inconsistent feedback for identical agent actions, leading to unstable policy updates and increased training difficulty. In contrast, while benign injection may, in certain cases, mislead detection models by introducing seemingly legitimate content without altering the original malicious functionality, it suffers from two major drawbacks: hardcoded benign bytes are increasingly recognized by modern detectors as evasion artifacts, and the method is computationally inefficient when applied at scale, particularly in large-scale adversarial training scenarios. On the other hand, deterministic padding offers a structured and repeatable transformation mechanism that eliminates noise during agent–environment interaction. As a result, it not only improves the convergence of reinforcement learning models but also significantly enhances the efficiency and reliability of adversarial sample generation.

Editing: The editing category refers to transformations that alter specific features of the original sample without injecting new content. As shown in Figure 2 and Table 2, compared to the other three action types, the number of editing-based operations accounts for the largest proportion, reaching 7/18. These operations span multiple structural components of the PE file, including the DOS header, IMAGE_NT_HEADER, section, and overlay, with a notable concentration in the file header.

Existing studies have shown that the most influential features for detection models are often located within the PE file headers [64–66]. The DOS header is a total of 64 bytes. Among these, the fields e_magic and e_lfanew are used to identify the file format and point to the offset of the "PE\0\0" signature, respectively. The remaining 58 bytes are typically filled with default or unused values in modern PE files, which allows attackers to embed adversarial perturbations through a DOS change operation.

The NT header, which includes the PE signature, file header, and optional header, serve as the foundational structure for the validation and loading of PE files. As discussed in Section 3.1, these headers define key aspects such as file legitimacy, basic attributes, and runtime memory management mechanisms. Structured as a multi-layer nested hierarchy, they provide a compact yet comprehensive metadata summary of the executable. By modifying selected fields, such as TimeDateStamp, Checksum, and certain entries in the data directory, the attacker can achieve high evasion success rates while introducing only minimal perturbations. For instance, Wei Song et al. [50] introduced the concept of an action minimizer, which replaces coarse-grained macro-actions with fine-grained micro-actions. Their work investigated how adversarial code can be generated by minimally altering

feature representations to evade detection systems. The experimental results demonstrate that even a single-byte modification can yield effective adversarial samples.

Extension: The extension operation was first introduced into the domain of RL-based adversarial malware generation by Buwei Tian et al. [59]. This technique involves modifying the offset values to shift specific regions of the original sample backward, thereby creating additional space for adversarial manipulation.

More specifically, extension operations can be categorized into two subtypes: section extension, which enlarges a section's allocated space, and header extension, which increases the sizes of header regions to accommodate additional metadata or padding.

In Buwei Tian et al.'s study, it was found that header extension proved to be the most effective transformation when targeting the MalConv-Pre [64] detector. As a result, this action was selected most frequently by the RL agent throughout the training process, with a total of 1504 executions recorded, significantly outperforming other action types in terms of usage frequency.

Obfuscation: Unlike the previously discussed operations, which typically target only localized regions of the PE file, obfuscation techniques such as encryption, packing, and encoding operate on the entire executable. These methods are widely adopted in real-world attack scenarios by hackers due to their high efficacy in evading detection [67,68].

The gym-malware framework was among the first to incorporate packing-based obfuscation into RL-driven malware generation, proposing the use of UPX [69] for the compression or decompression of PE files. Experimental results demonstrated that adversarial samples generated through UPX-based transformations were among the most effective evasion strategies. However, in practice, UPX is a well-known packer commonly used by attackers, and its compressed binaries often contain identifiable signatures. As a result, many modern detectors flag UPX-packed executables as potentially malicious without performing deep inspection, effectively reducing their stealthiness.

As an alternative, dead-code insertion, a prevalent code obfuscation technique, has been introduced into RL-based adversarial malware generation frameworks by researchers such as Gibert et al. and Lan Zhang et al. [49,54], aiming to increase the evasion success rates by embedding semantically inert instructions that do not alter program behavior but confound static analysis.

In another notable study, Brian et al. [56] employed the open-source tool Darkarmar [70], applying one or more rounds of XOR encryption cycles to the PE file to increase the code complexity. Their experiments showed that such XOR-based encryption significantly supports the generation of adversarial samples. Moreover, this operation was frequently selected by the RL agent as the final action in the transformation sequence, indicating its strategic importance in the evasion pipeline.

When designing the action space for RL-based malware evasion, researchers can choose from the above four types of operation. However, in practice, any operation must strictly adhere to the structural constraints of the PE file format. Early studies employed the LIEF library [71] to implement these operations within the PE structure. However, empirical results revealed a significant limitation: over 60% of the modified samples either became non-executable or exhibited altered runtime behavior after a single transformation [50,51]. Subsequent researchers have implemented operations using the PeFile library [72] and incorporated checks against the PE file format specification before making any modifications. This precautionary measure helps to prevent extreme cases where the file might become corrupted.

5.2. State Space Representation

In the RL framework, the environment must interact with the generated adversarial malware samples and provide feedback to the agent regarding the current state, prompting the agent to select the next action for interaction. As shown in Table 3, the most commonly targeted models by researchers can be categorized into three types: LightGBM, MalConv, and commercial anti-malware engines.

Table 3. Comparison of three different types of target models.

Target Model	LightGBM	MalConv	AV
Knowledge of model	Yes	Yes	No
Input data	Features extracted from PE files	Raw bytes extracted from PE files	PE files
Input preprocessing	2351d vectors	Fixed size (1M)	Unknown

During the generation of adversarial malware samples, variations in the target model lead to differentiated state representations during interactions with the environment. Compared to academic models such as LightGBM and MalConv, there are significant limitations when attacking commercial antivirus engines in real-world scenarios: attackers can only observe whether the modified adversarial sample is flagged as malicious by the engine and cannot access internal state representations of the engine. This black-box interaction characteristic renders traditional state modeling methods in reinforcement learning frameworks ineffective.

Therefore, to construct an RL model suitable for real-world adversarial environments, it is essential to address the following core issue: how to reasonably abstract and vectorize the state of black-box detection engines based on binary feedback information.

