Evading Machine Learning Botnet Detection Models via Deep Reinforcement Learning

Di Wu[1,2], Binxing Fang[3,4,5], Junnan Wang[1,2], Qixu Liu[1,2], Xiang Cui[3,1]
1 (Institute of Information Engineering, Chinese Academy of Sciences)
2 (School of Cyber Security, University of Chinese Academy of Sciences)
3 (Cyberspace Institute of Advanced Technology, Guangzhou University)
4 (Institute of Electronic and Information Engineering of UESTC in Guangdong)
5 (School of Cyberspace Security, Beijing University of Posts and Telecommunications)
Email: wudi6@iie.ac.cn, fangbx@bupt.edu.cn, wangruonan@iie.ac.cn, liuqixu@iie.ac.cn, cuixiang@iie.ac.cn

***Abstract*—Botnets are one of predominant threats to Internet security. To date, machine learning technology has wide application in botnet detection because that it is able to summarize the features of existing attacks and generalize to never-before-seen botnet families. However, recent works in adversarial machine learning have shown that attackers are able to bypass the detection model by constructing specific samples, which due to many algorithms are vulnerable to almost imperceptible perturbations of their inputs. According to the degree of adversaries' knowledge about the model, adversarial attacks can be classified into several groups, such as gradient- and score-based attacks. In this paper, we propose a more general framework based on deep reinforcement learning (DRL), which effectively generates adversarial traffic flows to deceive the detection model by automatically adding perturbations to samples. Throughout the process, the target detector will be regarded as a black box and more close to realistic attack circumstance. A reinforcement learning agent is equipped for updating the adversarial samples by combining the feedback from the target model (i.e. benign or malicious) and the sequence of actions, which is able to change the temporal and spatial features of the traffic flows while maintaining the original functionality and executability. The experiment results show that the evasion rates of adversarial botnet flows are significantly improved. Furthermore, with the perspective of defense, this research can help the detection model spot its defect and thus enhance the robustness.**

Keywords—botnet, adversarial, reinforcement learning

I. INTRODUCTION

Botnets represent a persistent threat to Internet security, and a common phenomenon is that bots always work in a coordinated way, which will lead to time-space similarities in communication contents and patterns. Based on this assumption, machine learning is widely used in botnet detection, especially in anomaly detection. Researchers utilize classification algorithms (e.g. SVM [1] and Random Forest [2]), clustering algorithms (e.g. DBSCAN [3] and X-means [4]), and deep learning (e.g. CNN [5] and LSTM [6]) to automatically mine the complex relationships between malicious and benign network flows and distinguish botnets. For the defenders' perspective, these models are effective since they achieve extremely high accuracy on test datasets.

However, For a well-established detection system, it is not only necessary to have an effective strategy and good performance, but also to anticipate the countermeasures from opponents. Attackers can construct targeted data to break many of the assumptions that practitioners make and the integrity of models. Generally the anti-detecting technologies based on machine learning include two types: contaminating the training set and building adversary samples. In this paper, we focus on the latter, for example, attackers may attempt to generate new malware deliberately designed to evade existing classifiers. Evasion attacks against machine learning have been discussed in many security fields, such as spam [7], malware [8], image classification [9] and voice recognition [10]. Motivated adversaries discover and exploit a set of features, utilizing them to modify the stationary data distribution and stochastic properties of the samples.

For bypassing botnet detection systems, some researchers try to create time-space perturbation by injecting randomized traffic into the botnet. Cui [11] et al. point out that the botmaster can randomize its C&C communication contents to eliminate space similarity (e.g. infecting packet and flow-level noise) and add a random delay to eliminate time similarity when responding to some interactive commands. Zhang [12] et al. present that it may be effective to avoid server clustering detection by letting bots randomly visit many specific benign domains with the same URI file. However, the studies have their own limitations. On the one hand, these methods are all made for particular targets, which means the premise is that the adversary should have some knowledge about the model structure, especially the feature space. But in a real attack scenario, the adversary is typically unaware of such details. On the other hand, adding perturbation manually is quite inefficiency and has a high requirement to the experience of the designer.

