Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2023.046658

ARTICLE

An Empirical Study on the Effectiveness of Adversarial Examples in Malware Detection

Younghoon Ban, Myeonghyun Kim and Haehyun Cho*

School of Software, Soongsil University, Seoul, 06978, Korea

*Corresponding Author: Haehyun Cho. Email: haehyun@ssu.ac.kr

Received: 10 October 2023 Accepted: 07 December 2023 Published: 11 March 2024

ABSTRACT

Antivirus vendors and the research community employ Machine Learning (ML) or Deep Learning (DL)-based static analysis techniques for efficient identification of new threats, given the continual emergence of novel malware variants. On the other hand, numerous researchers have reported that Adversarial Examples (AEs), generated by manipulating previously detected malware, can successfully evade ML/DL-based classifiers. Commercial antivirus systems, in particular, have been identified as vulnerable to such AEs. This paper firstly focuses on conducting black-box attacks to circumvent ML/DL-based malware classifiers. Our attack method utilizes seven different perturbations, including Overlay Append, Section Append, and Break Checksum, capitalizing on the ambiguities present in the PE format, as previously employed in evasion attack research. By directly applying the perturbation techniques to PE binaries, our attack method eliminates the need to grapple with the problem-feature space dilemma, a persistent challenge in many evasion attack studies. Being a black-box attack, our method can generate AEs that successfully evade both DL-based and ML-based classifiers. Also, AEs generated by the attack method retain their executability and malicious behavior, eliminating the need for functionality verification. Through thorogh evaluations, we confirmed that the attack method achieves an evasion rate of 65.6% against well-known ML-based malware detectors and can reach a remarkable 99% evasion rate against well-known DL-based malware detectors. Furthermore, our AEs demonstrated the capability to bypass detection by 17% of vendors out of the 64 on VirusTotal (VT). In addition, we propose a defensive approach that utilizes Trend Locality Sensitive Hashing (TLSH) to construct a similarity-based defense model. Through several experiments on the approach, we verified that our defense model can effectively counter AEs generated by the perturbation techniques. In conclusion, our defense model alleviates the limitation of the most promising defense method, adversarial training, which is only effective against the AEs that are included in the training classifiers.

KEYWORDS

Malware classification; machine learning; adversarial examples; evasion attack; cybersecurity

1 Introduction

Machine Learning (ML) and Deep Learning (DL) based models, a subset of artificial intelligence, have demonstrated exceptional performance in various domains, including recommendation systems, handling imbalanced datasets, bioinformatics, medical diagnosis, financial risk management, and

stock exchange [1]. Moreover, ML and DL models play a crucial role in cyber security systems, particularly in fraud detection, intrusion detection, spam detection, and malware detection [2–6]. ML and DL-based static analysis approaches offer significant efficiency and cost advantages when compared to traditional static analysis methods. Performing a thorough and meticulous static analysis of the ever-increasing array of emerging malware strains is a resource-intensive and time-consuming undertaking. Also, with the continuous emergence of new forms of malware, we are facing strong challenges in promptly and efficiently identifying these new threats [7–9]. Therefore, the research community is increasingly focusing on developing ML and DL-based static malware classifiers. We have been observing ML-based malware detectors offer notable advantages, primarily their high scalability and their ability to rapidly and effectively detect large volumes of malware [10,11]. Furthermore, ML-based static analysis is more cost-effective than dynamic analysis, many anti-virus vendors use static analysis to deal with a ton of malware emerging everyday [12]. In addition, security researchers believe that recently proposed ML-based approaches are able to detect zero-day malware with the high accuracy [10,11,13,14]. On the other hand, unfortunately, many researchers have reported that the ML/DL-based classifiers can be bypassed if we falsify malware that had been previously detected [10,12–15]. Cylance, a commercial antivirus system, has been identified as vulnerable to Adversarial Examples (AEs), which are created by subtly altering previously known malware [11].

In general, an AE is a specially crafted input generated to deceive ML or DL models. AEs are primarily designed for evasion attacks on models that process images, sound, and Portable Executable (PE) files. For example, in the case of an image-based model, AEs are created by perturbing each pixel of the image. By adding imperceptible noise to an image of a dog, an Image-based AE can be generated in a way that the model incorrectly classifies it as a different label [16,17]. Adversarial attack approaches can be categorized into three distinct levels, depending on the attacker's knowledge about the target model [16,18].

White-box attack: The attacker has access to the target model, knowing its structure, weights, parameters, output, features, etc., and is aware of the dataset used to train the target model.

Gray-box attack: The attacker possesses only a fraction of the knowledge about the target model compared to a white-box attack.

Black-box attack: The attacker sends a query to the target model and only has access to the query's response, representing the best-case scenario for real-world situations.

Under the white-box attack scenario, we exploit the ambiguity in the specifications of the Windows PE file format to inject adversarial noise or optimize the injected noise, creating AEs capable of misclassifying malware as benign. In the black-box attack on PE binaries, AEs are generated by applying various perturbation techniques to PE malware, rewriting the PE malware itself, or utilizing byte sequences or strings from benign binaries [10,19]. Previously proposed evasion attacks such as RAMEN [10], GAMMA [12], and Multi-Armed Bandit (MAB)-malware [11] used benign contents under the black-box scenario. Specifically, RAMEN [10] and GAMMA [12] achieved high evasion rates by adding strings, bytes, or Application Programming Interfaces (APIs) extracted from benign binaries. The use of benign content enhances computational efficiency by avoiding the generation of random byte patterns injected into the original PE binary. Additionally, it plays a crucial role in altering the byte entropy of the original malware in a specific direction, making it highly effective for attacking machine learning models. These findings demonstrate the effectiveness of using benign content in generating AEs with high evasion rates [10–12,20]. Most black-box evasion attacks rely on the effective concept of the transferability [17,18,21,22]. The transferability denotes that AE created to evade a specific model can also successfully evade models with different architectures [18,23].

On the other hand, a line of research on defense against evasion attacks has proposed various methods such as adversarial training [24], defensive distillation [25], and feature selection [26] to counter different types of evasion attacks [17]. Among them, adversarial training stands out as the most effective and promising approach to defend against AE [27]. The approach trains the model with AEs during the training process to make the model capable of classifying AE [15,24,27,28]. However, because adversarial training can only defend against the specific adversarial noise or perturbation techniques applied in the AEs used in the training process, it becomes challenging to defend against unknown AEs. Moreover, adversarial training may lead to significant changes in the performance of the existing classifier, potentially causing a degradation in model performance [27].

In this paper, we perform an evasion attack by generating AEs using various perturbations and benign content under the black-box scenario that reflects real-world environments, targeting well-known static analysis-based ML/DL malware classifiers. We use the attack method while preserving the PE format of the original malware, ensuring the executability of PE malware and preserving malicious behavior. Since our attack method directly applies perturbations and benign content to the malware, we do not need to consider the problem-feature space dilemma. Additionally, to verify the transferability of AEs generated for evading DL-based target models, we analyze the results of ML-based models with varying features in AE detection. Furthermore, we validate the transferability of AEs generated through our attack technique by employing models with different structures and features, as well as VirusTotal. Finally, to defend against target models vulnerable to evasion attacks, we construct a defense model by applying the fuzzy hash, Trend Locality Sensitive Hashing (TLSH), known for its capability to find similar files. We then execute our attack technique against the constructed defense model, measure the evasion rate of AEs generated by us, and evaluate the performance of the defense model. The contributions of this paper are as follows:

- We conduct adversarial attacks employing a total of seven PE format-preserving perturbation techniques, including overlay append, a technique utilized in various evasion attack studies. Our attack method directly applies perturbations to the PE binary, ensuring the integrity of the PE format of malware. Consequently, there is no need to validate the malicious behavior of AEs generated with applied perturbations.
- To assess the transferability of the AE we generated, we conducted experiments using AE produced from each target model. The results revealed that 99% of the AEs crafted to evade ML-based malware detectors were also successful in bypassing DL-based malware detectors, showcasing the capability of our attack method to create powerful transferability AE. Furthermore, when querying our generated AEs on VirusTotal (VT), we managed to evade detection by 17% of the 64 detection vendors participating in VT.
- We present a defense method utilizing TLSH, a robust similarity hashing technique recognized for its capacity to identify similar files with low false positives and high detection rates. Our proposed defense method effectively mitigates AE generation by our attack method, successfully defending against over 90% of the generated AEs.

2 Background

In this section, we discuss the limitations of research on evading ML-based malware detectors and explain the studies aimed at defending against evasion attacks.

Threats to Validity: In this paper, we have included studies that deal with deep learning-based malware classifiers or machine learning-based malware detection techniques and evasion attack by using combinations of search strings such as "deep learning," "PE malware," "adversarial examples," and "adversarial attack defense." We also explore attack methods for generating AEs which preserves the PE format and the malicious behavior of malware. Additionally, we discuss a novel defense strategy.

2.1 Learning-Based Malware Detectors

ML and DL models automatically learn complex relationships among file attributes from an extensive training dataset [14]. Research in PE malware detection leverages the learning capabilities of ML/DL models, renowned for their ability to generalize predictions for new instances, even for previously unobserved forms of malware, such as zero-day malware [16]. Many studies in malware detection employing ML/DL models typically encompass three stages: data collection, feature engineering, and model training/predictions [13,14,16].

ML/DL models operate on numerical inputs, necessitating a feature extraction process to convert various features of PE malware into numerical values. Static features of PE malware are extracted without executing the malware, involving the extraction of API calls, byte sequences, readable strings, and header information contained within the malware. Additionally, dynamic features involve executing the PE malware to extract API call sequences, system resource status, and file status during runtime. After extracting static or dynamic features from malware, these features are mapped into numeric values to serve as inputs to ML/DL models. The feature engineering process fundamentally represents the mapping of the problem space (original PE malware without feature extraction) to the feature space (numeric features). This mapping function, expressed mathematically as Eq. (1), can be defined as follows:

$$\phi : \mathbb{Z} \rightarrow \mathbb{X} \tag{1}$$

Here $\mathbb{Z}$ represents the problem space of the original PE malware before feature extraction, and $\mathbb{X}$ corresponds to the associated feature space. $\mathbb{X}$ numerically represents the intrinsic properties of the object (malware) from $\mathbb{Z}$.

The "model train/predictions" phase refers to the process of training a model using numeric features extracted from malware, following feature extraction. The goal is to create a discrimination function f with parameters that can classify malware or benignware. Here $\mathbb{X}$ represents the numeric features of malware generated after the feature engineering process. $\mathbb{Y}$ represents the label space for $\mathbb{X}$, and f denotes a DL/ML model with parameters. This process can be expressed in Eq. (2) as shown below:

$$f : \mathbb{X} \rightarrow \mathbb{Y} \tag{2}$$

ML/DL models have demonstrated the high generalization performance. These models were expected to be robust even to small perturbations. However, Szegedy et al. [21] discovered that imperceptible non-random perturbations applied to input images in object recognition tasks could arbitrarily alter the model's predictions. This finding revealed the vulnerability of ML/DL models to AE attacks, as widely discussed in numerous studies [16,29]. AE attacks were originally explored in image classification tasks, and the PE file and image domains exhibit distinct characteristics, giving rise to the problem-feature space dilemma [30]. Similar to Ling et al. [16], Fig. 1 illustrates the dilemma in the problem-feature space, specifically addressing the Inverse Feature Mapping Function that connects the problem space and feature space for PE files. Nevertheless, AE attacks have been introduced into

the cybersecurity domain, and researchers are proposing AE attacks in various security domains such as malware classification [10–12,31].