To achieve a more concise and efficient representation of the current state, the majority of existing studies adopt the eight primitive feature groups derived from the EMBER dataset as the state observations. As shown in Table 4, these features encapsulate elements that are widely used in both industrial and academic malware detection systems. They are composed of five parsed features and three format-agnostic features, specifically including general file information, header information, imported functions, exported functions, section information, byte histograms, byte entropy histograms, and string information. After vectorization, this set results in a high-dimensional feature space of 2351 dimensions.

However, some researchers argue that such high-dimensional observations may lead to the exponential expansion of the state space, increasing the complexity of learning and hindering convergence. To address this issue, DQEAF [35] employs a low-dimensional observation space consisting only of a byte histogram and byte entropy histogram. According to its experimental results, this approach reduces training instability caused by high-dimensional inputs and significantly improves the convergence speed. Nevertheless, the study does not provide detailed rationale for the selection of these two specific features, nor does it include a systematic analysis of the environmental observability.

Table 4. Eight primitive feature groups derived from EMBER.

Feature	Position	Information	Dim
General file information (Parsed feature)	PE header	file_size vsize has_debug exports imports has_relocations has_resources has_signature has_tls symbols	10
Header information (Parsed feature)	PE header	timestamp machine characteristics subsystem dll_characteristics magic major_image_version minor_image_version major_linker_version minor_linker_version major_operating_system_version minor_operating_system_version major_subsystem_version minor_subsystem_version sizeof_code sizeof_headers sizeof_heap_commit	62
Imported functions (Parsed feature)	Import address table	-	1280
Exported functions (Parsed feature)	Export table	-	128
Section information (Parsed feature)	Each section	entry name size entropy vsize props	255
Byte histogram (format-agnostic feature)	Counts of each bytes	-	256
Byte entropy histogram (format-agnostic feature)	Byte entropy	-	256
String information (format-agnostic feature)	Statistics about printable strings	numstrings avlength printabledist printables entropy paths urls registry MZ	104

From the perspective of feature engineering, Buwei Tian et al. [59] conducted a comprehensive analysis of how sample modifications affect each feature group. By eliminating redundant features, they identified that format-agnostic features have a significantly higher impact on adversarial samples than parsed features. Specifically, their ranking of influence is as follows: string information (highest impact), byte histograms, byte entropy histograms, imported functions, and exported functions (negligible impact). Based on these findings, the authors constructed a new 636-dimensional state observation space, which included all 616 dimensions from string, byte histogram, and byte entropy histogram features, along with selected sub-features from general file information, header information, and section information. Their experimental results demonstrate that this reduced-dimensional representation leads to a more stable and robust reinforcement learning training process.

5.3. Reward Function Construction

The reward function, as the feedback mechanism governing the interaction between the agent and the environment, plays a pivotal role in shaping the direction of policy optimization. A well-designed reward function is essential for the success of reinforcement learning algorithms, as it directly influences how the agent learns to modify malware samples effectively. In Table 1, we list the different reward components included in the reward functions of each framework, such as the evasion result, query number, similarity, intrinsic reward, and so on. In most adversarial malware generation methods, the reward function is typically computed dynamically based on the output of the detection model. For instance, in the gym-malware framework, if an adversarial sample is classified as malicious, the reward is set to 0; if the sample is classified as benign, the agent receives a positive reward of 10. This reward structure is directly aligned with the fundamental objective of minimizing the detectability of malicious behavior and thereby encourages the agent to evolve variants of the malware that can evade detection while preserving its intended functionality.

Building upon the foundation of gym-malware, subsequent studies have introduced multi-dimensional reward signals to mitigate the policy fragility caused by relying solely on a single reward metric. gym-malware-mini incorporates penalty-based rewards to accelerate the learning efficiency. Specifically, when an adversarial sample is classified as malicious, the agent receives a negative reward of -1, instead of the neutral 0 used in the original framework. This modification provides stronger negative feedback for failed evasion attempts, encouraging the agent to avoid ineffective strategies early in the training process. Similarly, DQEAF and RLAttackNet enhance the gym-malware reward mechanism by introducing a maximum interaction round limit (MAXTURN) to prevent excessive file modifications. If the agent fails to generate an evasive sample within MAXTURN steps, it receives a final reward of 0. This means that if, after reaching MAXTURN, the adversarial sample is still detected as malicious, the reward is set to 0. On the other hand, if the adversarial sample is classified as benign before reaching MAXTURN, the reward decreases as the number of modifications increases. In contrast, the AIMED-RL framework extends the reward function further by integrating three components: a constraint on the maximum number of modifications, the detection result feedback, and a file similarity metric that balances evasion effectiveness with stealthiness. Specifically, AIMED-RL calculates the proportion of modified regions relative to the original sample to measure the perturbation magnitude. Based on an empirical analysis, the authors found that a 40% modification ratio strikes an optimal trade-off, sufficiently evading detection while maintaining the malicious functionality. In their implementation, the weights assigned to the maximum modification count, detection feedback, and file similarity were set to 0.3, 0.5, and 0.2, respectively,

emphasizing the dominant role of detection feedback while dynamically balancing attack efficacy and concealment.

Unlike DQEAF, RLAttackNet, and AIMED-RL, which all impose explicit constraints on the maximum number of allowed actions, mab-malware identifies significant redundancy in action sequences during adversarial malware generation. It argues that the majority of modifications are unnecessary for successful evasion. To address this, mab-malware introduces a sparse reward allocation mechanism, where only the critical actions that ultimately lead to evasion receive positive rewards. Through action importance filtering, this framework reduces the average length of action sequences, thereby improving both the attack efficiency and model convergence without compromising the evasion success rates.

While these approaches significantly improve the robustness and stability of reinforcement learning policies, they largely rely on attacker expertise and subjective design choices in defining reward structures. To reduce the bias inherent in manual reward engineering, the AMG-IRL framework introduces inverse reinforcement learning (IRL) into the domain of adversarial malware generation. At its core, AMG-IRL initially uses the detection model to score each action. Once sufficient samples have been collected, the framework autonomously learns the reward function based on the observed state space, action space, and transition probabilities. This data-driven approach enables the system to infer intrinsic reward patterns from environmental interactions, reducing the dependence on human-defined heuristics and enhancing its generalization across diverse evasion scenarios.