This work is supported by the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDC02000000), the Youth Innovation Promotion Association CAS and the Foundation of Key Laboratory of Network Assessment Technology, Chinese Academy of Sciences (CXJJ-17S049), the Key Laboratory of Network Assessment Technology at Chinese Academy of Sciences and Beijing Key Laboratory of Network security and Protection Technology.

Qixu Liu and Xiang Cui are the corresponding authors.

978-1-5386-8088-9/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: California Polytechnic State University San Luis Obispo. Downloaded on January 23,2023 at 02:55:46 UTC from IEEE Xplore. Restrictions apply.

Moreover, for malicious network traffic, the adversarial samples have to remain the same functionality and usability as original samples. Unlike images, each byte in a flow has its own specific meanings, which means that adding perturbation in some sensitive fields may cause the flow unavailable, such as validation errors and illegal field. Therefore, the conditions of generating adversarial botnet flows are much stricter.

Due to these issues, we present a novel automatic evasion research in the field of botnet flows. The conditions of attackers and defenders in the scenario include:

- The attacker have no knowledge of the structure and feature space of the botnet detection classifier, which means the target model is a black box to the attacker.
- The attacker can only get the feedback about the malicious/benign label given by the target model for an arbitrary input, and no score needed.
- The defender will not flag the evasive variants and use them to retraining the detection system.

Above all, in this work, we make three contributions:

- We evaluate the mainstream existing adversarial attack approaches, and emphasise the decision-based attack (only rely on the final decision of the model) as an important category that are highly relevant for real-world applications and important to assess the robustness of detection models.
- We propose an novel adversarial botnet flows generator framework based on deep reinforcement learning. Through the RL algorithm and the Markov decision processes (MDPs), the equipped agent is able to add perturbations to the flows and change their spatial and temporal properties, thereby trying to deceive the target detector.
- We demonstrate evading two kinds of machine learning models, one is a decision tree classifier based on the manually extracted feature set, and the other is a classification model based on convolutional neural network (CNN) which automatically learns features from network flows. Besides, we use Snort [13], an open-source intrusion detection tool, to verify that the flows keep their original functions.

II. Related Work

There are a number of studies in the literature on attacking machine learning models in information security. The effects of adversarial samples are determined primarily by the dimension of an individual input feature and the structure of the model. For a labeled input sample x, the corresponding adversarial sample can be represented as $x^* = x + \eta$, η is the designed perturbation. The perturbation η should select properly, too small (smaller than the precision of the features) to make it not rational for the classifier to respond differently to the adversarial input, and too big to make the flow break its normal format and the executability. Based on the adversary's knowledge about the target model, the adversarial attacks can be roughly divided into three categories: *gradient-based*, *score-based* and *decision-based* attacks. Gradient-based and score-based attacks are often denoted as white-box and oracle attacks respectively.

1) Gradient-based attack: The model must be fully differentiable and the structure and weights must be known by the attacker. Gradient information of the loss function is a extremely useful resource for the attacker, which can be mainly used in two ways. The first approach is to perturb the sample x in the direction that would most increase the loss function $J(g_\theta(x^*), y)$,

$$x^* = \arg\max_{x^*:\|x^*-x\|_P \le \varepsilon} J(g_\theta(x^*), y) \quad (1)$$

Here, $g_\theta(x)$ is the score of the input sample based on the mapping relationship $g: R^x \times \theta \to R^Y$, which θ is the model parameter. Besides, ε is the maximum permissible range of the added perturbation. Grosse et al. [14] propose to use the gradient-based approach to generate adversarial Android malware examples, and use them to fool a neural network detection model. For different sizes of neural networks, the misclassification rates after adversarial crafting range from 40% to 84%.

The second class of gradient-based attacks connects the model under attack to a generator model in a generative adversarial network (GAN) [15]. During training of the GAN, the generator is encouraged to generate samples that are increasingly more difficult to detect. Anderson et al. [16] leverage the concept of GAN to construct a domain generation algorithm (DGA) that is designed to bypass a deep learning detector. Results show that domains generated from a GAN to bypass the target detector also bypass a random forest classifier that leverages hand-crafted features.

A simple way to defend against gradient-based attacks is to mask the gradients [17], for example by adding non-differentiable elements either through implicitly means like saturated non-linearities, or explicitly means like non-differentiable classifiers.

