GAN-Based Adversarial Patch for Malware C2 Traffic to Bypass DL Detector

Junnan Wang[1,2], Qixu Liu[1,2], Chaoge Liu[1,2(✉)], and Jie Yin[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
liuchaoge@iie.ac.cn

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. The constantly evolving malware brings great challenges to network security defense. Fortunately, deep learning (DL)-based system achieved good performance in the malware command and control (C2) traffic detection field due to its excellent representation capabilities. However, DL models have been shown to be vulnerable to evasion attacks, that is, DL models can easily be misled by adding subtle perturbations to the original samples. In this paper, we propose a GAN-based evasion method, which can help malware C2 traffic bypass the DL detector. Our main contributions contain: (1) directly generate adversarial traffic that can implement malicious functions by inserting additional adversarial patches in the original flow; (2) adaptively imitating victim's normal traffic by training GAN in victim environment, and introducing transfer learning to reduce the additional victim resource usage caused by GAN training. Results show that the adversarial patch generated by GAN can prevent malware C2 traffic from being detected with 51.4% success rate. The higher time efficiency and smaller malware impact make our method more suitable for real attacks.

Keywords: Malware C2 traffic · Evasion attacks · GAN · Transfer learning

1 Introduction

Malware allows attackers to remotely control computers to perform criminal activities using Command and Control (C2) channels, which has posed great

This work is supported by the Youth Innovation Promotion Association CAS (No. 2019163), the National Natural Science Foundation of China (No. 61902396), the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDC02040100), the Key Laboratory of Network Assessment Technology at Chinese Academy of Sciences and Beijing Key Laboratory of Network security and Protection Technology.

D. Gao et al. (Eds.): ICICS 2021, LNCS 12918, pp. 78–96, 2021.
https://doi.org/10.1007/978-3-030-86890-1_5

challenges to network security. Fortunately, it can be mitigated by detecting C2 channels on the network.

Among the rich malware C2 traffic detection methods, the deep learning (DL)-based detection method has been widely used and researched because it is an end-to-end solution that can automatically learn feature representations from raw traffic data [13,15,16,24,25]. In this paper, we mainly focus on the DL-based malware C2 detection model taking raw malware C2 traffic data as input, which is a state-of-the-art detection method [19].

After the success of DL in the field of malicious traffic identification, their robustness and security issues have become the subject of much discussion by security researchers. In 2014, Szegedy et al. [23] first discovered that due to their linear nature, well-performing DL models are vulnerable to adversarial examples, which are intentionally crafted by adding tiny perturbations to mislead the DL model. After that, how to use adversarial machine learning (AML) ideas to construct adversarial malicious traffic to bypass detection also received attention.

Different from AML in the image recognition field, the construction of adversarial malware traffic has many unique constraints and challenges:

1. Ensure that the generated adversarial traffic can retain the original malicious functions, and the basic network protocol format will not be destroyed.
2. Directly generate adversarial traffic without the help of other attachments, rather than generating adversarial features that are just intermediate results of evasion attacks.
3. How to make adversarial traffic adaptively imitate the normal traffic of individual victims, so as to ensure that it can be applied to a variety of terminals. While those imitations that are limited to specific normal application traffic will fail when the application is rarely used on some victims.

These three challenges are progressive. Challenge-1 represents effectiveness, challenge-2 means usability, and challenges-3 is a practical requirement that proposed based on real attack scenarios.

Unfortunately, none of the existing work can solve the above problems at the same time. [8] and [10] directly treat traffic samples as image samples, even cannot meet challenge-1. [14] and [7] can only generate adversarial features violate challenge-2. What counts is, most of the current work does not consider challenge-3, which is the most realistic requirement in the malware traffic evasion field.

In light of the challenges, we present an adaptive evasion attack on DL-based detectors in practical settings. Specifically, we propose a GAN-based method that can directly generate sample-independent adversarial patches (*adv_patch*es). Malware can directly send a packet encapsulating the *adv_patch* in C2 communication to bypass the DL-based detector, without other attachments' help or complex source code modification. And the C2 flow that encapsulates *adv_patch* is called adversarial flow, which can directly bypass the DL detector. Therefore, our method can solve challenge-1 and challenge-2 mentioned above.

In order to adaptively simulate a specific victim's traffic, there are two solutions. One is to collect large-scale normal traffics on the victim and send them back to train GAN, but that is unrealistic because it will increase the exposure risk of the C2 channel. The other is to train the GAN model on the bot, which

will increase the exposure risk of malware on the victim. We choose the latter to solve challenge-3. At the same time, in order to reduce the extra resource utilization caused by GAN training on the victim, we introduced transfer learning (TL) technology to further improve the similarity between malware C2 traffic and victim normal traffic at a small cost. Results show that our method can not only achieve a success rate of 51.4%, but also has a good performance in time efficiency and a minor negative impact on malware.

Our major contributions are elaborated as follows:

1. We propose a GAN-based black-box malware C2 traffic evasion method to bypass the DL detector. Under the premise of functionality preserving and network protocol compliance, we can directly obtain adversarial traffic by inserting an additional *adv_patch* packet, without other attachments or complex source code modifications.
2. Our method enables adversarial traffic to adaptively imitate host-side normal traffic, that is, dynamically adjust adversarial traffic according to the traffic characteristics of different victim terminals, which is more practical and concealed. We also introduce TL to alleviate the additional system resource occupation caused by GAN training on the victim.
3. We design a real-life experiment to evaluate the proposed method, and proved its practicability and efficiency from the perspectives of evasion performance, time performance, and impact on malware.

As far as we know, this is the first work on adaptive evasion method, that considers and comprehensively evaluates the negative impact of the evasion method on malware.

