ORIGINAL PAPER

A comparison of adversarial malware generators

Pavla Louthánová[1] · Matouš Kozák[1] · Martin Jureček[1] · Mark Stamp[2] · Fabio Di Troia[2]

Abstract

Machine learning has proven to be a valuable tool for automated malware detection, but machine learning systems have also been shown to be subject to adversarial attacks. This paper summarizes and compares related work on generating adversarial malware samples, specifically malicious Windows Portable Executable files. In contrast with previous research, we not only compare generators of adversarial malware examples theoretically, but we also provide an experimental comparison and evaluation for practical usability. We use gradient-based, evolutionary-based, and reinforcement-based approaches to create adversarial samples, which we test against selected antivirus products. The results show that applying optimized modifications to previously detected malware can lead to incorrect classification of the file as benign. Moreover, generated malicious samples can be effectively employed against detection models other than those used to produce them, and combinations of methods can construct new instances that avoid detection. Based on our findings, the Gym-malware generator, which uses reinforcement learning, has the greatest practical potential. This generator has the fastest average sample production time of 5.73 s and the highest average evasion rate of 44.11%. Using the Gym-malware generator in combination with itself further improved the evasion rate to 58.35%. However, other tested methods scored significantly lower in our experiments than reported in the original publications, highlighting the importance of a standardized evaluation environment.

Keywords Adversarial malware · Malware detection · Machine learning · PE files

1 Introduction

With the rapid development of information technology, computer systems have become increasingly important in people's daily lives. Unfortunately, the rapid development of these technologies is accompanied by a similarly rapid increase in cyberattacks.

✉ Matouš Kozák
matous.kozak@fit.cvut.cz

Pavla Louthánová
louthpav@fit.cvut.cz

Martin Jureček
martin.jurecek@fit.cvut.cz

Mark Stamp
mark.stamp@sjsu.edu

Fabio Di Troia
fabio.ditroia@sjsu.edu

[1] Faculty of Information Technology, Czech Technical University in Prague, Prague, Czechia

[2] Department of Computer Science, San Jose State University, San Jose, CA, USA

Malicious software (malware) is one of the most significant security threats today, comprising several different categories of malicious code, such as viruses, trojans, worms, spyware, and ransomware. To protect computers and the Internet from malware, early detection is necessary. However, this is problematic, as a large amount of new malicious code is generated every day [1]. Since it is not possible to analyze each sample individually, automatic mechanisms are required to detect malware.

Antivirus companies often rely mainly on signature-based detection techniques [2] for malware detection. Signatures are specific patterns that allow for the recognition of malicious files. For example, they can be a byte sequence, a file hash, or a string. When inspecting a file, the antivirus system compares its content with the signatures of already-known malware stored in the database. If a match is found, the file is reported as malware. Signature-based detection methods are fast and effective in detecting known malware. However, malware authors can modify their code to change the signature of the program, thereby avoiding detection. Some malware can hide in the system using various obfuscation techniques [3], such as encryption, oligomorphic, polymorphic, metamor-

phic, stealth, and packing methods, to make the detection process more difficult.

Machine learning (ML) models are commonly used today in various fields. Their application can be found, for example, in technologies such as self-driving cars, weather forecasting, face recognition, or healthcare [4, 5]. Machine learning has also proved to be a useful tool for automatic malware detection [6]. Unlike the signature-based method, it is capable of detecting previously unknown or obfuscated malware. However, it can be difficult to explain why the model classifies a certain file as malicious or benign [7], which can cause hidden vulnerabilities that attackers can exploit.

Machine learning models are vulnerable to adversarial attacks [8]. Attackers purposely design adversarial examples, which are deliberately designed inputs to a machine learning model, to cause the model to make a mistake in its predictions. Adversarial machine learning is a field that deals with attacks on machine learning algorithms and defenses against such attacks.

Malware detection is thus a battle between defenders and malware authors, in which each side attempts to devise new and effective ways to outwit the other. Each detection method has its own advantages and disadvantages. In various scenarios, one method may be more successful than another. Thus, the creation of an effective malware detection method is a very challenging task, and new research and methods are necessary.

The main contribution of this paper is to compare works that focus on adversarial machine learning in the area of malware detection. Specifically,

- we applied some existing methods in the field of adversarial learning to selected malware detection systems.
- we combined these methods to create more sophisticated adversarial generators capable of bypassing top-tier AV products.
- we evaluated the single and combined generators in terms of accuracy and usability in practice.

While several survey papers [9–13] summarizing current state-of-the-art knowledge in the domain of adversarial malware, this is the first comparison, which includes experimental evaluation of different techniques against top-tier antivirus engines in terms of practical usability.

The rest of the paper is organized as follows: In Sect. 2, we describe state-of-the-art techniques used to generate adversarial examples. Section 3 provides an overview of the publications focused on creating adversarial portable executable malware samples. In Sect. 4, we describe the experiments performed and the metrics used for evaluation. Section 5 presents the experimental results, and in Sect. 6, we compare our results with related work, address the limitations of our research, and propose ideas for future research.

Finally, we summarize this work and contributions in Conclusion.

2 Background

In this section, we describe the different methods used to create adversarial examples. We also introduce and describe selected attacks for experimentation.

2.1 Methods for creating adversarial examples

In this section, we describe various methods to create adversarial examples.

2.1.1 Gradient-based approaches

Gradient-based methods are a popular approach to generate adversarial examples. These methods work by computing the gradient of a loss function with respect to the input data. This gradient is then used to iteratively modify the input to minimize the loss. The Fast Gradient Sign Method [14] and the Jacobian-based Saliency Map Approach [15] are two popular gradient-based methods used for malware generation.

Given a trained model f and an input example x, gradient-based methods generate an adversarial example x' by adding a small perturbation δ to the input that maximizes the loss function $L(f(x'), y)$, where y is the true input label.

The perturbation δ is calculated as follows:

$$\delta = \epsilon \cdot \text{sign}(\nabla_x L(f(x), y))$$

where ϵ is a small constant that controls the size of the perturbation, and the sign function is used to ensure that the perturbation has the same sign as the gradient, allowing efficient computation and ensuring that the perturbation always increases the loss.

The gradient of the loss function with respect to the input ($\nabla_x L(f(x), y)$) is computed by backpropagation through the model f. This gradient gives the direction in which the loss function increases the most for a small change in the input and is used to determine the direction of the perturbation.

Gradient-based attacks are performed using the addition or insertion method for perturbations generated using the gradient of the cost function. When using the append method, the data (payload) is appended at the end of the file. When the insertion method is used, the payload is inserted into the slack region where the physical size is greater than the virtual size.

2.1.2 Generative adversarial network-based approaches

Generative adversarial networks (GANs) were developed and presented by Goodfellow et al. [16] in 2014.

GAN is a system consisting of two neural networks, a generator and a discriminator, which compete against each other. The goal of the generator is to create examples that are indistinguishable from the real examples in the training set, thus fooling the discriminator. In contrast, the objective of the discriminator is to distinguish the false examples produced by the generator from the real examples that come from the training data set, thus preventing it from being fooled by the generator. The generator learns from the feedback it receives from the discriminator's classification [17].

These two neural networks are trained simultaneously. The generator is constantly improving its ability to generate realistic samples, so the discriminator must continually improve its ability to distinguish between real and generated samples. This mutual competition forces both networks to continuously improve through the training process. Once this training process is completed, the generator can be used to generate new samples that are indistinguishable from the real samples [18].

Denote the generator as G and the discriminator as D. As described in [16], networks G and D play the following two-player minimax game with value function $V(G, D)$:

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[log D(x)]$$
$$+ \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

that G tries to minimize, while D tries to maximize. $D(x)$ is the discriminator's estimate of the probability that the original data x is real, $G(z)$ is the generator's output when it receives noise z as input, $D(G(z))$ is the discriminator's estimate of the probability that a synthetic sample $G(z)$ of data is real, $\mathbb{E}_x$ is the expected value over all real data instances, and $\mathbb{E}_z$ is the expected value over all generated fake instances.

