Crafting Adversarial Example to Bypass Flow-&ML- based Botnet Detector via RL

Junnan Wang
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
wangruonan@iie.ac.cn

Qixu Liu*
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
liuqixu@iie.ac.cn

Di Wu
Huawei Technologies Co., Ltd.
Shen Zhen, China
wudi94@huawei.com

Ying Dong
Beijing Venus Information Security Technology Incorporated Company
Beijing, China
dong_ying@venustech.com.cn

Xiang Cui
Cyberspace Institute of Advanced Technology, Guangzhou University
Guang Zhou, China
cuixiang@gzhu.edu.cn

ABSTRACT

Machine learning(ML)-based botnet detection methods have become mainstream in corporate practice. However, researchers have found that ML models are vulnerable to adversarial attacks, which can mislead the models by adding subtle perturbations to the sample. Due to the complexity of traffic samples and the special constraints that to keep malicious functions, no substantial research of adversarial ML has been conducted in the botnet detection field, where the evasion attacks caused by carefully crafted adversarial examples may directly make ML-based detectors unavailable and cause significant property damage. In this paper, we propose a reinforcement learning(RL) method for bypassing ML-based botnet detectors. Specifically, we train an RL agent as a functionality-preserving botnet flow modifier through a series of interactions with the detector in a black-box scenario. This enables the attacker to evade detection without modifying the botnet source code or affecting the botnet utility. Experiments on 14 botnet families prove that our method has considerable evasion performance and time performance.

CCS CONCEPTS

• **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**; • **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

Bypass Botnet Detector, Adversarial Machine Learning, Reinforcement Learning

ACM Reference Format:
Junnan Wang, Qixu Liu, Di Wu, Ying Dong, and Xiang Cui. 2021. Crafting Adversarial Example to Bypass Flow-&ML- based Botnet Detector via RL. In *24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '21), October 6–8, 2021, San Sebastian, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3471621.3471841

*Corresponding Author

RAID '21, October 6–8, 2021, San Sebastian, Spain

ACM ISBN 978-1-4503-9058-3/21/10…$15.00
https://doi.org/10.1145/3471621.3471841

1 INTRODUCTION

Machine learning(ML) has greatly promoted the development of botnet detection technology. Unlike signature-based detection methods, machine learning-based anomaly detection is able to efficiently and accurately identify malware-generated traffic when certain behavior patterns are recognized. In particular, the significant spatial-temporal similarity of botnets makes them easily detected by ML-based models. In addition, ML-based methods can identify unknown botnet families and are more suitable for dealing with large-scale network traffic data.

Unsurprisingly, attackers have started looking for methods and techniques that would allow them to overcome the progress in detection systems and bypass behavior-based detection. For example, attackers can frequently change the IP address corresponding to the domain name of the command and control (C&C) server to evade IP blacklist detection; use application layer protocols (HTTP, DNS) or external web services (Twitter, Facebook) for C&C communication to bypass the firewall; and encrypt C&C communications to avoid payload-based botnet detector. However, these evasion methods cannot bypass the widely used Flow-& ML-based botnet detectors, which distinguish malicious relying on the statistical characteristics of flows. Moreover, these methods require extensive or complicated direct source modifications, thereby placing high demands on attackers [38].

One direct way to bypass the ML-based botnet detector is using ML's vulnerability to attack ML detection model. Szegedy et al. [40] first discovered that well-performing ML models are susceptible to

adversarial attacks in the form of adding some tiny perturbations to inputs that fool a model into producing incorrect outputs. Many recent works have also proposed ingenious methods (L-BFGS [40], FGSM [18], Deepfool [28] et al.) to craft adversarial examples, but these methods are difficult to directly apply for crafting botnet adversarial samples.

In the context of botnet detection, the constraints on the perturbation that added to the adversarial example is no longer imperceptible to humans, but that it cannot affect the original malicious intention of botnet traffic. The complexity and specific format of botnet traffic data determine the need to develop new methods for constructing botnet adversarial samples to bypass ML-based botnet detectors.

In this paper, a reinforcement learning(RL) -based method to bypass an ML-based flow-level botnet detector is presented. We attempt to mislead the ML-based detector by adding some perturbations to the botnet flow. Specifically, we model the modification of botnet flow as a sequential decision problem, let the RL agent learns the optimal modification strategy during a series of interactions with the botnet detector. To ensure the preservation of functionality, we design an action space containing 14 incremental operations, each of which only adds a carefully crafted packet to the original flow in an attempt to change some of the flow-level characteristics. The detector deems these characteristics to be discriminating, but this may not be a causal indictor of benign traffic. Moreover, adding packets is an incremental operation at the transport layer, while malicious functions are generally encapsulated in the application layer. Therefore, it can be guaranteed that the original malicious intention will not be destroyed.

The advantages of our attack method include that (1) it is a black box attack, which is more in line with real attack scenarios than other methods; (2) it is general and can be used regardless of whether the detector's loss function is differentiable; (3) it is plug and play, the RL agent can exist as a proxy, the attack has a low evasion cost and is suitable for any botnet family.

Through extensive experiments, we prove that the current ML-based botnet detector is vulnerable. Attackers can avoid detection by only adding a few packets to the botnet flow at a relatively small cost and without any prior knowledge.

The contributions of this paper can be summarized as follows:

- We propose a general black box attack framework for ML-based botnet detectors. We assume that the attacker can access the detector and obtain the input binary discrimination result (malware/benign) but does not have any prior knowledge of the algorithm and feature space used by the detector. To the best of our knowledge, this work is the first study about black-box adversarial attacks in botnet evasion field.
- We design a series of universal action spaces and encapsulate them in the RL framework. On the one hand, all actions are incremental operations, thereby ensuring that the transmission of malicious information and functions is not affected. On the other hand, the actions are universal and well encapsulated, enabling attackers to escape detection without complex modifications to the botnet malware.
- We demonstrate how to train and deploy our system to avoid ML detection on a carefully constructed botnet flow dataset and comprehensively evaluate the evasion performance, time cost, and universality of the framework.

This paper is divided into 6 sections as follows: Section 2 introduces related works. Section 3 describes the system framework. Section 4 shows how we set up our experiment. Section 5 discusses the experimental results and conclusions. At the end of the paper, we summarize and prospect the work of this paper.

2 RELATED WORK

2.1 Adversarial machine learning

In the literature, there are many works focusing on adversarial attacks. Researchers have proposed various advanced attack methods based on how much detector information is available. Such knowledge of the detector can include [9]:

- the training set or part of it;
- the feature space of the ML algorithm;
- the type of ML algorithm (SVM, LR, DecisionTree, CNN, etc.)
- the trained classifier (details of the model, such as its architecture and hyperparameters);
- the feedback from the detector (score-based feedback consisting of probability- or binary-based feedback for the final label)

In this paper, we group recent adversarial attack methods into three attack scenarios based on different levels of adversary knowledge about the attacked system: perfect knowledge (PK), limited knowledge (LK) and zero knowledge (ZK).