The rest of the paper is organized as follows: We start by providing backgrounds and related works in Sect. 2. Section 3 introduces the overview of our evasion method. Section 4 elaborate experimental setting up. Experimental results and findings are shown in Sect. 5. Finally, we conclude in Sect. 7.

2 Background and Related Work

2.1 Background–Malware Traffic Detection

With the development of machine learning technology, DL technology has been widely used in the malware C2 traffic detection field. On the one hand, DL-based methods can automatically learn deep abstract feature representations, thereby solving the dilemma of manual feature engineering. On the other hand, compared with the traditional ML methods, DL-based methods also have a considerably higher capacity to learn complex patterns, so they can deal with large-scale encryption and unknown malicious traffic detection well.

According to the different model inputs, DL-based classifiers can be divided into statistic feature-based and raw data-based. [18] and [19] have proved that DL-based model, using raw flow representations as input, can outperform other detectors, while without requiring any prior knowledge.

In this work, we particularly focus on the vulnerability of DL-based malware C2 traffic detector, which taking raw byte stream flow data as input.

[25] proposed a stacked autoencoder (SAE) based network protocol identification method using raw traffic data, and achieved high accuracy.

[24] proposed an end-to-end malware traffic classification method with 2D-CNN taking the first 784 bytes of flow. Lotfollahi et al. [16] combines SAE and 1D-CNN, and takes the first 1500 bytes of IP header and payload data as input.

Byte Segment Neural Network (BSNN) [13] and Flow Sequence Network (FS-Net) [15] are both RNN-based traffic classification methods. The difference is that BSNN takes raw payload as input, while FS-Net's input is raw flow.

In summary, the current DL model for malware traffic detection often takes the first few bytes of the raw byte stream as input, then learns the abstract representation through multi-layer neural networks, and the final prediction is calculated by the softmax layer.

2.2 Related Work–Malware Traffic Evasion

While the malware traffic detection method is constantly improving, attackers are also exploring evasion techniques to avoid detection. Evasion and detection technologies are innovating in the tit-for-tat game, trying to be able to overwhelm the opponent.

In order to bypass blacklist-based detection, attackers introduced dynamic resolution technologies such as DGA and Fast-Flux to replace the hard-coding method. Introducing techniques such as encryption and data encoding to cover up the payload, so the payload-based detection is invalidation. To bypass the detector based on statistical characteristics, the attacker introduces technologies such as protocol tunnels and online-social networks (OSN) to construct covert channels and overwhelms malicious traffic in mass normal traffic.

In recent years, with the widespread application of DL in the field of malicious traffic detection, many researchers have also tried to use the inherent security vulnerabilities of DL to bypass DL-based detectors. We divide these tasks into two categories according to the adversarial output.

Feature-space attack refers to a type of attack method that can only generate adversarial feature vectors. However, the mapping process from traffic samples to traffic characteristics is irreversible and non-differentiable. So, it is difficult to reversely infer traffic samples, even if the adversarial feature vector is known. In other words, this attack method is just theoretical proof that DL-detector is vulnerable to evasion attacks, and cannot be directly used for malicious delivery. This attack method can only be used as theoretical proof that the detection system is vulnerable to attack.

Clements [8] and Ibitoye [10] used classic AML algorithms (FGSM [9], BIM [11], PGD [17], C&W [5], JSMA [21] etc.) to evaluate the robustness of DL-based network intrusion detection system (NIDS) against adversarial attacks in a white box scenario. They directly convert the traffic samples into gray images and perturb the 'pixel' indiscriminately. No consideration is given to the fine structure of traffic samples and the constraints of maintaining malicious functions.

Lin et al. [14] proposed a black-box evasion attack method-IDSGAN, which uses GAN to generate adversarial statistical features of malware traffic. Although IDSGAN can ensure the effectiveness of the intrusion by changing only non-functional features, it does not consider the dependence between statistical features. FENCE [7] solves this problem by combining gradient-based methods and mathematical constraints to maintain consistency in a family of dependencies.

Traffic-space attack refers to attack methods that can generate adversarial traffic samples. Unlike feature-space attacks, traffic-space attack methods can be powerful weapons for attackers to bypass malware traffic detectors.

Novo [20] used the classic AML algorithm FGSM [9] to perturb the encrypted C&C malware traffic characteristics and achieved a white-box adversarial attack against the detector. It requires additional traffic proxy or complex source code modification to obtain the final adversarial traffic. And white-box attacks require a full understanding of the detector, which is difficult to attain in real life.

Rigaki et al. [22] proposed a method that uses GAN to generate statistical features similar to Facebook traffic, thus adjust the behavior of the malware C2 traffic to avoid detection. FlowGAN [12] is no longer limited to Facebook traffic, can dynamically morph traffic features as any other "normal" network flow to bypass censorship. However, in these two works, GAN can only output adversarial features. If the attacker wants to obtain adversarial traffic based on these adversarial features, he needs to make complex modifications to the malware source code, which will cause delays to the malware's communication channel.

In Attack-GAN [6], the generator is viewed as an agent in RL, which can craft adversarial traffic conditioned to the security domain constraints to ensure attaining the attack functionality. But Attack-GAN needs to constantly access IDS to obtain prediction results, which is unrealistic in real-attack.

Unlike the works we reviewed in this section, in this paper we focus specifically on how to directly generate adaptive adversarial traffic without the help of any other additional components. Only by adaptively imitating victim traffic, can the adversarial traffic seemed to be normal-like in bots with different characteristics. Moreover, while most related work assesses the performance of the evasion attack on malware traffic detectors, they do not consider the impact of the proposed methods on malware, nor do they consider the practicality of the method. We properly solve these problems by performing a real-life experiment in this work.