2.1.3 Reinforcement learning-based approaches

Reinforcement learning (RL) is a type of machine learning technique along with supervised and unsupervised machine learning. In supervised machine learning, the model is trained using a training set of labeled examples. Based on the given inputs and expected outputs, the model creates a mapping equation that the model can use to predict the labels of the inputs in the future. In unsupervised machine learning, the model is trained only on inputs without labels. The model divides the input data into classes that have similar properties, and during prediction, the inputs are labeled based on the similarity of their properties to one of the classes. Unlike supervised and unsupervised machine learning, rein-forcement learning algorithms learn by interacting with an environment and getting feedback in the form of rewards or penalties rather than relying on pre-labeled instances.

A reinforcement learning model consists of two main parts: an agent and an environment. The agent learns to perform a task through repeated interactions with the environment through trial and error. In addition to the agent and the environment, the reinforcement learning model has four main subelements: a policy, a reward signal, a value function, and, optionally, an environment model. The policy defines the behavior of the agent at a particular time. In other words, it is a strategy that the agent uses to determine the next action based on the current state to achieve the highest reward. The reward signal is the feedback from the environment to the agent, indicating the success or failure of the agent's action in a given state. At each time step, the agent is in some state and sends the selected action as its output to the environment, which then returns a new state to the agent along with a reward signal. While the reward signal shows what is beneficial in the present, the value function describes what is beneficial in the long term. The value function provides an estimate of the expected cumulative reward from the current state of the environment in the future. The agent's objective is to maximize the total reward. The environment model mimics the behavior of the environment, making it possible to predict future states and rewards. This is an optional part of the system [19].

Reinforcement learning is defined as repeated interactions between an agent and an environment. Individual interactions (signals exchange) are performed in time steps. At time step t, the environment is in state $s_t \in S$, where S is the set of all possible states in the environment. The agent receives state s_t and then chooses action $a_t \in A$ based on policy π, where A is the set of all possible actions defined by the environment. After the environment receives information about the chosen action a_t from the agent, it calculates the reward $r_t = R(s_t, a_t, s_{t+1})$ and sends it to the agent in the form of feedback. At the same time, the environment transitions to a new state s_{t+1}. When this cycle is complete, we say that one time step has passed. This cycle can repeat forever or end when it reaches a terminal state or a maximum time step $t = T$. We call the triplet of signals (s_t, a_t, r_t) an experience. The time elapsed between $t = 0$ and the end of the environment is called an episode. A trajectory is a sequence of experiences during an episode, $\tau = (s_0, a_0, r_0), (s_1, a_1, r_1), \ldots$ [19].

In the field of creating adversarial malware samples, interactions between the agent and the environment occur in discrete time steps. The agent has a set of operations available for modifying PE files while maintaining the functionality of the malware. The goal of the agent is to perform a sequence of operations on the malware to prevent its detection.

2.1.4 Evolutionary algorithm-based approaches

Evolutionary algorithms (EA) are a useful tool for solving optimization problems. They are based on the Darwinian principle of evolution and attempt to mimic these processes. The search for the best or at least satisfactory solution to a problem takes the form of competition between gradually developing solutions within the population. Variants of EA include, for example, evolutionary strategies, evolutionary programming, genetic algorithms, and genetic programming. All of these variants share the same principle of operation but differ in their implementation.

When solving a problem, it is necessary first to define the representation of candidate solutions. These candidate solutions are called individuals or phenotypes. Since the phenotype can have a complex structure, an encoding is used to represent the individuals in an appropriate way, which is called a chromosome or genotype. Next, an initial population of individuals is created, with each individual representing a coded solution. Then, each member of the population is evaluated using a fitness function that numerically expresses the quality of the solution. The individuals with the best score are then selected and used to create a new generation. This is done using the crossover operator, which usually takes pairs of chromosomes and exchanges information between them to create new offspring. This is followed by the mutation operator, which changes a small portion of the offspring so that it is no longer just a mixture of parental genes. This introduces new genetic material into the new generation. This entire cycle (fitness evaluation, selection, crossover, mutation) is repeated until the termination condition is reached.

2.2 Selected attacks

To generate samples of malicious software, we utilized three distinct techniques: gradient-based techniques, evolutionary algorithm-based techniques, and reinforcement learning-based techniques. Specifically, we selected Partial DOS [20] and Full DOS [21] attacks from gradient-based techniques. From the evolutionary algorithms-based techniques, we chose GAMMA padding [22] and GAMMA section-injection [22] attacks. Finally, for reinforcement learning techniques, we selected the Gym-malware [23] attack. In this section, we describe each of these attacks in detail.

Partial and Full DOS attacks focus on modifying bytes in the DOS header of a portable executable (PE) file. The DOS header contains only two important pieces of information. The initial 2 bytes represent the magic number, while the final 4 bytes at offset 0x3C show the location of the PE signature in the NT headers. Partial DOS attacks modify only bytes within the range of 0x02 to 0x3B, inclusive. Full DOS attacks expand this range to include all bytes up to the PE signature.

The position of this signature may vary in individual files, but it can be found at offset 0x3C.

GAMMA padding and section-injection attacks are based on inserting parts extracted from benign files into malware files. These are black-box attacks. Gamma attacks are formalized as a constrained optimization problem. The goal is to minimize the probability of detection but also to minimize the size of the injected content. This optimization problem is solved using a genetic optimizer. First, a random matrix is created that represents the initial population of manipulation vectors. The algorithm then iterates in three steps: selection, crossover, and mutation. During selection, the objective function is evaluated, and the N best candidate manipulation vectors from the current population and the population created in the previous iteration are selected. This is followed by the crossover function, where the candidates from the previous step are modified by mixing the values of pairs of randomly selected candidate vectors, and a new set of N candidates is returned. The last operation is a mutation, which randomly changes the elements of each vector with a low probability. At each iteration, N queries are made to the target model to evaluate the objective function on new candidates and keep the best candidate population. When the maximum number of queries is reached or no further improvement in the objective function value is observed, the best manipulation vector from the current population is returned. The resulting adversarial malware sample is obtained by applying the optimal manipulation vector to the input malware sample through the manipulation operator.

The Gym-malware attack is based on reinforcement learning. The environment consists of a sample of malware and the attack target, which is an anti-malware engine. At each step, the agent receives feedback that is composed of a reward value and a vector of features that summarize the state of the environment. Based on the feedback, the agent selects mutations from a set of actions, such as adding a function to the import address table, manipulating existing section names, creating new sections, adding bytes to extra space at the end of sections, creating a new entry point that immediately jumps to the original entry point; manipulating debug information; packing or unpacking the file; modifying the header checksum; adding bytes to the end of the PE file. This process is repeated in several rounds, and rounds can be prematurely terminated if the agent bypasses the anti-malware engine before 10 mutations are completed.

3 Related Work

This section summarizes the publications on modern methods to create adversarial examples. The section is divided into several parts, depending on the area to which the method belongs, with all publications summarized and compared

Table 1 Summary of State-of-the-Art Adversarial Attacks against PE Malware Detection

Paper	Year	Attack framework	Technique	Target model	Knowledge	Dataset	Evasion rate (%)
[24]	2019	AIMED	Evolutionary algorithm	GBDT, AVs	Black-box	6880	82
[25]	2020	MDEA	Evolutionary algorithm	MalConv	Black-box	14288	93
[22]	2021	GAMMA	Evolutionary algorithm	GBDT, MalConv	Black-box	30000	65–96
[23]	2018	Gym-malware	Reinforcement learning	GBDT	Black-box	58300	10–24
[26]	2020	MAB-malware	Reinforcement learning	GBDT, MalConv, AVs	Black-box	5000	32–97
[27]	2019	DQEAF	Reinforcement learning	GBDT	Black-box	9600	70
[28]	2021	AIMED-RL	Reinforcement learning	GBDT	Black-box	4387	30–43
[29]	2018	AMB	Gradient	MalConv	White-box	13195	60
[30]	2018	FGSM	Gradient	MalConv	White-box	18239	92–98
[20]	2019	Partial DOS	Gradient	MalConv	White-box	60	86
[31]	2018	FGM Append/Slack	Gradient	MalConv	White-box	16300000	30–70
[21]	2020	RAMEN	Gradient	MalConv	white-box	104	80–93
[32]	2017	MalGAN	GAN	SVM, RF, MLP, …	Black-box	180000	92–95
[33]	2019	Improved-MalGAN	GAN	RF, MLP	Black-box	125029	100
[34]	2020	GAPGAN	GAN	MalConv	Black-box	42872	100

based on the attacker's knowledge, technique, dataset size, target model, and recorded results in Table 1.