In addition to the aforementioned reward engineering strategies, researchers have also explored intrinsic motivation mechanisms to address the challenge of sparse external rewards in reinforcement learning-based adversarial malware generation. Specifically, the Enhancing RLAMG framework incorporates the intrinsic curiosity module (ICM) [73] to improve policy optimization under sparse reward conditions. The reward function in Enhancing RLAMG comprises two components: a sparse extrinsic reward, provided by the environment, which follows the same mechanism as in gym-malware—a reward of +10 is assigned when an adversarial sample successfully evades detection, and 0 otherwise—and an intrinsic reward, computed by the ICM module, which is derived from the discrepancy between the agent's predicted next state and the actual observed state after executing an action. This intrinsic reward mechanism serves to drive exploration via curiosity, encouraging the agent to visit less familiar states and thereby improving both the data efficiency and generalization capabilities during policy learning. By incorporating this model-aware exploration bonus, Enhancing RLAMG effectively mitigates the limitations imposed by the binary and infrequent feedback typical of real-world malware detection systems.

5.4. Selection of RL Architectures

Based on the findings presented in Sections 5.1 and 5.2, the problem of adversarial malware generation exhibits a coupled structure in its mathematical formulation, characterized by a discrete action space and a continuous, high-dimensional state space. As a result, the adaptive selection of reinforcement learning algorithms becomes a critical factor in achieving effective evasion.

The chosen algorithm must strike a balance between discrete action decision making and the accurate representation of high-dimensional states, ensuring that the learning process satisfies three key requirements simultaneously: policy convergence, sample generation efficiency, and a high evasion success rate. In Table 5, several reinforcement learning algorithms that are frequently employed in the domain of adversarial malware generation are presented.

Table 5. A comparison of several typical reinforcement learning algorithms, including their state spaces, action spaces, and convergence speeds.

Category	Method	State Space	Action Space	Convergence Speed
Value-based	DQN	Continuous	Discrete	Low sample efficiency, requires a large number of environment interactions
	Double DQN	Continuous	Discrete	
	Dueling DQN	Continuous	Discrete	
	DiDDQN	Continuous	Discrete	
Actor–Critic	A3C	Continuous	Discrete	Policy gradient direct optimization objective function, converges faster
	SAC	Continuous	Continuous	
	PPO	Continuous or Discrete	Continuous	

In response to the demand for the modeling of discrete action spaces in adversarial malware generation, researchers have primarily focused on value-based reinforcement learning methods, particularly variants of Deep Q-Networks (DQN) and their extensions. By customizing the network architectures and fine-tuning the hyperparameters, these approaches aim to achieve dynamic adaptation between evasion strategies and evolving detection environments.

During the early stages of applying RL to adversarial malware generation, researchers explored various DQN variants, such as the Double DQN [74], Dueling DQN [75], and Distributional Double DQN (DiDDQN) [76], to iteratively improve the policy convergence speed and the precision of action selection. These algorithmic enhancements significantly contributed to the stability and effectiveness of discrete decision making under high-dimensional state representations.

However, due to the reliance of DQNs on an experience replay mechanism, which stores and samples historical interaction data, these improvements also introduce a notable computational overhead. In black-box detection scenarios where real-time adaptability is crucial, the storage inefficiency and delayed policy updates inherent in experience replay become significant bottlenecks, limiting the responsiveness and scalability of such frameworks in practical attack settings.

In response to the limitations of value-based methods in handling high-dimensional state spaces, researchers have gradually shifted focus toward modeling techniques that better accommodate such complex representations. To address the challenges posed by these high-dimensional states, a growing body of work has explored enhanced algorithms based on the actor–critic framework.

Ebrahimi et al. proposed the AMG-VAC algorithm, which extends the Variational Actor–Critic (VAC) algorithm [77] originally designed for continuous action spaces to discrete action settings. This adaptation addresses the inefficiency of the traditional VAC in selecting discrete actions by incorporating an action-masking mechanism, thereby enabling more effective policy learning within constrained action spaces.

On another front, to overcome the issue of insufficient exploration in complex state spaces, Buwei Tian et al. [59] introduced the RLAEG framework. Built upon the Soft Actor–Critic (SAC) algorithm [78], RLAEG leverages an entropy regularization mechanism to balance exploration and exploitation, enhancing the robustness of the agent's policy in high-dimensional environments. To adapt SAC for discrete action selection, RLAEG maps continuous action sampling to discrete decisions through a tailored discretization process.

Overall, the reinforcement learning modeling of adversarial malware generation faces a dual challenge: discrete action space constraints and high-dimensional state representation. Early approaches based on the DQN and its variants improved policy convergence by refining the value function approximation and experience replay mechanisms; however, they were limited by storage inefficiencies and poor dynamic adaptability, especially in

black-box detection scenarios. Subsequent research leveraging discretized actor–critic frameworks has demonstrated greater resilience in navigating high-dimensional state spaces, achieving a more balanced trade-off between exploration and action selection. These advancements mark a paradigm shift from historically data-driven, experience-reliant methods toward strategies that integrate policy optimization with dynamic exploration capabilities, thereby enhancing both their effectiveness and generalizability in real-world evasion tasks.

6. Evaluation System for RL-Driven Adversarial Malware Generation

This section introduces a multi-faceted evaluation system designed specifically for attack models that generate adversarial malware samples using reinforcement learning. The framework incorporates multiple evaluation criteria, including the evasion rate, transferability, functionality preservation, injection rate, interaction rounds, and operation count. The primary objective of this framework is to provide a holistic assessment of the performance characteristics and real-world applicability of these attack models. By evaluating across diverse dimensions, it enables systematic and objective comparisons, offering a solid foundation for future research and development in this domain.

As illustrated in Table 6, the evasion rate is designated as the primary evaluation metric, serving as a direct measure of the model's attack effectiveness. The remaining metrics are considered secondary indicators, which support auxiliary analysis and offer deeper insights into various aspects of model behavior, such as its stealthiness, efficiency, and robustness.

Table 6. Evaluation metrics for RL-driven adversarial malware generation.