3 Method

In this section, we use some technical terms to represent various roles in a malware C2 traffic evasion attack. *Malware* means the code used to achieve C2, *master* means the computer of the attacker, *victim* means malware-infected hosts. The *adversary* tries to control the victim by malware, while the *defender* tries to protect the victim through a DL-based malware C2 traffic detector.

3.1 Thread Model

Adversary's Goal. From the perspective of the CIA (confidentiality, integrity, and availability), attackers try to reduce the availability of detectors by camouflaging malware C2 flow.

Adversary's Knowledge. The attacker knows that the target network may be protected by a flow-level detector based on DL. However, the attacker does not need to master any prior knowledge about the detector, such as the architecture, parameters, or training data.

Adversary's Capability. The attacker has full control of the C2 server and partial control of the victims, so he can update victims to change their communication behaviors as he wants.

3.2 Framework

Our framework is inspired by [4], that attackers can mislead the classifier by placing a gradient-based sample-independent *adv_patch* in a specific area. *Adv_patch* is effective because it can calculate the most effective perturbation to the DL model by using gradient backpropagation according to the gradient passed by the discriminator. When inputting the detector, *adv_patch* can dominate the feature learning of the detector, thereby misleading the detector

The idea of *adv_patch*es suits malware C2 traffic evasion well. On the one hand, through this method, we can directly operate on the traffic samples and output traffic samples with actual attack functions.

On the other hand, traffic samples have more complex network protocol constraints than images, and there is a need to keep malicious functionality in the perturbed sample. That makes many AML algorithms designed for images unavailable. While our method can better meet the constraints of functionality preserving and network protocol.

Specifically, our method includes two modules, a GAN-based generation module and a TL-based transfer module.

To better illustrate our method, we propose two terms. We define **universal benign communication (UBC)** traffic as benign communication traffic that has multiple types benign communication traffic and can cover a variety of benign communication behavior characteristics, while **host benign communication (HBC)** traffic only includes benign communication traffic from a specific host. HBC is more specific and targeted, while UBC is more versatile and generalized.

In the generation module, we use GAN to imitate the normal traffic to generate *adv_patch*. By inserting it into the original flow, we can obtain adversarial malware C2 traffic that can mislead the DL-based malware C2 traffic detector.

In the transfer module, we retrain the GAN model in the victim environment to adaptively simulate the victim's normal flow. TL is used in this module because it can realize the transfer from imitating UBC traffic tasks to adaptively imitating specific HBC traffic tasks with a smaller data scale requirement and system resource cost.

The workflow of our method is shown in Fig. 1. It can be divided into three stages: the pre-training stage in the master environment, the fine-tuning stage in the victim environment, and the practical stage.

The pre-training stage in the master environment refers to the pre-training of GAN performed by the attacker before the weapon is delivered. In a fully

controllable master environment, the attacker can construct a training dataset by capturing the original malware C2 traffic and the UBC flow. After the pre-training stage is completed, the attacker compresses and packs the GAN model together with the malware, and delivers them to the victim.

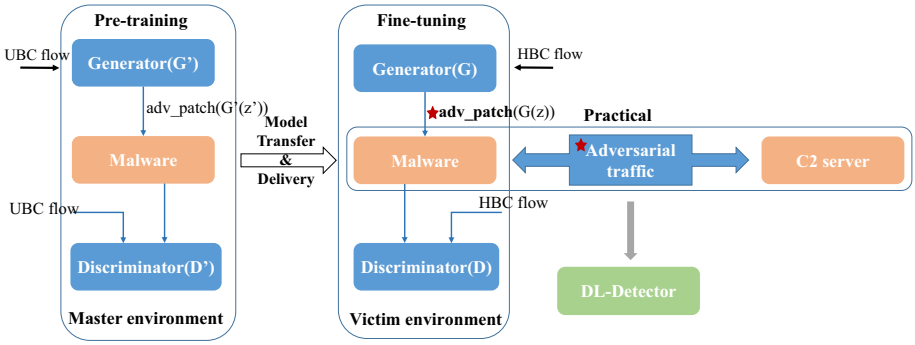

Fig. 1. System framework

Next is the fine-tuning stage in the victim environment. The malware will call the packet sniffer module to build up the HBC traffic profile, which is used for fine-tuning the GAN model so that the specific characteristics of the victim's normal traffic can be more accurately embedded in output *adv_patch*.

Finally, in the practical stage, the fine-tuned GAN model can be used to camouflage malware C2 traffic. Specifically, the malware will first access the generator to obtain the *adv_patch* before communicating with the C2 server, and send out the packet encapsulating the *adv_patch* after the TCP three-way handshake, followed by other original malicious packets.

3.3 Generation Module – WGAN

As the core of the method, we choose GAN as the generation module. Generative Adversarial Networks (GAN), are a class of DL-based generative model. The GAN model architecture involves two sub-models: a generator (G) that is trained to generate new examples, and a discriminator (D) that tries to classify examples as either real or fake. The final goal is to make the data obtained by the generator becoming more similar to the real data.

In the context of malware C2 traffic evasion, the generator is responsible for learning the characteristics of the normal communication traffic and generating fixed-length *adv_patch*es to help malware C2 traffic evading the DL-based detector. While the discriminator plays a similar role to the detector, which is used to determine whether the generated confrontation traffic is sufficiently similar to the normal traffic and pass gradients to the generator for parameter tuning.

Specifically, we use Wasserstein GAN (WGAN) [3]. Instead of JS divergence, WGAN introduces Wasserstein distance (calculate as Eq. 1) to calculate the

distance between the generated distribution and the real distribution as the loss function. WGAN can solve many problems of vanilla GAN, such as unstable training and collapse mode, and Wasserstein distance can be used as an indicator of training progress.