3.1 Evolutionary algorithm-based attacks

In [24], an AIMED system was designed and implemented to generate adversarial examples using genetic programming by Castro et al. The system enables the automatic finding of optimized modifications that are applied to previously detected malware and lead to its incorrect evaluation by the malware classifier. It is ensured that all generated adversarial examples are valid PE files. The system implements genetic operations such as selection, crossover, and mutation. If modified PE malware cannot bypass the malware detector, genetic operations are repeated until the generated adversary PE malware can bypass the malware classifier. Experiments have shown that the time to generate successful adversarial examples is reduced by up to 50% compared to random approaches. Furthermore, adversarial examples generated using the given malware classifier were shown to be successful against other malware detectors in 82% of cases.

Wang and Miikkulainen proposed an adversarial malware detection model named MDEA [25]. This model combines the convolutional neural network to classify raw data from malicious binary and evolutionary optimization to modify detected malware. The action space consists of 10 different methods to modify binary programs. The genetic algorithm evolves different action sequences by selecting actions from the action space until the generated adversarial malware can bypass the target malware detectors. After the successful discovery of action sequences, these sequences are applied to the corresponding malware samples and create a new training set for the detection model. Unlike AIMED, malware samples generated by MDEA are not tested for functionality.

Demetrio et al. introduced a black-box attack framework called GAMMA [22]. The black-box attack is the most challenging case since the attacker knows nothing about the target classifier besides the final prediction label. GAMMA attacks are based on the principle of injecting harmless content extracted from goodware into malicious files. Harmless content is inserted into some newly created section or at the end of the file while the functionality of the file is preserved. The attack is formalized as a constrained optimization problem that minimizes the probability of escaping detection and also penalizes the size of the injected content through a specific penalty.

3.2 Reinforcement learning-based attacks

Anderson et al. focused on automating the manipulation of malicious PE files in [23]. The goal is to modify the original malicious PE file so that it is no longer detected as malicious and, at the same time, its format and functionality are not violated. They proposed an attack known as Gym-malware. This is a black-box attack based on reinforcement learning. The authors defined the RL agent's action space as a set of binary manipulation actions. Over time, the agent learns which combinations of actions make malware more likely to bypass antivirus systems.

Song et al. proposed the MAB-Malware framework [26] based on reinforcement learning to generate adversarial PE malware examples. The action selection problem is modeled as a multi-armed bandit problem. The results showed that the MAB-Malware framework achieves an evasion rate of 74% to 97% against machine learning detectors (EMBER GBDT [35], and MalConv [36]) and an evasion rate of 32% to 48% against commercial antivirus (AV). Furthermore, they also showed that the transferability of adversarial attacks between ML-based classifiers (i.e., adversarial examples generated against one classifier can be used successfully against another) is greater than 80%, and the transferability of attacks between pure ML and commercial AV is only up to 7%.

Fang et al. proposed a framework named DQEAF [27] that uses reinforcement learning to evade antimalware engines. DQEAF is similar in methodology to Gym-malware but has many benefits and a higher evasion success rate of adversarial examples compared to it. DQEAF uses a subset of modifications used in Gym-malware and ensures that all modifications will not cause corruption in the modified malware files. DQEAF is able to reduce instability caused by higher dimensions by representing executable files with only 513 features, which is much lower than that in Gym-malware. DQEAF takes priority into account when replaying past transitions. This helps to replay higher-value transitions more frequently and thus optimize RL algorithm.

Another reinforcement learning approach is presented in [28], in which Labaca-Castro et al. presented the AIMED-RL adversarial attack framework. This attack can generate adversarial examples that lead machine learning models to misclassify malicious files without compromising their functionality. The authors demonstrated the importance of a penalty technique and introduced a new penalization for the reward function with the aim of increasing the diversity of the generated sequences of modifications while minimizing the number of modifications. The results showed that the agents with penalty outperform the agents without penalty in terms of both the best and the average evasion rates.

3.3 Gradient-based attacks

Kolosnjaji et al. in [29] introduced a gradient-based white-box attack to generate adversarial malware binaries against MalConv. A white-box scenario occurs when the attacker gets access to the system and may examine its internal configuration or training datasets. The basic idea of the attack

is to manipulate some bytes of each malicious software to maximize the likelihood that the input samples are classified as benign. To ensure that malicious binary functionality is maintained, only padding bytes at the end of the file are considered. The results show that the evasion rate is correlated with the length of the injected sequence and, despite the fact that less than 1% of the bytes are modified, the modified binary evades the target network with high probability and that the precision of MalConv can be reduced by more than 50%.

In [30], Kreuk et al. presented an improved gradient-based white-box attack method against MalConv. The authors proposed two methods for inserting a sequence of bytes into a file; the payload is inserted either at the end of the file or into slack regions. Unlike [29], the evasion rate of [30] is invariant to the length of the injected sequence.

Demetrio et al. in [20] presented a gradient-based variant white-box attack that is similar to [29]. The main difference is that [29] injects adversarial bytes to the end of the PE file, while this attack is limited to changing bytes within a specific disk operating system (DOS) header in the PE header. The results show that a change of a few bytes is sufficient to evade MalConv with high probability.

Suciu et al. in [31] describe the FGM append and slack attacks and compare their effectiveness against MalConv. This attack is similar to [30], and the experimental results show that inserting adversarial perturbation between sections (FGM slack attack) is more efficient than appending modified bytes at the end of the file (FGM append attack). However, the FGM append attack is able to achieve higher evasion rates, albeit with a larger perturbation size.

Demetrio et al. in [21] propose RAMEN, a general adversarial attack framework against PE malware detectors. This framework generalizes and includes previous attacks against machine learning models, as well as three new attacks based on manipulations of the PE file format that preserve its functionality. The first attack is a full DOS attack, which edits all the available bytes in the DOS header. The second attack is called Extend, which enlarges the DOS header, thus enabling manipulation of the extra DOS bytes. The third is the Shift attack, which shifts the content of the first section, creating additional space for the adversarial payload.

3.4 Generative adversarial network-based attacks

Hu and Tan proposed in [32] a model called MalGAN, which is based on a generative adversarial network (GAN). This model enables the generation of adversarial malware examples that are capable of bypassing black-box ML-based detection models. MalGAN includes two neural networks: a generator and a substitute detector. The generator is used to generate adversarial examples, which are dynamically generated according to the feedback of the malware detector and

are able to fool the substitute detector. The results showed that MalGAN can reduce the detector's accuracy to almost zero.

Later, Kawai et al. presented Improved MalGAN in [33]. The authors discuss the problems of MalGAN and try to improve them. For example, the original MalGAN uses multiple malware samples to train MalGAN, in contrast, the improved MalGAN uses only one malware sample. Furthermore, the original MalGAN trains the generator and the malware detector with the same application programming interface (API) call list, while the Improved MalGAN trains with different API call lists.

Yuan et al. in [34] introduced GAPGAN, a GAN-based black-box adversarial attack framework. GAPGAN allows end-to-end black-box attacks at the byte level against deep learning-based malware binaries detection. In this approach, a generator and a discriminator are trained concurrently. The generator is used to generate adversarial payloads that are appended to the end of the original data to craft a malware adversarial sample while ensuring the preservation of their functionality. The discriminator tries to imitate the black-box malware detector to recognize both the original benign samples and the adversarial samples generated. When the training process is completed, the trained generator is able to generate an adversarial sample in less than 20 milliseconds. Experiments show that GAPGAN is capable of achieving a 100% attack success rate against the MalConv malware detector by only inserting adversarial payloads with the size of 2.5% of the total length of the input malware samples.

4 Experiments

This section describes the setup and procedure for each experiment. First, the hardware configuration is introduced, followed by a description of the datasets used. Next, the setup of the different algorithms used to generate adversarial samples is described. Finally, the experiments performed are described.

4.1 Experimental setup

The experiments are carried out on the NVIDIA DGX Station A100 server. This server is equipped with an AMD EPYC 7742 processor with a base frequency of 2.25 GHz, 64 cores, and 512 gigabytes of RAM. We also use a virtual machine with a Windows 11 operating system and another virtual machine with a Kali Linux operating system for testing and analysis.