	Metric	Description
Primary evaluation metric	Evasion rate	The ratio of the number of adversarial samples that maintain their malicious functionality and successfully evade detection to the total number of adversarial samples
Auxiliary evaluation metrics	Functionality preservation	Adversarial samples can evade detection while fully retaining the original malicious functions
	Transferability	Whether the samples generated by one model can escape from other models
	Injection rate	The ratio of adversarial perturbation to the original sample size
	Interaction rounds	The number of rounds of confrontation between the agent and the detection model
	Operation count	The cumulative modification steps of adversarial samples

The evasion rate is defined as the ratio of the number of adversarial samples that maintain their malicious functionality and successfully evade detection to the total number of adversarial samples generated by the model. This can be mathematically expressed using the following formula:

$$EvasionRate = \frac{|E \cap F|}{N} \quad (1)$$

E represents the set of adversarial malware samples that successfully evade detection. F denotes the set of adversarial samples that retain their malicious functionality after modification. N indicates the total number of adversarial malware samples generated by the attack model.

Functionality preservation serves as a fundamental prerequisite in evaluating the validity of adversarial malware samples. It directly defines the effective boundary of evasion rate computation, as an adversarial sample is only considered a genuine threat if it successfully evades detection while retaining its original malicious capabilities. In contrast to the evasion rate, which can be quantitatively measured, functionality preservation lacks a formal mathematical formulation and typically relies on a combination of static and dynamic analysis techniques.

Researchers commonly employ two complementary verification approaches to ensure that neither the file structure nor the malicious behavior has been compromised during the evasion process:

1. Static Verification: The control flow graphs (CFGs) of the original and modified binaries are compared using tools such as IDA Pro [79], aiming to detect whether critical logic paths and function boundaries remain intact.
2. Dynamic Verification: Samples are executed in controlled environments such as Cuckoo Sandbox [80] to monitor the activation of malicious behaviors.

Both methods traditionally depend on manual inspection and behavioral comparison, making them labor-intensive and difficult to scale. To address these limitations, Buwei Tian et al. proposed an automated functionality verification framework based on the Angr symbolic execution engine [81]. This approach enables the automatic analysis of code path coverage and data flow dependencies, effectively embedding functionality validation into the adversarial generation pipeline. As a result, this framework significantly reduces human intervention while improving both the reliability and efficiency of adversarial malware synthesis.

Transferability refers to the ability of an adversarial sample to generalize across different detection models—that is, whether a sample crafted to evade one model can also evade others without further modification. In real-world attack scenarios, security vendors often employ heterogeneous detection engines with varying architectures and training data. As a result, attackers typically lack prior knowledge of the specific model deployed in the target environment. In this context, high transferability significantly enhances the practical effectiveness of adversarial attacks. By improving the multi-engine evasion rate, it broadens the range of exploitable targets and reduces the need for model-specific crafting, thereby increasing both the attack coverage and operational efficiency. Consequently, transferability serves as a key indicator in evaluating the real-world applicability and strategic value of adversarial malware generation techniques. Maximizing transferability not only improves the likelihood of successful evasion but also enhances the scalability and cost-effectiveness of automated attack frameworks.

The injection rate, interaction rounds, and operation count are essential auxiliary metrics in evaluating the effectiveness of adversarial attacks. These three indicators collectively influence the evasion rate through distinct but interrelated mechanisms. The injection rate is defined as the ratio of the size of injected adversarial perturbations to the original file. A lower injection rate reduces structural anomalies in the modified sample, thereby enhancing the stealthiness of the attack and minimizing the risk of heuristic-based detection [82]. Aryal K et al. observed that, when the injection rate exceeds 30%, the generation of adversarial samples requires addressing byte conflicts and format verification, leading to an increase in the sample generation time [64]. Interaction rounds refers to the number of adversarial interactions between the agent and the detection model during the evasion process. A strategy that minimizes the interaction rounds can reduce the model's sensitivity to abnormal behavior patterns while simultaneously improving the sample generation efficiency. The operation count denotes the cumulative number of modification steps applied to generate an adversarial sample. Streamlining the operation sequence eliminates

redundant actions, improves the attack feasibility, and helps to preserve the integrity of the malicious functionality.

7. Discussion

This section offers a systematic review of the research framework and provides comprehensive answers to the research questions initially formulated in Section 2.

Response to RQ1: The action space defines the set of operations that an agent can perform on an adversarial malware sample at any given state. While the scale of the action space directly affects the training efficiency, other factors, including the action diversity, operation determinism, execution order, and modification ratio, also play critical roles in shaping the quality of the generated adversarial samples. A well-designed action space should adhere to three core principles:

1. Simplicity: Under the premise of maintaining operational simplicity, the action space should systematically explore how variations in the modification magnitude and deterministic behavior of a single operation affect the feature decision boundary.
2. Validity: All operations must strictly follow the PE format specification while ensuring that its malicious functions are not damaged to avoid sample failure due to operations. This has two implications: first, samples that do not adhere to the PE file format specifications may be unable to run properly on Windows systems; second, even if the samples are able to execute successfully, their original malicious functionality may be compromised or eliminated during the modification process.
3. Comprehensiveness: Avoid limiting modifications to a single type of transformation. Instead, the action space should incorporate diverse strategies across multiple dimensions, including code obfuscation techniques. This facilitates the comprehensive perturbation of the model's decision boundary and enhances the robustness of adversarial samples against detection mechanisms.

Response to RQ2: Within the RL framework, the environment module serves as the core component responsible for providing feedback to the agent. Specifically, it returns the immediate reward and the next state based on the current action taken by the agent. In realistic black-box malware detection scenarios, the environment model must integrate two essential components: a malware detection engine and a state space modeler. The former acts as the primary source of feedback by delivering detection outcomes, typically benign or malicious, while the latter dynamically models multi-dimensional state features to compensate for the limited information conveyed by the binary feedback signal. During the construction of the state representation, two critical aspects must be carefully considered:

1. Sensitivity to Agent Actions: The observed state features should be responsive to the agent's modifications, ensuring that changes in the adversarial sample are meaningfully reflected in the state transition dynamics.
2. Comprehensiveness of State Features: The feature set must provide broad coverage of the malware sample's structural and behavioral properties, ensuring that no critical information is lost during the adversarial perturbation process.

Response to RQ3: In the domain of adversarial malware generation, in response to the mathematical modeling of discrete action spaces and high-dimensional continuous state spaces, researchers have made targeted improvements to traditional reinforcement learning methods. From a methodological perspective, recent trends reflect a clear shift away from Deep Q-Networks (DQN), which rely heavily on historical data and experience replay mechanisms, toward more flexible and expressive actor–critic architectures. These models combine the advantages of policy optimization with those of value-based estimation, enabling the more effective exploration of complex, high-dimensional state–action spaces

under dynamic environmental feedback. This transition not only underscores the research community's pursuit of higher algorithmic efficiency but also highlights the growing demand for flexibility and adaptability in handling sophisticated adversarial tasks such as evasion-oriented malware synthesis.