We choose WGAN not only because of its excellent learning ability, but also because its adversarial fits well with the confrontation scenarios of malicious traffic detection and evasion attacks.

$$W(p_r, p_g) = \inf_{\gamma \sim \prod(p_r, p_g)} E_{(x,y)\sim\gamma}\left[\|x - y\|\right] \tag{1}$$

The loss function of WGAN is:

$$L^D = E_{x\sim p_{data}}[D(x)] - E_{z\sim p_{(z)}}[D(G(z))] \tag{2}$$

$$L^G = E_{z\sim p_{(z)}}[D(G(z))] \tag{3}$$

$$W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01) \tag{4}$$

In our method, during the training process, the generator will take benign communication flow as input, attempt to generate a fixed-length *adv_patch*, and return it to the malware. The discriminator takes the new malware C2 flow and the benign communication flow as input, and learns how to distinguish between them. During the application process, the generator will be requested by malware to obtain a new *adv_patch*.

We adopted the classic model in [3] as our generation module. One small difference is that in order to avoid that the discriminator is too powerful to guide the parameter learning of the generator well, we have removed several convolutional layers in the discriminator to reduce the complexity of the discriminator. At the same time, this can also further reduce the size and parameter number of the GAN model. It is worth mentioning that in order to insert *adv_patch* into

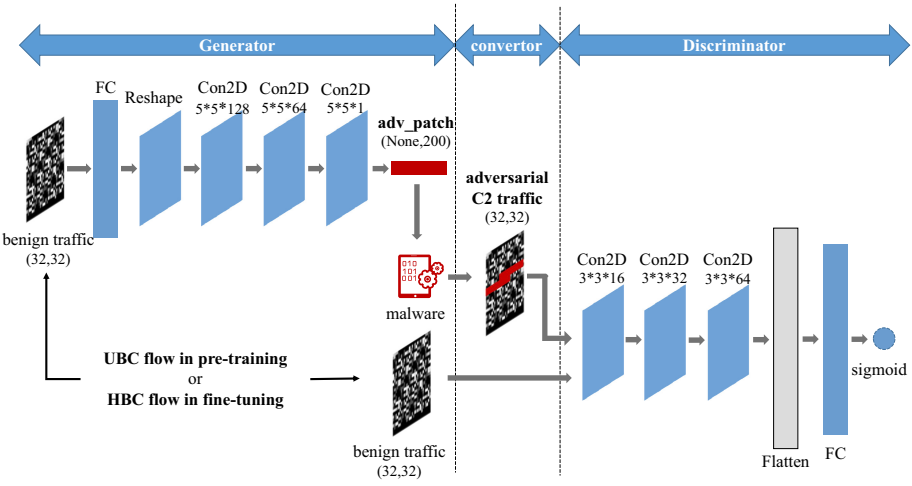

Fig. 2. The architecture of the GAN model we used

the original malicious traffic, we built a concatenate layer between the generator and the discriminator to facilitate gradient propagation. The architecture and hyperparameters setting of the GAN model are shown in Fig. 2.

3.4 Transfer Module–Transfer Learning

Transfer Learning is a machine learning method that transfers knowledge from the source domain (task A) to the target domain (task B), so that task B can achieve better learning results. Usually applicable to situations where the amount of data in the source field is sufficient, but in the target field is small.

In the context of malware C2 traffic evasion, we regard the pre-training process in a fully controllable master environment as task A. Task A attempts to train the generator to generate a fixed-length payload and insert it into the malicious communication flow, making it difficult for the discriminator to distinguish the newly constructed malicious flow from the UBC flow.

Task B is a fine-tuning process that occurs in the victim environment. In this process, the generator will use the HBC traffic captured in the victim as a template to learn how to construct malware C2 traffic.

The difference between the two tasks is that the traffic distribution of task B is more specific and concentrated. To some extent, the distribution of UBC traffic and HBC traffic is similar, so it is very suitable to use parameter-TL.

Specifically, on the premise of further improving evasion performance, applying TL has the following two advantages:

(1) Reduce the training cost in the victim environment: Parameter-TL can reduce the later training cost, by only training a small part of the parameters. Therefore, we can reduce the victim's perception of the fine-tuning process and avoid being detected due to taking up too many system resources.
(2) Suitable for small datasets: It is unrealistic to train a large neural network from scratch to capture a large amount of communication traffic in the victim environment. While TL can handle this problem well because there are fewer parameters to learn. Besides, we can rely on TL to generate more victim-specific adversarial C2 traffic.

4 Experiment

4.1 Dataset

In order to evaluate the performance of our method, we constructed a data set by selecting 12 botnet traffic from the public dataset CTU and the UBC traffic from the ISOT dataset. The detail of the dataset we summarized is shown in Table 1.

The dataset can be divided into two parts, one part is used to train and test the DL-based detector, the other part is used to train and test our proposed evasion method. Each part includes both malware and benign traffic.

The dataset for the detector is the dataset used by the defender. In this part, malware traffic includes 9 malware families from the CTU covering a variety of commonly used C2 channels. The benign traffic is captured from the 10 computers in our laboratory environment, which can cover many different types of normal traffic. The reason for this setting is that in order to protect the internal network in a targeted manner, the defender often uses the specific internal normal traffic to train the DL-based detector.

The dataset for WGAN is the dataset used by the attacker. The malicious traffic is 5 malware families selected from CTU, Neris and Virut are also used to train the detector, but the Neris traffic files come from different captures. The benign traffic includes UBC traffic from the ISOT for pre-training, and internal capture id01 for fine-tuning. The reason for this setting is the fact that it is difficult for an attacker to obtain a large amount of internal traffic. Therefore, pre-training can only use public datasets, and in fine-tuning stage, a small volume of traffic samples can be used to adaptively simulate specific HBC traffic.

