24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Black-box Adversarial Attacks Against Deep Learning Based Malware Binaries Detection with GAN

Junkun Yuan [1], Shaofang Zhou [1], Lanfen Lin* [1] and Feng Wang[2] and Jia Cui[3]

Abstract. For efficient malware detection, there are more and more deep learning methods based on raw software binaries. Recent studies show that deep learning models can easily be fooled to make a wrong decision by introducing subtle perturbations to inputs, which attracts a large influx of work in adversarial attacks. However, most of the existing attack methods are based on manual features (e.g., API calls) or in the white-box setting, making the attacks impractical in current real-world scenarios. In this work, we propose a novel attack framework called GAPGAN, which generates adversarial payloads (padding bytes) with generative adversarial networks (GANs). To the best of our knowledge, it is the first work that performs end-to-end black-box attacks at the byte-level against deep learning based malware binaries detection. In our attack framework, we map input discrete malware binaries to continuous space, then feed it to the generator of GAPGAN to generate adversarial payloads. We append payloads to the original binaries to craft an adversarial sample while preserving its functionality. We propose to use a dynamic threshold for reducing the loss of the effectiveness of the payloads when mapping it from continuous format back to the original discrete format. For balancing the attention of the generator to the payloads and the adversarial samples, we use an automatic weight tuning strategy. We train GAPGAN with both malicious and benign software. Once the training is finished, the generator can generate an adversarial sample with only the input malware in less than twenty milliseconds. We apply GAPGAN to attack the state-of-the-art detector MalConv and achieve 100% attack success rate with only appending payloads of 2.5% of the total length of the data for detection. We also attack deep learning models with different structures under different defense methods. The experiments show that GAPGAN outperforms other state-of-the-art attack models in efficiency and effectiveness.

1 INTRODUCTION

Deep neural networks have achieved great success, more and more work prefers to use deep learning for efficient malware detection. Among them, some work (e.g., [5] and [12]) detects malware based on manual features (e.g., API calls) which may contain malicious behaviour of a program, some work (e.g., [21], [24], and [4]) directly uses information of software without running it, and other work (e.g., [13] and [20]) integrates the above strategies or uses other methods, like visualization. Recently, there is a trend of using raw binaries for malware detection, which can efficiently mine the latent relationships among different sections of the file. With the rapid development of malware, the defense efficiency becomes crucial in today's realistic scenarios, making the end-to-end detection based on raw binaries more promising.

However, many research work ([25], [7], [17], [9], and [27]) has demonstrated that deep neural networks are susceptible to adversarial attacks. The attackers add small perturbations to the original data that is imperceptible to humans, which can mislead the classifiers to do wrong decisions. These studies point out a serious threat to the security of deep learning algorithms and AI applications. In malware detection, most of the adversarial attacks (e.g., [14], [15], and [3]) rely on the complete information of the detector (i.e., white-box attacks). However, there are limitations to this kind of attack, e.g., the target model must be fully exposed to the attackers. Meanwhile, previous attack work (e.g., [11], [2], and [23]) are based on manual features that are speculated to be used for training the detector. If the speculation is wrong or once the defender changes its training strategies, this kind of attacks will be invalid. The wide use of raw binaries based detection also makes such an attack that needs plenty of resources and time to extract features inapplicable.

Different from the manual features, the original binaries data cannot be simply changed even with small modifications, or their functionality will be damaged. Besides, the size of binaries data varies widely, which further increases the attack difficulty. We also find that subtle perturbations will be ignored when transforming adversarial payloads in continuous space back to discrete binary when we save the generated adversarial samples, which affects the effectiveness of adversarial attacks. Therefore, how to perform effective and practical black-box attacks to the deep learning models based on malware binaries while protecting the original functionality remains a great challenge.

In this paper, we put forward a novel attack framework GAPGAN which generates adversarial payloads via GANs. To the best of our knowledge, it is the first work that performs end-to-end black-box attacks at the byte-level against deep learning based malware binaries detection. We apply GAPGAN to attack the state-of-the-art detector MalConv [21] as well as other deep learning models with different structures. The experiments show that our model can achieve a high attack success rate, and it outperforms other state-of-the-art attack methods in efficiency and effectiveness.

We have the following contributions:

1. We propose a novel adversarial attack framework GAPGAN, which performs end-to-end black-box attacks at the byte-level against deep learning based malware binaries detection, making the attacks more efficient and effective.
2. In GAPGAN, the generator generates adversarial payloads and ap-

[1] College of Computer Science and Technology, Zhejiang University, Zhejiang, China, {yuanjk, tanes, llf}@zju.edu.cn
* Corresponding author is Lanfen Lin
[2] Department of Computer and Information Technology Zhejiang Police College, Zhejiang, China, wangfeng@zju.edu.cn
[3] China Information Technology Security Evaluation Center, Beijing, China, cuij@itsec.gov.cn

pends it to the original data to craft a malware adversarial sample while preserving its functionality. Once the training process is finished, the generator can efficiently generate each adversarial sample in less than twenty milliseconds.

3. We propose to use a dynamic threshold for reducing the loss of the effectiveness of the payloads when mapping it from continuous space back to discrete binaries. For balancing the attention of the generator to the payloads and the adversarial samples, we adopt an automatic weight tuning strategy.
4. We apply GAPGAN to attack the state-of-the-art malware detector MalConv. The experiments show that the adversarial samples generated by GAPGAN can achieve an attack success rate of 100% when appending adversarial payloads of 2.5% of the length of the data for detection. The experiments also show that GAPGAN outperforms other state-of-the-art attack methods in efficiency and effectiveness under different defenses.

The remaining part of the paper is composed of five parts: In Section 2, we introduce the background and related work. In Section 3, we explain the details of our attack framework GAPGAN. In Section 4, we describe the experimental setup, including datasets, metrics, and target models. In Section 5, we show the results of our experiments. In Section 6, we sum up our work and give a conclusion.