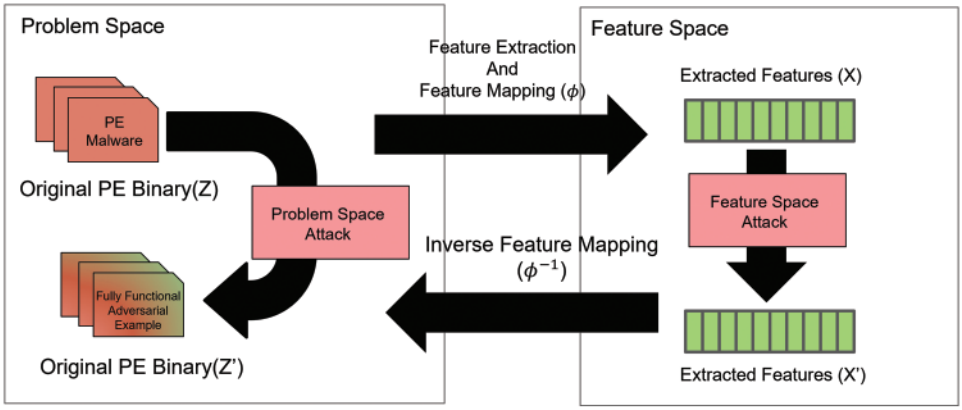

Figure 1: This figure illustrates the relationship between the problem space and feature space through a feature mapping function. The Feature mapping function and Inverse feature mapping function serve to connect the problem space and feature space. However, obtaining the Inverse feature mapping function in the context of PE binary proves challenging, rendering it unsuitable for application in evasion attacks

Attackers employ various methods to generate AEs to evade ML/DL-based malware classifiers. For instance, in the white-box attack (feature-space attack), the continuous nature of feature representation in the feature space is typically leveraged to create AEs using the gradient information of ML/DL malware classifiers [10,12]. Fig. 2 demonstrates the performance of an evasion attack in the feature space to generate PE file AE using white-box attacks, while showing the impossibility of evasion in the problem space. In the end, gradient-based attacks in the feature space reveal their impracticality in real-world PE malware detection. In the black-box attack (problem-space attack), where information about the ML/DL malware classifier is unavailable, attacks are conducted in the problem space. This attack involves generating AEs by modifying the original malware or creating a surrogate model with performance similar to the black-box model and then using methods from the white-box attack [11,22,31]. Therefore, our attack method performs a problem-space attack on PE malware, preserving the PE format, and applies perturbations to create AEs that evade ML/DL models.

2.2 Perturbing Malware to Evade Classifiers

Black-box attacks are particularly well-suited for real-world scenarios where the attacker lacks information about the target model and can only make queries. Many black-box attacks capitalize on the ambiguities within the PE format to execute problem-space attacks. Leveraging these ambiguities provides the advantage of generating AE without impacting the original binary's behavior, as emphasized in [10]. Exploiting this advantage enables us to maintain the PE format of the original PE malware, ensuring the executability and malicious behavior of the generated AE, thereby eliminating the need for functionality verification.

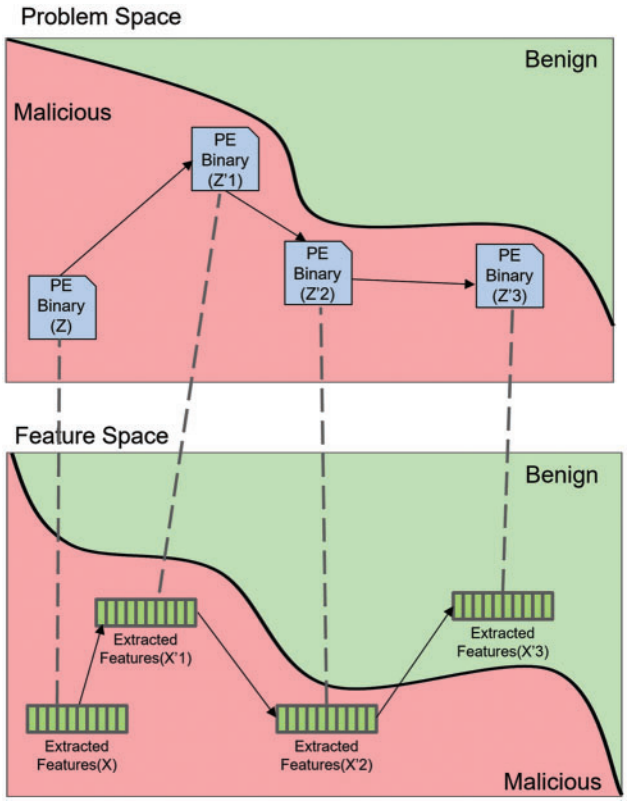

Figure 2: This figure illustrates the distinction between problem space attacks and feature space attacks. The original PE malware, represented as Z, undergoes continuous modifications in the problem space, resulting in the generation of AE denoted as Z'1, Z'2, Z'3. The illustration demonstrates how these AEs are mapped to AE Features, specifically X'1, X'2, X'3, in the feature space. In the feature space, X'3 is misclassified as benign, whereas it is evident that the corresponding Z'3 in the problem space has not been evaded

We introduce perturbations that exploit the ambiguities within the PE format and incorporate them into our attack method. These perturbations, leveraging the uncertainties within the PE format, include *Overlay Append, Section Append, Break Checksum, Section Rename, Section Add, Remove Debug, Remove Certificate, Perturb Header Fields, Filling Slack Space, Padding, Manipulating DOS Header and Stub Extend the DOS Header, Content Shifting, Import Function Injection, Section Injection* [10–12,31]. As mentioned earlier, these perturbations have been employed in previous research and offer advantages due to their capability to preserve the executability and behavior of the original PE malware by leveraging the ambiguities within the PE format. We selectively incorporate some of these perturbations from this set into our attack method.

2.3 Preventing AEs of Malware

As a line of research has been conducted on evasion attacks, defense research has also been active. Methods to defend against evasion attacks include feature selection [26], defensive distillation [25], and adversarial training [15,24,27]. Feature selection originally aims to reduce the classifier's space-time complexity and enhance its generalization capability by choosing a subset of relevant features for

classification and detection. Zhang et al. [26] proposed a feature selection model that harnesses the advantages of feature selection in real-time processing tasks without compromising security.

Defensive distillation was initially introduced to reduce computational complexity by transferring knowledge from a larger model to a smaller one. Distillation involves two models: the teacher and the student. The teacher model undergoes conventional training, while the student model is trained using the probabilities (soft class labels) learned from the teacher model. Moreover, distillation is controlled by the distillation temperature, T. When T is large, it causes the teacher model to assign higher probabilities to each class. Papernot et al. [25] developed a model that exhibits robustness to evasion attacks through distillation. However, it was observed that the distillation-based approach wasn't particularly beneficial in the field of malware security, and the model's accuracy declined as the distillation temperature increased [17].

Although adversarial training enhances robustness against various perturbations depending on the AE used, it comes with limitations, including the requirement for a separate AE dataset for training, challenges in generalizing to new AEs even after retraining, and the overhead associated with model retraining [32]. Furthermore, the most promising defenses, knowledge distillation, and adversarial training, were not found to be effective in defense [17]. Therefore, in this paper, we undertake a defense against AEs with various perturbations to minimize the loss of malware detection accuracy and mitigate the limitations of adversarial training. We implement a similarity-based defense method that compares the original malware and the AE's similarity using TLSH [33], a locality-sensitive hash with excellent detection capabilities for identifying similar files.

3 Overview

Researcher's endeavors focused on Evasion Attacks often generate AEs by injecting noise into the original malware within a white-box attack, employing Genetic Algorithms (GAs) for optimization, thus introducing minimal perturbation [10,12]. However, GAs encounter limitations when applied to target models that return hard labels, as they heavily rely on fitness functions during the optimization process. Furthermore, as depicted in Fig. 2, a white-box attack may be feasible in the feature space but may not succeed in evading the target model in the actual problem space. In the real world, attackers conducting evasion attacks typically find themselves in a state of uncertainty, lacking knowledge regarding the parameters, structure, and features employed by the target model. Therefore, this paper performs evasion attacks, a type of black-box attack. Our attack method involves incorporating benign content and perturbations into the original malware, ensuring that the original malware's PE format remains unchanged while guaranteeing executability and malicious behavior. Furthermore, our attack method eliminates the need for functionality verification when generating AEs.

We have also proposed a defense method to protect models vulnerable to evasion attacks. Our proposed defense method assesses similarity values through the computation of a fuzzy hash value called TLSH [33]. TLSH, a high-performance similarity hash, offers outstanding detection capabilities for identifying similar byte streams or files while minimizing false positives associated with malware detection [34]. As our proposed defense method relies on measuring similarity values, we can mitigate the limitations of adversarial training, which can only defend against limited perturbations within the dataset, and address the issue of accuracy degradation found in defensive distillation. To validate our defense method, we apply our attack method, using various types of perturbations, to our defense model and detect AE generated by our attack method as a test of the defense model's effectiveness. Fig. 3 provides an overview of both our attack and defense methods.

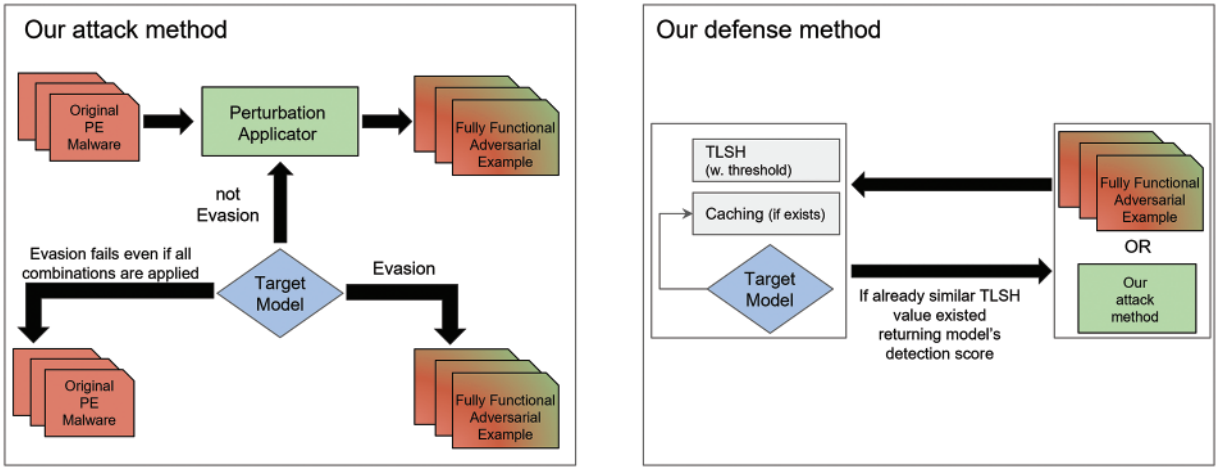

Figure 3: Our attack and defense method overview

3.1 Target Model

Many antivirus vendors claim to detect malware threats through static analysis without executing binaries [12]. Therefore, in our quest to select well-known ML/DL-based malware classifiers as target models, we explored malware classifiers that use static features to detect malware threats without the need to run the malware. As a result, we chose to utilize two target models: Malconv [13], a DL-based malware classifier, and Gradient Boosting Decision Tree (GBDT) [14], an ML-based malware classifier. Malconv uses raw bytes as features and leverages convolutional layers to learn these features, subsequently classifying input binaries as malicious or benign. GBDT, on the other hand, learns manually engineered features to classify input binaries into malicious or benign categories. Both of our selected target models are implementations from the Machine Learning Static Evasion Competition (MLSEC) 2019 [35].