The original data needs to be preprocessed before inputting into the model. The data preprocessing process mainly includes three steps.

1. Split. The captured pcap file is divided into bidirectional flows according to the five-tuple $<sip, dip, sport, dport, protocol>$. We use the open-source tool pkt2flow [1] to complete this operation.
2. Filter. After the split, we only keep the flow with valid data transmission, and filter out the flow that the TCP connection is not fully established or is closed immediately after establishment.
3. Anonymization. We perform anonymization on traffic data to avoid specific information such as IP and MAC misleading the detection model. Specifically, we replace them all with 0.

Table 1. Details of the dataset

		Malware family	Flow num.	C2 channel
Detector	Malware	CTU-44-Rbot	2745	IRC
		CTU-47-Menti	216	TCP
		CTU-49-Murlo	1986	TCP
		CTU-42-Neris	1583	HTTP
		CTU-54-Virut	3451	HTTP
		CTU-127-Miuref	1286	HTTP
		CTU-125-Geodo	6320	HTTP
		CTU-141-1-Bunitu	6143	HTTP/HTTPS
		CTU-348-1-HTbot	10000	HTTP/HTTPS
	Benign	id01-id10	39452	–
WGAN	Malware	CTU-50-Neris	19282	HTTP
		CTU-54-Virut	3451	HTTP
		CTU-264-2-Emotet	10000	HTTPS
		CTU-346-1-Dridex	8022	HTTPS
		CTU-327-1-Trickbot	25924	HTTPS
	Benign	UBC traffic	17144	–
		HBC traffic-id01	9706	–

4.2 Hyperparameters

The Length of the New Packet

The output dimension of GAN is fixed, so we need to determine the length of the generated *adv_patch*.

In order to simulate the normal data packet as much as possible, the mode of the payload length of the UBC traffic is selected as the length of the *adv_patch*. This can make the newly added packet look closer to the normal data packet at least in terms of statistical characteristics.

We perform statistics on the captured UBC traffic, and obtain the distribution of its payload length, which is shown in Fig. 3.

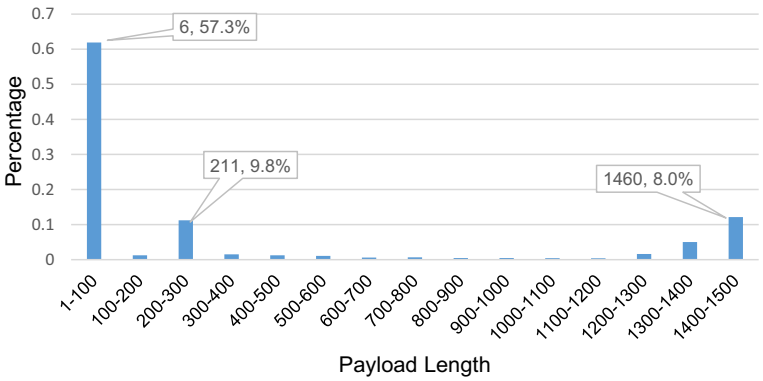

Fig. 3. The payload length distribution of universality normal traffic

Through the inspection of the original traffic data, we found that the reason for many packets with 6 bytes payload is that the Ethernet data link layer will automatically pad the frame with 0 to ensure that the minimum length of the frame is 64 bytes. While 1460 is the maximum segment size of TCP transmission. TCP segmentation will be performed when large-size data is transmitted, resulting in a large number of packets with a payload length of 1460.

Based on the above findings, we set the length of the *adv_patch* to 200, which is close to the second most frequent payload length 211, and it is also convenient for quantification and calculation.

Insertion Position of the New Packet

After getting *adv_patch*, we need to decide when to send it. Through the investigation of the current DL-based detection work, we found that in order to balance the model accuracy and complexity, researchers often intercept part of the traffic data for learning deep representations as the basis for classification. Wang et al. [24] proved that the first few packets, up to the first 20 packets, are sufficient for correct accuracy, even for encrypted traffic.

In this case, to guarantee the impact on the DL-based detector, we decided to add the *adv_patch* packet after the TCP three-way handshake process. That means malware should send out the packet encapsulating the *adv_patch* once the TCP connection is established. In this way, it can be ensured that the carefully crafted *adv_patch* can appear in the visual field of the detector.

4.3 Detector

Marín et al. [19] designed a series of experiments to prove that the raw flows & DL-based malicious traffic detection models outperform traditional ML-based models, which use specific hand-crafted features based on domain expert knowledge as input.

RawFlows's input is a tensor of size (n, 1, m), where n is the number of bytes, and m represents the number of packets. They set $n = 100$ and $m = 2$, that is, only the first 100 bytes of the first two packets in a flow are considered.

In our work, we refer to their DL architecture and make certain extensions on it. Our adjustment is mainly reflected in the hyperparameters of the model. On the one hand, considering that our purpose is to detect malware C2 traffic, only sampling the first 2 packets may lose a lot of flow information. Moreover, because the proportion of C2 traffic is relatively small, our data volume does not reach the scale of RawFlows. In order to provide more flow information to the model, we set $n = 200$ and $m = 8$.

At the same time, to accommodate the expansion of input, we also need to adjust the model structure accordingly to increase the expressive ability of the model. Specifically, we refer to the DL architecture of raw packets in [19] to reshape our DL-based malware C2 traffic detector. The DL architecture of the target detector in this article is shown in Fig. 4.