We use two datasets for our experiments. The first dataset contains a total of 3,625 harmless executable files, which were collected from a newly installed Windows 11 system. The second dataset contains 3,625 malicious executable files

obtained from the VirusShare repository [37]. All the files in both datasets are PE files.

4.2 Attack settings

In total, we compare five adversarial attack strategies. Partial DOS and Full DOS attacks are performed in a white-box setting against the MalConv detector, and the maximum number of iterations is set to 50. GAMMA padding and GAMMA section-injection attacks are performed in a black-box setting against the MalConv detector with the maximum number of queries set to 500, regularization parameter set to 10^{-5}, and a total of 100.data sections extracted from benign programs used as injection content. The last attack we use is the Gym-malware attack, with its default configuration performed in a black-box setting against the GDBT detector. The Gym-malware model was trained on a dataset of 3,000 malicious samples and a validation set of 1,000 files.

4.3 Evaluation metrics

We use several metrics to evaluate the experiments. A key metric in the area of adversarial machine learning is the evasion rate (ER), the proportion of malware files misclassified by the target malware classifier, and can be calculated as follows:

$$ER = \frac{misclassified}{total} \tag{1}$$

where $misclassified$ is the number of malware samples misclassified as benign, and $total$ is the total number of files submitted to the target classifier after discarding files that were already incorrectly predicted before modification.

The evasion rate mentioned above is a universal metric that can be used to evaluate both single attacks and combinations of attacks. In both cases, we are interested in the percentage of malware that escaped detection by the antivirus program. Additionally, we use the following metrics to evaluate the combination of attacks.

The first two metrics that we chose to evaluate the success of the combination are the absolute improvement and the relative improvement in the evasion rate when using the second attack in the combination compared to the first attack. Absolute improvement (AI) can be described by the following formula:

$$AI = ER_C - ER_1 \tag{2}$$

where ER_C is the total evasion rate when using a combination of methods, and ER_1 is the evasion rate after using the first attack in the combination alone. The result is the percentage increase in evasion rate between the first and second attacks

in the combination. For example, if the evasion rate after the first attack in the combination is 0.01 and after the second attack is 0.1, then the absolute improvement is 0.09, which means that the second attack improved the overall evasion rate by 9%.

Similarly, the relative improvement (RI) can be expressed using the formula:

$$RI = \frac{ER_C - ER_1}{ER_1} \tag{3}$$

where the meaning of the variables is the same as in the previous formula (2). However, in the previous case, the result expressed a percentage increase over all samples tested. In the case of relative improvement, we limit ourselves to the set of samples that escaped the antivirus program after the first attack. For example, if the evasion rate after the first attack of the combination is 0.01 and after the second attack it is 0.1, then the relative improvement is 9, i.e., the second attack improved the evasion rate of the first attack by 900%.

Next, we need to compare the combination of attacks with performing the attacks separately to see if the combination of attacks adds any value. To do this, we use a simple comparison of the evasion rate of the combination of attacks with the evasion rate of the attacks performed separately. We call it evasion rate comparison (ERC), and it has the following calculation:

$$ERC = ER_C - \max(ER_1, ER_2) \tag{4}$$

where ER_C is the evasion rate of the combination attack, while ER_1 and ER_2 are the evasion rates of the first and second attacks, respectively. If the result is positive, it means that the combination of attacks performed better than the combination of the two attacks in the combination that would have been performed alone. If the result is negative or zero, it means that the execution of the attack combination was pointless because one of the attacks that were part of the combination performed better or was equal to the combination in terms of evasion rate.

4.4 Experiments description

We present four experiments that explore different characteristics of the adversarial attacks mentioned above. We designed the experiments to capture the fundamental characteristics that efficient and successful adversarial attacks should have if applicable in real-world scenarios. Firstly, we focus on the efficiency of generators, i.e., how fast they can generate new adversarial samples and how large the samples will be. Next, we focus on the key feature of any adversarial example, evasiveness against malware detectors. Lastly, we

explore how a combination of different generators can create new and evasive adversarial malware samples.

4.4.1 Sample generation time

In the first experiment, we measure the time it takes to generate individual samples using all the aforementioned selected algorithms. These results, along with other data collected during the experiments, can help compare the effectiveness of various generators and determine the most effective method to generate adversarial malware samples.

4.4.2 Sample size

In the second experiment, we investigate how the size of the original malware samples changes by applying various adversarial malware generators. Generally, the attacker's goal is to minimize the increase in the size of the generated adversarial files to make it harder to distinguish them from the original malware samples.

4.4.3 Bypassing commercial AV products

In the third experiment, we analyze the effectiveness of created adversarial malware samples against real-world AV detectors. Based on a comparative study [38], 10 AV programs were selected for experimentation, and their names were intentionally anonymized in the following results. Note that in the subsequent results, only nine AVs are listed as two of the selected AVs reported the same results.

The modified malware files from different adversarial algorithms are submitted to the VirusTotal server [39] to obtain the evasion rate for each adversarial malware generator. To avoid bias in the results, we only analyzed malware samples that were correctly classified by all selected AV products before modification. We also discarded samples from which we were unable to obtain file analysis from VirusTotal, e.g., due to the broken behavior of modified examples. In total, we use a set of 530 genuine malware samples along with a modified version for each malware generator.

4.4.4 Combination of multiple techniques

In the last experiment, we test the effectiveness of using a combination of methods to generate malware samples [40]. The goal of this experiment is to test whether using multiple adversarial example generators per malware sample would significantly increase the malware evasion rate.

An overview of the experiment is shown in Fig. 1. First, the original malware samples are processed by the first generator. These modified samples are then tested against a real AV detector that is not part of the generator. The result is a set of samples divided into two sets. The first set consists of
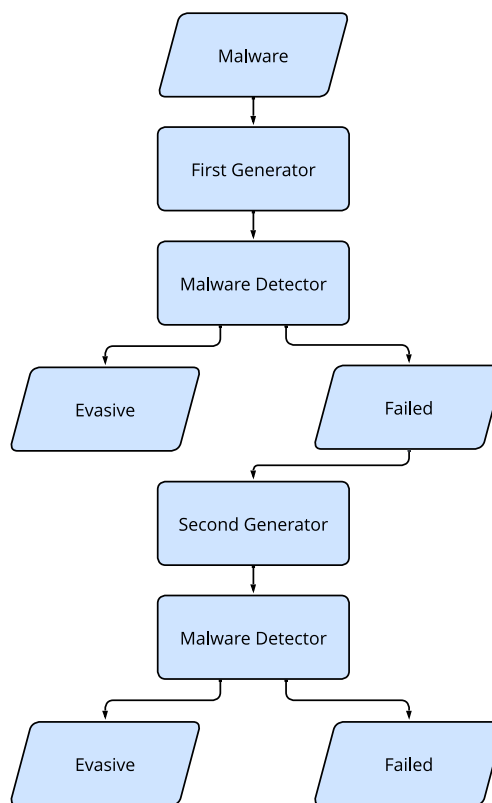
Fig. 1 Method for generating adversarial examples by combining two generators

evasive examples that successfully evaded the given malware detector, and this set is no longer processed. In contrast to *adversarial examples*, which are generated against the target classifier, *evasive examples* are samples that have evaded detection by the AV program, although this AV program was not used to generate these samples. The second set consists of *failed examples* that failed to evade the detector and are used as input to the second generator. The second generator processes the *failed examples*, and the resulting modified samples are again tested against the real AV detector. The result is again a set of samples divided into two sets: *evasive* and *failed*. The set of *evasive examples* produced by the first generator and the set of *evasive examples* produced by the second generator together form the set of resulting successful adversarial examples produced by combining these two generators. The *failed examples* obtained after using the second generator are the resulting samples that did not evade detection.

5 Results

This section presents the recorded results of the individual experiments described in Sect. 4.

5.1 Sample generation time

Firstly, we look at the results of the sample generation time. Average times in seconds and standard deviation for each attack are listed in the box plot in Fig. 2.

From Fig. 2, we can see that Gym-malware took the least amount of time, on average less than 6 s, to generate adversarial examples with some outliers below 100 s. It should be noted that Gym-malware requires a preceding training phase that is not taken into account in this experiment. On the contrary, the Full DOS attack recorded the longest time to create adversarial examples, with an average duration of more than 160 s. The remaining three methods achieved similar sample generation times of around 100 s in spite of the fact that the results of the GAMMA section-injection attack contain several outliers with extensive long durations of more than 300 s. However, the measured times may be affected by the settings of individual algorithms.