3.1.1 Deep Learning Model

The malware classifier model proposed by Raff et al. [13] employs the raw bytes of the PE binary as features to simplify the malware detection tool and enhance its accuracy. The rationale behind using raw inputs is their demonstrated exceptional performance across various domains, including images, signals, and text. Malconv takes the first 2 MB of raw bytes from a PE binary as input, processes them through an embedding layer, two 1D convolution layers, one dense layer, and an output layer to calculate the probability that the given PE binary is malware. If the raw bytes of the input binary exceed 2 MB in length, they are truncated to fit this size; if they are less than 2 MB, they are padded with zeros. The raw byte input is divided into 500-byte segments without overlapping and passed through a total of 128 convolution filters. Furthermore, Malconv, composed of convolution layers, exhibits an O(n) time complexity. Consequently, Malconv's time complexity increases linearly depending on the input size used [36].

3.1.2 Machine Learning Model

Anderson et al. [14] utilized the GBDT malware classifier, trained on the Elastic Malware Benchmark for Empowering Researchers (EMBER) dataset, which comprises various static features extracted from 1.1 million PE malware samples through static analysis. GBDT incorporates diverse

static features from the PE binary, such as general file information (e.g., the number of imported and exported functions, and the virtual size of the file), header information (including executable characteristics like architecture and version), byte histograms (indicating the occurrences of each byte divided by the total number of bytes), information extracted from strings, and various hand-engineered features, including section information. These features are trained to classify a PE binary input into either malware or benign. Significantly, the GBDT model exhibited superior performance compared to Malconv [14]. Many evasion attack works have also targeted Malconv and GBDT as vulnerable models, and their experiments illustrate the susceptibility of these models to evasion attacks [10–12,27]. In this paper, we introduce a novel defense approach designed to address the vulnerability of the two target models to evasion attacks.

4 For Bypassing Malware Classifiers: Attack Method

Our attack method aims to generate an AE that maintains the original malware's executability and malicious behavior, eliminating the necessity for functional verification. This preservation is achieved without compromising the original malware's PE format. We meticulously choose and apply perturbations that do not interfere with the PE format of the original malware. For instance, we utilize perturbations that can impact the static features of the original malware, such as file hash, section hash, section count/name/padding, checksum, and byte entropy.

- Overlay Append (OA): In OA, benign content is appended to the end of the original malware binary. Consequently, the AE generated with OA exhibits newly added bytes at the end of the binary, distinguishing it from the original malware binary. As a result, the AE created with OA and the original malware binary have distinct hash values and byte entropy.
- Section Append (SP): SP performs appending random bytes or benign content to the unused space between two sections of the original malware binary. This perturbation influences the hash value of the original malware binary, the section hash, and also impacts the section padding.
- Break Checksum (BC): BC replaces the checksum value of the optional header in the original malware binary with zero. When comparing the AE generated with BC and the original malware binary, differences arise in the file hash value and the checksum of the original malware binary. Since the checksum value of the optional header is unique in the original malware binary, this perturbation should be applied only once to the original malware binary.
- Section Rename (SR): SR utilizes the section name from a benign binary to change the section name of the original malware binary. As a result, the AE generated through this perturbation differs from the original malware binary in terms of file hash value and section name.
- Section Add (SA): SA performs adding content extracted from a benign binary to the original malware binary. It is one of the perturbations designed to obscure the distinction between the static features of the original malware binary and those of the benign binary given as input. Notably, the file hash value, section count, and byte entropy of the SA applied AE are distinct from those of the original malware binary.
- Remove Debug (RD): RD modifies the debug information of the original malware binary to zero. This leads to disparities in the file hash value, section hash, and debug information between the AE generated with RD and the original malware binary. Furthermore, given that the debug information in the original malware binary is unique, RD should be executed only once on the original malware binary provided as input.

- Remove Certificate (RC): RC sets the signed certificate of the original malware binary to zero. The AE subjected to RC exhibits distinct file hash value and certificate compared to the original malware binary. Additionally, since the original malware binary possesses a unique signed certificate, RC should be applied only once.

Fig. 4 illustrates the flow of our attack method. We employ a Perturbation Applicator and a Perturbation Combination Dictionary to generate an AE when given the original malware as input.

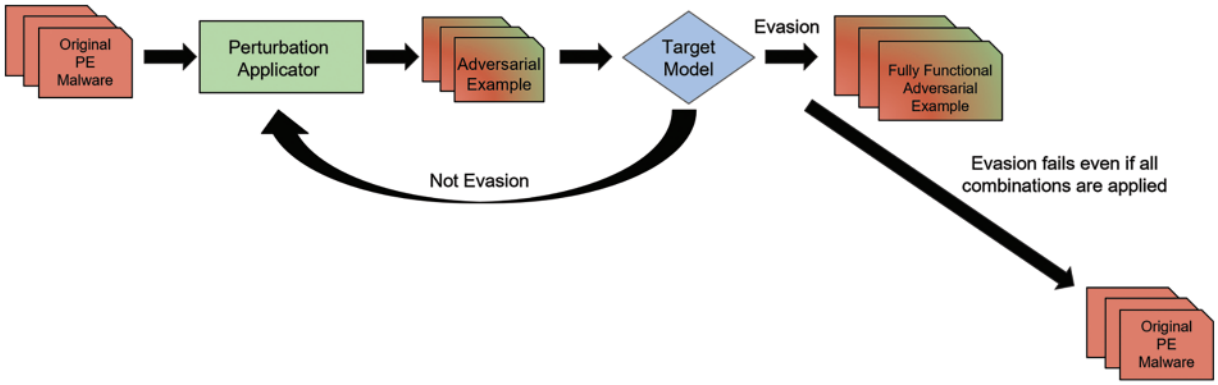

Figure 4: Our attack method flow

Perturbation Combination Dictionary. The perturbation combination dictionary utilized in our attack method is a repository that includes perturbation combinations along with corresponding sets of benign content, crucial for generating AEs. To construct this dictionary, we implemented specific modifications to the MAB-malware framework [11], a research initiative dedicated to generating Reinforcement Learning-based AEs. The following adjustments were implemented: the exclusion of code randomization perturbation, and for every AE generation, we updated the log file to document the perturbations and benign content used, along with the detection score of identified AEs. We conducted a thorough analysis of the AEs generated using the modified MAB-malware. This analysis yielded a total of 998 pairs, each comprising perturbation combinations and benign content applied during AE generation. We categorized the perturbation combinations into various lengths, ranging from 1 to 10, and selected the top 10 perturbation combinations for each length. As shown in Fig. 5, a single perturbation can be paired with various benign contents. In the perturbation combination dictionary, we have stored 21 distinct perturbation combinations, each associated with 206 unique benign contents, all of which are utilized in the AE generation process.

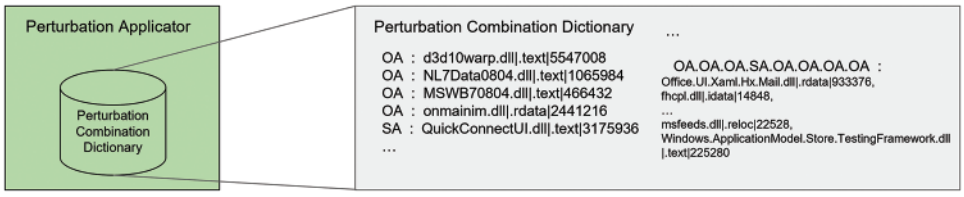

Figure 5: Perturbation applicator and perturbation combination dictionary are used in our attack method

Perturbation Applicator. The perturbation applicator generates AEs by applying perturbation combinations and benign content to the original malware. The perturbation combinations are applied

sequentially, starting from length 1. The perturbation applicator randomly selects benign content corresponding to the currently chosen perturbation combination from the perturbation combination dictionary. Subsequently, the perturbation applicator applies the selected perturbation combination and its corresponding benign content to the original malware. This process ensures the preservation of the PE format, executability, and malicious behavior, aligning with our objectives and resulting in a AE that does not require functional verification. Subsequently, the detection results of the AE, as identified by the target model, are examined if the target model classifies the generated AE as benign, it indicates a successful evasion of the target model by our created AE In such cases, the perturbation applicator discontinues the application of the remaining perturbation combinations to the generated AE and prepares a new malware sample for the creation of a new AE. Conversely, if the generated AE is unsuccessful in evading the target model, the remaining perturbation combinations that have not been applied yet are sequentially used to create a new AE. The evasion capability of the new AE is then assessed. If the perturbation applied to the generated AE is the last one in the currently selected perturbation combination, the input malware is initialized to its initial state without perturbation. Then, a new perturbation combination and its corresponding benign content are prepared to create a fresh AE. This iterative process continues until all 21 perturbation combinations have been applied to the input malware or all prepared malware samples have been exhausted.

4.1 Evaluation

Experimental Environments. The experimental environment consisted of two Intel(R) Xeon(R) Gold 6230 20-core 2.10 GHz CPUs with Ubuntu 20.04 LTS, 256 GB of RAM, and four NVIDIA GeForce RTX 2080 Ti GPUs. The language used was Python 3.7. We have adapted and employed MAB-Malware to build the perturbation combination dictionary. The pefile library has proven its capability to apply perturbations to binaries reliably, without introducing unnecessary changes that could compromise the executability of the PE binary [11]. Consequently, we have opted to use the pefile library, as it aligns with the objectives of our attack method. Additionally, for our defense method, we have utilized python-tlsh 4.5.0 to construct the defense model.

Dataset. The dataset used in the experiment was acquired through MAB-malware. We collected 1,000 malware samples and 27,167 benign contents from this dataset. Among them, 206 benign contents were included in our perturbation combination dictionary. All 1,000 malware samples were utilized in our experiments.