2 BACKGROUND AND RELATED WORK

2.1 Adversarial Attacks Against Malware Detection

Most of the traditional machine learning and deep learning methods for malware detection (e.g., [5] and [12]) focus on manual features that are extracted from programs' behavior information, like signature and API calls. For this kind of detection methods, earlier attack work is mainly based on the manual features which are supposed to be used by the defender. Some work proposes to use APIs as binary features, then adopt deep learning models to generate adversarial samples ([11] and [8]). A different approach based on API call sequences uses an optimization process to perform adversarial attacks [23]. [2] proposes to use reinforcement learning for attacking, it comprises numerous manual information as features, e.g., PE header metadata, section metadata and byte histogram. Xu et al. [29] put forward a genetic programming based attack method to perform stochastic manipulations on the structures of file. However, these attacks need expert experience and plenty of time to obtain effective features, and once the features used for attacking is known by the defenders, the fast update detectors can easily evade the attacks.

Recent malware detection work (e.g., [21], [24], and [4]) pays more attention to use deep learning models on raw software binaries, as deep neural networks can efficiently mine latent characteristics in raw data without mass data preprocessing and prior experience. To catch up with the updated malware detection technologies, the attackers start to seek new methods that can be applied to raw software binaries (e.g., [14], [15], and [3]). Different from the extracted features, the raw binaries data cannot be simply changed or it may lose important functionality. Besides, the raw binaries have variable input sizes, which can further make these attacks more tricky than previous.

[14] proposes the first adversarial attack work at the byte-level, which combines gradient ascent and greedy strategies. It appends bytes one by one to the end of the file for preserving their functionality. However, it performs white-box attacks that have limitations in real-world scenarios, and the model needs to calculate the gradient for each padding byte which consumes a lot of time and resources. [15] also puts forward an approach for discrete sequences by injecting local perturbations. However, it is in the white-box setting and not efficient. [3] proposes both white-box and black-box methods. In the black-box method, it randomly selects and appends benign data blocks to the malware data, tests the results at each time. It consumes plenty of time to get the effective blocks before performing attacks. This approach is simple but tedious and inefficient, which is not applicable for effective malware adversarial samples generation. In contrast, we will show that our end-to-end framework can attack in the black-box setting and generates adversarial samples in far less time.

2.2 Generative Adversarial Networks (GANs)

Generative adversarial networks (GANs) [6] are widely used in computer vision tasks (e.g., [30], [16] and [1]) in recent years. According to their high level of imitation ability, some work (e.g., [11] and [28]) adopts GANs for adversarial attacks. The most representative attack methods use the approach called distillation [10] to fit the discriminator with the outputs of the target model, train the generator for generating data that can mislead the discriminator. In this way, the adversarial samples can attack the target model indirectly, i.e., the transferability of the adversarial samples [19]. Different from previous work, we use the generator to generate adversarial payloads, which is used to craft an adversarial sample without damaging its functionality. In our model, once the training process of GANs is finished, the generator can independently generate malware adversarial samples in a very short time with only the input malware binaries.

3 BLACK-BOX ATTACKS TO MALWARE DETECTION WITH GAN

In this section, we will briefly explain the formal definition of the input binaries and the adversarial samples, then introduce the framework and strategies details of GAPGAN.

3.1 Problem Definition

Binary file of software consists of a sequence of bytes belonging to the discrete space $\mathcal{X} = \{0, ..., 255\}$. Let $\boldsymbol{b} = (b_1, ..., b_n) \in \mathcal{X}^n$ denote a binary, where n is the length of byte sequence, varying from file to file. The binary file $\boldsymbol{b}$ has labels $y \in \{-1, 1\}$, where $y = 1$ indicates that it is a benign software $\boldsymbol{b}_{ben}$, otherwise it is a malware $\boldsymbol{b}_{mal}$.

The malware detector aims at learning a mapping function $f : x \to \{-1, 1\}$ which satisfies $f(\boldsymbol{b}_{mal}) = -1$ and $f(\boldsymbol{x}_{ben}) = 1$. On the contrary, the goal of the adversarial attacks is to find a model g and generate an effective adversarial sample $\boldsymbol{b}_{adv} = g(\boldsymbol{b}_{mal})$ to make the malware detector classify it as benign software, i.e., $f(\boldsymbol{b}_{adv}) = 1$. In the meanwhile, $\boldsymbol{b}_{adv}$ must preserve the original function of $\boldsymbol{b}_{mal}$.

3.2 GAPGAN Framework

Figure 1 shows the overview of the proposed framework GAPGAN. It contains two stages: training process and attack process. In the training process, we train the generator network $\mathcal{G}$ and the discriminator $\mathcal{D}$ concurrently, where $\mathcal{G}$ intends to generate adversarial payloads for input malware and concatenate them to craft adversarial samples, while $\mathcal{D}$ tries to distill the target black-box detector f and