Perfect knowledge. White box attack. In this scenario, the attacker has complete information about the detector. The attacker's goal is to minimize the sample misclassification function.

[40] leveraged L-BFGS to solve the optimization problem of finding the minimum amount of required perturbations. C&W attack [13] designed a new loss function with a small value in the anti-sample but a larger value in the clean sample so the adversarial example could be searched by minimizing the loss function. FSGM [18] assumes that the loss function in the neighborhood of the clean sample is linear.

Deepfool [28] was the first method to use the L2 norm to limit the disturbance size to obtain the minimum perturbation. Universal perturbation [27] extends DeepFool to craft image-agnostic and universal perturbations.

Limited knowledge. Gray box attack. The attacker only has limited knowledge about the detector and cannot use gradient-based approaches. However, by knowing the type or feature space of the model, attackers can mislead the detector by finding a set of features that not discriminating enough via the structure features of the model, the feature space or the feedback score sequence.

Apruzzese et al. [6] used a mixed integer linear program solver to construct an optimal adversarial example for evading flow and random forest based botnet detectors.

Papernot et al. [31] trained a substitute model after performing Jacobian_based dataset augmentation to accurately simulate the decision boundary of the detector.

Table 1: Taxonomy of the latest academic studies on adversarial attack methods.

PK/LK/ZK	Method	Object	Disadvantage
PK	L-BFGS [40]	image	Unable to handle non-differentiable loss functions
	FSGM [18] [24]	image/malware	
	SLEIPNIR [4]	Windows PE	
	DeepFool [28]	image	
	JBSM [32] [19]	image/malware	
	C&W attack [13]	image	
	ATN [7]	image	
	Adv_MalConv [22]	malware	
	Universal perturbation [27]	image	
LK-score feedback	Train substitute model [31]	image	Targeting against a particular type of ML model
LK-model type	Evade-RF [6]	botnet flow	
ZK	MalGAN [21]	malware	Actually belong to LK scenario, the attacker needs to know the complete feature space
ZK	EvadeHC [14]	pdf malware	Need help with verifying information
ZK	Boundary attack [11]	image	Need the most iterations to converge

PK: Perfect knowledge; LK: Limited knowledge; ZK: Zero knowledge.

Zero knowledge. . Black box attack. Assume the attacker has zero knowledge of the model except for the binary decision of the detector. The attacker can only evade the detector through trial and error or by crafting adversarial examples against a joint classifier. The basic idea is that if the adversarial examples can bypass each model in the collection, then it may bypass every single detector.

MalGAN [21] introduced GAN to generate PE malware to bypass a black-box detector. The GAN discriminator is a substitute detector model built by the attacker, while the generator is utilized to generate adversarial malware samples.

Boundary attack [11] starts from a large adversarial perturbation and then seeks to reduce it while remaining adversarial by performing a random walk along the boundary between the adversarial and the non-adversarial regions.

From Table 1, we can see that most of the existing white box attacks can only deal with differentiable loss functions, while gray box attacks can only bypass specific types of detectors. A few of the existing black box attack methods do not need any prior knowledge at all; instead, these methods rely somewhat on external information or model-related information. This challenge motivates us to build a general black box adversarial attack method that can bypass any ML-based botnet detector.

2.2 Botnet Evasion

While the botnet detection method is constantly improving, attackers are also exploring techniques to avoid ML-based botnet detection. Traditional botnet evasion techniques make C&C traffic difficult to detect by encrypting traffic, hiding C&C information in redundant fields of TCP/IP protocols, or using online-social-networks(OSN) to construct covert channels [30] [44] [5] [1] [2]. However, these methods require complicated modifications to the botnet architecture or source code, which places high capability requirements on botnet controllers, and also has a certain impact on the availability and market value of the botnet.

With the widespread application of machine learning in botnet detection field [23] [10] [16] [20] [34] [42] [41], security researchers have gradually sought adversarial machine learning(AML) methods to bypass botnet traffic detectors. That is, constructing botnet traffic confrontation samples by adding perturbations to the botnet traffic samples, thereby bypassing the ML-based botnet traffic detector. These methods can be achieved by applying traffic proxies instead of modifying the botnet source code, which seems to be a more attractive and lower-cost solution.

Furthermore, AML-based evasion methods can be divided into two categories according to different output.

Feature space attack. refers to methods that can only generate adversarial traffic feature vectors. However, considering that the process of mapping traffic samples to feature vectors is irreversible, such an attack cannot cause actual security threats and can only use to prove the vulnerability of the ML-based detector [25]. Evade-RF [6] attempts to make botnet traffic different from the malicious flows contained in the detector training dataset by slightly altering flow-level statistical characteristics, such as flow duration, as well as the numbers of exchanged bytes and exchanged packets. The author trained a random forest as a botnet detector on the CTU dataset and evaluated the performance of the generated adversarial samples. [29] discussed the learning and evasion consequences of the gap between the generated and crafted adversarial samples, and they achieved a white box adversarial attack against an encrypted C&C malware traffic detector. However, these works made the unrealistic assumption that the attacker has perfect knowledge of the classifier, and this is obviously inconsistent with real attack scenarios.

End-to-end attack. refers to an attack method that can generate real traffic as output. This attack method is more suitable for real network attack scenarios, because the output can be directly used by the attacker. However, some stricter constraints will be introduced when applying AML to generate malicious traffic samples:

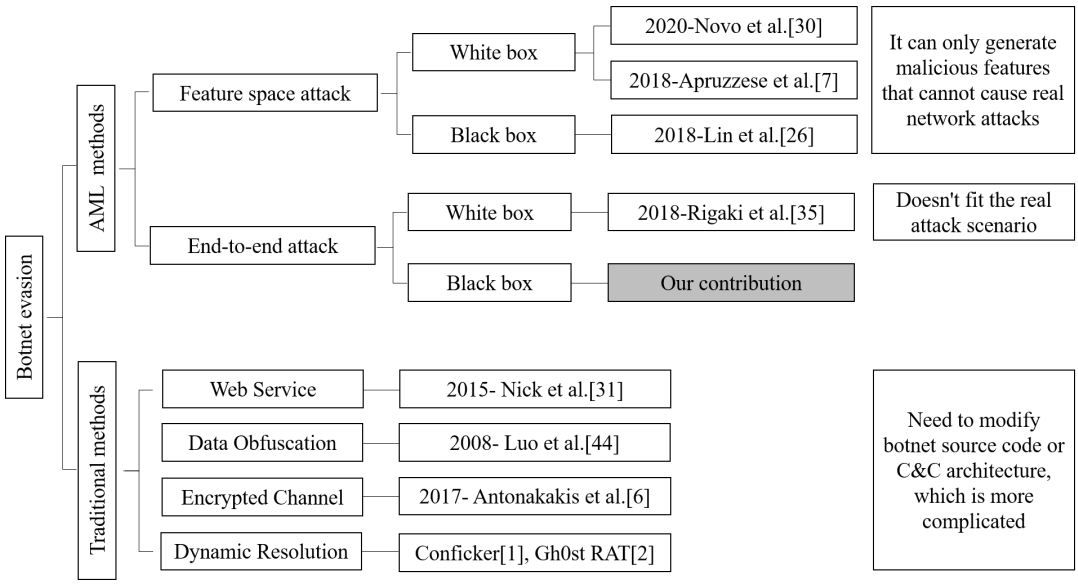

Figure 1: Taxonomy of the Botnet Evasion

(1) Keep the malicious functionality intact in the perturbed sample. (2) Cannot destroy the file structure and network protocol structure of pcap files. This leads to many AML methods in the image field that cannot be directly applied, because they perturb image pixels indiscriminately.