Evaluation Metrics. To quantitatively assess the number of malware samples that evaded the target model, we employed the Evasion Rate as an evaluation metric for AEs. Eq. (3) represents the formula for the evasion rate, as follows. This metric will be utilized in our paper for analysis and evaluation purposes.

$$Evasion\ Rate = \frac{Evasion\ \ Adversarial\ \ Example\ \ Count}{Total\ \ Sample\ \ Count} \tag{3}$$

4.1.1 Target Model Classification Performance Measurement

Both well-known malware classifiers, Malconv and GBDT, were trained on the EMBER 2018 dataset [14] and implemented in MLSEC 2019 [35]. Before generating AEs, we evaluate the classification performance of these two models. The malware samples we acquired are executables, whereas benign content consists of specific byte sequences extracted from particular sections of binary or .dll files. To evaluate the classification performance of our target models, we randomly selected 1,000 executable benignware samples from the DikeDataset [37]. In total, we utilized 2,000 executable

datasets. We use several evaluation metrics, including Accuracy, macro-recall, macro-precision, and macro-F1-score, to assess the performance of the target model. Recall measures the proportion of samples belonging to the benign class that the model correctly classifies as benign. Precision represents the ratio between the model's classification results and the number of samples that actually belong to the corresponding class. The F1-score is computed as the harmonic mean of precision and recall. Accuracy, being the most commonly used metric for evaluating a model, indicates how closely the model's predictions align with the actual class labels of the samples. Table 1 presents the classification performance results of both target models. The GBDT model, which utilizes various types of features extracted through static analysis of PE binaries, uses raw bytes as input and exhibits overall higher performance compared to Malconv, which employs a relatively lower threshold. Consequently, Malconv, despite showing slightly inferior classification performance to GBDT, maintains a precision of 0.79, indicating satisfactory classification performance. Both models are employed as our target models for our attack and proposed defense method.

Table 1: Classification performance of our target model

Model	Precision	Recall	F1-score	Accuracy
Malconv	0.79	0.64	0.59	0.64
GBDT	1	1	1	1

4.1.2 Adversarial Example Generation for Bypassing Deep Learning Classifiers

Using a dataset of 1,000 malware samples, our goal is to generate AEs that evade the target model, Malconv, ensuring both viability and malicious behavior without requiring functionality verification. When our attack method was applied to the target model, the AEs generated through our attack method achieved a 99.9% evasion rate. This indicates that out of the 1,000 malware samples in our dataset, 999 AEs were successfully generated to evade the target model. We achieved this high evasion rate by analyzing the AEs generated by Malconv after modifying the MAB-malware and then storing perturbation combinations and benign content using a perturbation combination dictionary. Consequently, we can confirm the effectiveness of our attack method against Malconv, as it achieved a high evasion rate. Additionally, among the AEs generated by our method, the AE with the lowest detection score for the target model applied a single OA perturbation. The original malware associated with this AE had a detection score of 0.72, while this AE achieved a significantly lower detection score of 0.00027, well below Malconv's threshold of 0.5. This result clearly demonstrate that our attack method can effectively circumvent Malconv.

We analyzed the 999 generated AEs to identify the applied perturbation combinations. Fig. 6 illustrates the distribution of perturbation combination lengths used in the 999 generated AEs. Among them, 802 AEs were evaded by applying a single perturbation with a length of 1, constituting 80% of all AEs. Interestingly, no AEs showed perturbation combinations of lengths 5, 6, 7, or 9 with perturbation lengths greater than 1.

We investigated the reason why our attack method predominantly generates AEs with a single perturbation. Fig. 7 provides insights into the perturbation combination lengths that should have been applied to the original malware, revealing instances where the length is 2 or 3. Our attack method leverages the detection results of the target model each time an AE is generated. As a result, many AEs are produced by applying the initial perturbation from perturbation combinations with lengths

of 1 or more. Among the 802 single-perturbation AEs we generated, 707 had OA applied, while 55 had SA applied. It is evident that Malconv is susceptible to evasion attacks using both SA and OA. This vulnerability arises from Malconv's utilization of raw bytes as features. Furthermore, Malconv is susceptible to append-based attacks (OA, SA) and benign content due to its composition, which consists of a convolutional layer and a lack of learning regarding the location information of input features.

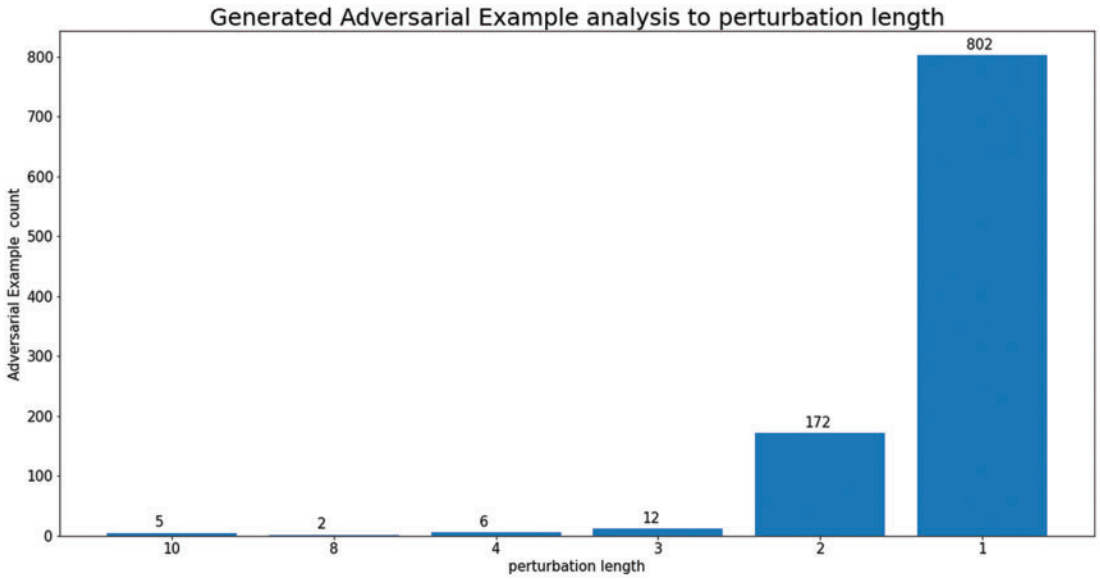

Figure 6: To bypass Malconv, distribution of perturbation combination length applied to generated AEs

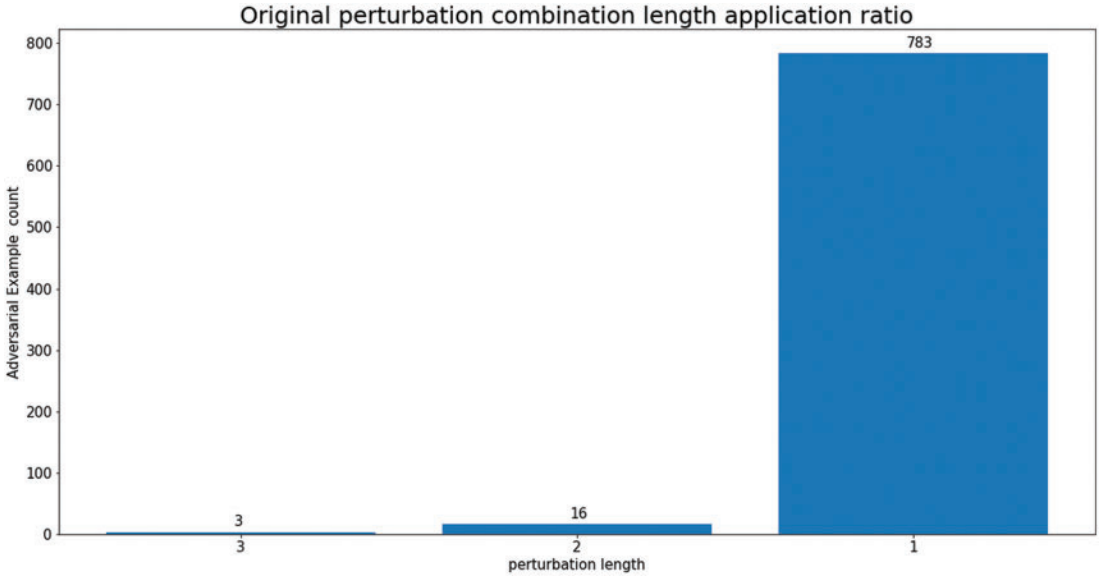

Figure 7: Distribution of original perturbation combination lengths for generated AEs

Therefore, our attack method achieved a high evasion rate against Malconv due to its use of benign content and append-based attacks, such as OA and SA. Table 2 presents a list of benign content used in the 802 single-perturbation AEs, demonstrating that the size of benign content does not significantly impact AE generation. Notably, "System.ni.dl" was employed in generating 112 of the 802 AEs, "NL7Data0804.dll" in 104 AEs, and "Microsoft.VisualBasic.Compatibility.dll" in 100 AEs. As a result, our attack method successfully evaded Malconv with a high evasion rate using a perturbation combination dictionary. Additionally, due to the target model's composition with a convolutional layer and a lack of learning about input feature locations, it remains vulnerable to our attack method using benign content, OA, and SA perturbations.

Table 2: List of benign contents used to generate AE that evades Malconv

	Benign content	Number of generated AEs
1	System.ni.dll|.reloc|309248	112
2	NL7Data0804.dll|.text|1065984	104
3	Microsoft.VisualBasic.Compatibility.dll|.text|479232	100
4	d3d10warp.dll|.text|5547008	98
5	WindowsBase.dll|.text|1195520	86
6	Solitaire.exe|.rdata|1912320	81
7	onmainim.dll|.rdata|2441216	75
8	QuickConnectUI.dll|.text|3175936	55
9	MSWB70804.dll|.text|466432	41
10	Windows.SharedPC.AccountManager.dll|.idata|6144	37
11	TileDataRepository.dll|.text|433152	6
12	rpcrt4.dll|.text|741376	3
13	offlinesam.dll|.rsrc|94208	2
14	winshfhc.dll|.text|10240	1
15	clrhost.dll|.reloc|1024	1

4.1.3 Adversarial Example Generation for Bypassing Machine Learning Classifiers

We generated AEs using 1,000 malware samples to evade the machine learning-based malware classifier, GBDT. Employing our attack method to evade GBDT resulted in 656 AEs successfully evading GBDT out of a dataset of 1,000 malware samples. As a result, our attack method achieved a 65.6% evasion rate against GBDT. This evasion rate is lower than the result presented in Section 4.1.2. The low evasion rate can be attributed to the fact that the perturbation combinations and benign contents stored in the perturbation combination dictionary did not exert sufficient influence on the input features used by GBDT.

We analyzed the original perturbation combination lengths that should have been applied to the 341 AEs with a perturbation combination length of 1, along with the benign content used in these AEs. Fig. 8 illustrates the distribution of perturbation combination lengths applied to the 656 generated AEs. Approximately 52% (341 cases) had a perturbation combination length of 1, while about 24%

(158 cases) had a length of 4. The reason why more than half of the generated AEs have a single perturbation applied is as explained earlier. Our attack method detects each AE by the target model every time a perturbation is applied to the original malware for AE generation, and that is why evasion primarily involves single perturbations.