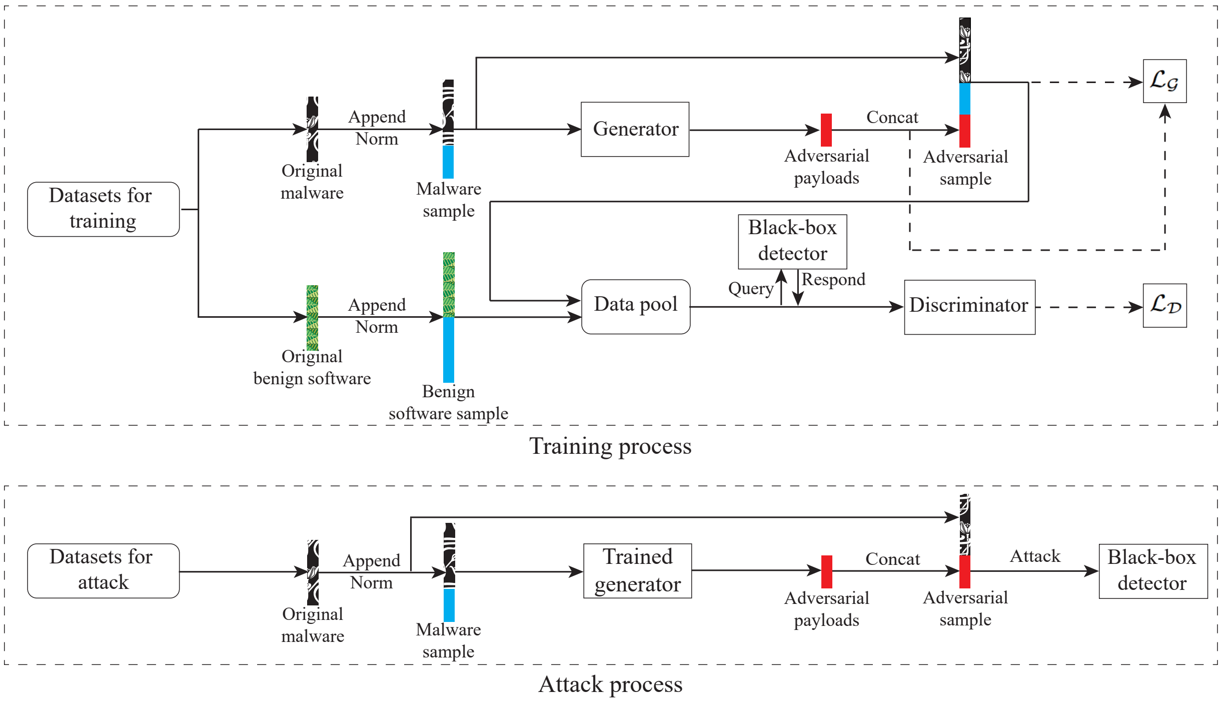

Figure 1: Overview of GAPGAN

imitates the decision of f for both orignial benign samples and the generated adversarial samples. In the attack process, we only need the trained generator to attack black-box detector.

For protecting original functionality of malware when crafting its adversarial sample, there have been some popular methods like using debug logs and compressing data before runtime, but they are time-consuming and laborious. Other attacks performed by carefully choosing and manipulating are sophisticated that may require specific experience and not applicable for efficient adversarial attacks. Inspired by previous work ([3] and [14]), we choose to append bytes (payloads) at the end of file to preserve their functionality, which is simple and does not require any expert experience.

Since the length of software file n varies greatly, we first append zeros (represented as the blue part in Figure 1) to the end of input binaries to match the input size t of the network as $\boldsymbol{b'} = (b_1, .., b_n, 0, ..., 0) \in \mathcal{X}^t$, where $t \geq n$. In this way, we can feed every sample with different lengths to a specific network with fixed size. Then, we map each byte in discrete binaries to a continuous space $[-1, 1]$ by normalization. We define the normalized input as $\boldsymbol{x}$, where $\boldsymbol{x} = (x_1, ..., x_t) \in \mathbb{R}^t$.

After data preprocessing, the normalized malware $\boldsymbol{x}_{mal}$ is fed to $\mathcal{G}$. Then $\mathcal{G}$ generates adversarial payloads $\boldsymbol{a}_{adv}$ (represented as the red part in Figure 1) based on the corresponding characteristics of $\boldsymbol{x}_{mal}$:

$$\boldsymbol{a}_{adv} = \mathcal{G}(\boldsymbol{x}_{mal}) \tag{1}$$

We append $\boldsymbol{a}_{adv}$ to the end of $\boldsymbol{x}_{mal}$ to craft an adversarial malware sample $\boldsymbol{x}_{adv}$:

$$\boldsymbol{x}_{adv} = [\boldsymbol{x}_{mal}, \boldsymbol{a}_{adv}] \tag{2}$$

where $[\cdot, \cdot]$ denotes concatenation operation.

For training $\mathcal{D}$, both $\boldsymbol{x}_{adv}$ and $\boldsymbol{x}_{ben}$ are integrated into the data pool. In each iteration, we sample a batch of mixed examples from the data pool, then use them to query the black-box detector f. Next, we use the label responded by f to fit $\mathcal{D}$, making the decision boundary of $\mathcal{D}$ as close to f as possible.

During training, the generator $\mathcal{G}$ learns to create samples which can evade the discriminator $\mathcal{D}$. In addition, with the improvement of the similarity between $\mathcal{D}$ and our target model f, the adversarial attack ability of $\mathcal{G}$ to f will improve as well. Finally, the adversarial samples generated by $\mathcal{G}$ can also evade f effectively because of the transferability of adversarial attacks.

Once the training process is finished, we can use the trained $\mathcal{G}$ to generate adversarial samples in a very short time with only the input malware. It is worth noticing that we abandon the padding zeros for reducing the whole length of payloads in the attack process. To our practical experience, it will make the attack success rate decrease a little, but the loss is acceptable. In addition, we need to convert the adversarial samples back to discrete space as executable file.

In order to make our framework adapt to malware binaries and payloads with different lengths, the generator network is designed to have variable input and output size. To be more specific, the generator first extracts features of inputs with two convolution layers. Then, it resizes the high-level features with fully-connected layers. After two layers of deconvolution and one layer of $1 * 1$ convolution, the adversarial payloads are generated. On the other hand, the discriminator performs binary classification with convolutional layers and fully-connected layers. Notice that if the size of the input data and the length of payloads that we decide to generate are determined, we can use them as input to easily tune the structure of GAPGAN because of the fully-connected layers in both the generator and the discriminator.