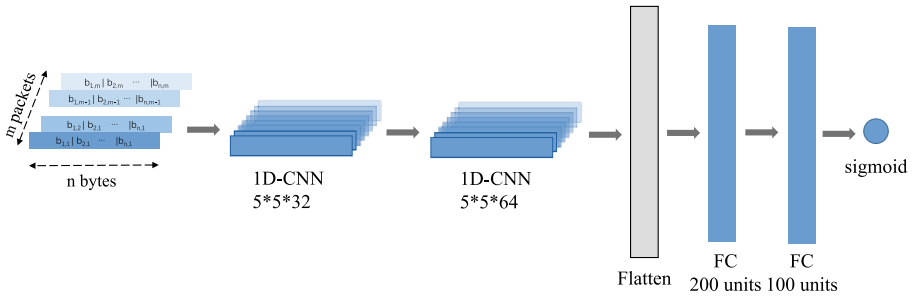

Fig. 4. DL architecture for detector

5 Results

As mentioned earlier, we designed comprehensively evaluate the effectiveness of our method from the perspectives of evasion performance, time performance, and impact on malware.

Specifically, in the pre-training stage, we will use the traffic of a specific family and the UBC traffic to train the GAN model. Then fix the discriminator and all convolutional layers of the generator, and fine-tune the model using the victim's locally captured benign samples and 1000 malicious samples. Finally, in the practical stage, the generator will directly or be called by malware to generate the *adv_patch*, which is inserted into the original malicious C2 flow to construct the adv-C2 traffic. The newly crafted samples are input to the detector for prediction. So we can evaluate the evasion performance through the change in the detector's recall score.

5.1 Evasion Performance

We randomly select 3000 flow samples from each of the 5 families for testing, and get the results in Table 2. We use *DetectionRate* and *EvasionRate* to measure the performance of the detector and our method, respectively. They are calculated according to Eq. 5 and Eq. 6.

$$DetectionRate = Recall = \frac{Number\ of\ detected\ malware\ flow}{Number\ of\ all\ malware\ flow} \tag{5}$$

$$EvasionRate = \frac{Successful\ evasion\ attempts}{All\ evasion\ attempts} \tag{6}$$

The first column in Table 2 is the original detector recall score for 5 malware families, indicating that the detector we use has good accuracy and generalization performance.

From the table, we can see that the proposed method can achieve an evasion rate up to 51.4%, while reducing the recall of the detector to 45.4%, less than 50%, which means that it is difficult for the detector to resist our attack method.

Table 2. The evasion performance of our attack method

Family	Detection rate		Evasion rate	
	init_sample	adv_sample	Pre-training	Fine-tuning
CTU-50-Neris	99.00%	62.63%	30.07%	36.37%
CTU-54-Virut	99.83%	67.07%	24.93%	32.77%
CTU-264-2-Emotet	97.47%	48.67%	45.37%	48.80%
CTU-346-1-Dridex	96.80%	45.40%	47.50%	51.40%
CTU-327-1-Trickbot	99.93%	63.37%	31.40%	36.57%

By comparing the last two columns in the table, we can find that after the fine-tuning process in the victim environment, the evasion rate generally increases by 7%–21%, which can prove the effectiveness of the TL.

In addition, we have also observed that there are certain differences in the evasion performance of different malware families, from 51.4% in Dridex to 32.77%

in Virut. Through the analysis of the original flow samples, we believe that the bypass rate is mainly affected by the following factors: 1) The detection rate of the detector to the malware. Take Dridex and Neris as examples. The detector has seen the Neris samples during the training phase, so it can easily get the characteristics of Neris and achieve a relatively high recall, while the initial recall of the unseen Dridex family is relatively low. Therefore, a carefully generated *adv_patch* is more likely to mislead the detector, who has not seen Dridex before.

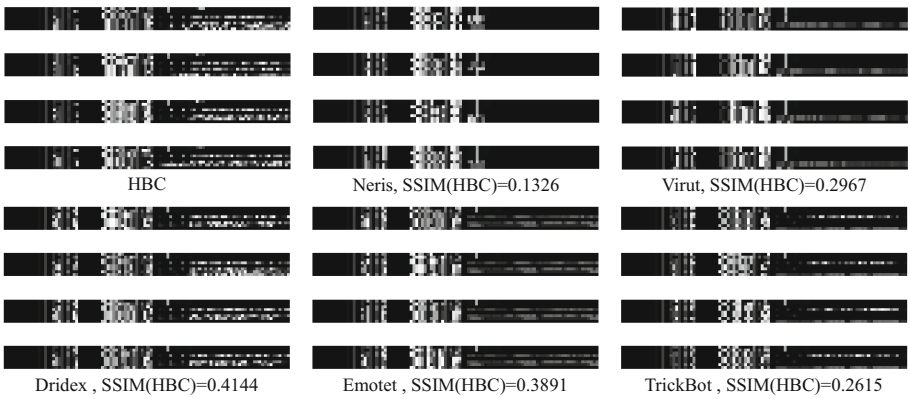

Fig. 5. Visualize flow samples of different families

2) The similarity between the malware C2 sample and the target benign sample. We turn the input of the detector into a grayscale image for direct observation, which is shown in Fig. 5. We also calculate the average structural similarity score (SSIM, a common measure of image similarity) between each malware family and HBC to measure how similar each family's traffic is to HBC. Take Dridex and Trickbot as examples, both of them are mostly TLS traffic and have high similarities. But through the visualization of the samples, we found that the Dridex C2 samples are more similar to the target benign C2 samples, so it is easier to fool the detector by adding disturbances on Dridex C2 flow.

5.2 Time Performance

In order to evaluate the time performance of our method, we recorded the fine-tuning time elapsed and the evasion rate of 5 CTU malware families under different training hyperparameters (batch_size, epochs).