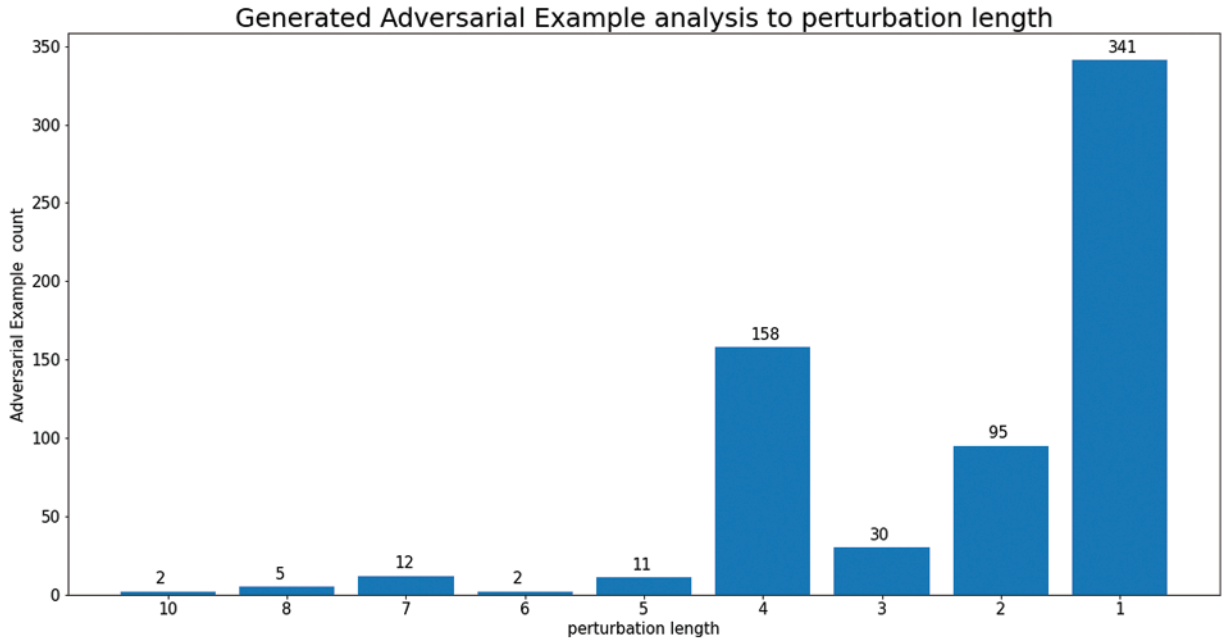

Figure 8: To bypass GBDT, distribution of perturbation combination length applied to generated AEs

Fig. 9 shows that 323 AEs were generated using a single perturbation with a perturbation combination length of one. Additionally, out of the 341 AEs with a perturbation combination length of 1, 340 AEs were generated by applying OA, while only 1 AE was generated by applying SA. The dominance of OA over SA can be attributed to the fact that OA affects format-agnostic features among GBDT's input features, such as raw byte histograms and byte entropy histograms. This observation indicates that OA was more frequently employed in AE generation.

Table 3 presents the benign content utilized for AEs with a perturbation combination length of 1, along with the number of AEs generated using each content. In contrast to the results in Section 4.1.2, it becomes evident that the size of the benign content employed directly affects the number of AEs generated. The benign content size that generated the highest number of AEs is 1867.5 KB. The combination of OA and substantial benign content has an impact on format-agnostic features, such as raw byte histograms and byte entropy histograms, used by GBDT as input. Consequently, this leads to the creation of AEs capable of evading GBDT. In simpler terms, GBDT employs a diverse range of feature types, making it more challenging to evade than Malconv, which relies solely on raw bytes as input. Through an analysis of the 656 generated AEs, we discovered that GBDT can be evaded by applying a single perturbation, specifically OA and SA. Additionally, the benign content size used in the AEs generated by GBDT is larger than that used in the AEs generated by Malconv.

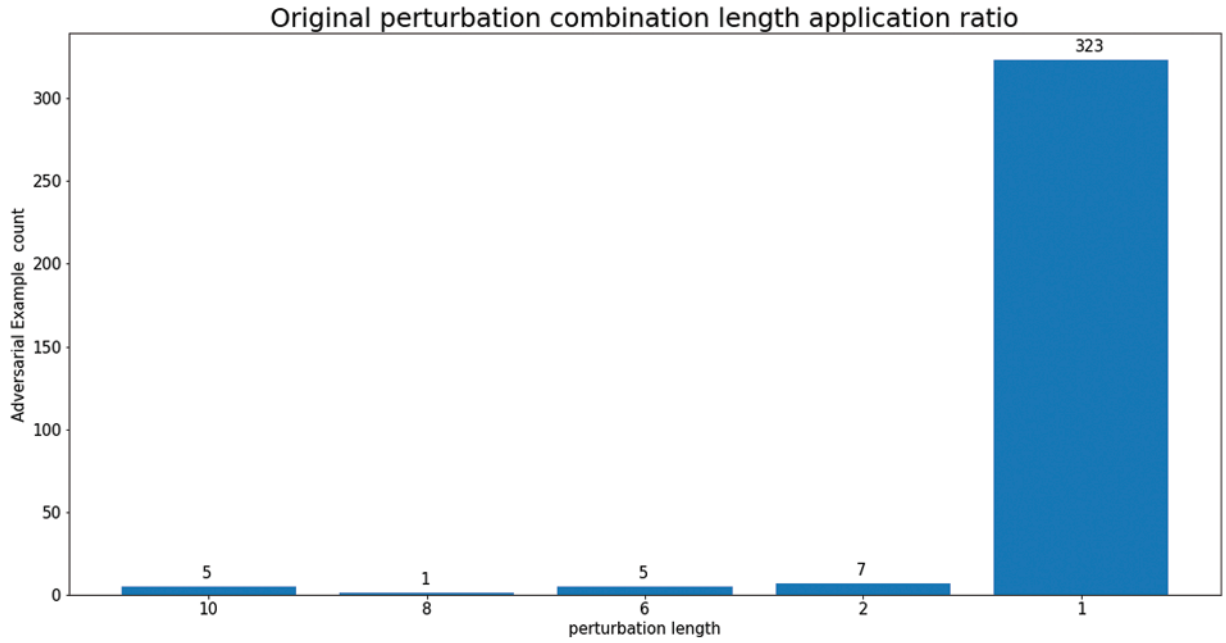

Figure 9: Distribution of original perturbation combination lengths for GBDT evasion AEs

Table 3: List of benign contents used to generate AE that evades GBDT

	Benign Content	Number of generated AEs
1	Solitaire.exe|.rdata|1912320	79
2	WindowsBase.dll|.text|1195520	66
3	Microsoft.VisualBasic.Compatibility.dll|.text|479232	54
4	onmainim.dll|.rdata|2441216	50
5	NL7Data0804.dll|.text|1065984	25
6	d3d10warp.dll|.text|5547008	25
7	MSWB70804.dll|.text|466432	13
8	UnityEngineDelegates.dll|.rdata|815104	10
9	System.ni.dll|.reloc|309248	9
10	TileDataRepository.dll|.text|433152	5
11	offlinesam.dll|.rsrc|94208	2
12	Windows.SharedPC.AccountManager.dll|.idata|6144	1
13	QuickConnectUI.dll|.text|3175936	1
14	Office.UI.Xaml.Hx.Mail.dll|.rdata|933376	1

4.2 AE Transferability

In this section, we assess the transferability of our generated AEs. Initially, we identify AEs that do not necessitate functional verification, ensuring both guaranteed executability and the manifestation of malicious behavior, using a malware detection model with distinct structures and features.

To assess whether the 999 AEs that evaded Malconv could also bypass GBDT, we subjected these 999 AEs to detection by GBDT. The detection results revealed an evasion rate of 33.2%, indicating that out of the total 999 AEs, 332 AEs successfully evaded GBDT. The lower evasion rate can be attributed to the perturbation and benign content applied to the generated AEs. The AEs were initially designed to evade Malconv, which relies on raw bytes as features. Consequently, many AEs may not effectively evade GBDT, which employs a diverse set of static features, unlike Malconv.

We then evaluated whether the 656 AEs that evaded GBDT could also bypass Malconv. For this assessment, Malconv was utilized to detect these 656 AEs. The results demonstrated that 650 AEs, constituting a 99% evasion rate, confirmed their capability to evade Malconv. It is noteworthy that the AEs generated with GBDT as the target exhibited a relatively higher evasion rate compared to those generated with Malconv as the target. The high evasion rate of the AEs generated to evade GBDT is inferred to be due to the benign content and perturbations applied to these AEs, which also influenced the raw bytes used by Malconv.

In conclusion, the AEs that evaded the ML-based malware classifier GBDT achieved a high evasion rate against the DL-based malware classifier Malconv. We observed that the transferability of AEs depends more on the features used by the target model rather than its structural complexity. Additionally, we have confirmed that the AEs generated by our attack method exhibit transferability.

4.2.1 AE Transferability at Real-World Malware Detector

From the pool of 650 AEs that evaded both GBDT and Malconv, we randomly selected one AE to be submitted to VirusTotal (VT) to assess how effectively an AE generated by our attack method could evade various antivirus vendors participating in VT. For this evaluation, we submitted a total of two binaries to VT: a randomly selected AE and its corresponding original malware.

Figs. 10 and 11 illustrate the number of antivirus vendors that participated in the detection process and the number of vendors that actually detected the original malware and the AE within the VT results. Notably, the AE managed to evade detection by 11 antivirus vendors compared to the detection results of the original malware. Fig. 12 provides a list of the antivirus vendors that were evaded. It can be inferred that these 11 vendors utilize static features and ML or DL models for malware detection.

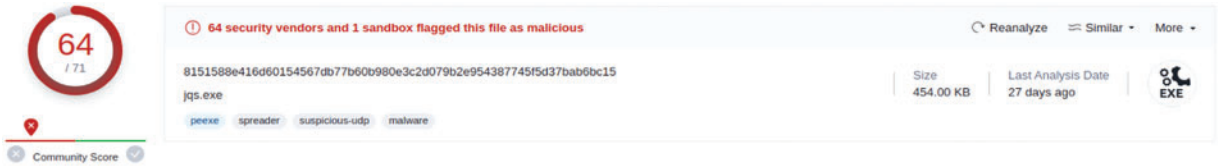

Figure 10: Part of the VT report of the original malware

Figure 11: Part of the VT report of the AE

Figure 12: List of antivirus vendors participating in VT that our generated AE evaded

Among the 53 vendors that detected the AE as malicious, some vendors provided detailed detection scores. For instance, "MAX", an antivirus vendor, displays a score in the detection label. The original malware was detected by "MAX" with an AI score of 100, while the AE was detected with an AI score of 87, indicating a 13-point drop in the AI score.

We conducted an analysis of the 53 vendors that detected the AE as malicious and observed that six vendors identified the AE with the same label, namely, "Trojan.Ransom.Cerber.1." These vendors include ALYac, Arcabit, BitDefender, eScan, GData, and VIPRE. It is plausible that these six vendors either shared the same detection label or employed similar logic for malware detection. Additionally, seven vendors classified the AE under the same malware family label, "Cerber." These vendors are AhnLab-V3, ClamAV, Cyren, Emsisoft, ESET-NOD32, Ikarus, and Microsoft. This suggests that these vendors may have utilized similar detection logic or shared the same detection label. In conclusion, our analysis revealed that some vendors on VirusTotal share detection labels, as previously noted in Peng et al. [38].