3.3 Black-box Attacks Strategy

Generator. The generator $\mathcal{G}$ aims at learning the characteristics of $\boldsymbol{x}_{mal}$ whose original label is $y = -1$, and generate corresponding effective sample $\boldsymbol{x}_{adv}$, which can mislead $\mathcal{D}$ to predict it as "benign" whose label is $y = 1$. In our practical experience, $\mathcal{G}$ often pays more attention to $\mathcal{D}$'s prediction result of the $\boldsymbol{x}_{adv}$ which brings a serious problem, i.e., the effectiveness of adversarial payloads $\boldsymbol{a}_{adv}$ cannot be improved well. Therefore, we make $\mathcal{G}$ consider both the global and the local (i.e., $\boldsymbol{x}_{adv}$ and $\boldsymbol{a}_{adv}$) effectiveness of $\boldsymbol{x}_{adv}$. The adversarial loss function of the generator $\mathcal{G}$ is:

$$\mathcal{L}_{\mathcal{G}} = -(1-\beta)\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}_{adv}}}(\mathcal{D}(\boldsymbol{x})) - \beta\mathbb{E}_{\boldsymbol{a}\sim p_{\boldsymbol{a}_{adv}}}(\mathcal{D}(\boldsymbol{a})) \quad (3)$$

where β is a hyperparameter that keeps a balance of the generator's attention between $\boldsymbol{x}_{adv}$ and $\boldsymbol{a}_{adv}$. We try to find the best value of β, however, a fixed β can not always perform the best, because the conditions of networks are different every time the attack program runs. Therefore, the best β should have the ability of adaptive adjustment. Inspired by [18], we consider automatically tuning β based on the outputs of $\mathcal{D}$ to $\boldsymbol{x}_{adv}$ and $\boldsymbol{a}_{adv}$, which represent the attack effectiveness of them respectively. We give the automatic tuning mechanism:

$$\beta = \frac{exp(\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}_{adv}}}(\mathcal{D}(\boldsymbol{x})))}{exp(\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}_{adv}}}(\mathcal{D}(\boldsymbol{x}))) + exp(\mathbb{E}_{\boldsymbol{a}\sim p_{\boldsymbol{a}_{adv}}}(\mathcal{D}(\boldsymbol{a})))} \quad (4)$$

If x_{adv} is more effective than $\boldsymbol{a}_{adv}$, then the expectation of the output of $\mathcal{D}$ to $\boldsymbol{x}_{adv}$ is larger. The automatic tuning mechanism will increase β to improve the learning rate of $\boldsymbol{a}_{adv}$ indirectly. We will show its efficacy in our experiments.

Discriminator. We use the discriminator $\mathcal{D}$ to dynamically distill the target black-box model f. To be more specifically, we sample a batch of mixed data from the data pool, get labels by querying f. The samples and their corresponding labels are used for fitting $\mathcal{D}$ based on the distance metric $\mathcal{H}$. The distillation loss of $\mathcal{D}$ is:

$$\mathcal{L}_{\mathcal{D}} = \mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{x}_{adv}}\mathcal{H}(\mathcal{D}(\boldsymbol{x}), f(\boldsymbol{x})) + \mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{x}_{ben}}\mathcal{H}(\mathcal{D}(\boldsymbol{x}), f(\boldsymbol{x})) \quad (5)$$

$\mathcal{D}$ tries to learn the decision strategies of f on $\boldsymbol{x}_{ben}$ and $\boldsymbol{x}_{adv}$. In this way, $\mathcal{D}$ is treated as a substitute detector, which is used for transferring the attack effectiveness of adversarial samples to the ultimate target black-box model f.

Dynamic threshold strategy. In the attack process, we will generate adversarial samples and save them locally. However, we find that subtle perturbations will be ignored when we map the adversarial samples from adversarial continuous space back to the discrete space of binaries. A large part of payloads containing attack effectiveness will be ignored because of their small values. To solve this problem, we propose to use a dynamic threshold strategy to limit the minimum value of payloads:

$$e = \begin{cases} e, & if\,|e| > \epsilon * \frac{i}{T_{max}} \\ 0, & else \end{cases} \quad (6)$$

where e represents each byte in payloads, i is the current training iteration time, T_{max} is the maximum training iteration time, and ϵ is the maximum threshold value. We directly set the bytes with small values as zeros below the threshold. However, if we use a static threshold, the learning process of $\mathcal{G}$ will get lost, leading to terrible adversarial attack results (see experimental results in section 5). It is because most of the bytes generated by G which is just after initialization are very small, they will be set as zeros in the beginning. Hence we use $\epsilon * \frac{i}{T_{max}}$ to dynamically increase the threshold, in this way, $\mathcal{G}$ can gradually adjust its attack strategy to the constraints. More concretely, if a byte with small value but certain attack effectiveness, it will first be set to zero with the threshold. Then $\mathcal{G}$ will continue to add the perturbations to the byte or other adjacent byte for improving the adversarial attack effectiveness in this area. Finally, the byte or other adjacent byte will be modified to fill the adversarial attack loss of the byte which is set to zero. It is worth noting that all the adjustment process will perform automatically with the gradient descent algorithm after we set ϵ.

On the basis of the above work, the overall algorithm of black-box attacks to malware detection is shown in Algorithm 1. In Section 5, we will show the details of our attack experiments.