[35] proposed a method of using a generative adversarial network (GAN) to mimic the behavior of Facebook chat network traffic to bypass a self-adapting Stratospheric IPS. Their article attempted to adapt the behavior of its communication channel to mimic the behavior of Facebook chat network traffic according to the characteristics (total byte size, duration, and time delta between the current flow and the next flow) received from a GAN. However, the article did not clearly explain how to modify the source code to achieve the changes indicated by the GAN, and the IPS mentioned in the article is not a flow detector but rather a 3-tuple (victim IP, server IP, server port) detector.

There are also some works [36] that have use the generation ability of the GAN to generate network traffic that looks as real as possible. The purpose of these works is to improve the quality of the dataset for training malicious traffic detectors to deal with the data imbalance problem, yet it cannot be guaranteed that the generated network traffic actually occurs in the data link. Furthermore, these methods do not consider whether the crafted traffic can carry out the malicious function of the botnet. This leads to a completely different scenario from that of our work, in which we aim to bypass botnet traffic detection by constructing botnet adversarial samples.

3 THREAT MODEL AND SYSTEM FRAMEWORK

We use the Markov decision process (MDP) to model the problem of crafting adversarial botnet flow samples and implement general black box attacks via RL algorithms. This section first explains the threat model of our attack method, then introduces the overall framework of the proposed system, and finally introduces the important system components in detail.

3.1 Threat model

We describe our threat model according to the method proposed in [9].

Adversary's goal. The attacker's goal is to be able to mislead the detector by generating adversarial samples, thereby increasing its invisibility. From the perspective of the CIA triad (confidentiality, integrity and availability), attackers try to reduce the availability of network intrusion detection systems by camouflaging botnet flow.

Adversary's knowledge. The attacker understands that the target network may be protected by a flow-level network intrusion detection system based on machine learning. However, the attacker does not need to master any prior knowledge about the detector, such as the algorithm, parameters, features or training data.

Adversary's capability. In the evasion scenario based on adversarial attacks, the attacker has the ability to modify the test data but not the detector's training data. Even so, because the attacker has full control of the botmaster and partial control of the bots, we believe that the attacker can also update bots to change their communication behaviors by setting up a proxy. At the same time, we assume that the attacker can continuously access the detector to obtain the binary prediction result from the detector.

3.2 System design

In the context of evading botnet detection, we instantiate the feedback loop of RL, as shown in Figure 2. Our task is training the agent through a series of interactions with the botnet detector so that it can learn how to craft adversarial botnet flow example to bypass the botnet detector.

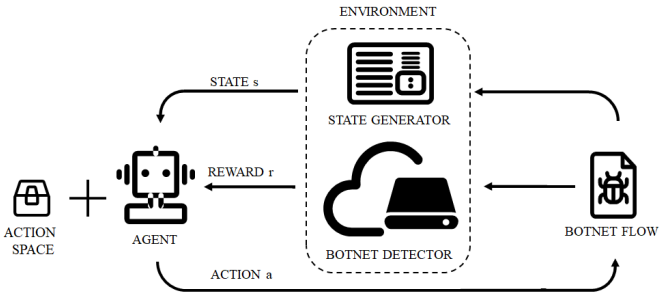

Figure 2: System framework

Generally, our system setup is composed of two components: an *agent* and an *environment*. The *agent* receives feedback from the *environment* and then selects actions from the action space to modify the botnet flow according to the RL algorithm.

The *environment* mainly includes two modules: a botnet detector and a state generator. The botnet detector is the detector we want to bypass. Its task is to continuously predict the malicious traffic in the loop and feed the binary result back to the agent as a reward. It should be noted that the reward is a real number $\in \{0, R\}$; if the detector is successfully misdirected, the reward is R; otherwise, it is 0. We set $R = 10$ to give a strong positive feedback when the agent successfully modifies the botnet traffic sample for bypassing detector. The state generator is used to generate a description of the current sample state and help the agent make action decisions. Generally, the more comprehensive and accurate the state is, the better the agent's decision will be.

When the system starts to run, the state generator generates a state s based on the input sample. Then, the *agent* perceives the state and selects an action a from the action space to modify the botnet flow sample according to the RL algorithm. Then, the modified sample is input into the botnet detector, and a prediction result is obtained. Next, *agent* receives the reward r according to the binary feedback, and the state generator generates the next state s' according to the modified sample. The loop continues until the detector is successfully misled or the number of operations reaches the upper limit (if the detector is still not successfully bypassed, we start another sample modification attempt).

3.3 Reinforcement Learning Algorithm

Reinforcement learning is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences [39]. It is employed by various software and machines for finding the best possible behavior or path they should take in a specific situation.

To solve the RL problem, the agent chooses an RL algorithm to learn to take the best action in each of the possible states it encounters. It senses a given state from the environment and maximizes its long-term reward value by learning to select an appropriate action based on the enhanced signal provided by the environment. Considering that our action space is not continuous, we choose two classic value-based RL algorithms to compare which is more suitable for proposed application scenario.

DQN. Q-learning is an off-policy temporal-difference (TD) method that uses a Q-table to estimate the Q-value for selecting the best behavior. However, when the state space or action space is large, Q-learning has the problem of "dimensionality disaster". DQN leverages a neural network to estimate the Q-value function [26] and switch to a function $Q(s, a; \theta)$ to approximate $Q(s, a)$ instead of Q_table, where θ is the network parameter. The loss function for the network $L(\theta)$ is defined as the squared error between the target Q-value and the Q-value output from the network.

$$L(\theta) = E[(TargetQ - Q(s, a; \theta))^2] \quad (1)$$

where

$$TargetQ = r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a; \theta)$$

where γ is the discount factor, s and a represent the current state and action, respectively, s_{t+1} represents the next state and r_{t+1} is the reward of action a under state s (These are the same in the following equation).