8. Open Issues and Future Directions

Investigating adversarial malware generation techniques is of critical importance in developing robust malware detection systems capable of withstanding evasion attempts. Despite the growing adoption of RL in the domain of adversarial malware generation and the promising progress achieved thus far, the current methodologies still face substantial limitations that hinder their practical deployment in real-world environments. This section provides a critical review of these challenges and outlines potential research directions that may address these shortcomings.

8.1. Emerging Trends in Adversarial Malware Generation Techniques

8.1.1. Toward Practical Evaluation of RL in Adversarial Malware Generation

The current research on RL-driven adversarial malware generation predominantly focuses on evasion rate comparisons against specific detection models when evaluating the experimental performance. While the evasion rate remains an important metric, this narrow focus has become a major bottleneck that limits the rapid advancement of RL-based adversarial malware synthesis.

In real-world black-box detection scenarios, it is not only important to achieve a high evasion rate, but it is also crucial to enhance the practicality and adaptability of the generation model. While ensuring that adversarial samples can effectively evade detection, it is equally important to ensure that these samples maintain their effectiveness across different environments and detection systems.

8.1.2. Addressing the Gap Between Static and Dynamic Detection in Adversarial Malware Generation

Despite the development of numerous efficient RL algorithms for the generation of adversarial malware samples, most of these approaches are primarily designed for static detection environments. In reality, contemporary detection models encompass both static analysis and dynamic analysis mechanisms. The latter not only examines the characteristics of malicious code but also evaluates its behavior across multiple dimensions, providing a more comprehensive threat assessment.

Moreover, modern detection engines are increasingly incorporating cloud-based detection capabilities, enabling them to monitor and analyze malicious software uploaded by users to cloud services. Frequent submissions of adversarial samples can trigger enhanced scrutiny or alerts, thereby increasing the risk of detection. Consequently, when targeting real-world detection systems, it is imperative to develop evasion techniques that can effectively circumvent dynamic analysis mechanisms while minimizing the number of queries made to the target model. This approach reduces the likelihood of detection and enhances the stealthiness and success rate of adversarial attacks. To achieve this goal, new research must not only focus on technological advancements but also deepen our understanding of existing security measures.

8.2. Reflections on the Evolution of Malware Detection Techniques

RL-driven adversarial malware generation techniques inherently rely on multiple queries to the target model, introducing subtle perturbations to malicious software to generate adversarial samples that disrupt the decision-making processes of detection models, causing them to misclassify the malware as benign. As illustrated in Table 7, currently,

defensive measures against adversarial attacks can be classified into three categories according to their respective stages of application. Firstly, during the data preprocessing stage, potential adversarial samples can be filtered out by leveraging the distinguishable characteristics between adversarial and benign inputs. Secondly, in the model training stage, the detection model is retrained with a controlled proportion of adversarial examples to improve its robustness against input perturbations. Lastly, at the model detection stage, specific features of adversarial samples are extracted and integrated into tailored classification mechanisms, aiming to enhance both the detection accuracy and classification performance.

Table 7. Defensive measures against adversarial attacks.

Application Stage	Description	Methods
Data preprocessing	Eliminate possible adversarial samples	[83–85]
Model training	Add a certain number of adversarial samples to retrain the model	[86–88]
Model detection	Directly extract the features of the adversarial samples	[89,90]

Through a systematic analysis of the adversarial sample generation mechanisms for PE malware based on reinforcement learning, this paper provides new insights into the development of malware detection techniques.

During the data preprocessing phase, the interactive nature of reinforcement learning agents with their environment can be utilized to identify possible adversarial attacks through the detection of frequent sample submission patterns. A key challenge, however, lies in differentiating benign repetitive queries from malicious probing behaviors. Therefore, the effective modeling of normal user behavior is essential in designing detection systems that reduce false alarms while maintaining high sensitivity to actual threats.

In addition, with the rapid advancement of large language models (LLMs) in code understanding and analysis, they have shown great potential and application prospects in the field of malware detection, offering new insights into the detection of adversarial malware samples. Specifically, during the model detection phase, the natural language processing capabilities of LLMs can be leveraged to efficiently parse dynamic behavior reports generated by sandboxes, assisting security analysts in quickly identifying hidden attack characteristics in adversarial samples, thereby significantly improving the analysis efficiency and accuracy. Moreover, benefiting from their strong ability to understand code semantics, in the data preprocessing phase, LLMs can effectively identify and filter out redundant or irrelevant code segments that are deliberately injected during the generation of adversarial malware samples. This provides high-quality input for subsequent feature extraction and classification tasks, enhancing the discriminative capabilities of detection models. These advantages not only improve the model's ability to distinguish adversarial samples from benign ones but also lay a solid foundation for the construction of more robust and interpretable defense mechanisms.

9. Conclusions

As the application of RL in the generation of adversarial malware samples continues to expand, a systematic overview and timely update of the field have become increasingly essential. Such efforts not only consolidate existing knowledge but also pave the way for future advancements in malware detection technologies.

This paper aims to address the current gap in the literature by providing a comprehensive review of RL-driven adversarial malware generation. We first summarize the

foundational frameworks used in RL-based adversarial malware generation and establish a set of evaluation criteria for the assessment of model performance. Furthermore, we conduct an in-depth analysis of existing approaches from four key perspectives: action space design, state space representation, reward function construction, and the selection of RL frameworks. In addition to technical synthesis, this work also explores the major challenges facing the field of adversarial malware research.

Overall, this study seeks to provide valuable guidance for the advancement of security strategies and reinforcement of cyber defense mechanisms, ultimately contributing to a more secure and resilient digital ecosystem.