Algorithm 1 Black-box Attacks to Malware Detection

```
Input: Training set $S = \{(\boldsymbol{x}_0, y_0), ..., (\boldsymbol{x}_{k-1}, y_{k-1})\}$, generator $\mathcal{G}(\boldsymbol{x}; \boldsymbol{\theta}_{\mathcal{G}_0})$, discriminator $\mathcal{D}(\boldsymbol{x}; \boldsymbol{\theta}_{\mathcal{D}_0})$, target black-box model $f$, max training iteration $T_{max}$, maximum threshold $\epsilon$, weight $\beta$
Output: A well-trained Generator $\mathcal{G}(\boldsymbol{x}; \boldsymbol{\theta}_{\mathcal{G}})$
  for $i = 0 \to T_{max} - 1$ do
      Sample $m$ examples from $S$ and get training set for $\mathcal{D}$, $S_d = \{(\boldsymbol{x}_{d_0}, y_{d_0}), ..., (\boldsymbol{x}_{d_{m-1}}, y_{d_{m-1}})\}$
      for $x_{d_i}$ in $S_d$ do
          Query $f$ and get $f(x_{d_i})$
      end for
      Use $S_d$ to update $\boldsymbol{\theta}_{\mathcal{D}}$ with $\nabla_{\boldsymbol{\theta}_{\mathcal{D}}}(\mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{x}_{mal}}\mathcal{H}(\mathcal{D}(\boldsymbol{x}), f(\boldsymbol{x})) + \mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{x}_{ben}}\mathcal{H}(\mathcal{D}(\boldsymbol{x}), f(\boldsymbol{x})))$
      Sample $m$ examples ($y_{adv_i} = -1$) from $S$ and get training set for $\mathcal{G}$, $S_{adv} = \{(\boldsymbol{x}_{adv_0}, y_{adv_0}), ..., (\boldsymbol{x}_{adv_{m-1}}, y_{adv_{m-1}})\}$
      for $(\boldsymbol{x}_{adv_i}, y_{adv_i})$ in $S_{adv}$ do
          Generate adversarial payloads $\boldsymbol{a}_{adv_i} = \mathcal{G}(\boldsymbol{x}_{adv_i})$
          for $e$ in $\boldsymbol{a}_{adv_i}$ do
              if $|e| < \epsilon * \frac{i}{T_{max}}$ then
                  $e \leftarrow 0$
              end if
          end for
          $\boldsymbol{x}_{adv_i} \leftarrow [\boldsymbol{x}_{adv_i}, \boldsymbol{a}_{adv_i}]$
      end for
      Calculate $\beta = \frac{exp(\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}_{adv}}}(\mathcal{D}(\boldsymbol{x})))}{exp(\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}_{adv}}}(\mathcal{D}(\boldsymbol{x})))+exp(\mathbb{E}_{\boldsymbol{a}\sim p_{\boldsymbol{a}_{adv}}}(\mathcal{D}(\boldsymbol{a})))}$
      Use $S_{adv}$ to update $\boldsymbol{\theta}_{\mathcal{G}}$ with $\nabla_{\boldsymbol{\theta}_{\mathcal{G}}} -(1-\beta)\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}_{adv}}}(\mathcal{D}(\boldsymbol{x})) - \beta\mathbb{E}_{\boldsymbol{a}\sim p_{\boldsymbol{a}_{adv}}}(\mathcal{D}(\boldsymbol{a}))$
  end for
  return $\boldsymbol{\theta}_{\mathcal{G}}$
```

Table 1: Malware and benign software data are collected from different sources and split into four datasets according to their length distributions.

Datasets	Class	Number	Max	Mean	Source
1	Malware	3,436	93,986	51,715	VirusTotal
	Benign	3,436	98,304	41,651	Chocolatey
2	Malware	5,000	195,584	80,707	VirusTotal
	Benign	5,000	196,608	98,072	Chocolatey
3	Malware	10,000	394,128	126,276	VirusTotal
	Benign	10,000	393,640	128,808	Chocolatey
4	Malware	3,000	196,189	117,812	Kaggle 2015
	Benign	3,000	195,320	92,526	Chocolatey

4 EXPERIMENTAL SETUP

This section introduce the preparation of our attack experiments, including datasets, evaluation metrics, and the target models which we choose and train.

4.1 Datasets and Evaluation Metrics

Malware and benign software data are collected from different sources for our adversarial attack experiments, as shown in Table

Table 2: The detection accuracies of MalConv and other deep learning models used as target black-box models. A: CNN-based model; B: CNN-LSTM-based model; C: CNN-GRU-based model; D: Parallel-CNN-based model.

Datasets	MalConv	A	B	C	D
1	96.40%	-	-	-	-
2	96.42%	94.94%	95.99%	95.30%	94.70%
3	97.22%	-	-	-	-
4	95.55%	95.02%	95.27%	95.24%	95.30%

1. The malware samples are downloaded from VirusTotal[4] and Microsoft Malware Classification Challenge (Kaggle 2015) [22]. The benign software samples are downloaded by Chocolatey Software[5], a package manager for Windows. The data are split into four datasets to make the binaries length distributions of malware close to benign software in each dataset. Dataset 1, 2, and 3 with different maximum and mean length are used for exploring the impact of binaries length on attacks. Dataset 2 and 4 with different sources but close length distributions are used for evaluating the generalization of attack algorithms.

We randomly split each dataset into two parts, one (70%) is for training the black-box model, the other (30%) is for adversarial attacks. In this way, the data used for training black-box model and for adversarial attacks are disjoint.

To evaluate the performance of adversarial attack methods, we choose the attack success rate (ASR) metric:

$$ASR = \frac{\sum_{i=1}^{n} I(f(\boldsymbol{x}_{mal_i}) = -1 \wedge f(\boldsymbol{x}_{adv_i}) = 1)}{\sum_{j=1}^{n} I(f(\boldsymbol{x}_{mal_j}) = -1)} \quad (7)$$

where I is indicator function, it equals to 1 if the expression is true, or 0 otherwise. ASR represents the rate of malware samples that are detected by the black-box model but evade successfully after being crafted to be adversarial samples.

4.2 Target Black-box Models

We choose the state-of-the-art malware detector MalConv [21] as our primary target black-box model. MalConv first embeds each byte in input binaries to 8-dimensional vector, then uses two convolution layers with different activation functions for classification. We train a MalConv detector with input size 2,000,000 for each dataset. Table 2 shows the test accuracy of each MalConv detector after training. It demonstrates that the trained MalConv detectors have similar performance with that in [21].