5.2 Sample size

Secondly, we present the results of the sample size experiment in Table 2, which contains the average and standard deviations of sample size changes in bytes for each tested attack. Note that Partial and Full DOS attacks are based on changing bytes in the DOS header. Thus, modifying the malware samples by these methods does not alter the size of the resulting adversarial examples. On the other hand, the remaining tested methods can change the initial file size.

The GAMMA padding and section-injection attacks are based on inserting parts extracted from benign files into the malware file. In the case of the GAMMA padding attack, the file size increased by 223,605 bytes on average, and in the case of the GAMMA section-injection attack, it increased by 1,940,352 bytes. As we can see from the Standard Deviation column, the adversarial examples generated by the GAMMA section-injection attack exhibit significant differences in the final file sizes.
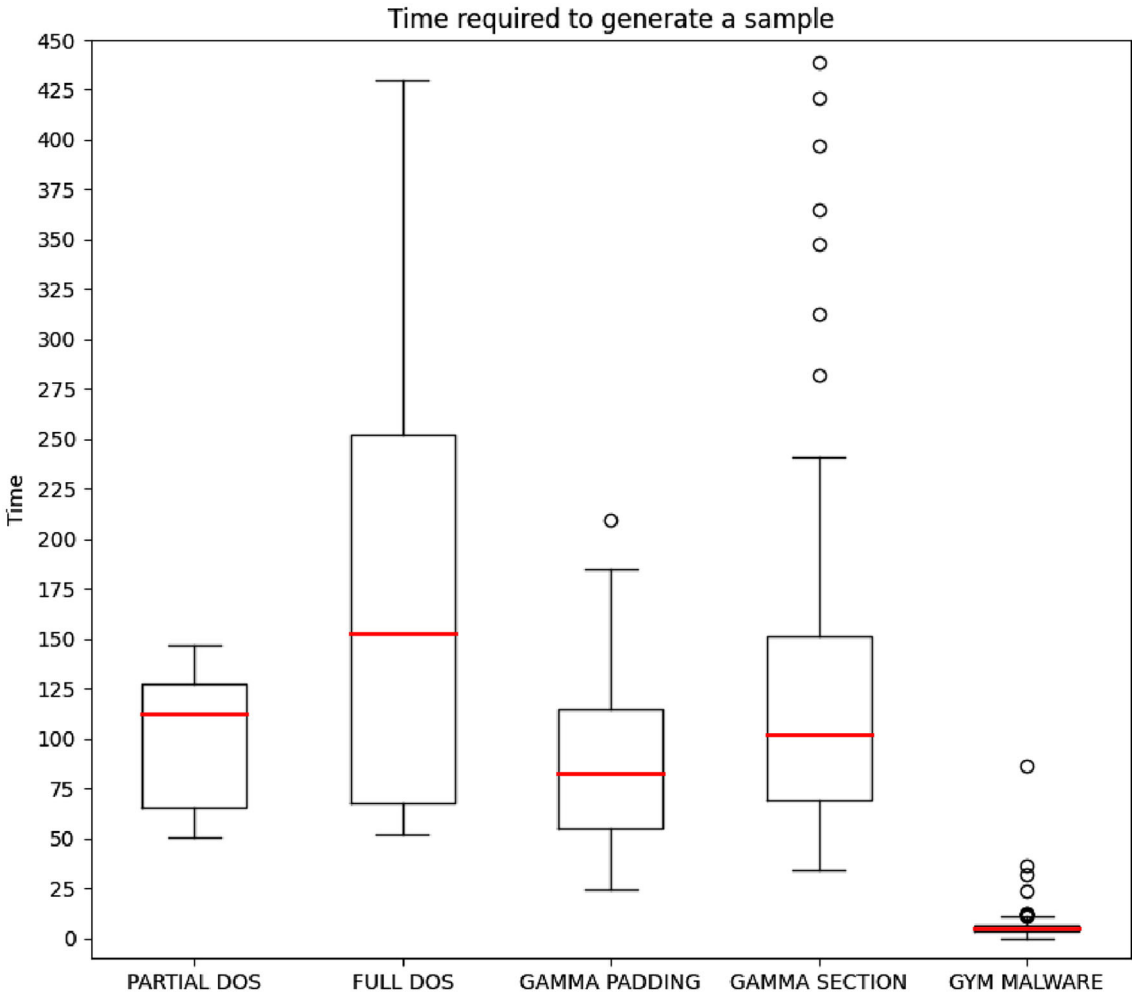

Fig. 2 Time required to generate a sample for each sample generator

Table 2 Changes to the sample size of generated adversarial samples for each generator

Attack	Average Size [B]	Standard Deviation [B]
Partial DOS	0	0
Full DOS	0	0
GAMMA Padding	223,604.57	48,403.25
GAMMA Section-injection	1,940,351.63	78,088,981.28
Gym-malware	− 149,272.75	754,660.5

Table 3 Evasion rate (in %) of generated adversarial samples achieved against AV products on the VirusTotal server

Attack	AV1	AV2	AV3	AV4	AV5	AV6	AV7	AV8	AV9	Average
GAMMA padding	0.00	1.79	0.45	0.22	0.45	0.90	1.34	0.56	0.45	0.68
Partial DOS	0.78	2.57	0.78	1.01	0.78	0.78	1.90	1.45	0.78	1.21
Full DOS	0.67	1.34	0.78	0.90	0.78	0.78	4.14	1.23	0.78	1.27
GAMMA section-injection	18.46	5.37	6.38	4.36	4.47	9.06	43.62	1.23	5.37	10.92
Gym-malware	45.53	19.02	44.86	67.23	41.61	53.58	53.80	26.51	44.86	44.11

The Gym-malware attack uses various types of file manipulations, which can cause the file size to reduce, increase, or remain unchanged based on the chosen modification. On average, the file size was reduced by 149,273 bytes. The observed file size reduction was probably due to the authors' implementation of file manipulations, as they used the LIEF library [41], which significantly alters the structure of the initial malware file.

5.3 Bypassing commercial AV products

Thirdly, we list the effectiveness of generated adversarial samples on commercially available AV programs. The results are shown in Table 3, where each column represents one of the selected AV programs, and each row represents one of the algorithms used to generate adversarial samples. The values in the table represent the achieved evasion rates, expressed as a percentage, for the corresponding algorithm and AV products.

The Gym-malware attack achieved the highest evasion rates among all selected AV programs, successfully bypassing the top AVs 19.02% to 67.23% of the time. The second-best results were recorded by the GAMMA section-injection attack, which recorded evasion rates between 1.23% and 43.62%.

In contrast, the GAMMA padding attack achieved the worst results, failing to mislead any detector tested in more than 1.8% of cases. Full and Partial DOS attacks scored slightly better than the GAMMA padding attack, with the Full DOS marginally outperforming the Partial DOS attack.

5.4 Combination of multiple techniques

Based on the results from the previous experiment, we chose the three most successful adversarial example generators and tested all nine possible combinations. Namely, we used Gym-malware, GAMMA section-injection, and Full DOS adversarial malware generators.

This section contains two types of tables. The first type of table lists the measured minimum, average, and maximum values for particular AVs across the nine combinations of the three selected generators. The second type of table lists the measured minimum, average, and maximum values for a particular combination of generators across the nine AVs. The First Generator column contains the first generator used, and the Second Generator column contains the second generator used as described in Sect. 4.4.4.

Evasion rate

First, we present the results of the evasion rate metric. For all AVs, we examine the minimum, average, and maximum of the results of all combinations of generators tested in the experiment. These results can be found in Table 4. For the minimum values of the evasion rate, we can see that none of the antivirus programs reached a detection rate of 100%. On the other hand, all these values are relatively low compared to the average and maximum values, which tells us that some of the nine generator combinations were not very successful. The average evasion rates range from about 18% to 49%, and the maximum values range from 30% to 78%. The best result was achieved by combinations of generators against AV7, where the average evasion rate is around 49%, while the least successful was against AV2, where the average evasion rate is around 18%.