SARSA. State–action–reward–state–action (SARSA) is an on-policy algorithm for TD learning. The optimal Q-value, denoted as $Q(s_t, a_t)$, can be expressed as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (2)$$

There are many differences between Q-learning and SARSA. First, on-policy SARSA learns the Q-value based on the action performed by the current policy, while off-policy Q-learning does so relative to the greedy policy. Taking the cliff walking problem as an example, DQN tends to choose a large negative reward while exploring, while SARSA tends to take the safe path and avoid a dangerous optimal path. This means that a Q-learning agent may fall off the cliff at a safe point as a result of choosing exploration. DQN is bolder, while SARSA is more conservative.

Second, under some common conditions, they both converge to the real value function but at different rates. Q-learning tends to converge slightly slower than SARSA because of the greedy policy, but it has the capability to continue learning while changing policies. Q-learning directly converges to the optimal policy, while SARSA only learns a near-optimal policy by exploring.

For the above reasons, we implemented these two RL algorithms and evaluated their performances through a series of experiments. The results are shown in Section 5.

3.4 Action space

The action space includes a series of modification actions for botnet flow.

ML-based botnet flow-level detection is anomaly-based that identify botnet based on the differences in some features between botnet flow and normal flow. ML models perform feature extraction before classification, which will cause somewhat information compression and information loss, no matter it is done manually or automatically based on a neural network. We hope that the perturbation added to the original sample can confuse certain features of the malicious sample with the normal sample after feature extraction, which the botnet detector deems as an indicator of benign samples.

But as aforementioned, the most important thing to modify the botnet flow is to not affect its malicious functions. Therefore, we

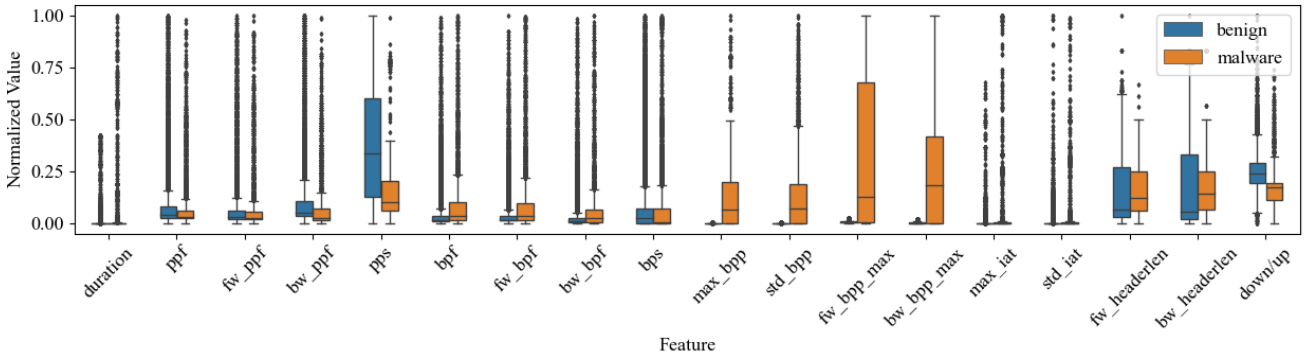

Figure 3: Boxplot of the 18 normalized flow features for botnet and normal flow

cannot delete flow packets at will. The only choice is to apply modifications to areas that do not affect the implementation of malicious functions or add new data packets. Through the analysis of botnet flow, we find that the malicious content of botnet is encapsulated in the application layer, so incremental operations at the transport layer will not affect the original malicious functions.

To determine which features should be disturbed, we refer to works on ML-based botnet detection, such as [23] [10], [16] [20]. We find that researchers tend to extract some discriminative features based on the working mechanism of the botnet, and these features often have high degrees of overlap in different jobs.

Taking the difficulty of action designing and into account, we choose 18 features from the set of features commonly used in botnet detection, including duration, packet per flow(ppf), packet per second(pps), bytes pre flow(bpf), bytes per second(bps), inter-arrival time(iat), down/up ratio and so on(fw:forward, bw:backward). These characteristics can be easily affected by carefully designing the time, size and direction of the added packet. Figure 3 shows the boxplot of the 18 normalized feature values in our training dataset for botnet and benign flows.

We can observe that due to the unique working mechanism of botnets, there are some differences between the ranges of values for some features in the botnet and normal flows. For example, botnet will send a large number of short heartbeat packets to confirm whether the connection is maintained, so it has a smaller ppf; downloading malicious applications and transmitting private information on the bot side will result in a larger bpp; bots need to send a large amount of secret information in response to short commands from botmaster, so botnet traffic tends to have a small down/up ratio.

Based on this discovery, our action space includes 14 actions, which can affect the above-mentioned statistical characteristics of transport layer by simply modifying the data packet timestamp or adding carefully constructed packets. When constructing new packets, we mainly consider three attributes: timestamp, direction, and packet size. These 14 actions are divided into 5 categories according to the objects they intend to affect, as summarized in Figure 4:

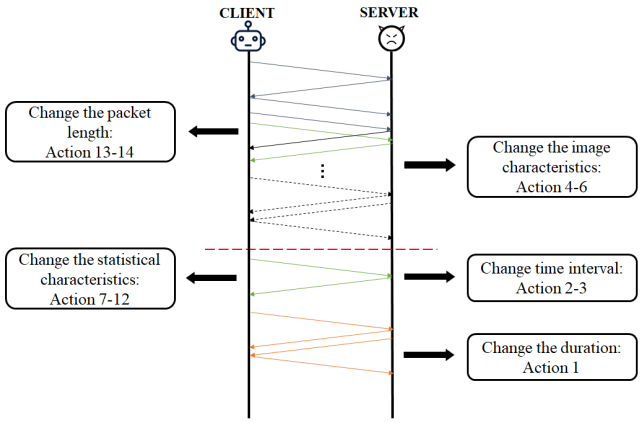

Figure 4: Action space

- Change the duration
  1) Withhold the final TCP FIN packet for randomly 1-3s
- Change the time interval
  2) Add a forward packet with an interval of 20s at the end
  3) Add a backward packet with an interval of 20s at the end
- Change the image characteristics: (for the DNN model) pay attention to the location and content of the packet
  4) Add an empty packet at random location ($0 < loc < 8$)
  5) Add a random packet at random location ($0 < loc < 8$)
  6) Add a full 0 packet at random location ($0 < loc < 8$)
- Change the statistical characteristics: (for the non-DNN model) pay attention to the direction and size of the packet
  7) Add a forward large-size 0 packet
  8) Add a backward large-size 0 packet
  9) Add a forward avg-size packet with no content at the head
  10) Add a backward avg-size packet with no content at the head
  11) Add a forward empty packet
  12) Add a backward empty packet
- Change the packet length
  13) Add a random length of 0 at the end of the packet with a probability of 0.2
  14) Select two packets without payloads, add the character '0' to avg-size

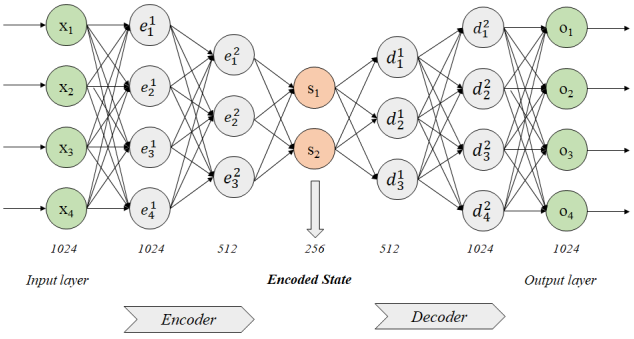

Figure 5: State Generator Details

The purpose of the "Change the duration" category is to affect the duration; "Change the time interval" is to change IAT; "Change the packet length" is used to disturb bpp; "Change the image characteristics" contains actions to change the input of a deep neural network-based botnet detector, so they focus on the location and payload of the newly inserted data packet; The actions in "Change the statistical characteristics" are to comprehensively disturb the above statistical characteristics, so the direction and size of the data packet are mainly considered.