4.2.2 Section Injection

As observed in Sections 4.1.2 and 4.1.3, it is evident that the most frequently used perturbations in the AEs generated by our attack method are two single perturbations: OA and SA. Both OA and SA involve adding benign content to the original malware binary. These two methods are akin to the simple section injection approach, which injects adversarial noise by adding a dummy section to a PE binary and creates an AE by manipulating that section. The simple section injection attack is a type of white-box-based attack that optimizes the injected section to reduce the logit value of the target model [10,39].

We generated AEs capable of evading four target models using a simple section injection method that does not incorporate benign content, and subsequently, we assessed their evasion rates. Out of the four selected target models, three of them-Malconv [13], NonNeg [40], and IBLTMC-ff [41]-are designed for malware detection on PE binaries. The fourth target model, DexRay [42], specializes in detecting Android malware through image analysis. For our experiments, we utilized datasets consisting of PE binaries, randomly selecting 100 samples from the DikeDataset [37]. In the case of the DexRay model, we employed the DexRay dataset provided by the model itself. The Simple Section Injection attack is a method that optimizes the injected noise. Therefore, to generate AEs evading the Malconv, NonNeg, and IBLTMC-ff models, we injected 2 KB of noise into the original malware binary through the optimization process. For the DexRay model, noise was introduced equivalent to 0.5% of the input size in bytes. The results of the Simple Section Injection attack are presented in Table 4. Remarkably, even without the use of benign content, Simple Section Injection achieves a high evasion rate against the target model by introducing adversarial noise into the original malware and optimizing it.

4.3 Compare with the State-of-the-Art Approaches

We compare our attack method to MAB-malware, a state-of-the-art research method that utilizes reinforcement learning to generate AEs. To ensure a fair comparison, we adjusted our code to exclude the code randomization perturbation used by MAB-malware and utilized the same malware dataset.

Table 5 shows the results of the evasion rate comparison between our attack method and MAB-malware. In the table, bold indicates a high-performance evasion rate, and an asterisk indicates that the p-value, evaluated by a paired t-test, is significant at 0.05.

Table 4: Simple section injection experiment results

Model	Evasion rate	Query usage
Malconv [13]	98%	100
NonNeg [40]	97%	100
IBLTMC-ff [41]	98%	140
DexRay [42]	99%	150

Table 5: Comparison of our attack and a state-of-the-art's evasion rate

Model	MAB-malware [11]	Our attack method
Malconv [13]	96.7%	**99.9%**∗
EMBER [14]	**81.2**%	65.6%∗

Both MAB-malware and our attack methodology utilized the same set of 1,000 malware samples, with a common focus on generating AEs targeted at the Malconv and GBDT models. MAB-malware successfully generated 812 AEs that evaded GBDT, achieving an evasion rate of 81.2%. In contrast, our attack method generated around 65.6% of AEs, amounting to a total of 656 AEs when applied to the identical set of malware samples. These findings underscore that our attack method, relying on perturbation combinations, demonstrates a lower evasion rate compared to MAB-malware, which utilizes individual perturbations. Moreover, our achieved evasion rate was relatively lower, given that we employed a limited set of benign content stored in the Perturbation Combination Dictionary compared to MAB-malware.

Similarly, when generating AEs targeting Malconv, MAB-malware excelled by producing 967 AEs out of 1,000 malware samples, achieving an impressive 96.7% AE evasion rate. In contrast, our attack method outperformed MAB-malware by achieving a 99.9% AE creation rate with 999 AEs when targeting Malconv. According to the comparison results, our attack method demonstrated limitations in evading ML-based classifiers compared to the state-of-the-art research utilizing reinforcement learning. We attribute this limitation to the perturbation combinations included in our Perturbation Combination Dictionary and the limited number of benign content we employed. However, despite the restricted set of benign content and perturbation combinations, our study clearly outperformed recent research utilizing reinforcement learning in evading DL-based models.

5 Mitigating Adversarial Examples of Malware

To defend against repetitive queries in black-box attacks and protect the target model from AEs with various applied perturbations, we employ TLSH, a similarity score-based approach. Oliver et al. [43] conducted experiments to evaluate the efficiency of TLSH in comparison to other similarity score comparison schemes. Their objective was to assess how effectively TLSH can identify changes in files when their contents have been altered. Their findings confirmed that TLSH exhibits a

lower false positive rate for malware detection and a higher detection rate compared to other similarity score comparison schemes, specifically, SSDEEP [44] and SDHASH [45]. Overall, their experimental results demonstrate that TLSH is robust in identifying altered files and offers greater resistance to evasion compared to other similarity-based approaches. Additionally, TLSH, which employs the k-skip-n-gram method, can generate a 72-digit change-sensitive locality-sensitive hash for files of at least 50 bytes, facilitating similarity comparison [46].

Therefore, we utilize TLSH [33], a high-performance similarity hash known for its excellent detection rate in identifying similar byte streams or files while minimizing malware false positives, to construct our defense model. Fig. 13 illustrates the flow of our proposed defense method. We begin by storing the TLSH values and detection scores of the malware samples provided as input to the target model in a database. Subsequently, when a new input binary is given to the target model, we check the database for similar TLSH values. If a similar TLSH value is found, we return the corresponding detection score associated with that TLSH value. Conversely, if the TLSH value is not found in the database, we store the TLSH value along with its detection score in the database, thereby constructing our defense model.

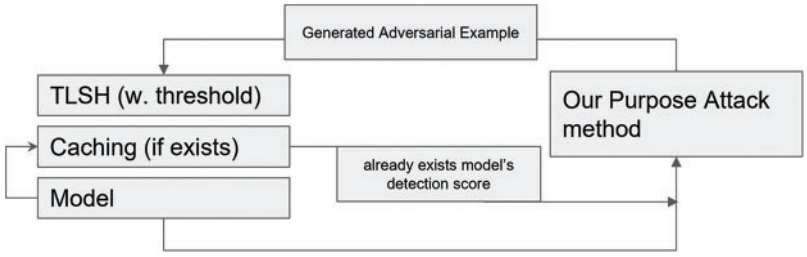

Figure 13: The flow of our proposed defense method

TLSH employs a distance metric for calculating similarity scores, resulting in scores that range from 0 to potentially 1,000, where 0 represents a complete perfect match and 1,000 signifies a mismatch [33]. Due to TLSH's wider range of similarity scores compared to SSDEEP and SDHASH, it has been demonstrated that a broader range of thresholds can be effectively utilized to achieve low false positives and high detection rates, distinguishing it from existing similarity comparison methods. Hence, we set the threshold to the average of the smallest and largest TLSH similarity scores between the original malware and the AEs generated by our attack method. Specifically, we set the threshold to 450 for Malconv and 543 for GBDT.

5.1 Evaluation Results of Mitigating Adversarial Examples of Malware

To evaluate the effectiveness of our defense model, we applied our attack method to it. The results demonstrate that our attack method was unsuccessful in generating an AE capable of bypassing the defense model, highlighting the robustness of our defense approach. Additionally, our defense model detected a total of 1,655 AEs generated by our attack method that evaded the two target models without the application of TLSH. Subsequently, when TLSH was incorporated into Malconv, all 1,655 AEs were no longer able to evade detection and were correctly classified as malicious. However, when TLSH was applied to GBDT, 9 AEs out of the 1,655 still managed to evade GBDT and were misclassified as benign. We analyzed the reasons why these 9 evaded AEs were successful in bypassing detection. Firstly, we checked the similarity scores between the evaded AEs and their corresponding original malware. The results showed an average similarity score of 640, exceeding the TLSH threshold set for GBDT. The AE with the highest differ similarity score in our dataset scored 726 when compared

to its original malware, indicating significant dissimilarity. Upon analyzing the 9 AEs that evaded detection, we identified two types of perturbations applied: OA and SA. Two AEs were perturbed using OA, while seven AEs were perturbed using SA. The benign content associated with OA is "d3d10warp.dll," and for SA, it is "QuickConnectUI.dll."

These AEs, generated using simple perturbations and corresponding benign content, exhibited substantial dissimilarity to the original malware when TLSH-based similarity scores were calculated using raw bytes as input.

Figs. 14 and 15 present sections of the VT report for the original malware and an AE with the highest similarity score of 726. In particular, Fig. 15 displays a segment of the VT report for the AE generated with the application of OA, showcasing the Overlay section. Notably, this section is absent in the VT report for the original malware. The entropy of the AE's overlay measures at 6.3, and in Fig. 14, the hash values in the Basic properties are all different. This verification supports the conclusion that the benign content used in the OA perturbation significantly influenced the raw byte histogram and byte entropy histogram, which are features utilized by GBDT, resulting in evasion beyond the established threshold. The notable variance in similarity scores can be attributed to the utilization of substantial benign content sizes, such as"d3d10warp.dll" (5,417 KB) and "QuickConnectUI.dl" (3,101 KB), which impact the format-agnostic features employed by GBDT, namely the raw byte histogram and byte entropy histogram. In summary, the AEs generated using our attack method showcase significantly distinct TLSH values compared to the original malware, providing evidence of their successful evasion of GBDT, with similarity scores surpassing the established threshold.

Basic properties

MD5	02aadd0984b007cdea27c046a09f1ec4
SHA-1	1fdc0e98e45dd995652121943e9f4232bb0e1bad
SHA-256	8151588e416d60154567db77b60b980e3c2d079b2e954387745f5d37bab6bc15
Vhash	045046151d751184z26008d1e5z37ze0c006bfz
Authentihash	fc9dc64e3cd7d902cf32964506e8f89807c6363a96fcd81759ca8038dae9e1f7
Imphash	d3f1135c76c904a6c2c482f929f2f77b
Rich PE header hash	032fe337725df3aad0c87216e3dcc3e1
SSDEEP	6144:YHHGR5D8kmKrA8QIQZi8vn64QR2bMgvp:YnyZ8kmKoi8v64FbMgv
TLSH	T16AA47B6561293011F3A6F3B40DA9FA320D693C50079604FB95A13ED41B17FFAA9B93F2
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable (GUI) Intel 80386, for MS Windows
TrID	Win32 Executable MS Visual C++ (generic) (37.8%) Microsoft Visual C++ compiled executable (generic) (20%) Win64 Executable (generic) (12.7%) Win32 Dynamic Link Library (generic) (7.9%) Win16 NE executable (generic) (6.1%)
DetectItEasy	PE32 Compiler: Microsoft Visual C/C++ (2008) Compiler: Microsoft Visual C/C++ (15.00.21022) [C] Linker: Microsoft Linker (9.00.21022) Tool: Visual Studio (2008)
File size	454.00 KB (464896 bytes)

Basic properties

MD5	9eb0d20db2bb62219c642e80bf0af049
SHA-1	dfaa89799b5e117a017a45042cceed1b5f77f938
SHA-256	73ce58d288b4d4efd57f2ea6db2f9ddde45dd432faf1973ab66f5311176f98e5
Vhash	066046151d751184z26008d1e5z37ze0c006bfz
Authentihash	539a2c6041ed38d38ed444b7cd78618d8b3f066001d2814ca0e34b2af864c946
Imphash	d3f1135c76c904a6c2c482f929f2f77b
Rich PE header hash	032fe337725df3aad0c87216e3dcc3e1
SSDEEP	98304:OnLhJ3+Wrpd3p6ngGETtB3MSGPNqGPwZbeznviOZ+kk6j3eu+Kp6H:i3+WrpFpogGUB3MSGPNqGPQeznviCJTs
TLSH	T11B56F772E386F02AE14E92B07909E6699D547D35036600C7B2D17EEA37F70E39A35E13
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable (GUI) Intel 80386, for MS Windows
TrID	Win32 Executable MS Visual C++ (generic) (37.8%) Microsoft Visual C++ compiled executable (generic) (20%) Win64 Executable (generic) (12.7%) Win32 Dynamic Link Library (generic) (7.9%) Win16 NE executable (generic) (6.1%)
DetectItEasy	PE32 Compiler: Microsoft Visual C/C++ (2008) Compiler: Microsoft Visual C/C++ (15.00.21022) [C] Linker: Microsoft Linker (9.00.21022) Tool: Visual Studio (2008)
File size	5.73 MB (6011904 bytes)

Figure 14: The above VT report is from the original malware and the below report is from the AE

Overlay

entropy	6.3215789794921875
offset	464896
chi2	53310884
filetype	unknown
md5	baaf4def364bbac72635a611e74638a4
size	5547008

Figure 15: Overlay section of AE's VT report

6 Related work

In this section, we review research that employs diverse methods for conducting evasion attacks and present studies that have proposed defense mechanisms against such attacks. Table 6 offers a concise summary of studies focused on evasion attacks.