Table 4 Evasion rate (in %) for each AV using all combinations of generators

AV	Minimum	Average	Maximum
AV1	0.78	32.39	55.26
AV2	1.45	17.79	29.53
AV3	0.90	31.15	63.09
AV4	1.45	38.59	78.19
AV5	0.78	26.76	57.61
AV6	0.90	35.91	73.60
AV7	5.26	49.32	74.50
AV8	1.57	17.80	41.39
AV9	0.78	30.79	62.75

Table 6 Absolute improvement (in %) for each AV using all combinations of generators

AV	Minimum	Average	Maximum
AV1	0.11	10.84	29.98
AV2	0.11	9.21	22.48
AV3	0.11	13.81	41.16
AV4	0.00	14.43	60.74
AV5	0.00	11.14	35.01
AV6	0.11	14.77	50.22
AV7	0.90	15.46	40.72
AV8	0.34	8.14	26.51
AV9	0.00	13.78	41.39

Table 5 shows the results of the evasion rate achieved by each combination of generators. Here, we can see that the most successful combination was the one in which the Gym-malware generator was used twice in a row. This achieved a minimum evasion rate of around 30% for all AVs. The average value for this combination is around 58%, and for at least one AV, we achieved an evasion rate of around 78% with this combination. On the other hand, the worst combination in terms of evasion rate is the one in which the Full DOS generator was used twice. In this case, we have an average evasion rate of about 2%, while for all other combinations, this value exceeds 10%, and for some even more significantly. The maximum value of the evasion rate for this combination is about 5%, while for the others, we have at least about 44%.

Absolute improvement

Next, to determine the effectiveness of using a combination of generators instead of individual generators alone, we evaluate the metric that we identified as an absolute improvement in Sect. 4.3.

First, we focus on Table 6, which shows the absolute improvement values for each AV in the nine combinations of generators. Here, we can see that for some AVs, we were

unable to improve the evasion rate by applying the second generator. Specifically, this refers to the minimum value for AV4, AV5, and AV9. Table 7 helps us to identify the relevant generators. We can see that the only non-improving combinations are the ones in which we use the same generator twice, namely, the Full DOS generator or the GAMMA section-injection generator. This may indicate that the use of these combinations is not entirely effective. For the average values in Table 6, we can see that they do not differ significantly, in all cases between about 8% and 15%. For the maximum values, we have a slightly higher range, approximately 22% to 61%. This means that if we use a second generator, we will improve the evasion rate by about 10% on average after using the first generator.

However, significantly more intriguing is Table 7, which shows the absolute improvement for each generator combination against all AVs. Here, we can see that the use of Full DOS as the second generator does not result in a significant absolute improvement in the evasion rate for the minimum, average, and maximum values. This means that using Full DOS as the second generator in a combination is the least effective. On the other hand, we can see that we get the best absolute improvement by using the Gym-malware method as the second generator in the combination.

Table 5 Evasion rate (in %) for each generator combination against all AVs

First Generator	Second Generator	Minimum	Average	Maximum
Full DOS	Full DOS	0.78	1.54	5.26
Full DOS	GAMMA section-injection	1.57	10.48	43.96
Full DOS	Gym-malware	23.15	38.69	61.63
GAMMA section-injection	Full DOS	6.26	15.05	45.30
GAMMA section-injection	GAMMA section-injection	1.90	14.03	44.52
GAMMA section-injection	Gym-malware	25.39	46.97	74.50
Gym-malware	Full DOS	26.51	46.16	67.34
Gym-malware	GAMMA section-injection	27.18	49.22	67.79
Gym-malware	Gym-malware	29.53	58.34	78.19

Table 7 Absolute improvement (in %) for each generator combination against all AVs

First Generator	Second Generator	Minimum	Average	Maximum
Full DOS	Full DOS	0.00	0.27	1.12
Full DOS	GAMMA section-injection	0.67	9.21	39.82
Full DOS	Gym-malware	21.92	37.42	60.74
GAMMA section-injection	Full DOS	0.78	4.13	5.93
GAMMA section-injection	GAMMA section-injection	0.00	3.11	10.29
GAMMA section-injection	Gym-malware	20.02	36.04	51.23
Gym-malware	Full DOS	0.11	2.05	7.49
Gym-malware	GAMMA section-injection	0.56	5.11	10.63
Gym-malware	Gym-malware	7.72	14.23	20.02

In Table 7, we can observe another interesting fact. As we have already mentioned, the use of two identical generators in combination is not very effective. However, this statement does not apply to the Gym-malware generator. In contrast, if we use the Gym-malware generator as the first generator in the combination, then the best choice to select the second generator seems to be the Gym-malware generator again. Full DOS and GAMMA section-injection generators have minimal absolute improvements in the role of the second generator. These results show that the Gym-malware generator is successful in both cases, used alone and in combination with itself.

The best absolute improvement values are achieved when we choose Full DOS as the first generator and Gym-malware as the second. This results in an absolute improvement in the evasion rate in all AVs of at least around 22%, on average 37%, and up to a maximum of 61%. On the other hand, the worst results in terms of absolute evasion rate improvement are obtained when we choose Full DOS as both generators. In this case, we obtain a minimum absolute improvement of 0%, an average of 0.3%, and a maximum of 1.1% across all antivirus programs. We can conclude that the Full DOS generator is likely to be the least successful generator, both when used alone and in combination.

Relative improvement

We follow the absolute improvement results with a relative improvement in the evasion rate. Analogously to the improvement in the absolute evasion rate, we can see the results of relative improvement in Table 8. We can see that AV7 performed the best on average in terms of relative improvement of the evasion rate. Conversely, AV4 performed the worst, where, in some cases, the second generator managed to increase the number of successfully modified samples (which evaded detection) by more than 67 times.

Table 9 confirms that the Full DOS generator, which was used as the second generator in combination, does not increase the evasion rate significantly. On the other hand, if

Table 8 Relative improvement (in %) for each AV using all combinations of generators

AV	Minimum	Average	Maximum
AV1	0.61	788.88	4466.67
AV2	8.33	307.66	1675.00
AV3	3.74	732.66	5014.29
AV4	0.00	914.67	6787.50
AV5	0.00	633.29	4257.14
AV6	1.46	846.29	6000.00
AV7	2.05	232.70	983.78
AV8	3.80	510.39	2154.55
AV9	0.00	758.01	5285.71

this generator is chosen as the first generator in the combination, the GAMMA section-injection or Gym-malware generator can increase the number of samples that evade detection by up to 67 times. This shows that the Full DOS generator is not a very strong generator on its own. Regarding the Gym-malware generator, we can see that when it is used as the first generator in a combination, the second generator does not significantly increase the result, even when Gym-malware is used again as the second generator. However, the Gym-malware generator is very successful on its own as it achieves a significantly higher evasion rate than other generators. Therefore, a relative improvement in the evasion rate from 39% to 56% can cause a drastic increase in the total evasion rate of the combination (up to 20%). We can also see that when the Gym-malware generator is used as a second generator in other combinations, there are huge relative improvements over the first generator. Again, we can see that Gym-malware is very effective when used in any combination of generators.

Evasion rate comparison

The last metric examined, evasion rate comparison, helps us find the answer to the question of whether it is better to use

Table 9 Relative improvement (in %) for each generator combination against all AVs

First Generator	Second Generator	Minimum	Average	Maximum
Full DOS	Full DOS	0.00	18.93	62.50
Full DOS	GAMMA section-injection	75.00	715.51	2400.00
Full DOS	Gym-malware	983.78	4027.99	6787.50
GAMMA section-injection	Full DOS	3.85	104.90	409.09
GAMMA section-injection	GAMMA section-injection	0.00	58.50	181.25
GAMMA section-injection	Gym-malware	70.77	741.93	2154.55
Gym-malware	Full DOS	0.17	7.31	39.41
Gym-malware	GAMMA section-injection	0.83	13.60	42.94
Gym-malware	Gym-malware	16.31	35.90	56.12

Table 10 Evasion rate comparison (in %) for each AV using all combinations of generators

AV	Minimum	Average	Maximum
AV1	−14.88	0.87	9.73
AV2	0.11	5.28	10.52
AV3	−4.81	4.02	18.23
AV4	−11.63	−0.31	10.96
AV5	−7.49	2.06	16.00
AV6	−5.82	3.03	20.02
AV7	−8.95	4.43	20.69
AV8	−3.36	2.52	14.88
AV9	−2.69	3.99	17.90

individual generators separately or to use them in combination. A negative value of this metric indicates that we would achieve a better evasion rate by using the better of the two generators individually (better in terms of evasion rate). On the contrary, a positive value indicates that we recorded a better result using the combination of both generators.