3.5 State space

Considering that the binary feedback of the detector contains too little information for use by the agent, we need a state generator to deliver the state of the botnet flow to the agent. To describe the state of the current botnet flow samples concisely, we use a feature encoder with a deep structure — a stacked autoencoder (SAE) — to automatically extract botnet flow features and feed them back to the agent as the state.

SAE is a neural network consisting of several layers of sparse autoencoder, where the output of each hidden layer is connected to the input of the successive hidden layer. SAE was first proposed by Bengio et al. [8]. To avoid the potential vanishing gradient problem of the deep network, SAE training is performed using unsupervised pretraining and supervised fine-tuning. To some extent, the pre-trained networks facilitate iterative convergence in the supervised phase because they fit the structure of the training data, making the initial value of the entire network have a suitable state. Because each layer is based on the features extracted by the previous layer, SAE is able to extract highly abstract and complex features. SAE has achieved noteworthy performances on many feature preprocessing and dimensionality reduction tasks.

Specifically, we take the first 1024 bytes of each botnet flow file (because the first few packets, up to the first 20 packets, have been shown to be sufficient for correct accuracy, even for encrypted traffic [42]) as an input for the SAE model. After several epochs of training, the SAE model can automatically learn a 256-dimensional state vector of the botnet flow.

When determining the state dimension, we test 128 and 256 dimensions. Under the trade-off between the training time cost and evasion effects, we finally set the number of feature dimensions to 256.

4 EXPERIMENTAL SETUP

4.1 Implementation

By referring to the implementation of Tor, we deploy our system as an adversarial proxy in the network environment, as shown in Figure 6. The attacker can easily deploy an adversarial proxy on the botmaster side, while on the bot side, the attacker can achieve this by updating the bot through the original C&C channel. Therefore, our method can be implemented without complex modifications to the original malware.

Under this deployment scenario, all communication traffic between the attacker and the bot reaches the proxy first. Therefore, the attacker can monitor the botnet communication traffic, and the adversarial proxy equipped with the trained RL agent can perform incremental actions against the botnet flow until it successfully bypasses the detector. In this way, what the detector obtains is the botnet communication traffic that has been processed by the adversarial agent and is very likely to bypass the it.

In such an attack and defense architecture, the attacker can achieve adversarial attack-based botnet detector evasion in a completely black box scenario.

With an aim to engage the community, we implement our RL framework with OpenAI gym [12]. Specifically, we implement SARSA and DQN agents using keras-rl [33].

4.2 Dataset

Assessing the performance of any detection approach requires experimentation with data that are heterogeneous enough to simulate real traffic at an acceptable level. We choose two public datasets: *CTU* [17], captured by the Malware Capture Facility Project, which is a research project in charge of continuously monitoring the threat landscape for new emerging threats, retrieving malicious samples and running them in facilities to capture the traffic; and *ISOT* [37], created by merging different available datasets. It contains both malicious (traces of the Storm and Zeus botnets) and non-malicious traffic (gaming packets, HTTP traffic and P2P applications). The dataset contains trace data for a variety of network activities spanning from web and email to backup and streaming media. This variety of traffic makes it similar to real-world network traffic.

We select 10 botnet families from these public datasets to form a new dataset with the following considerations:

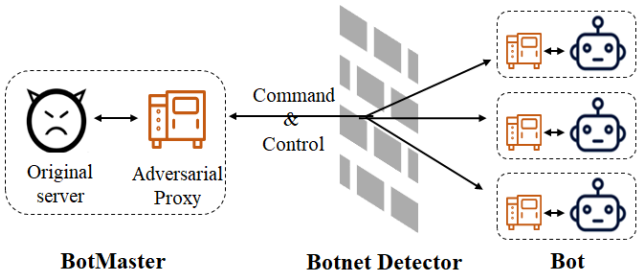

Figure 6: Implementation of our system

Table 2: Details of the dataset

Used for	Channel	Botnet Family	File Serial No.	Original sessions	Used sessions	Captured year	Notes
Train & Test	IRC	Menti	CTU-47	4,507	4,000	2011	
		Rbot	CTU-44/45/52	31,736	10,000	2011	An IRC-based botnet with 40,000 active members
		Murio	CTU-49	8,381	8,000	2011	
	HTTP	Virut	CTU-46/54	31,173	30,000	2011	Responsible for 5.5 percent of malware infections during the third quarter of 2012
		Miuref(3ve)	CTU-127	13,426	10,000	2015	It infected around 1.7 million computers
		Neris	CTU-42/43	31,133	30,000	2011	
		HTbot	CTU-348	36,855	30,000	2018	One of the largest reported Android banking botnets known to date
		Dridex/Necurs	CTU-346	10,029	10,000	2018	One of the world's largest spam botnets
		Trickbot	CTU-327	30,052	30,000	2018	The top business threat in 2018
	P2P	Storm	ISOT-Storm	4,672	4,672	2012	Infected more than 1 million systems
Train	Normal	–	ISOT-normal	511,322	80,000	2010	

- Diversity, the dataset covers the most mainstream botnet communication channel types (IRC, HTTP, P2P), and the traffic characteristics are significantly different.
- Typically, the dataset includes typical botnet families' traffic, they're either causes major attacks, has a large number of controlled hosts, or adopts advanced hiding methods.
- Large time span, for a single family, each traffic file takes a long time to capture. Overall, that dataset covers botnet traffic from 2011-2018. This makes the dataset more versatile and novel than other existing datasets.

We summarize the botnet families and their capture times, brief introductions and numbers of session samples used for the experiment in Table 2.

After obtaining the dataset, we perform data preprocessing progress, as shown in Figure 7.

Step 1: Integration & pruning. We combine the collected traffic belonging to the same botnet family. If the pcap file is too large, it is cropped. The purpose of this step is to balance the sample size of each family.

Step 2: Splitting pcap into sessions. This is done to obtain more complete communication information. A session refers to all packets consisting of bidirectional flows, that is, the source IP and destination IP are interchangeable in a 5-tuple (*SIP*, *SPort*, *DIP*, *DPort*, *Protocol*). Specifically, we use SplitCap [3] to split each pcap file into sessions.