In order to test the generalization of the adversarial attack methods, we also use four deep learning models with different structures as target models. For reducing the large dimensions of input binaries, we consider adding CNN structures to each deep learning model. Each byte in input binaries are embedded to 8-dimensional vector in the same way as MalConv. We train these four models on dataset 2 and 4, the detection accuracies are shown in Table 2. It shows that the four detectors also reach good classification accuracy.

5 EXPERIMENTAL RESULTS

In this section, we show the effectiveness of GAPGAN in adversarial attack experiments. We also compare it with other state-of-the-art attack methods under different defenses.

[4] http://www.virustotal.com
[5] https://chocolatey.org/

Table 3: Attack success rate (ASR) of the adversarial samples generated by GAPGAN against MalConv models on different datasets. Payloads rate represents the rate of the length of payloads to that of binaries for detection.

Payloads Rate	Dataset 1	Dataset 2	Dataset 3	Dataset 4
1%	64.66%	6.28%	2.15%	4.13%
2.5%	**100.00%**	36.10%	18.14%	30.99%
5%	**100.00%**	77.78%	43.27%	53.49%
10%	**100.00%**	**98.21%**	72.89%	76.88%
20%	**100.00%**	**100.00%**	**88.95%**	**87.41%**

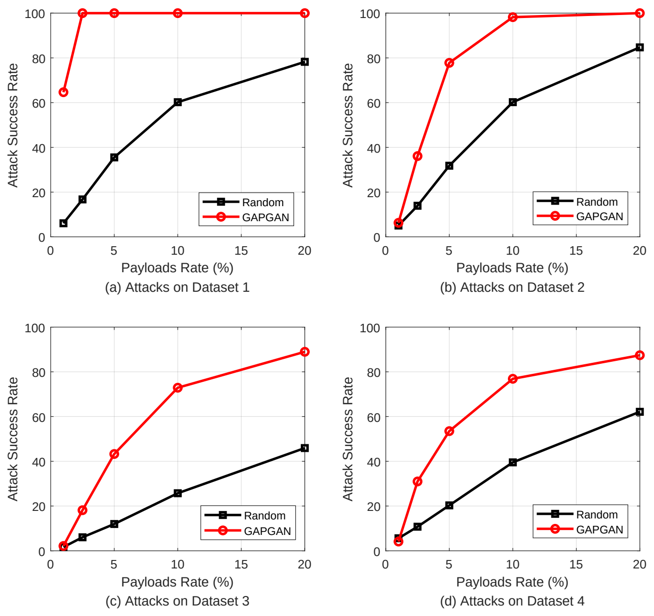

Figure 2: Comparison of ASR of adversarial samples generated by GAPGAN and generated randomly against MalConv detectors on different datasets with different payloads rates.

5.1 Black-box Attacks with GAPGAN

We first apply GAPGAN to attack the four trained MalConv detectors (we have shown them in the previous section) with different lengths of adversarial payloads. Payloads rate is used for representing the rate of the length of payloads to that of binaries for detection. According to the performance of attacks on different datasets shown in Table 3, it shows that GAPGAN can perform effective black-box attacks against MalConv models. As can be seen from the results on dataset 2 and dataset 4, the adversarial samples have a high attack success rate on different data. Besides, the adversarial binaries whose original length is shorter may have better attack effectiveness, because of the increasement of the payloads rate. It is worth mentioning that the ASR of adversarial samples generated from dataset 1 can reach 100% with only a small proportion of payloads, i.e., 2.5% of the total length of the data for detection.

We find that when ASR has already reached high value, using larger payloads may just improve little attack success rate (e.g., ASR is 98.21% when appending payloads with a rate of 10%, but it just improve 1.79% when appending the payloads with twice the length). However, the risk of being detected and the cost will increase with the increasement of payloads' length. Meanwhile, in order to prove the effectiveness of adversarial payloads generated by GAPGAN, we compare it with random payloads as shown in Figure 2. We see that

Table 4: Comparison of the state-of-the-art adversarial attack methods in the malware detection task. Run time is the time for generating an adversarial samples in attack process, which is evaluated by generating 3000 adversarial samples with each method.

	Adversarial attack methods			
	Opt. [14]	AdvSeq [23]	MalGAN [11]	GAPGAN
Black-box		✓	✓	✓
Run time	>2h	-	**0.02s**	**0.02s**
Attack level	**Bytes**	API calls	API calls	**Bytes**

Table 5: Comparison of the attack performance of Random, Opt. and GAPGAN against different detectors on dataset 2 and 4. Random: random payloads. The payloads rates are 10% in these experiments.

Detector	Dataset 2			Dataset 4		
	Random	Opt.	GAPGAN	Random	Opt.	GAPGAN
MalConv	60.21%	**99.87%**	98.21%	57.52%	68.34%	**76.88%**
A	57.84%	**90.41%**	76.04%	17.10%	**85.09%**	51.31%
B	44.04%	93.32%	**99.35%**	46.50%	**77.24%**	68.67%
C	64.25%	**92.74%**	84.40%	55.72%	**78.17%**	64.96%
D	70.47%	97.23%	**99.93%**	9.03%	74.49%	**87.80%**

adversarial samples with payloads generated by GAPGAN have far better attack effectiveness compared with payloads generated randomly. It can also be seen that ASR of random payloads is proportional to the payload rates, while adversarial payloads increase rapidly with the increasement of the payloads rates and the growth rate of ASR slows down when it reaches high value. We consider that there exists optimum payload rates for each dataset, i.e., the growth rate of ASR will decline fast when appending larger payloads.