Author Contributions: Conceptualization, S.Z.; methodology, Y.T.; investigation, S.Z.; resources, Y.T.; data curation, Y.T.; writing—original draft preparation, Y.T.; writing—review and editing, H.L. and H.M.; supervision, H.L.; project administration, X.Y.; revision, Y.T. and H.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Major Scientific and Technological Research Project of Henan Province in China, funded by the Songshan Laboratory (241110210200).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Cletus, A.; Opoku, A.A.; Weyori, B.A. An evaluation of current malware trends and defense techniques: A scoping review with empirical case studies. *J. Adv. Inf. Technol.* **2024**, *15*, 649–671. [CrossRef]
2. Hamza, A.A.; Abdel-Halim, I.T.; Sobh, M.A.; Bahaa-Eldin, A.M. A survey and taxonomy of program analysis for IoT platforms. *Ain Shams Eng. J.* **2021**, *12*, 3725–3736. [CrossRef]
3. Monnappa, K.A. *Learning Malware Analysis: Explore the Concepts, Tools and the Techniques*; Packt Publishing Ltd.: Birmingham, UK, 2018.
4. CHECK POINT. Available online: https://itwire.com/images/articles/andrew-matler/2024_Security_Report.pdf (accessed on 29 April 2025).
5. Sophos. Available online: https://news.sophos.com/enus/2024/03/12/2024-sophos-threat-report/ (accessed on 29 March 2025).
6. AV-Test. Available online: https://portal.av-atlas.org/malware/statistics (accessed on 15 March 2025).
7. Sathyanarayan, V.S.; Kohli, P.; Bruhadeshwar, B. Signature generation and detection of malware families. In Proceedings of the 13th Australasian Conference, Information Security and Privacy, Wollongong, Australia, 7–9 July 2008.
8. Ucci, D.; Aniello, L.; Baldoni, R. Survey of machine learning techniques for malware analysis. *Comput. Secur.* **2019**, *81*, 123–147. [CrossRef]
9. Anderson, B.; McGrew, D. Machine learning for encrypted malware traffic classification: Accounting for Noisy Labels and Non-Stationarity. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, 13–17 August 2017.
10. Cui, Z.; Xue, F.; Cai, X.; Cao, Y.; Wang, G.; Chen, J. Detection of Malicious Code Variants Based on Deep Learning. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3187–3196. [CrossRef]
11. Botacin, M.; Domingues, F.D.; Ceschin, F.; Machnicki, R.; Alves, Z.A.M.; Geus, P.; Gregio, A. Antiviruses under the microscope: A hands-on perspective. *Comput. Secur.* **2022**, *112*, 102500. [CrossRef]
12. Qiang, W.; Yang, L.; Jin, H. Efficient and robust malware detection based on control flow traces using deep neural networks. *Comput. Secur.* **2022**, *122*, 102871. [CrossRef]
13. Jha, S.; Prashar, D.; Long, H.V.; Taniar, D. Recurrent neural network for detecting malware. *Comput. Secur.* **2020**, *99*, 102037. [CrossRef]
14. SL, S.D.; Jaidhar, C.D. Windows malware detector using convolutional neural network based on visualization images. *IEEE Trans. Emerg. Top. Comput.* **2019**, *9*, 1057–1069.
15. Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; Nicholas, C.K. Malware Detection by Eating a Whole EXE. In Proceedings of the 32nd AAAI National Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.
16. Raff, E.; Fleshman, W.; Zak, R.; Anderson, H.S.; Filar, B.; McLean, M. Classifying sequences of extreme length with constant memory applied to malware detection. In Proceedings of the 35nd AAAI Conference on Artificial Intelligence, Online, 2–9 February 2021.

17. Krcál, M.; Švec, O.; Bálek, M.; Jašek, O. Deep convolutional malware classifiers can learn from raw executables and labels only. In Proceedings of the 6th International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
18. Kalash, M.; Rochan, M.; Mohammed, N.; Bruce, N.; Wang, Y.; Iqbel, F. Malware classification with deep convolutional neural networks. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 26–28 February 2018.
19. Vu, D.L.; Nguyen, T.K.; Nguyen, T.V.; Nguyen, T.N.; Massacci, F.; Phung, P.H. A convolutional transformation network for malware classification. In Proceedings of the 2019 6th NAFOSTED Conference on Information and Computer Science (NICS), Hanoi, Vietnam, 12–13 December 2019.
20. Vasan, D.; Alazab, M.; Wassan, S.; Naeem, H.; Safaei, B.; Zheng, Q. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Comput. Netw.* **2020**, *171*, 107138. [CrossRef]
21. Saxe, J.; Berlin, K. Deep neural network based malware detection using two dimensional binary program features. In Proceedings of the 2015 10th international conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, 20–22 October 2015.
22. Chen, X.; Hao, Z.; Li, L.; Cui, L.; Zhu, Y.; Ding, Z. Cruparamer: Learning on parameter-augmented api sequences for malware detection. *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 788–803. [CrossRef]
23. Maniriho, P.; Mahmood, A.N.; Chowdhury, M.J.M. API-MalDetect: Automated malware detection framework for windows based on API calls and deep learning techniques. *J. Netw. Comput. Appl.* **2023**, *218*, 103704. [CrossRef]
24. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. *arXiv* **2014**, arXiv:1412.6572.
25. Demetrio, L.; Biggio, B.; Lagorio, G.; Roli, F.; Armando, A. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 469–3478. [CrossRef]
26. Kong, Z.; Xue, J.; Wang, Y.; Huang, L.; Niu, Z.; Li, F. A survey on adversarial attack in the age of artificial intelligence. *Wirel. Commun. Mob. Comput.* **2021**, *2021*, 4907754. [CrossRef]
27. Park, D.; Yener, B. A survey on practical adversarial examples for malware classifiers. In Proceedings of the 4th Reversing and Offensive-Oriented Trends Symposium, Vienna, Austria, 19 November 2020.
28. Pierazzi, F.; Pendlebury, F.; Cortellazzi, J.; Cavallaro, L. Intriguing properties of adversarial ml attacks in the problem space. In Proceedings of the 41st IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 18–20 May 2020.
29. Zhan, D.; Duan, Y.; Hu, Y.; Li, W.; Guo, S.; Pan, Z. MalPatch: Evading DNN-based malware detection with adversarial patches. *IEEE Trans. Inf. Forensics Secur.* **2023**, *19*, 1183–1198. [CrossRef]
30. Wang, S.; Fang, Y.; Xu, Y.; Wang, Y. Black-box adversarial windows malware generation via united puppet-based dropper and genetic algorithm. In Proceedings of the 24th IEEE International Conference on High Performance Computing & Communications, Hainan, China, 18–20 December 2022.
31. Gibert, D.; Planes, J.; Le, Q.; Zizzo, G. A wolf in sheep's clothing: Query-free evasion attacks against machine learning-based malware detectors with generative adversarial networks. In Proceedings of the 2023 IEEE European Symposium on Security and Privacy Workshops, Delft, The Netherlands, 3–7 July 2023.
32. Zhong, F.; Cheng, X.; Yu, D.; Gong, B.; Song, S.; Yu, J. MalFox: Camouflaged adversarial malware example generation based on conv-GANs against black-box detectors. *IEEE Trans. Comput.* **2023**, *73*, 980–993. [CrossRef]
33. Hu, W.; Tan, Y. Generating adversarial malware examples for black-box attacks based on GAN. In Proceedings of the 7th International Conference on Data Mining and Big Data, Beijing, China, 11 November 2022.
34. Anderson, H.S.; Kharkar, A.; Filar, B.; Evans, D.; Roth, P. Learning to evade static PE machine learning malware models via reinforcement learning. *arXiv* **2018**, arXiv:1801.08917.
35. Fang, Z.; Wang, J.; Li, B.; Wu, S.; Zhou, Y.; Huang, H. Evading anti-malware engines with deep reinforcement learning. *IEEE Access* **2019**, *7*, 48867–48879. [CrossRef]
36. Fang, Y.; Zeng, Y.; Li, B.; Liu, L.; Zhang, L. DeepDetectNet vs. RLAttackNet: An adversarial method to improve deep learning-based static malware detection model. *PLoS ONE* **2020**, *15*, e0231626. [CrossRef]
37. Geng, J.; Wang, J.; Fang, Z.; Zhou, Y.; Wu, D.; Ge, W. A survey of strategy-driven evasion methods for PE malware: Transformation, concealment, and attack. *Comput. Secur.* **2024**, *137*, 103595. [CrossRef]
38. Ling, X.; Wu, L.; Zhang, J.; Qu, Z.; Deng, W.; Chen, X.; Qiao, Y.; Wu, C.; Ji, S.; Luo, T. Adversarial attacks against Windows PE malware detection: A survey of the state-of-the-art. *Comput. Secur.* **2023**, *128*, 103134. [CrossRef]
39. Afianian, A.; Niksefat, S.; Sadeghiyan, B.; Baptiste, D. Malware dynamic analysis evasion techniques: A survey. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–28. [CrossRef]
40. Bulazel, A.; Yener, B. A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In Proceedings of the 1st Reversing and Offensive-Oriented Trends Symposium, Vienna, Austria, 16 November 2017.
41. Jadhav, A.; Vidyarthi, D. Evolution of evasive malwares: A survey. In Proceedings of the 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), New Delhi, India, 11–13 March 2016.