Step 3: Anonymization & cleaning. Our traffic file is produced from different network environments. To eliminate the IP and MAC address' effects on the detector, the unique information of the traffic data needs to be randomized. Specifically, we replace them with a randomly generated new address.

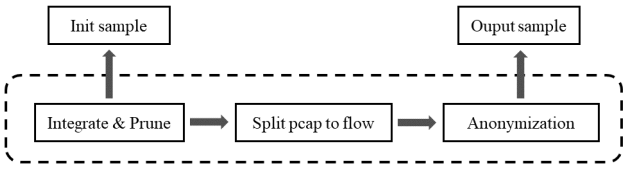

Figure 7: Data preprocessing

4.3 Detector

In our system, the function of the detector is to predict the modified botnet flow and feed the binary result back to the agent. For comparison, we choose two state-of-art detection models in our experiments: the composite DL detection model combining CNN with LSTM and the non-differentiable ML detection model based on XGBoost as our botnet detector.

BotCatcher detection model. The communication mode between the C&C server and the bot is significantly different from the communication mode between normal users, so the abnormal traffic generated by the botnet can be detected through traffic analysis. BotCatcher [43] uses a deep learning algorithm to automatically extract temporal and spatial features from network traffic and trains a softmax classifier accordingly. We show the system framework in Figure 8.

Considering that CNN has the ability to extract local features and recognize spatial similarities, RNN has the ability to process sequence data and recognize temporal similarities. In the feature learning module, BotCatcher uses CNN to extract spatial features by converting the botnet session data into a gray image and leverages LSTM to learn the temporal features of packet sequences. After processing these two kinds of features through the multilayer neural network, BotCatcher puts them into a softmax layer to identify abnormal traffic patterns. The author used the CTU dataset to verify the effectiveness of the proposed model, while we obtain a 99.6% detection rate on our dataset.

XGBoost detection model. XGBoost stands for eXtreme Gradient Boosting, which is an efficient implementation of the gradient boosting machines created by Tianqi Chen. It is designed for speed and performance, which is the reason why it has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data.

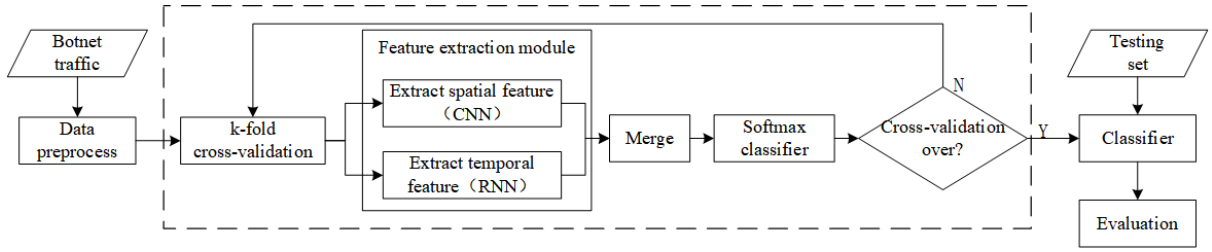

Figure 8: BotCatcher system framework

Table 3: XGBoost detector's feature set.

Feature	Importance	Feature	Importance
max_fiat	0.188713	max_forward_pkt_len	0.003005
std_fiat	0.112026	std_backward_pkt_len	0.002809
duration	0.110066	total_pkt_len	0.002221
forward_header_len	0.103142	std_pkt_len	0.001829
total_fiat	0.074335	mean_forward_pkt_len	0.001568
mean_fiat	0.071723	mean_pkt_len	0.001502
flowPktsPerSecond	0.066497	min_pkt_len	0.001372
fPktsPerSecond	0.050493	NPEx	0.000719
bPktsPerSecond	0.03671	max_backward_pkt_len	0.000523
IOPR	0.028872	mean_backward_pkt_len	0.000457
backward_header_len	0.02652	total_forward_pkt_len	0.000392
SameLenPktRatio	0.021164	flowBytesPerSecond	0.000327
max_biat	0.01633	min_forward_pkt_len	0.000261
total_biat	0.015546	std_forward_pkt_len	0.000261
total_packets	0.013848	total_backward_pkt_len	0.000196
mean_biat	0.012542	fBytesPerSecond	0.000131
std_biat	0.011431	min_backward_pkt_len	6.53E-05
max_pkt_len	0.010255	min_fiat	0
total_fpackets	0.007773	min_biat	0
total_bpackets	0.004377	bBytesPerSecond	0

Dhaliwalet et al. [15] chose XGBoost to build an IDS system. The performance of XGBoost is better than those of many other models because it can effectively deal with the problem of data surplus and can be processed in parallel. Therefore, XGBoost is very suitable for dealing with real-world networks.

In their work, the authors performed a cross-validation experiment using the NSL-KDD dataset (a csv file containing the 41-dimensional characteristics of malicious traffic), achieving a result (98.7%) better than those of other machine learning algorithms. The model structure and hyperparameter settings used in this paper are completely consistent with they used (they tuned the hyperparameter settings), and we obtain detection result (98.9%) that is as good as the original work on our dataset. We list the extracted features and their importance in Table 3.

5 RESULTS

To evaluate our system, we divide the dataset into four disjoint subsets: a detector training set, an agent training set, a detector testing set, and an agent testing set, at a ratio of 4:4:1:1. The disjointness is to test the generalization ability of our attack model, and to better simulate the real attack scenario, where the attacker may not be able to obtain the training data of the target detector.

To compare the performances of different RL algorithms and detectors, we implement the following four system instances: SARSA agent–BotCatcher detector, SARSA agent–XGBoost detector, DQN agent–BotCatcher detector, and DQN agent–XGBoost detector. Each agent is trained for $action\_num * sample\_num$ rounds.

5.1 Evasion performance

The evasion rate illustrates the probability that the botnet flow adversarial example can successfully evade the detector. Table 4 compares the testing evasion rates of four system examples after training on 10 botnet families. Among them, XGBoost-random and BotCatcher-random represent the situations in which we use a random strategy to select actions from the action space for modifying the botnet flow. On the one hand, they are the baselines for measuring whether the agent is effective. In other words, if the agent's performance is not as good as the random strategy, it means that the agent has not learned anything useful. From Figure 1, we can see that the results of the RL algorithm are better than those of the random strategy in any case. On the other hand, this can also prove that our action space is indeed effective because even random selection can bypass the detector with a certain probability.

Contrasts between families. From Table 4, we can find that the evasion rates vary among different botnet families; Storm even has an evasion rate of 0 with XGBoost-SARSA. With an unchanging action space, the influence of existing actions on different family samples vary. Through a statistical analysis of each family sample, we find that a lower evasion rates may be due to the sessions containing the largest numbers of packets (such as Storm, which is a P2P-based botnet that has a large session size), so the effect of adding packets or changing the timestamps on eigenvalues is relatively small. Alternatively, the characteristics of botnet samples are very diverse from those of benign samples, causing the agent to fail to convert it into a benign sample within a limited set of $action\_num$ steps. In practical applications, the attacker can trade-off between the evasion rate and the size of the perturbation to the traffic sample, and increase the $action\_num$ appropriately.