2) Score-based attack: More generally, attackers have no knowledge about the classification algorithms and the feature information of the model, but can repeatedly probe the model and get predicted scores (e.g. class probabilities and logits). Through the score for each query, the attacker is able to directly measure the efficacy of any perturbation to the target black-box model. Xu et al. [18] propose a general method to automatically find variants that can evade structural feature-based PDF malware classifiers which only rely on the classification score feedback. An oracle is used to determine if a generated variant preserves maliciousness.

To resist this kind of attack, the defenders should severely impede the numerical gradient estimate by adding stochastic elements like dropout into the model.

3) Decision-based attack: Finally, the most generic attack scenario is that the detection model reports only the decision (i.e. malicious or benign) for an input. Hoerver,

Authorized licensed use limited to: California Polytechnic State University San Luis Obispo. Downloaded on January 23,2023 at 02:55:46 UTC from IEEE Xplore. Restrictions apply.

most of the exsiting well performing studies in this field need different levels information about the model. Hu et al. [8] propose a framework named MalGAN to generate adversarial malware samples, which are able to bypass black-box models. For mainstream classification algorithms, the true positive rate of MalGAN can reach nearly zero. But a limitation of this research is that the attacker must know the complete feature space of the model.

Another relevant form of decision-based attacks is the variant of transfer attack in which the training set is used to train a fully observable substitute model. Papernot et al. [19] use inputs synthetically generated by adversaries to bypass a DNN model. Results show that the model misclassifies 84.24% of the adversarial examples crafted with the substitute. However, the requirement of this approach is the information about the training data.

To the best of our knowledge, at present there is no mature research on evading the black-box botnet detection model. In this work, we present an adversarial framework against the most harsh attack conditions. The limitations of information which available to the attackers are:

- Attackers have no knowledge of the structure and feature space of the target model.
- The feedback of the target model for an query is strictly Boolean (malicious or benign).
- There is no external party to ensure the validity of adversarial samples, such as an oracle.

III. Framework Design

The designed black-box attack in this paper is based on deep reinforcement learning, which generates adversarial samples by MDPs.

A. Deep Reinforcement Learning

The goal of reinforcement learning is to learn good policies for sequential decision problems, by optimizing a cumulative future reward signal [20]. A MDP is defined as a tuple (S, A, P, R, γ) where: S is the state space of the process; A is a finite set of actions; P is a Markovian transition model, where $P(s, a, s')$ is the probability of making a transition to state s' when taking action a in state s; R is a reward (or cost) function, such that $R(s, a)$ is the expected reward for taking action a in state s; $\gamma \in [0, 1)$ is the discount factor for future rewards. A reinforcement learning model consists of an agent and an environment. For each turn, the environment receives the action a chosen by the agent, and feeds back the observed state s' (after executing a) and reward r. The agent follows a policy $\pi(a|s)$ based on the estimated value determined by the s and r. The process stops when a target state is reached through a series of exploration and exploitation.

Some early actions/states produce no immediate reward but are important to the final outcome. For a given policy, we can further define the Q-value function, which represents the expected future discounted reward. An optimal policy can be built from the optimal Q-function by choosing, for a given state, the action with highest Q-value (i.e. Q-learning). However, because the space in the Arcade Learning Environment (ALE) is too large to tractably store a tabular representation of the Q-function, the Deep Q-Network (DQN) which uses a deep function (e.g. CNN) approximator to represent the state-action value function was proposed [21]. The key contribution of the deep reinforcement learning framework is its ability, as in deep learning, for the agent to learn a value function in an end-to-end way.

B. Framework Structure

In the context of botnet traffic evasion, we apply DQN in a reinforcement learning framework, as shown in Figure 1. The environment is composed of the botnet sample variation in each iteration and the target botnet detection model. The agent gets a reward (benign or botnet) given by the target model and an estimate of the environment state which represented by a feature vector of the sample. Through such information, the Q-function and action policy determine which action to select next. The actions in action space A can modify the flow file but cannot break (a) the integrity and availability, and (b) the original function of the flow sample. Moreover, in the environment, there is a feature extraction module used to extract features from the modified sample according to input requirements of the detection model and a state generation module used to create the state feature vector of the sample.