Table 6: Researches conducting evasion attacks

Name	AE generating method	Benign content or benignware usage	Attack type (white box/black box)	Target model type
RAMEn [10]	Genetic algorithm	X	White box	ML & DL
GAMMA [12]	Genetic algorithm	O	Black box	ML & DL
Anderson et al. [15]	Reinforcement learning	X	Black box	ML
Song et al. [11]	Reinforcement learning	O	Black box	ML & DL
MalGAN [47]	Generative adversarial networks	X	White box	ML
GAPGAN [48]	Generative adversarial networks	O	Black box	DL

6.1 Related Work on Evasion Attack Research

In this section, we categorize and introduce studies on evasion attacks into four distinct categories.

6.1.1 Research on Evasion Attacks Based on Optimization Methods

The optimization method utilizing a GA exploits the inherent ambiguities in the PE format to inject adversarial noise either into the slack space of the PE format or at the end of the binary file. Subsequently, AEs are generated by optimizing this adversarial noise using GA. AEs generated through GA capitalize on the PE format's ambiguities, offering the advantage of not requiring functional verification. However, these studies do not guarantee that the AE exhibits the same behavior as the original malware. Additionally, since GA employs a fitness function, it faces limitations when applied to models that provide a binary output label [10,12].

6.1.2 Research on Evasion Attacks Based on RL Methods

In Reinforcement Learning (RL), the agent applies perturbations to the original malware by executing predefined actions, resulting in the generation of AEs and earning rewards. The RL agent learns to maximize its rewards. Anderson et al. [15] introduced an RL-based black-box attack method. In this approach, the authors automatically generated new variants of Windows PE exploit malware by modifying binary files. However, as the number of actions an agent must undertake to create an AE increases, so does the search space for actions, making it challenging to apply with current RL algorithms [49]. Nevertheless, in reference [15], RL-based AE generation method demonstrated a 15% improvement in performance compared to a method that randomly selects perturbations (actions). It should be noted that this method does not guarantee the preservation of AE functionality. Song et al. [11] proposed an RL-based framework for generating AEs that can evade PE malware classifiers and antivirus engines. By considering the adversarial attack as a MAB problem, the authors aimed to strike a balance between discovering evasion patterns and generating more variants. The results showed an evasion rate ranging from more than 74% to 97% against ML detectors. However, MAB-Malware requires a significant amount of time to attack commercial AV systems, making it less suitable for directly targeting such systems [50].

6.1.3 Research on Evasion Attacks Based on GAN Methods

Research on Generative Adversarial Network (GAN)-based AE generation offers the advantage of being able to target various types of models due to the use of generative models, such as GANs. GAN-based AE generation research employs two primary approaches: one uses malware and noise as input to generate AEs, while the other employs benign binaries and noise as input to GANs to generate adversarial payloads, which are then injected into malware [32,47,48,51]. It is worth noting that AEs generated by MalGAN [47] and GAPGAN [48] do not result in executable binaries [51]. Additionally, MalGAN generates AEs based on API calls as features, limiting their effectiveness in evading models that use different features. Conversely, GAPGAN demonstrates that generating AEs using bytes is more efficient for conducting black-box attacks compared to using API calls as features. Lastly, it is important to acknowledge that relying solely on GANs for AE generation can be challenging due to stability issues during GAN training, which may hinder effective convergence on specific datasets [22].

6.1.4 Research on Evasion Attacks Based on Packing and Encryption Methods

Most researchers employ packers or binary encryption methods to generate AEs that can evade the target model. They either pack and unpack malware or modify the encoding of the original malware's binary bytes, creating a new malware variant in the form of a dropper using techniques like XOR encryption. Studies utilizing these methods have demonstrated effective evasion results [11,15,20,31]. However, the goal of our attack method in this paper is to generate AEs by leveraging the advantages of preserving the original malware's PE format, including maintaining executability, preserving malicious behavior, and avoiding unnecessary functional verification. Therefore, we do not employ packing or obfuscation techniques that alter the PE format.

6.2 Related Work on Research for Defenses against Evasion Attacks

Universal Adversarial Perturbations (UAP) is a type of adversarial perturbation that applies the same perturbation to various inputs to induce errors in ML classifiers. Labaca-Castro et al. [27] proposed an attack method in feature space and problem space using UAP. Additionally, they introduced a defense method utilizing adversarial training with AEs generated through the attack

method. Evaluation results demonstrated that UAP attacks represent a serious and real threat to ML-based malware detection systems. The defense method leverages the generated AEs for adversarial training, considered the most promising defense strategy. The authors used a dataset comprising an equal mix of original malware and generated AEs for adversarial training. However, following adversarial training, the classifier exhibited a slight degradation in performance when detecting the original malware. The defense method still grapples with the limitation inherent in adversarial training—the challenge of defending against various types of perturbations.

Chen et al. [52] introduced EvnAttack, an effective evasion attack model designed to target PE malware detectors using Windows API calls as input. They also proposed SecDefender, a security learning framework aimed at effectively countering evasion attacks. EvnAttack assesses the significance of API calls and categorizes them into two sets: M comprising API calls highly relevant to malware and B comprising API calls highly relevant to benignware. It then proposed an attack method to either inject API calls from set B into the extracted malware API calls or remove API calls from set M. SecDefender, as a defense method, suggests retraining the classifier using EvnAttack and incorporates a defense strategy that employs a security regularization term to recover any performance degradation of the model post-retraining. Both research efforts demonstrate the potential to mitigate one or multiple adversarial attacks. However, it is important to note that adversarial retraining entails significant additional costs for generating AEs and retraining the model, rendering it impractical in some attacks [16,53].

Quiring et al. [54] presented a combinatorial framework for adversarial defense that achieved the top position in Microsoft's 2020 Machine Learning Security Evasion Competition. The Peberus framework begins by inspecting input malware based on predefined heuristics. It then leverages an ensemble comprising existing malware detectors, a monotonic skip-gram model, and a majority voting mechanism involving signature-based models to make predictions. Finally, Peberus employs a stateful nearest-neighbor detector to continuously assess whether the PE file exhibits similarity to previously identified malware samples.

Lucas et al. [53] introduced an attack method that modifies the instructions within the original binary without injecting maliciously crafted bytes. Additionally, the authors proposed a defense approach employing binary normalization and instruction masking to counter adversarial attacks that insert runtime-executable instructions into different sections of the binary. The defense method initially reverses any potential adversarial manipulations applied to the PE malware, ensuring that the input to the PE malware detector is in a form that enables accurate classification. However, it is worth noting that attackers could potentially overcome the defense by using obfuscation techniques or transformations that the normalization algorithm does not recognize.

7 Conclusion

In this paper, we employ perturbations utilized in prior AE generation research, particularly those perturbations that leverage the ambiguity present in the original malware's PE format, in conjunction with benign content. Our attack method aims to generate AEs that preserve the PE format of the original malware, ensuring both executability and malicious behavior without the need for unnecessary functional verification. In contrast to traditional AE generation research, our attack method does not employ complex optimization processes or intricate attack methods. Instead, we showcase the significant threat posed by our approach, which simply employs perturbations and benign content to generate AEs capable of effectively evading well-known malware classifiers. Additionally, we analyzed

the generated AEs and confirmed that the inclusion of benign content in AE generation is beneficial for evading the target model in a black-box attack.

To assess the transferability of AEs generated by our attack method, we evaluate their detection capabilities against two distinct target models. The results indicate that AEs achieved evasion rates of 33.2% and 99%, respectively. Subsequently, we examined the effectiveness of our generated AEs in evading antivirus vendors participating in VT. The results demonstrated that our AEs successfully evaded 11 antivirus vendors, estimating that these 11 vendors employ static features and utilize ML or DL models for malware detection. In conclusion, our attack method has clearly demonstrated its effectiveness in generating AEs with robust transferability in black-box attacks. This success is attributed to the modification of the PE format of the original malware and the incorporation of benign content.

To mitigate the transferability of our generated AEs and defend against our attack method, which utilizes various perturbations, we built a defense model using TLSH, which is sensitive to changes and demonstrates excellent performance in finding similar files. To validate our proposed defense method, we performed our attack method on the defense model we constructed. The results confirmed the inability to generate AEs capable of evading the target model when the defense was applied. The impossibility of AE generation can be attributed to the effectiveness of TLSH in detecting similar byte streams or files, thereby preventing the creation of AEs capable of evading the target model through the use of various perturbations and benign content. Therefore, we can infer that the TLSH employed in our proposed defense method is effective in countering persistent queries in black-box attacks. Finally, we subjected the generated 1,655 AEs to the defense-applied model for detection. The detection results revealed that, among these AEs, 9 were successful in evading the GBDT defense model. However, it was confirmed that not a single sample could bypass the Malconv defense model.

8 Limitations

Our attack method is tailored to target static analysis-based classifiers such as Malconv or GBDT. Therefore, it may not be applicable for evasion when targeting a dynamic analysis-based classifier. Moreover, the AEs we have generated are theoretically guaranteed to be executable and malicious, eliminating the necessity for functional verification. However, it is worth noting that, since we did not execute these AEs within a sandbox environment, there remains a possibility that the original malware and the AEs may exhibit different behaviors. The proposed defense method relies on a similarity score comparison of the TLSH value between the original malware and the AE to protect the model from vulnerability to evasion attacks. Nevertheless, it is important to note that TLSH's similarity score comparison is distance-based, resulting in a score range of 0 to 1,000, which is broader than other similarity comparison methods. Therefore, while the thresholds we employed in our experiments may be suitable for our specific attack, they might not be directly transferable to real-world applications.

Acknowledgement: The authors extend their appreciation to all the authors who contributed to this paper and to the reviewers and editors who have assisted in its improvement. The authors wish to express their appreciation to the reviewers for their helpful suggestions which greatly improved the presentation of this paper.