Contrasts between agents. By comparing the evasion rates of the system instances with the same detector but different agents in Figure 1, we find that in most cases, the SARSA agent performs better than the DQN agent. We think that this is caused by the intrinsic difference between the two RL algorithms. SARSA is an on-policy algorithm that is more cautious than Q-learning. Q-learning always thinks about maximizing Q functions, regardless of other non-maxQ results. SARSA is a conservative algorithm that cares about every step of the decision and is sensitive to errors and death. Therefore, when we aim to generate adversarial examples with

Table 4: Evasion performances of system instances.

	Menti	Rbot	Murio	virut	Miuref	Neris	HTBot	Dridex	Trickbot	Storm
XGBoost-SARSA	87%	83%	76%	86%	66%	66%	61%	41%	31%	0%
XGBoost-DQN	85%	77%	75%	85%	60%	56%	49%	41%	30%	1%
XGBoost-Random	75%	72%	68%	71%	54%	42%	45%	34%	24%	0%
BotCatcher-SARSA	21%	26%	24%	40%	42%	34%	54%	38%	59%	73%
BotCatcher-DQN	21%	22%	22%	41%	38%	28%	52%	50%	51%	64%
BotCatcher-Random	17%	19%	20%	37%	37%	27%	48%	37%	42%	59%

Table 5: Time performances of system instances.

	Menti	Rbot	Murio	Virut	Miuref	Neris	HTBot	Dridex	Trickbot	Storm
XGBoost-SARSA	1.35	3.21	1.14	2.27	1.42	2.93	2.56	1.81	2.33	-
XGBoost-DQN	2.14	5.26	1.87	3.76	2.34	4.45	4.11	3.01	4	1.61
XGBoost-Random	6.42	10.26	5.14	6.05	6.17	8.05	7.12	8.56	7.12	-
BotCatcher-SARSA	6.54	4.24	7.11	3.05	4.73	6.04	3.06	2.19	2.57	4.93
BotCatcher-DQN	8.3	6.36	7.48	3.33	5.82	4.44	3.33	2.38	3.54	4.02
BotCatcher-Random	11.43	11.91	11.14	9.03	9.18	10.11	8.05	7.13	9.06	10.16

We use the average number of queries that are necessary for the agent to craft effective adversarial samples to evaluate the system's time performance.

fewer steps and tiny perturbances, SARSA agents may be more suitable than DQN agents.

Contrasts between detectors. As shown in Table 4, the evasion rate with XGBoost as the detector is higher than that with BotCatcher in most of the botnet families' results, and we believe this is because actions have a greater impact on statistical features than image features. By analyzing the statistical features of the XGBoost model shown in Table 3 and the action space described in section 3, we can see that (i) actions in the action space all directly or indirectly affect the statistical features, whether they are designed for statistical features or image features; (ii) many actions in the action space even change the most dependent features of XGBoost (action 1-duration, action 2-foward iat, action *add*-pps, action 9&10-header_len, etc.). Therefore, we conclude that the targeted modification and high consistency between the action space and the detector's feature space are largely responsible for the vulnerability of the detector.

5.2 Time performances

A major point in black box adversarial attacks is to issue the least amount of queries to the target model: if the proposed approach requires a very high number of queries, then its feasibility in real-world context would affected. We use the average number of queries that are necessary for the agent to craft effective adversarial samples to evaluate the system's time performance. The results are shown in Table 5.

From the perspectives of the botnet families. , the time performances between different families does not change significantly, meaning that as long as our agent is well trained, it will have a better performance than that of the baseline regardless of what botnet family it deals with. This result also further shows that our system has high availability. If our system has a large gap in terms of its time performances when facing different botnet families, the attacker may carefully consider whether our system can adapt to his botnet.

From the perspective of the RL algorithms. , the agent equipped with the DQN algorithm often requires more steps to bypass the detector than the agent equipped with the SARSA algorithm. This means that the action selected by DQN according to the greedy policy may not be the optimal action in the current state, so the agent needs to perform additional actions to accumulate interferences to bypass the detector.

From the perspective of the detectors. , the number of steps required for the agent to bypass BotCatcher is higher than that of XGBoost, meaning that the agent needs more iterations to mislead BotCatcher than to mislead XGBoost. In other words, XGBoost not only has a higher evasion rate but the evasion samples also require relatively fewer steps than those of BotCatcher. Therefore, we can conclude that XGBoost is more vulnerable than BotCatcher according to the experiments of this work.

5.3 Dominant actions

Dominant mutations refer to the most frequent actions taken by the agent when successfully evading the detector. Each botnet family has its own unique features, so the flows of different families also differ in terms of certain characteristics. For example, the ppf of Trickbot is 8.66, while that of HTBot is 28.34. Therefore, we guess that the influences produced by different actions should be different for each family, so each botnet family's dominant actions should

Table 6: Dominant action of the BotCatcher-SARSA agent

Action No.	Menti	Rbot	Murio	Virut	Miuref	Neris	HTBot	Dridex	Trickbot	Geodo	Storm	Waledac
1	**0.51**	0.02	**0.49**	0.00	0.03	**0.29**	0.06	0.00	0.00	0.06	0.01	0.06
2	0.01	0.00	0.01	0.01	0.00	0.03	0.01	0.06	**0.42**	0.05	0.01	0.06
3	0.01	0.02	0.00	**0.32**	0.03	0.01	0.02	0.07	0.00	0.01	0.06	**0.28**
4	0.00	0.14	0.02	0.01	0.09	0.01	0.00	0.06	0.00	0.07	0.01	0.04
5	0.00	0.04	0.02	0.19	0.02	0.03	0.00	0.00	0.00	0.08	0.01	0.02
6	0.09	0.08	0.04	0.01	0.02	0.01	**0.31**	0.00	0.00	0.03	**0.82**	0.02
7	0.01	0.01	0.00	0.00	**0.26**	0.00	0.00	**0.21**	0.02	0.01	0.01	0.02
8	0.02	0.01	0.01	0.12	0.00	0.13	**0.34**	0.19	0.32	**0.27**	0.03	0.01
9	0.00	0.00	0.00	0.00	0.01	0.02	0.00	0.12	0.02	0.06	0.01	0.01
10	0.00	**0.55**	0.10	0.00	**0.35**	0.01	0.00	**0.21**	0.07	0.02	0.01	0.01
11	0.00	0.05	0.20	0.00	0.01	0.04	0.06	0.00	0.15	0.15	0.01	**0.24**
12	0.20	0.01	0.01	0.01	0.09	0.01	0.00	0.00	0.00	0.01	0.01	0.08
13	0.07	0.08	0.07	0.03	0.03	0.12	0.19	0.08	0.02	0.16	0.01	0.03
14	0.07	0.00	0.02	**0.29**	0.04	0.29	0.00	0.00	0.00	0.04	0.01	0.12

be different. To test this hypothesis, during the test, we record the action list taken by the agent and count the frequency of each action. Table 6 shows the result under the BotCatcher-SARSA instance, where the action number corresponds to that described in section 3.