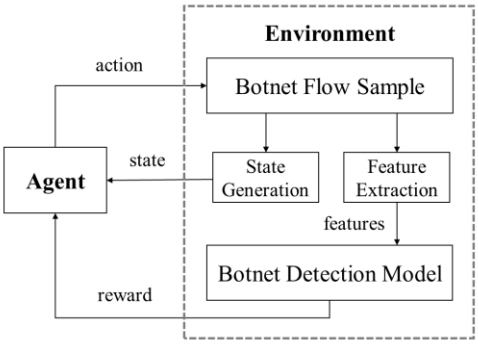

Fig. 1. Structure of the framework

To construct the above framework, we use OpenAI gym [22] as the environment interface, which is a toolkit for developing and comparing reinforcement learning algorithms. The gym framework provides a standardized environment to produce benchmarks and trains the RL agent through some methods: *reset*, *step*, and *render*. For the agent, we release a DQN agent by extending keras-rl [23], which implements several algorithms and works with OpenAI gym out of the box. In practice one botnet flow will be trained as a "game", and each step provides following feedback to the agent:

- *Reward*: a reward value $\in \{0, R\}$, where 0 denotes that the botnet flow was detected by the detection model and R is the reward for evading the model. In our experiments, we use R=10.
- *Observation*: a feature vector summarizing the status of the current flow sample.

Authorized licensed use limited to: California Polytechnic State University San Luis Obispo. Downloaded on January 23,2023 at 02:55:46 UTC from IEEE Xplore. Restrictions apply.

- *Done*: a Boolean variable that represent whether the environment needs to be reset, i.e., whether the current step is successful.

C. State Space

The botnet flow sample exists as raw bytes in the environment. In order to represent the state of sample more conveniently and concisely, it is necessary to express the state in the form of feature vector. Due to the relatively long length and complexity of each flow, we utilize an auto-encoders (AE) to map flows to short binary codes. An auto-encoder is a type of feedforward neural network used to learn efficient data codings in an unsupervised manner, which is primarily used for dimensionality reduction and feature extraction.

Before encoding, each flow is trimmed to 1024 bytes, thus, converting flows to grey images with size of 32*32. If it is not long enough, the 0x00 will be added in the end. Generally, the previous bytes in a flow will contain main connection information and a handful of content exchange data, so they can better reflect the characteristics of the flow. Then we use convolutional auto-encoder, an auto-encoder uses CNN as the encoder and decoder, to transform the "image" in to a 32 dimensional feature vector. The resulting vector represents a fairly holistic view of the botnet flow sample, so we use it as the state in the MDP.

D. Action Space

The agent takes an available modification to alter the flow sample through the policy in each iteration. There are a number of actions that can be made to a flow, which will not change the original function and not break the flow file format. These actions are able to change the communication pattern in temporal and spatial dimension. Some of the actions selected in this paper include:

- Modifying the timestamp of the first packet. The timestamp (the first 4 bytes in a packet header) increases or decreases for 0.1 to 0.3ms randomly.
- Adding the length of the packet payload. The modified packet is randomly added 1 to 5 0x00 bytes after the payload. The total amount of modified packets in a flow will be less than 30%.
- Appending a new 4 bytes packet composed of 0x00;
- Appending a new 4 bytes packet composed of a sequence of random characters;
- Appending a benign packet that extracted from a normal flow.

The common botnet detection models generally equate pattern recognition systems, and make judgments based on the specific feature or rule set. The above actions can create perturbations to the sample and most of them are stochastic in practice. For example, when appending a packet, the new packet is automatically generated or choose from a list of packets found in benign flows. The relevant fields in headers that are associated with data length should be specified to guarantee the flow received successfully by the target. For example, modifying the *Caplen* and *Len* fields in the packet header and the checksum of the TCP header, which corresponding to the packet data

Moreover, to bypass more strict detection models, there are some other fields that may be useful. For example, we can set different action rules for incoming and outgoing traffic. Besides, some of the stochastic actions can be replaced by specific modifications (e.g. using meaningful bytes instead of 0x00 when adding the payload length).