Funding Statement: This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) Grant funded by the Korea government, Ministry of Science and ICT (MSIT) (No. 2017-0-00168, Automatic Deep Malware Analysis Technology for Cyber Threat Intelligence).

Author Contributions: The authors confirm contribution to the paper as follows: study conception, design and data collection: Younghoon Ban; coding: Younghoon Ban, Myeonghyun Kim; data collection, analysis and interpretation of results: Younghoon Ban; draft manuscript preparation: Younghoon Ban, Haehyun Cho. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The datasets and tools utilized in this paper were sourced from publicly accessible repositories, guaranteeing accessibility. Access to these resources is not constrained, and they can be obtained through the sources listed in the References section of this paper.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Shaukat, K., Iqbal, F., Alam, T. M., Aujla, G. K., Devnath, L. et al. (2020). The impact of artificial intelligence and robotics on the future employment opportunities. *Trends in Computer Science and Information Technology, 5(1),* 50–54.
2. Javed, U., Shaukat, K., Hameed, I. A., Iqbal, F., Alam, T. M. et al. (2021). A review of content-based and context-based recommendation systems. *International Journal of Emerging Technologies in Learning (iJET), 16(3),* 274–306.
3. Luo, S., Shaukat, K. (2022). *Computational methods for medical and cyber security*. Basel, Switzerland: MDPI Books.
4. Shaukat, K., Luo, S., Varadharajan, V., Hameed, I. A., Chen, S. et al. (2020). Performance comparison and current challenges of using machine learning techniques in cybersecurity. *Energies, 13(10),* 2509.
5. Shaukat, K., Luo, S., Varadharajan, V., Hameed, I. A., Xu, M. (2020). A survey on machine learning techniques for cyber security in the last decade. *IEEE Access, 8,* 222310–222354.
6. Shaukat, K., Luo, S., Varadharajan, V. (2023). A novel deep learning-based approach for malware detection. *Engineering Applications of Artificial Intelligence, 122,* 106030.
7. Shaukat, K., Rubab, A., Shehzadi, I., Iqbal, R. (2017). A socio-technological analysis of cyber crime and cyber security in Pakistan. *Transylvanian Review, 1,* 84.
8. Javed, I., Tang, X., Shaukat, K., Sarwar, M. U., Alam, T. M. et al. (2021). V2X-based mobile localization in 3D wireless sensor network. *Security and Communication Networks, 2021,* 1–13.
9. Shaukat, K., Alam, T. M., Hameed, I. A., Khan, W. A., Abbas, N. et al. (2021). A review on security challenges in Internet of Things (IoT). *2021 26th International Conference on Automation and Computing (ICAC)*, Portsmouth, UK, IEEE.
10. Demetrio, L., Coull, S. E., Biggio, B., Lagorio, G., Armando, A. et al. (2021). Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM Transactions on Privacy and Security, 24(4),* 1–31.
11. Song, W., Li, X., Afroz, S., Garg, D., Kuznetsov, D. et al. (2022). Mab-malware: A reinforcement learning framework for blackbox generation of adversarial malware. *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, Nagasaki, Japan.
12. Demetrio, L., Biggio, B., Lagorio, G., Roli, F., Armando, A. (2021). Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security, 16,* 3469–3478.
13. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B. et al. (2018). Malware detection by eating a whole EXE. *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, USA.

14. Anderson, H. S., Roth, P. (2018). EMBER: An open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637.
15. Anderson, H. S., Kharkar, A., Filar, B., Evans, D., Roth, P. (2018). Learning to evade static PE machine learning malware models via reinforcement learning. arXiv preprint arXiv:1801.08917.
16. Ling, X., Wu, L., Zhang, J., Qu, Z., Deng, W. et al. (2023). Adversarial attacks against windows PE malware detection: A survey of the state-of-the-art. *Computers & Security, 128,* 103134.
17. Verma, U., Huang, Y., Woodward, C., Schmugar, C., Ramagopal, P. P. et al. (2022). Attacking malware detection using adversarial machine learning. *2022 4th International Conference on Data Intelligence and Security (ICDIS)*, Shenzhen, China, IEEE.
18. Hu, Y., Wang, N., Chen, Y., Lou, W., Hou, Y. T. (2022). Transferability of adversarial examples in machine learning-based malware detection. *2022 IEEE Conference on Communications and Network Security (CNS)*, Austin, TX, USA, IEEE.
19. Kreuk, F., Barak, A., Aviv-Reuven, S., Baruch, M., Pinkas, B. et al. (2018). Deceiving end-to-end deep learning malware detectors using adversarial examples. arXiv preprint arXiv:1802.04528.
20. Ceschin, F., Botacin, M., Lüders, G., Gomes, H. M., Oliveira, L. et al. (2020). No need to teach new tricks to old malware: Winning an evasion challenge with XOR-based adversarial samples. *Reversing and Offensive-Oriented Trends Symposium*, Vienna, Austria.
21. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D. et al. (2013). Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.
22. Rosenberg, I., Shabtai, A., Rokach, L., Elovici, Y. (2018). Generic black-box end-to-end attack against state of the art API call based malware classifiers. *Research in Attacks, Intrusions, and Defenses*, pp. 490–510. Heraklion, Greece, Springer.
23. Demontis, A., Melis, M., Pintor, M., Jagielski, M., Biggio, B. et al. (2019). Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks. *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, USA.
24. Goodfellow, I. J., Shlens, J., Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.
25. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. *2016 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, IEEE.
26. Zhang, F., Chan, P. P., Biggio, B., Yeung, D. S., Roli, F. (2015). Adversarial feature selection against evasion attacks. *IEEE Transactions on Cybernetics, 46(3),* 766–777.
27. Labaca-Castro, R., Muñoz-González, L., Pendlebury, F., Rodosek, G. D., Pierazzi, F. et al. (2021). Realizable universal adversarial perturbations for malware. arXiv preprint arXiv:2102.06747.
28. Shaukat, K., Luo, S., Varadharajan, V. (2022). A novel method for improving the robustness of deep learning-based malware detectors against adversarial attacks. *Engineering Applications of Artificial Intelligence, 116,* 105461.
29. Yan, S., Ren, J., Wang, W., Sun, L., Zhang, W. et al. (2022). A survey of adversarial attack and defense methods for malware classification in cyber security. *IEEE Communications Surveys & Tutorials, 25,* 467–496.
30. Quiring, E., Maier, A., Rieck, K. (2019). Misleading authorship attribution of source code using adversarial learning. *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, USA.
31. Jin, B., Choi, J., Hong, J. B., Kim, H. (2023). On the effectiveness of perturbations in generating evasive malware variants. *IEEE Access, 11,* 31062–31074.
32. Rosenberg, I., Shabtai, A., Elovici, Y., Rokach, L. (2020). Query-efficient black-box attack against sequence-based malware classifiers. *Annual Computer Security Applications Conference*, pp. 611–626. Austin, USA.

33. Oliver, J., Cheng, C., Chen, Y. (2013). TLSH–A locality sensitive hash. *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, Sydney, NSW, Australia, IEEE.
34. Oliver, J., Ali, M., Hagen, J. (2020). HAC-T and fast search for similarity in security. *2020 International Conference on Omni-Layer Intelligent Systems (COINS)*, Barcelona, Spain, IEEE.
35. MLSec2019 (2019). https://github.com/endgameinc/malware_evasion_competition (accessed on 06/25/2023)
36. Shaukat, K., Luo, S., Chen, S., Liu, D. (2020). Cyber threat detection using machine learning techniques: A performance evaluation perspective. *2020 International Conference on Cyber Warfare and Security (ICCWS)*, Islamabad, Pakistan, IEEE.
37. DikeDataset (2019). https://github.com/iosifache/DikeDataset (accessed on 08/26/2023)
38. Peng, P., Yang, L., Song, L., Wang, G. (2019). Opening the blackbox of virustotal: Analyzing online phishing scan engines. *Proceedings of the Internet Measurement Conference*, Amsterdam, Netherlands.
39. Castro, R. L., Schmitt, C., Rodosek, G. D. (2019). ARMED: How automatic malware modifications can evade static detection? *2019 5th International Conference on Information Management (ICIM)*, Cambridge, UK, IEEE.
40. Fleshman, W., Raff, E., Sylvester, J., Forsyth, S., McLean, M. (2018). Non-negative networks against adversarial attacks. arXiv preprint arXiv:1806.06108.
41. Prajapati, P., Stamp, M. (2021). An empirical analysis of image-based learning techniques for malware classification. *Malware Analysis Using Artificial Intelligence and Deep Learning,* 411–435.
42. Daoudi, N., Samhi, J., Kabore, A. K., Allix, K., Bissyandé, T. F. et al. (2021). DEXRAY: A simple, yet effective deep learning approach to android malware detection based on image representation of bytecode. *Deployable Machine Learning for Security Defense*, pp. 81–106.
43. Oliver, J., Forman, S., Cheng, C. (2014). Using randomization to attack similarity digests. *Applications and Techniques in Information Security*, pp. 199–210. Melbourne, Australia, Springer.
44. Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation, 3,* 91–97.
45. Roussev, V. (2010). Data fingerprinting with similarity digests. *Advances in Digital Forensics VI*, pp. 207–226. Hong Kong, China, Springer.
46. Oliver, J., Hagen, J. (2021). Designing the elements of a fuzzy hashing scheme. *2021 IEEE 19th International Conference on Embedded and Ubiquitous Computing (EUC)*, Shenyang, China, IEEE.
47. Hu, W., Tan, Y. (2022). Generating adversarial malware examples for black-box attacks based on GAN. *International Conference on Data Mining and Big Data*, Beijing, China, Springer.
48. Yuan, J., Zhou, S., Lin, L., Wang, F., Cui, J. (2020). Black-box adversarial attacks against deep learning based malware binaries detection with GAN. *ECAI 2020*, pp. 2536–2542. Santiago de Compostela, Spain, IOS Press.
49. Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T. et al. (2015). Deep reinforcement learning in large discrete action spaces. arXiv preprint arXiv:1512.07679.
50. Rigaki, M., Garcia, S. (2023). Stealing and evading malware classifiers and antivirus at low false positive conditions. *Computers & Security, 129,* 103192.
51. Zhong, F., Cheng, X., Yu, D., Gong, B., Song, S. et al. (2023). Malfox: Camouflaged adversarial malware example generation based on Conv-GANs against black-box detectors. *IEEE Transactions on Computers,* 1–14.
52. Chen, L., Ye, Y., Bourlai, T. (2017). Adversarial machine learning in malware detection: Arms race between evasion attack and defense. *2017 European Intelligence and Security Informatics Conference (EISIC)*, Athens, Greece, IEEE.

53. Lucas, K., Sharif, M., Bauer, L., Reiter, M. K., Shintre, S. (2021). Malware makeover: Breaking ML-based static analysis by modifying executable bytes. *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, Hong Kong, China.
54. Quiring, E., Pirch, L., Reimsbach, M., Arp, D., Rieck, K. (2020). Against all odds: Winning the defense challenge in an evasion competition with diversification. arXiv preprint arXiv:2010.09569.