From Table 6, we can find that there are large differences in the distributions and the dominant actions of different families, and these are often related to the characteristics and main functions of different family flows.

Taking Rbot and Menti family as examples, by statistics, we obtain that the median of the Rbot family's *duration* is 9.02 s, while that of the Menti family is 2.91 s. At the same time, we can see from Table 6 that the action *changetimestamp*'s impact on the Rbot samples (0.02) is significantly less than that on the Menti samples (0.51). That is, family flows with short durations may be affected more by the *changetimestamp* action than family flows with long durations. Therefore, we determine that the influences of actions on different families are closely related to the statistical characteristics or image characteristics of each family.

6 CONCLUSION

In this paper, we propose a general RL-based framework to craft adversarial botnet flow examples,so as to launch black box adversarial attacks against ML-based botnet flow detectors.

To ensure that the original malicious functions of the botnet flow will not be affected when modifying the botnet flow, we design an action space with 14 functionality-preserving actions. These actions can change some important transport layer characteristics but will not affect the application layer information that contains malicious functions. We select 14 botnet families to build a new botnet dataset for evaluating our method. Through experiments, we prove that ML-based botnet detectors are indeed susceptible to adversarial attacks, and our system can obtain considerable evasion rates for different botnet detection models with fewer queries.

Although we achieve remarkable performance, our methods can be improved in some ways. First, we can explore additional actions that can mislead the detector by exploring the feature set that the detector depending on to add perturbations to these features in a targeted manner. Second, the current actions have a large impact on the botnet flow, and we could reduce the perturbations through an iterative method.

We believe the framework proposed in this paper can promote the research of adversarial botnet flow examples and have a positive impact on the botnet detection field.

ACKNOWLEDGMENTS

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.

REFERENCES

[1] 2008. https://en.wikipedia.org/wiki/Conficker
[2] 2008. https://en.wikipedia.org/wiki/Gh0st_RAT
[3] 2011. SplitCap. https://www.netresec.com/?page=SplitCap.
[4] Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. 2018. Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 76–82.
[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1093–1110. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis
[6] Giovanni Apruzzese and Michele Colajanni. 2018. Evading botnet detectors based on flows and random forest with adversarial samples. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 1–8.
[7] Shumeet Baluja and Ian Fischer. 2018. Learning to Attack: Adversarial Transformation Networks.. In *AAAI*, Vol. 1. 3.
[8] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. In *Advances in neural information processing*

systems. 153–160.
[9] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 387–402.
[10] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*. 129–138.
[11] Wieland Brendel, Jonas Rauber, and Matthias Bethge. 2017. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248* (2017).
[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).
[13] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. IEEE, 39–57.
[14] Hung Dang, Yue Huang, and Ee-Chien Chang. 2017. Evading classifiers by morphing in the dark. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 119–133.
[15] Sukhpreet Singh Dhaliwal, Abdullah-Al Nahid, and Robert Abbas. 2018. Effective intrusion detection system using XGBoost. *Information* 9, 7 (2018), 149.
[16] Jérôme François, Shaonan Wang, Thomas Engel, et al. 2011. BotTrack: tracking botnets using NetFlow and PageRank. In *International Conference on Research in Networking*. Springer, 1–14.
[17] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. 2014. An empirical comparison of botnet detection methods. *computers & security* 45 (2014), 100–123.
[18] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).
[19] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*. Springer, 62–79.
[20] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. 2008. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. (2008).
[21] Weiwei Hu and Ying Tan. 2017. Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983* (2017).
[22] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 533–537.
[23] Satoshi Kondo and Naoshi Sato. 2007. Botnet traffic detection techniques by C&C session classification using SVM. In *International Workshop on Security*. Springer, 91–104.
[24] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236* (2016).
[25] Zilong Lin, Yong Shi, and Zhi Xue. 2019. IDSGAN: Generative Adversarial Networks for Attack Generation against Intrusion Detection. arXiv:1809.02077 [cs.CR]
[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
[27] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1765–1773.
[28] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2574–2582.
[29] Carlos Novo and Ricardo Morla. 2020. Flow-Based Detection and Proxy-Based Evasion of Encrypted Malware C2 Traffic. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security* (Virtual Event, USA) *(AISec'20)*. Association for Computing Machinery, New York, NY, USA, 83–91. https://doi.org/10.1145/3411508.3421379
[30] Nick Pantic and Mohammad I. Husain. 2015. Covert Botnet Command and Control Using Twitter *(ACSAC 2015)*. Association for Computing Machinery, New York, NY, USA, 10 pages. https://doi.org/10.1145/2818000.2818047
[31] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 506–519.
[32] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387.
[33] Matthias Plappert. 2016. keras-rl. https://github.com/keras-rl/keras-rl.
[34] Shahbaz Rezaei and Xin Liu. 2019. Deep learning for encrypted traffic classification: An overview. *IEEE communications magazine* 57, 5 (2019), 76–81.
[35] M. Rigaki and S. Garcia. 2018. Bringing a GAN to a Knife-Fight: Adapting Malware Communication to Avoid Detection. In *2018 IEEE Security and Privacy Workshops (SPW)*. 70–75. https://doi.org/10.1109/SPW.2018.00019
[36] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. 2018. Flow-based Network Traffic Generation using Generative Adversarial Networks. *Computers & Security* 82 (12 2018). https://doi.org/10.1016/j.cose.2018.12.012
[37] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix, and Payman Hakimian. 2011. Detecting P2P botnets through network behavior analysis and machine learning. In *2011 Ninth annual international conference on privacy, security and trust*. IEEE, 174–180.
[38] Elizabeth Stinson and John C Mitchell. 2008. Towards Systematic Evaluation of the Evadability of Bot/Botnet Detection Methods. *WOOT* 8 (2008), 1–9.
[39] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[40] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).
[41] Pablo Torres, Carlos Catania, Sebastian Garcia, and Carlos Garcia Garino. 2016. An analysis of recurrent neural networks for botnet detection behavior. In *2016 IEEE biennial congress of Argentina (ARGENCON)*. IEEE, 1–6.
[42] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. 2017. Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*. IEEE, 712–717.
[43] Di Wu, Binxing Fang, Xiang Cui, and Qixu Liu. 2018. BotCatcher:Botnet detection system based on deep learning. *Infocomm-journal* 39, 8 (2018), 18–28.
[44] Xiapu Luo, E. W. W. Chan, and R. K. C. Chang. 2008. TCP covert timing channels: Design and detection. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. 420–429. https://doi.org/10.1109/DSN.2008.4630112