Looking at the minimum values in Table 10, we can see that for almost all AVs, there was a combination of generators that was not effective due to the negative values in this column. The exception is AV2, where we achieved a better evasion rate for all combinations of generators used. However, the average and maximum values show that, in most cases, the combination is more effective than the better of the two generators used separately. Only AV4 has a negative average value, which means that a separate use of a single generator is a better option against this AV. On the other hand, when attacking AV2, it is better to use a combination of generators as it achieves the highest average evasion rate comparison value.

Table 11 shows that combining generators using Full DOS as the first generator and either GAMMA section-injection or Gym-malware as the second generator yields negative results, as evidenced by both the minimum and average values being negative. Based on the results from the previous parts of this section, we can say that it is more advantageous to use GAMMA section-injection and Gym-malware generators separately in such cases. We can also see other negative values in the minimum value column for the combination of GAMMA section-injection and Gym-malware generators used in this order. Nonetheless, the average value of the combination is positive, indicating that it is beneficial to use this combination on average. For the remaining combinations, we can conclude that using them results in a better evasion rate than using the better of the two generators separately.

6 Discussion and future work

To the best of our knowledge, this is the first paper that evaluates adversarial malware generators in terms of generation time, sample size, and, most importantly, evasion rate against commercially available AVs.

In general, our measured results (see Table 3) are significantly worse than the measurements done by the original authors (see Table 1). For example, the reported evasion rates in the original publications are almost always higher than the average evasion rates we measured against real-world AVs. This is expected as samples are generated against different malware detectors (GBDT or MalConv), which are most definitely easier to bypass than top-tier AVs. However, the decrease in evasion rate between the authors' reported numbers and measured values in our experiments is significant. The Partial and Full DOS attacks evade detection of MalConv in over 80% of cases yet can mislead AVs in no more than 1.27% of time. An even bigger difference can be seen for the GAMMA padding attack, which reported an evasion rate of over 94% against MalConv but achieved less than 0.7% against our tested AVs. Less significant contrast could be found for the GAMMA section-injection generator, which evaded real-world AVs in over 10% of cases while reporting over 96% evasion rate against MalConv. The only exception in our experiments is the Gym-malware generator, where Enderson et al. reported in the original publication

Table 11 Evasion rate comparison (in %) for each generator combination against all AVs

First Generator	Second Generator	Minimum	Average	Maximum
Full DOS	Full DOS	0.00	0.27	1.12
Full DOS	GAMMA section-injection	−2.80	−0.45	1.57
Full DOS	Gym-malware	−14.88	−5.42	4.81
GAMMA section-injection	Full DOS	0.78	4.13	5.93
GAMMA section-injection	GAMMA section-injection	0.00	3.11	10.29
GAMMA section-injection	Gym-malware	−11.63	2.86	20.69
Gym-malware	Full DOS	0.11	2.05	7.49
Gym-malware	GAMMA section-injection	0.56	5.11	10.63
Gym-malware	Gym-malware	7.72	14.23	20.02

evasion rates of only 10–24% against the GBDT classifier yet scored significantly higher in our testing, evading AVs in over 44% of cases on average. Furthermore, the authors of the GAMMA attacks reported that the GAMA section-injection attack augments the size of the original malware file by up to 880 KB, while our measurements show up to a 1.9 MB increase. These findings suggest that MalConv may not be a suitable option as a surrogate model when trying to bypass top-tier AVs.

However, the Partial and Full DOS attacks could be effective in specific scenarios. For example, when only a single malware detector is targeted, it can achieve high evasion rates while minimally modifying input binary as reported in the original publications. Similarly, the GAMMA generator seems to be highly successful in evading target classifier, and we believe it could be further improved by incorporating more sophisticated PE file manipulations to better generalize to unseen antivirus engines. Additionally, it should be added that some of the samples generated by Gym-malware were found not to preserve the original functionality [42], thus substantially hindering the real-world usability of this generator.

Contrary to the original publications, we explored the possibility of combining different generators to further improve the evasive capabilities of adversarial examples. Based on the results of our experiments, we have seen that even less-performing attacks (e.g., Full DOS) can significantly improve when combined with a more proficient generator (e.g., Gym-malware). Interestingly, we have seen measurable differences even when the same technique was repeated twice, suggesting that the default configuration used by the authors could be altered for higher evasion rates.

It must be noted that the discrepancy between results reported by the authors of the generators and our experiments could be caused by the datasets used. This highlights the importance of using representative samples for evaluation to ensure the most accurate and reproducible measurements.

This work compared the three main types of adversarial malware generators: reinforcement learning-based, gradient-based, and evolutionary algorithm-based. We did not include GAN-based attacks because, at the moment, these attacks are not yet fully explored, and many GAN generators cannot create functioning malicious adversarial examples. Nonetheless, these attacks seem promising and should be considered in future work.

Even though our selection of AVs is limited, we chose the current top-tier AVs that represent the majority of the consumer market. Based on the reported results, Gym-malware is a threat even to the leading AVs, and better safeguards should be implemented to defend against this attack. For future research, we encourage authors to include other malware detectors, e.g., academic ML-based classifiers, to gain better insight into the accuracy and sensibility of these new detection models against adversarial attacks.

For future experiments, we propose to study in more detail how the sample generation time is affected by the input genuine malware size and investigate the correlation between the sample generation time and the evasion rate of the resulting adversarial examples. Further experiments could be done in the area of combining generators where more than two generators could be combined to achieve even higher evasion rates.

7 Conclusion

In this paper, we explored the use of adversarial learning techniques in malware detection. Contrary to contemporary survey papers from the area of adversarial malware, we focused on experimental evaluation of state-of-the-art methods rather than theoretical comparison. Our goal was to apply existing methods for generating adversarial malware samples, test their effectiveness against selected malware detectors, and compare the evasion rates achieved and the practical applicability of these methods.

For our experiments, we chose five adversarial malware sample generators: Partial DOS, Full DOS, GAMMA padding, GAMMA section-injection, and Gym-malware.

This selection represents a spectrum of adversarial techniques based on gradient, evolutionary algorithms, and reinforcement learning. These adversarial malware generators were evaluated on nine commercially available antivirus products.

To validate and compare the different characteristics and properties of the methods used, we performed four experiments. These included tracking the time taken to generate samples, changes in sample size after applying adversarial modifications, testing effectiveness against antivirus programs, and evaluating combinations of generators.

The results indicate that making optimized modifications to previously detected malware can cause the classifier to misclassify the file and label it as benign. Furthermore, the study confirmed that generated malware samples could be used successfully against detection models other than those used to generate them. Using combination attacks, a significant percentage of new samples that could evade detection by antivirus programs were created.

Experiments showed that the Gym-malware generator, which uses a reinforcement learning approach, has the greatest practical potential. This generator produced malware samples in the shortest time, with an average sample generation time of 5.73 s. The Gym-malware generator also achieved the highest evasion rate against all selected antivirus products, with an average evasion rate of 44.11% against nine AVs. Furthermore, the Gym-malware generator was effective when combined with another generator, especially with itself, where it achieved the highest average evasion rate of 58.35%. Additionally, this generator could significantly improve the performance of other generators, with absolute and relative improvements ranging between $36.04\% - 37.42\%$ and $741.93\% - 4027.99\%$, respectively.