42. Mahmood, A.N.; Chowdhury, M.J.M.; Maniriho, P. A survey of recent advances in deep learning models for detecting malware in desktop and mobile platforms. *ACM Comput. Surv.* **2024**, *56*, 145.
43. Wu, C.; Shi, J.; Yang, Y.; Li, W. Enhancing machine learning based malware detection model by reinforcement learning. In Proceedings of the 8th International Conference on Communication and Network Security, Qingdao, China, 2–4 November 2018.
44. Anderson, H.S.; Roth, P. Ember: An open dataset for training static pe malware machine learning models. *arXiv* **2018**, arXiv:1804.04637.
45. Chen, J.; Jiang, J.; Li, R.; Dou, Y. Generating adversarial examples for static PE malware detector based on deep reinforcement learning. *J. Phys. Conf. Ser.* **2020**, *1575*, 012011. [CrossRef]
46. Ebrahimi, M.; Pacheco, J.; Li, W.; Hu, J.L.; Chen, H. Binary black-box attacks against static malware detectors with reinforcement learning in discrete action spaces. In Proceedings of the 2021 IEEE security and privacy workshops (SPW), San Francisco, CA, USA, 27 May 2021.
47. Labaca-Castro, R.; Franz, S.; Rodosek, G.D. AIMED-RL: Exploring adversarial malware examples with reinforcement learning. In Proceedings of the Machine Learning and Knowledge Discovery in Databases, Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, 13–17 September 2021.
48. Li, X.; Li, Q. An IRL-based malware adversarial generation method to evade anti-malware engines. *Comput. Secur.* **2021**, *104*, 102118. [CrossRef]
49. Gibert, D.; Fredrikson, M.; Mateu, C.; Plances, J.; Le, Q. Enhancing the insertion of NOP instructions to obfuscate malware via deep reinforcement learning. *Comput. Secur.* **2022**, *113*, 102543. [CrossRef]
50. Song, W.; Li, X.; Afroz, S.; Grag, D.; Kuznetsov, D.; Yin, H. MAB-Malware: A reinforcement learning framework for blackbox generation of adversarial malware. In Proceedings of the 2022 ACM on Asia conference on computer and communications security, Nagasaki, Japan, 30 May 2022.
51. Quertier, T.; Marais, B.; Morucci, S.; Fournel, B. MERLIN-Malware Evasion with Reinforcement LearnINg. *arXiv* **2022**, arXiv:2203.12980.
52. Zhong, F.; Hu, P.; Zhang, G.; Li, H.; Cheng, X. Reinforcement learning based adversarial malware example generation against black-box detectors. *Comput. Secur.* **2022**, *121*, 102869. [CrossRef]
53. Virus Total. Available online: https://virustotal.com/ (accessed on 29 May 2025).
54. Zhang, L.; Liu, P.; Choi, Y.H.; Chen, P. Semantics-preserving reinforcement learning attack against graph neural networks for malware detection. *IEEE Trans. Dependable Secur. Comput.* **2022**, *20*, 1390–1402. [CrossRef]
55. Zhan, D.; Bai, W.; Liu, X.; Hu, Y.; Zhang, L.; Guo, S.; Pan, Z. PSP-Mal: Evading malware detection via prioritized experience-based reinforcement learning with Shapley prior. In Proceedings of the 39th Annual Computer Security Applications Conference, Austin, TX, USA, 4–8 December 2023.
56. Etter, B.; Hu, J.L.; Ebrahimi, M.; Li, W.; Li, X.; Chen, H. Evading Deep Learning-Based Malware Detectors via Obfuscation: A Deep Reinforcement Learning Approach. In Proceedings of the 2023 IEEE International Conference on Data Mining (ICDM), Shanghai, China, 1–4 December 2023.
57. Zhan, D.; Zhang, Y.; Zhu, L.; Chen, J.; Xia, S.; Guo, S.; Pan, Z. Enhancing reinforcement learning based adversarial malware generation to evade static detection. *Alex. Eng. J.* **2024**, *98*, 32–43. [CrossRef]
58. Coull, S.E.; Gardner, C. Activation analysis of a byte-based deep neural network for malware classification. In Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 23 May 2019.
59. Tian, B.; Jiang, J.; He, Z.; Yuan, X.; Dong, L.; Sun, C. Functionality-Verification Attack Framework Based on Reinforcement Learning Against Static Malware Detectors. *IEEE Trans. Inf. Forensics Secur.* **2024**, *19*, 8500–8514. [CrossRef]
60. Zhan, D.; Liu, X.; Bai, W.; Li, W.; Guo, S.; Pan, Z. GAME-RL: Generating Adversarial Malware Examples against AP I Call Based Detection via Reinforcement Learning. *IEEE Trans. Dependable Secur. Comput. (Early Access)* **2025**, 1–17. [CrossRef]
61. Microsoft, Inc. Available online: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format (accessed on 9 June 2025).
62. Paterson, T. An inside look at MS-DOS. *Byte* **1983**, *8*, 230.
63. Szegedy, C.; Sutskever, I.; Goodfellow, I.; Zaremba, W.; Fergus, R.; Erhan, D. Intriguing properties of neural networks. In Proceedings of the 2nd International Conference on Learning Representations, Banff, AB, Canada, 14–16 April 2014.
64. Aryal, K.; Gupta, M.; Abdelsalam, M.; Saleh, M. Intra-section code cave injection for adversarial evasion attacks on windows pe malware file. *arXiv* **2024**, arXiv:2403.06428.
65. Demetrio, L.; Biggio, B.; Lagorio, G.; Roli, F.; Armando, A. Explaining vulnerabilities of deep learning to adversarial malware binaries. *arXiv* **2019**, arXiv:1901.03583.
66. Demetrio, L.; Coull, S.E.; Biggio, B.; Lagorio, G.; Armando, A.; Roli, F. Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM Trans. Priv. Secur. (TOPS)* **2021**, *24*, 27. [CrossRef]