From Fig. 6, we can find that the time elapsed of batch_size = 128 is roughly 1.4 times that of batch_size = 256, but what needs to be noted is that larger batch_size often means larger memory consumption. The overall fine-tuning time consumption will increase linearly with the increase of epochs number, while the evasion rate is different. The evasion rate at epochs = 100 is significantly higher than that of epochs = 50, but the evasion rate at epochs = 200 is not significantly

improved compared to epochs = 100. Therefore, for efficiency considerations, we think it is reasonable to set the training hyperparameters to (batch_size = 128, epochs = 100) or (batch_size = 256, epochs = 100). The attacker can trade-off between shorter training time and lower resource occupancy as needed.

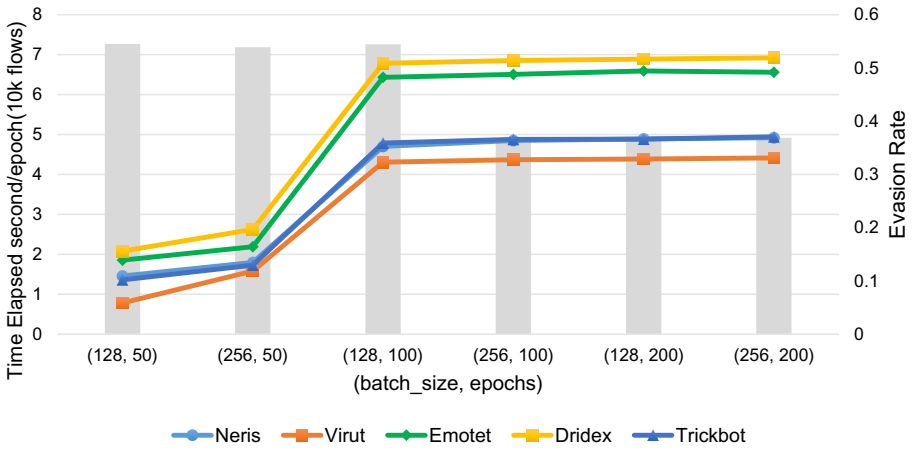

Fig. 6. Evasion rate and time elapsed under different setup of fine-tuning process

6 Real-Life Experiment

We believe that in order to develop evasion methods that can be applied in real attacks, a comprehensive evaluation from the perspective of practicality must be conducted, rather than just proving the effectiveness.

In this work, we complete the evaluation of practicability by designing a real-life experiment. Specifically, we built a custom malware on the basis of Byob. By requesting GAN in real-time to obtain *adv_patch*es and communicating with the server through C2 channel, we obtain a real-life scenario.

It should be pointed out that the real-life experimental settings are exactly the same as those described in Sect. 4, except that the source of the malware C2 traffic. The effectiveness experiment uses public traffic dataset, while the real-life experiment uses traffic that generated by our custom malware.

6.1 Custom Malware

To evaluate our method we used the open-source post-exploitation framework called Byob [2]. Byob was modified to receive the *adv_patch* from the GAN generator and send it after TCP three-way handshake. Byob consists of a client and a C2 server that is written in python. We deploy the C2 server in a Linux virtual machine and the infected victim in a Windows 8 virtual machine respectively.

The communication between client and server is established over HTTP. In order to allow the client to receive the payload generated by the generator, we modify the core module of the client so that every time the client communicates with the server, it will first call the trained generator to obtain the *adv_patch*, and send it once the connection is successfully established.

To continuously obtain malicious communication traffic, we write a script to let Byob client performs the following actions in sequence:

- checks if the server is online.
- sends a heartbeat message with a unique identifier.
- retrieves a command id from server.
- executes the corresponding module.

In this way, we obtained 17,536 Byob C2 flow, which is used as the malware C2 flow dataset for training GAN.

6.2 Impact on Malware

We evaluated the practicability of our method from two perspectives: malware C2 channel efficiency impact and resource utilization.

Malware C2 Channel Efficiency

For malware, the transmission efficiency of the C2 channel is a very important requirement. The significant C2 channel delay caused by evasion methods will reduce the communication efficiency of the victim and the C2 server.

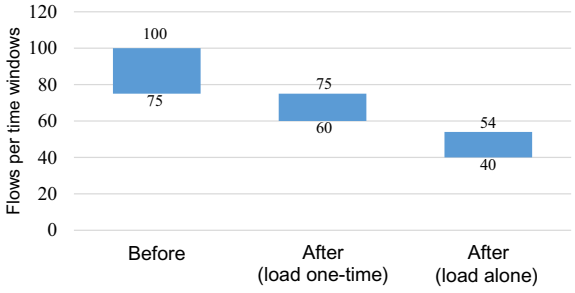

Fig. 7. Malware C2 channel efficiency before and after applying our method

Therefore, we evaluated malware C2 channel efficiency before and after applying our method. By calculating the number of C2 flows sent by malware before and after applying our method in a time window, we obtained Fig. 7.

Before applying, we program the malware to communicate with the C2 server every 3 s, so there will be 75–100 flows within a 5 min time window. After applying our method, this number dropped to 40–75. That is to say, whether we choose to access a large number of *adv_patch*es at one time, or request the generator every time before the start of each communication, we can guarantee at least 8 C2 communications per minute, which makes it a feasible channel for a C2.

Resource Utilization

Another major impact of the evasion method on malware is that it will increase resource usage in the victim environment. To measure the resource utilization of our method, we recorded the CPU and memory usage of GAN training and inferring, as shown in Fig. 8. Figure 8 shows that a large amount of resource occupancy is mainly caused by GAN training, and the resource usage of the GAN generation process (malware calls GAN to generate adv-patch) is relatively equivalent to malware calling other malicious functions.