5.2 Comparison with State-of-the-art Attack Methods

We compare GAPGAN with other state-of-the-art adversarial attack methods in malware detection task, i.e., Opt. method based on gradient optimization [14], AdvSeq method based on sequences of API calls [23], and MalGAN method based on API calls and GANs [11]. The results of them is shown in Table 4. It can be seen that only Opt. method performs attacks in the white-box setting. From the perspective of attack efficiency, once the attack models are trained, GAPGAN and MalGAN generate adversarial samples far faster than other methods (AdvSeq is also based on sophisticated optimization processes, which is supposed not efficient). However, only GAPGAN performs efficient black-box attacks at the byte-level, which is more threatening in real-world scenarios.

In order to further explore the effectiveness of attack approaches against binaries based detection, we choose to compare GAPGAN with Opt. method, i.e., the byte-level attack methods. The adversarial samples with random payloads are chosen for comparison. It can be seen from Table 5, both of the two attack methods have good attack performance against different detectors. However, GAPGAN performs efficient black-box attacks which is considered crucial for adversarial attacks in application.

5.3 Attack Performance Under Different Defense Methods

A lot of defense methods have been proposed to defend various of attacks. The most popular way to make the model robust to adversarial samples is adversarial training [7], which introduces adversarial perturbations in the training process to make the deep learning models

Table 6: Comparison of the attack performance of Random, Opt. and GAPGAN to different detectors under defenses. RND: random nullification data defense method; Adv.: adversarial training defense method. The payloads rates are 10% in these experiments.

Defense	Detector	Dataset 2			Datast 4		
		Random	Opt.	GAPGAN	Random	Opt.	GAPGAN
RND	Malconv	24.64%	51.23%	**63.69%**	49.59%	41.25%	**75.73%**
	A	20.67%	**57.84%**	45.00%	0.76%	**37.14%**	23.64%
	B	0.00%	62.29%	**87.47%**	5.79%	37.82%	**41.07%**
	C	7.65%	**39.47%**	34.57%	22.91%	29.74%	**39.22%**
	D	9.52%	43.58%	**92.35%**	3.06%	54.41%	**71.09%**
Adv.	Malconv	23.87%	29.78%	**57.04%**	13.10%	22.17%	**30.46 %**
	A	0.00%	15.14%	**23.72%**	0.00%	7.72%	**9.49%**
	B	0.00%	27.17%	**39.17%**	0.00%	9.38%	**15.82%**
	C	1.04%	19.77%	**24.18%**	4.99%	13.47%	**18.13%**
	D	0.00%	31.65%	**41.73%**	0.00%	17.97%	**27.60%**

Table 7: Attack success rate of adversarial samples generated by GAPGAN with different ϵ and β. ϵ is set to 0.06. β is set to 0.5 in the static case. W: without using the method; S: using static parameter; D: using dynamic parameter.

β	ϵ	Dataset 1	Dataset 2	Dataset 3	Dataset 4
W	W	70.94%	70.97%	22.94%	58.47%
W	D	**100.00%**	94.09%	61.38%	72.12%
D	W	83.12%	87.95%	46.44%	74.56%
S	D	**100.00%**	96.06%	69.38%	75.33%
D	S	8.74%	6.49%	1.38%	7.98%
D	D	**100.00%**	**98.21%**	**72.89%**	**76.88%**

tune the decision strategies. Another efficient defense method [26] randomly nullifies the input data to eliminate the attack effectiveness of adversarial samples. We compare the attack effectiveness of random payloads with that of adversarial samples generated by GAPGAN and Opt. under these defenses.

For simulating real-world scenarios, we assume that the attacker does not know any information on the defenses. In the experiments of RND defense methods, we randomly nullify 10% of input data and test the attack success rate of adversarial samples. Since the structures of the detectors contain an embedding layer, the gradient cannot be transferred in the adversarial training defense method. Therefore, we propose to use substitute models that are used for distilling the detectors to generate training data with adversarial perturbations. The new training data are used for improving the robustness of the detectors. The adversarial samples generated with the previous detectors will be evaluated on the retrained detectors. Table 6 shows the results of attacks under defenses. We show that in most cases, the attack performance of GAPGAN outperforms Opt. method, especially under the defense of adversarial training. A possible explanation is that Opt. method overly relies on the structures and gradient information of the target models. In addition, a byte in payloads is generated by the gradients of the current adversarial sample by Opt. method. The attack effectiveness is greatly damaged when the connections between bytes are cut off, i.e., in the random nullification process of RND defense. By comparison, GAPGAN considers the attack ability of the whole adversarial samples, making it more effective under defenses.

5.4 Effectiveness of Dynamic Threshold and Automatic Weight Tuning

As we explained in Section 3, we put forward a dynamic threshold strategy to limit the minimum value of payloads and an automatic weight tuning mechanism to balance the attention of the generator to both the payloads and the adversarial samples. We perform ablation studies to verify the effectiveness of these strategies with different parameters, i.e., without, static and dynamic, as shown in Table 7. It shows that our dynamic threshold and automatic weight tuning strategies significantly improve the effectiveness of adversarial samples. However, if we set the static threshold, then most of the bytes generated by the generator will directly be set as zeros at the beginning of the experiments. It makes the generator lose the correct direction of attacks, leading to poor results.

6 CONCLUSION

In this paper, we propose an adversarial attack framework GAPGAN to generate adversarial samples against binaries based malware detection via GANs. In our model, we append adversarial payloads generated by the generator to the original malware binaries to craft an adversarial sample without damaging its original functionality. The experiments show that GAPGAN can effectively attack the state-of-the-art detector MalConv as well as other deep learning models with different structures. The results also show that our model outperforms other state-of-the-art attack approaches under current defenses in efficiency and effectiveness.

GAPGAN is the first practical end-to-end black-box attack framework against malware detection, posing a threat to the next generation of popular detection technology, i.e., raw binaries based malware detection. While our work focuses on malware binaries, it can easily be extended to other fields, such as adversarial text or graph generation. This makes GAPGAN a promising attack framework for improving the robustness of the defense methods of malware detection or other tasks that it can be used for.