67. Badhwar, R. Polymorphic and metamorphic malware. In *The CISO's Next Frontier: AI, Post-Quantum Cryptography and Advanced Security Paradigms*; Springer International Publishing: Berlin/Heidelberg, Germany, 2021.
68. Simple Malware Obfuscation Techniques. Available online: https://resources.infosecinstitute.com/topic/simple-malware-obfuscation-techniques/ (accessed on 29 March 2025).
69. Upx. Available online: https://github.com/upx/upx (accessed on 29 May 2025).
70. Darkarmour Computer Software. Available online: https://github.com/bats3c/darkarmour (accessed on 29 March 2025).
71. Lief. Available online: https://github.com/lief-project/LIEF (accessed on 29 May 2025).
72. pefile. Available online: https://github.com/erocarrera/pefile (accessed on 29 March 2025).
73. Pathak, D.; Agrawal, P.; Efros, A.A.; Darrell, T. Curiosity-driven exploration by self-supervised prediction. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 11 August 2017.
74. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, Phoenix, AZ, USA, 12–17 February 2016.
75. Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.V.; Lanctot, M.; Freitas, N.D. Dueling network architectures for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016.
76. Bellemare, M.G.; Dabney, W.; Munos, R. A distributional perspective on reinforcement learning. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017.
77. Fellows, M.; Mahajan, A.; Rudner, T.G.J.; Whiteson, S. Virel: A variational inference framework for reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 1–15.
78. Christodoulou, P. Soft actor-critic for discrete action settings. *arXiv* **2019**, arXiv:1910.07207.
79. IDA Pro. Available online: https://www.hex-rays.com/products/ida/ (accessed on 29 March 2025).
80. Guarnieri, C. cuckoosandbox. Available online: https://cuckoosandbox.org/ (accessed on 29 March 2025).
81. Angr. Available online: https://angr.io/ (accessed on 29 March 2025).
82. Ding, Y.; Shao, M.; Nie, C.; Fu, K. An efficient method for generating adversarial malware samples. *Electronics* **2022**, *11*, 154. [CrossRef]
83. Chen, S.; Carlini, N.; Wagner, D. Stateful detection of black-box adversarial attacks. In Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence, Taiwan, China, 6 October 2020.
84. Albulayhi, K.; Sheldon, F.T. An adaptive deep-ensemble anomaly-based intrusion detection system for the internet of things. In Proceedings of the 2021 IEEE world AI IoT congress (AIIoT), Seattle, WA, USA, 13 May 2021.
85. Chen, B.; Ren, Z.; Yu, C.; Hussain, I.; Liu, J. Adversarial examples for cnn-based malware detectors. *IEEE Access* **2019**, *7*, 54360–54371. [CrossRef]
86. Shaukat, K.; Luo, S.; Varadharajan, V. A novel method for improving the robustness of deep learning-based malware detectors against adversarial attacks. *Eng. Appl. Artif. Intell.* **2022**, *116*, 105461. [CrossRef]
87. Zhang, Y.; Li, H.; Zheng, Y.; Yao, S.; Jiang, J. Enhanced DNNs for malware classification with GAN-based adversarial training. *J. Comput. Virol. Hacking Tech.* **2021**, *17*, 153–163. [CrossRef]
88. Wang, X.; Miikkulainen, R. MDEA: Malware detection with evolutionary adversarial learning. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19 July 2020.
89. Albulayhi, K.; Abu Al-Haija, Q.; Alsuhibany, S.A.; Jillepalli, A.A.; Ashrafuzzaman, M.; Sheldon, F.T. IoT intrusion detection using machine learning with a novel high performing feature selection method. *Appl. Sci.* **2022**, *12*, 5015. [CrossRef]
90. Mishra, A.; Gupta, B.B.; Perakovic, D.; Yamaguchi, S.; Hsu, C.H. Entropy based defensive Mechanism against DDoS attack in SDN-Cloud enabled online social networks. In Proceedings of the 2021 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 10 January 2021.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.