IV. Experiment

A. Dataset

The experiment dataset is composed of botnet flows and benign flows, which served as the trainset to train the target model and the benchmarks used by the agent. Particularly, botnet samples are extracted from the Malware Capture Facility Project [24] established by Czech Technical University. We choose several representative botnet flows as the trainset from it, and clip the larger size pcap files (more than 1 GB), as shown in TABLE I. The benign samples are selected from IOST 2010 dataset [25], which contains multiple types benign flows, such as HTTP, P2P download and games activity.

TABLE I. Dataset

Botnet	CTU Sequence Number	Operation
Nerris	42	Original (56 MB)
Rbot	44	Original (123 MB)
Virut	46	Original (30 MB)
Zeus	78-2	Clipped (309 MB)
Conficker	90	Original (13 MB)
Geodo	125-1	Clipped (327 MB)
Miuref	127-1	Original (16 MB)
Bunitu	141-1	Original (323 MB)

The raw traffic files are in pcap format and composed of multiple packets. However, the input datasets for target model and the RL agent are flows. Therefore, it is necessary to preprocess the data before training. The *pkt2flow* tool is used to turn pcap files into flows arranged in a time sequence, and the packets in each flow have the same 5-tuple (i.e. the source IP & port, the destination IP & port, the protocol).

B. Target Model

For systematically verifying the effect of the proposed framework, in our experiments, we attack two kinds of botnet detection models: (a) a decision tree model based on manually selected features (the 12 types of features are shown in TABLE Ⅱ) and (b) a deep learning model based on CNN [5]. Instead of the normal classification machine learning models, this CNN model turn the raw flows into images and directly learning features automatically from them, whose architecture is similar to LeNet-5 [26]. The details about the CNN model is refer to [5].

The target models are trained on 100000 botnet and benign flows, whose accuracy achieve 0.99. Through the two models, the flow are turned into a 12 dimensional vector or

Authorized licensed use limited to: California Polytechnic State University San Luis Obispo. Downloaded on January 23,2023 at 02:55:46 UTC from IEEE Xplore. Restrictions apply.

10 dimensional vector, respectively. However, if we want to produce more generous results than can be expected in practice, we can train the agent on the features used to represent the environment state. Generally, if the feature set used by the model under attacks has much overlap with the feature set used by the agent to represent the state, the effect of the framework will be better.

TABLE II. FEATURE SET

Feature	Operation	Type
srcIP	The source IP address of the flow	string
srcPort	The source port of the flow	string
dstIP	The destination IP address of the flow	string
dstPort	The destination port of the flow	string
bpf	The number of bytes per flow	int
ppf	The number of packets per flow	int
bpp	The average number of bytes per packet	int
bof	The number of bytes of the first packet	int
duration	The flow duration	float
bps	The average number of bytes per second	int
pps	The average number of packets per second	int
proto	The protocol of the flow	int

C. Relevant Parameters

In the exploration/exploitation strategy, the exploitation process makes the best decision given by current information and the exploration process gathers more information. Therefore, the trade-off between exploring the environment and exploiting that which the agent has already learned need to be noticed, for example the small rewards can cause the agent to become myopic. In our experiments, we adopt Boltzmann exploration to select next action and allow the agent to perform up to ten actions before declaring failure. The relatively few number of the action sequences in motivated by several considerations. On the one hand, the perturbations added to samples should be relatively small, thus ensure that no new characteristic forms in the process. On the other hand, long sequences moves that finally produce a reward may lead the *credit assignment problem*.

We set the total round of agent training session to 40000 (i.e. 40000 seed samples), which means that the maximum number of actions for training a model is 400000. Each round begins with a known botnet flow, which modified through a series of mutations. Rounds stop early if the current sample bypass the target model in less than ten allotted mutations. Finally, we use Snort as an aided verification tool to check the functionality of the evasions.

D. Result

We train an agent on 40000 flows (each type of botnet contribute 5000 flows), and there is no overlap between these flows and the dataset used to train the detection models. The flow used to train the agent is randomly selected from the botnet dataset in each round. For each category of botnet, whose flow used as the seed, we save the number of evasive variants discovered during training and the average actions operated. TABLE III shows the percentage of evasive variants in each category. Figure 1 shows that for the two kinds of target models, the average mutations added to a successful adversarial sample.