Our work highlights the importance of developing new techniques to detect malware and identify adversarial attacks. More research is needed in this area to successfully combat these novel threats and attacks.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. AV-TEST: Malware Statistics & Trends Report | AV-TEST. AV-TEST (2022). https://www.av-test.org/en/statistics/malware

2. Al-Asli, M., Ghaleb, T.A.: Review of signature-based techniques in antivirus products. In: 2019 International Conference on Computer and Information Sciences (ICCIS), pp. 1–6. IEEE (2019). https://doi.org/10.1109/ICCISci.2019.8716381

3. Singh, J., Singh, J.: Challenge of malware analysis: malware obfuscation techniques. Int. J. Inf. Secur. Sci. **7**, 100–110 (2018)

4. Al-Janabi, S., Alkaim, A.: A novel optimization algorithm (lion-ayad) to find optimal dna protein synthesis. Egypt. Inform. J. **23**(2), 271–290 (2022). https://doi.org/10.1016/j.eij.2022.01.004

5. Kadhuim, Z.A., Al-Janabi, S.: Codon-mrna prediction using deep optimal neurocomputing technique (dlstm-dsn-woa) and multivariate analysis. Results Eng. **17**, 100847 (2023). https://doi.org/10.1016/j.rineng.2022.100847

6. Singh, J., Singh, J.: A survey on machine learning-based malware detection in executable files. J. Syst. Architect. **112**, 101861 (2021). https://doi.org/10.1016/j.sysarc.2020.101861

7. Dolejš, J., Jureček, M.: Interpretability of machine learning-based results of malware detection using a set of rules, 107–136 (2022). https://doi.org/10.1007/978-3-030-97087-1_5

8. Rosenberg, I., Shabtai, A., Elovici, Y., Rokach, L.: Adversarial machine learning attacks and defense methods in the cyber security domain. ACM Comput. Surv. (CSUR) **54**(5), 1–36 (2021). https://doi.org/10.1145/3453158

9. Aryal, K., Gupta, M., Abdelsalam, M.: A survey on adversarial attacks for malware analysis. arXiv preprint arXiv:2111.08223 (2022). https://doi.org/10.48550/arXiv.2111.08223

10. Yan, S., Ren, J., Wang, W., Sun, L., Zhang, W., Yu, Q.: A survey of adversarial attack and defense methods for malware classification in cyber security. IEEE Commun. Surv. Tutor. **25**(1), 467–496 (2023). https://doi.org/10.1109/COMST.2022.3225137

11. Li, D., Li, Q., Ye, Y.F., Xu, S.: Arms race in adversarial malware detection: a survey. ACM Comput. Surv. (2021). https://doi.org/10.1145/3484491

12. Macas, M., Wu, C., Fuertes, W.: Adversarial examples: a survey of attacks and defenses in deep learning-enabled cybersecurity systems. Expert Syst. Appl. **238**, 122223 (2024). https://doi.org/10.1016/j.eswa.2023.122223

13. Ling, X., Wu, L., Zhang, J., Qu, Z., Deng, W., Chen, X., Qian, Y., Wu, C., Ji, S., Luo, T., Wu, J., Wu, Y.: Adversarial attacks against windows pe malware detection: a survey of the state-of-the-art. Comput. Secur. **128**, 103134 (2023). https://doi.org/10.1016/j.cose.2023.103134

14. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2015). https://doi.org/10.48550/arXiv.1412.6572

15. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy

(EuroS&P), pp. 372–387. IEEE (2016). https://doi.org/10.1109/EuroSP.2016.36

16. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. arXiv preprint arXiv:1406.2661 (2014). https://doi.org/10.48550/ARXIV.1406.2661

17. Dutta, I.K., Ghosh, B., Carlson, A., Totaro, M., Bayoumi, M.: Generative adversarial networks in security: a survey. In: 2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), pp. 0399–0405. IEEE (2020). https://doi.org/10.1109/UEMCON51285.2020.9298135

18. Wang, K., Gou, C., Duan, Y., Lin, Y., Zheng, X., Wang, F.-Y.: Generative adversarial networks: introduction and outlook. IEEE/CAA J. Automatica Sinica **4**(4), 588–598 (2017). https://doi.org/10.1109/JAS.2017.7510583

19. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction (2018). https://doi.org/10.1016/S1364-6613(99)01331-5

20. Demetrio, L., Biggio, B., Lagorio, G., Roli, F., Armando, A.: Explaining vulnerabilities of deep learning to adversarial malware binaries. arXiv preprint arXiv:1901.03583 (2019). https://doi.org/10.48550/arXiv.1901.03583

21. 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. arXiv preprint arXiv:2008.07125 (2020). https://doi.org/10.1145/3473039

22. Demetrio, L., Biggio, B., Lagorio, G., Roli, F., Armando, A.: Functionality-preserving black-box optimization of adversarial windows malware. IEEE Trans. Inf. Forensics Secur. **16**, 3469–3478 (2021). https://doi.org/10.1109/TIFS.2021.3082330

23. Anderson, H.S., Kharkar, A., Filar, B., Evans, D., Roth, P.: Learning to evade static pe machine learning malware models via reinforcement learning. arXiv preprint arXiv:1801.08917 (2018). https://doi.org/10.48550/arXiv.1801.08917

24. Castro, R.L., Schmitt, C., Dreo, G.: Aimed: evolving malware with genetic programming to evade detection. In: 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 240–247. IEEE (2019). https://doi.org/10.1109/TrustCom/BigDataSE.2019.00040

25. Wang, X., Miikkulainen, R.: Mdea: Malware detection with evolutionary adversarial learning. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2020). https://doi.org/10.1109/CEC48606.2020.9185810

26. Song, W., Li, X., Afroz, S., Garg, D., Kuznetsov, D., Yin, H.: Mab-malware: a reinforcement learning framework for attacking static malware classifiers. arXiv preprint arXiv:2003.03100 (2020). https://doi.org/10.48550/arXiv.2003.03100

27. Fang, Z., Wang, J., Li, B., Wu, S., Zhou, Y., Huang, H.: Evading anti-malware engines with deep reinforcement learning. IEEE **7**, 48867–48879 (2019). https://doi.org/10.1109/ACCESS.2019.2908033

28. Labaca-Castro, R., Franz, S., Rodosek, G.D.: Aimed-rl: exploring adversarial malware examples with reinforcement learning. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 37–52. Springer (2021). https://doi.org/10.1007/978-3-030-86514-6_3

29. Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., Roli, F.: Adversarial malware binaries: evading deep learning for malware detection in executables. arXiv preprint arXiv:1804.04637 (2018). https://doi.org/10.48550/arXiv.1803.04173

30. Kreuk, F., Barak, A., Aviv-Reuven, S., Baruch, M., Pinkas, B., Keshet, J.: Deceiving end-to-end deep learning malware detectors using adversarial examples. arXiv preprint arXiv:1802.04528 (2018). https://doi.org/10.48550/arXiv.1802.04528

31. Suciu, O., Coull, S.E., Johns, J.: Exploring adversarial examples in malware detection. arXiv preprint arXiv:1810.08280 (2018). https://doi.org/10.48550/arXiv.1810.08280

32. Hu, W., Tan, Y.: Generating adversarial malware examples for black-box attacks based on gan. arXiv preprint arXiv:1702.05983 (2017). https://doi.org/10.48550/arXiv.1702.05983

33. Kawai, M., Ota, K., Dong, M.: Improved malgan: avoiding malware detector by leaning cleanware features. In: 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), pp. 040–045. IEEE (2019). https://doi.org/10.1109/ICAIIC.2019.8669079

34. Yuan, J., Zhou, S., Lin, L., Wang, F., Cui, J.: Black-box adversarial attacks against deep learning based malware binaries detection with gan, 2536–2542 (2020). https://doi.org/10.3233/FAIA200388

35. Anderson, H.S., Roth, P.: Ember: an open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637 (2018). https://doi.org/10.48550/ARXIV.1804.04637

36. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.: Malware detection by eating a whole exe. arXiv preprint arXiv:1710.09435 (2017). https://doi.org/10.48550/ARXIV.1710.09435

37. VirusShare: VirusShare. https://www.virusshare.com/ (2023)

38. AV-Comparatives: Malware Protection Test September 2022. https://www.av-comparatives.org/tests/malware-protection-test-september-2022/ (2022)

39. VirusTotal: VirusTotal. https://www.virustotal.com/ (2023)

40. Kozák, M., Jureček, M.: Combining generators of adversarial malware examples to increase evasion rate. In: Proceedings of the 20th International Conference on Security and Cryptography - SECRYPT, pp. 778–786 (2023). https://doi.org/10.5220/0012127700003555

41. Thomas, R.: LIEF - Library to Instrument Executable Formats (2017). https://lief.quarkslab.com/

42. Kozák, M., Jurecek, M., Stamp, M., Troia, F.D.: Creating valid adversarial examples of malware. arXiv preprint arXiv:2306.13587 (2023). https://doi.org/10.48550/arXiv.2306.13587