As for GAN training, although the fine-tuning stage has obvious optimizations in memory utilization compared to the pre-training stage, it seems that there is no improvement in CPU utilization. That is constrained by the maximum CPU capacity. The CPU utilization in the fine-tuning stage and the pre-training stage is close to 100%, but by comparing the training time of each epoch, we can find that the pre-training is about 14.35 times that of the fine-tuning.

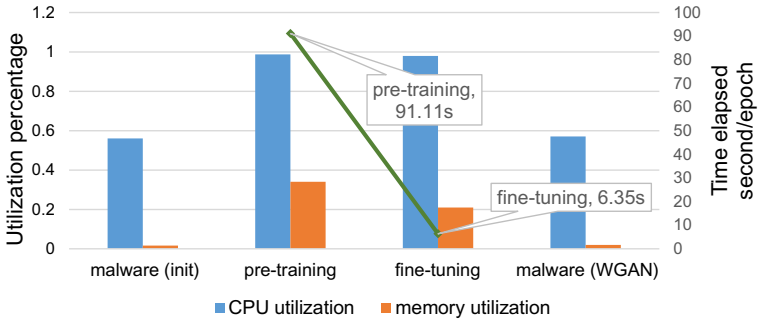

Fig. 8. Changes in resource utilization caused by GAN model training

Therefore, we can conclude that TL can reduce the resource utilization in the victim environment, thereby reducing the additional exposure risk caused by our evasion method. Although our method still brings additional resource consumption, it is inevitable. In any case, we believe that our method has certain advantages over other methods in terms of impacts on victim environment.

7 Conclusion

In this paper, we focus on how to use the DL model's vulnerability to craft adversarial samples in the field of malware C2 traffic, and propose a GAN-based evasion attack method. Specifically, GAN generates *adv_patch* by simulating the distribution of benign samples, so that malware C2 traffic containing that *adv_patch* can mislead DL-based detectors. Our method is not only able to adaptively simulate the normal traffic of the victim, but also has less negative impact on the malware. These two advantages make our method more suitable for real attack scenarios. The results show that our method can not only achieve a bypass

rate of 51.4%, but also has relatively little impact on malware C2 channel and less victim resource usage.

In future work, we plan to explore the influence of hyperparameters such as patch length and embedding position on the evasion rate. At the same time, we will seek ways to further reduce the negative impact of evasion methods on malware, such as model size and resource utilization.

References

1. https://github.com/caesar0301/pkt2flow
2. https://github.com/malwaredllc/byob
3. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN (2017)
4. Brown, T.B., Mané, D., Roy, A., Abadi, M., Gilmer, J.: Adversarial patch. CoRR abs/1712.09665 (2017). http://arxiv.org/abs/1712.09665
5. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57. IEEE (2017)
6. Cheng, Q., Zhou, S., Shen, Y., Kong, D., Wu, C.: Packet-level adversarial network traffic crafting using sequence generative adversarial networks (2021)
7. Chernikova, A., Oprea, A.: FENCE: feasible evasion attacks on neural networks in constrained environments (2020)
8. Clements, J., Yang, Y., Sharma, A., Hu, H., Lao, Y.: Rallying adversarial techniques against deep learning for network security (2019)
9. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2015)
10. Ibitoye, O., Shafiq, O., Matrawy, A.: Analyzing adversarial attacks against deep learning for intrusion detection in IoT networks. In: 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1–6 (2019)
11. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial machine learning at scale. arXiv preprint arXiv:1611.01236 (2016)
12. Li, J., Zhou, L., Li, H., Yan, L., Zhu, H.: Dynamic traffic feature camouflaging via generative adversarial networks. In: 2019 IEEE Conference on Communications and Network Security (CNS), pp. 268–276 (2019)
13. Li, R., Xiao, X., Ni, S., Zheng, H., Xia, S.: Byte segment neural network for network traffic classification. In: 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), pp. 1–10 (2018)
14. Lin, Z., Shi, Y., Xue, Z.: IDSGAN: generative adversarial networks for attack generation against intrusion detection (2019)
15. Liu, C., He, L., Xiong, G., Cao, Z., Li, Z.: FS-Net: a flow sequence network for encrypted traffic classification. In: IEEE Conference on Computer Communications, IEEE INFOCOM 2019, pp. 1171–1179 (2019)
16. Lotfollahi, M., Siavoshani, M.J., Zade, R.S.H., Saberian, M.: Deep Packet: a novel approach for encrypted traffic classification using deep learning. Soft. Comput. **24**(3), 1999–2012 (2020)
17. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
18. Marín, G., Casas, P., Capdehourat, G.: RawPower: deep learning based anomaly detection from raw network traffic measurements. In: Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, pp. 75–77 (2018)

19. Marín, G., Casas, P., Capdehourat, G.: Deep in the dark - deep learning-based malware traffic detection without expert knowledge. In: 2019 IEEE Security and Privacy Workshops (SPW), pp. 36–42 (2019)
20. Novo, C., Morla, R.: Flow-based detection and proxy-based evasion of encrypted malware c2 traffic. In: Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security, AISec 2020, pp. 83–91. Association for Computing Machinery, New York (2020)
21. 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)
22. Rigaki, M., Garcia, S.: Bringing a GAN to a knife-fight: adapting malware communication to avoid detection. In: 2018 IEEE Security and Privacy Workshops (SPW), pp. 70–75 (2018)
23. Szegedy, C., Zaremba, W., Sutskever, I.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
24. Wang, W., Zhu, M., Zeng, X., Ye, X., Sheng, Y.: Malware traffic classification using convolutional neural network for representation learning. In: 2017 International Conference on Information Networking (ICOIN), pp. 712–717 (2017)
25. Wang, Z.: The applications of deep learning on traffic identification. BlackHat USA **24**(11), 1–10 (2015)