ACKNOWLEDGEMENTS

This work was supported in part by the Application Program of New Model of Intelligent Manufacturing under the Grant No. 2016ZNZZ01, in part by Major Scientific Research Project of Zhejiang Lab under the Grant No.2018DG0ZX01.

REFERENCES

[1] Kenan E Ak, Joo Hwee Lim, Jo Yew Tham, and Ashraf A Kassim, ‘Attribute manipulation generative adversarial networks for fashion images’, *ICCV*, (2019).
[2] Hyrum S Anderson, Anant Kharkar, Bobby Filar, and Phil Roth, ‘Evading machine learning malware detection’, *Black Hat*, (2017).
[3] Bingcai Chen, Zhongru Ren, Chao Yu, Iftikhar Hussain, and Jintao Liu, ‘Adversarial examples for cnn-based malware detectors’, *IEEE Access*, **7**, 54360–54371, (2019).
[4] Scott E. Coull and Christopher Gardner, ‘Activation analysis of a byte-based deep neural network for malware classification’, in *IEEE Security and Privacy Workshops*, pp. 21–27, (2019).
[5] Chun-I Fan, Han-Wei Hsiao, Chun-Han Chou, and Yi-Fan Tseng, ‘Malware detection systems based on API log data mining’, in *IEEE Computer Society Signature Conference on Computers, Software and Applications*, pp. 255–260, (2015).
[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio, ‘Generative adversarial nets’, in *NIPS*, pp. 2672–2680, (2014).
[7] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, ‘Explaining and harnessing adversarial examples’, in *International Conference on Learning Representations*, (2015).
[8] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel, ‘Adversarial perturbations against deep neural networks for malware classification’, *arXiv:1606.04435*, (2016).
[9] Xiang He, Sibei Yang, Guanbin Li, Haofeng Li, Huiyou Chang, and Yizhou Yu, ‘Non-local context encoder: Robust biomedical image segmentation against adversarial attacks’, in *AAAI*, pp. 8417–8424, (2019).
[10] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, ‘Distilling the knowledge in a neural network’, *arXiv preprint arXiv:1503.02531*, (2015).
[11] Weiwei Hu and Ying Tan, ‘Generating adversarial malware examples for black-box attacks based on GAN’, *arXiv:1702.05983*, (2017).
[12] Youngjoon Ki, Eunjin Kim, and Huy Kang Kim, ‘A novel approach to detect malware based on API call sequence analysis’, *International Journal of Distributed Sensor Networks:11*, 659101:1–659101:9, (2015).
[13] Jin-Young Kim, Seok-Jun Bu, and Sung-Bae Cho, ‘Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders’, *Inf. Sci.:460-461*, 83–102, (2018).
[14] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli, ‘Adversarial malware binaries: Evading deep learning for malware detection in executables’, in *EUSIPCO*, pp. 533–537, (2018).
[15] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet, ‘Deceiving end-to-end deep learning malware detectors using adversarial examples’, *arXiv preprint arXiv:1802.04528*, (2018).
[16] Dongwook Lee, Junyoung Kim, Won-Jin Moon, and Jong Chul Ye, ‘Collagan: Collaborative GAN for missing image data imputation’, in *CVPR*, pp. 2487–2496, (2019).
[17] Juncheng Li, Frank R. Schmidt, and J. Zico Kolter, ‘Adversarial camera stickers: A physical camera-based attack on deep learning systems’, in *The International Conference on Machine Learning*, pp. 3896–3904, (2019).
[18] Shikun Liu, Edward Johns, and Andrew J. Davison, ‘End-to-end multi-task learning with attention’, in *CVPR*, pp. 1871–1880, (2019).
[19] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song, ‘Delving into transferable adversarial examples and black-box attacks’, in *International Conference on Learning Representations*, (2017).
[20] Fabio Martinelli, Francesco Mercaldo, Andrea Saracino, and Corrado Aaron Visaggio, ‘I find your behavior disturbing: Static and dynamic app behavioral analysis for detection of android malware’, in *International Conference on Privacy, Security and Trust*, pp. 129–136, (2016).
[21] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas, ‘Malware detection by eating a whole EXE’, in *AAAI Workshops*, pp. 268–276, (2018).
[22] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi, ‘Microsoft malware classification challenge’, *arXiv preprint arXiv:1802.10135*, (2018).
[23] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici, ‘Generic black-box end-to-end attack against state of the art API call based malware classifiers’, in *International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 490–510, (2018).
[24] Arindam Sharma, Pasquale Malacaria, and M. H. R. Khouzani, ‘Malware detection using 1-dimensional convolutional neural networks’, in *IEEE European Symposium on Security and Privacy Workshops*, pp. 247–256, (2019).
[25] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, ‘Intriguing properties of neural networks’, *arXiv preprint arXiv:1312.6199*, (2013).
[26] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G. Ororbia II, Xinyu Xing, Xue Liu, and C. Lee Giles, ‘Adversary resistant deep neural networks with an application to malware detection’, in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1145–1153, (2017).
[27] Xingxing Wei, Siyuan Liang, Ning Chen, and Xiaochun Cao, ‘Transferable adversarial attacks for image and video object detection’, in *IJCAI*, pp. 954–960, (2019).
[28] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song, ‘Generating adversarial examples with adversarial networks’, in *IJCAI*, pp. 3905–3911, (2018).
[29] Weilin Xu, Yanjun Qi, and David Evans, ‘Automatically evading classifiers: A case study on PDF malware classifiers’, in *The Network and Distributed System Security Symposium*, (2016).
[30] Minfeng Zhu, Pingbo Pan, Wei Chen, and Yi Yang, ‘DM-GAN: dynamic memory generative adversarial networks for text-to-image synthesis’, in *CVPR*, pp. 5802–5810, (2019).