TABLE III. THE PERCENTAGE OF EVASIVE VARIANTS

Botnets	DT Model evasions	CNN Model evasions
Nerris	215 (4.3%)	1959 (39.2%)
Rbot	298 (6.0%)	2294 (45.9%)
Virut	129 (2.6%)	1827 (36.5%)
Zeus	336 (6.7%)	2373 (47.5%)
Conficker	359 (7.2%)	2522 (50.4%)
Geodo	194 (3.9%)	1990 (39.8%)
Miuref	166 (3.3%)	1652 (33.0%)
Bunitu	103 (2.1%)	2034 (40.7%)

As shown in the table above, for the CNN model, the evasion rates of adversarial samples is about 35% ~ 50%. But the average evasion rate for the decision tree model is less than 10%. This probably because that the 4-tuple feature set (i.e. *srcIP*, *srcPort*, *dstIP*, *dstPort*). Although during the training process, the temporal and spatial characteristics of the flow have changed, the 4-tuple features are exactly the same as the seed due to the requirement that maintaining the original function. However, the perturbations added to the sample can reflect on the changes of pixels when preprocess the flow into an image for the CNN model, which will be more obvious in the detection procedure. The evasion rates have a direct correlation with the length and complexity of the sample. For example, if the flow is relatively large, the new added packet may be appended after the 1024 bytes which trimmed by the CNN model, thus cannot generating the immediate successful feedback.

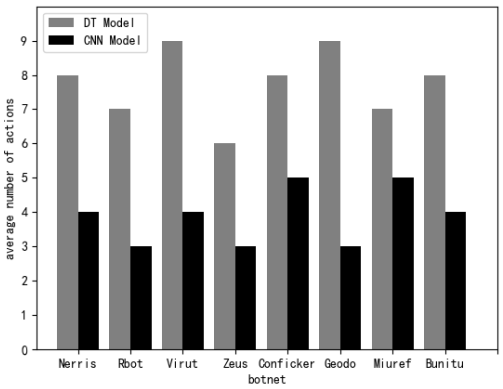

Fig. 1. The average number of actions operated

As shown in Figure 1, on average, the actions token by the agent for the decision tree model are 6 ~ 9, far more than the CNN model environment. In the process of training, the changes of the sample will gradually increase and accumulate automatically with the actions. This cumulative process can be represented more quickly in the form of local variations in images. While due to the 4-tuple features, the agent targeted to the decision tree model requires more actions to evade the detection.

In order to determine which mutation is most dominant, we assign different value to each kind of action (e.g. “0” for the timestamp modification, “1” for adding the length of the packet payload, and “2” for adding a new packet composed of 0x00). When the adversarial variant bypass the model, we record the median number of all the actions which have been carried out so far (the correlated function is integrated with

Authorized licensed use limited to: California Polytechnic State University San Luis Obispo. Downloaded on January 23,2023 at 02:55:46 UTC from IEEE Xplore. Restrictions apply.

keras-rl). The median number can directly reflect the action weight, in other words, which action get the most operations while training. TABLE IV shows the results of several botnet datasets. As we can see, the agent generally founds the models struggle with the action *new_packet*, which illustrates the significance of spatial features.

TABLE IV. THE DOMINANT MUTATON

Botnets	DT Model	CNN Model
Nerris	new_packet_0x00 (1.5)	new_packet_ran_char (2.5)
Rbot	new_packet_0x00 (2.0)	new_packet_ran_char (2.5)
Zeus	new_packet_0x00 (2.0)	new_packet_0x00 (2.0)
Geodo	new_packet_ran_char (3.0)	new_packet_ran_char (2.5)

Moreover, taking decision tree model as an example, we make a comparison experiment to test the effect of the framework when the target model do not extract the 4-tuple features. In this case, the detection model will focus on the communication patterns of botnet flows. The results is shown in TABLE V, and we can see that the evasion rate of such pattern-based model is significantly increased.

TABLE V. THE EFFECT FORTHE PATTERN-BASED MODEL

Botnets	Evasions	Average number of actions
Nerris	3770 (75.4 %)	6
Rbot	3954 (79.1%)	5
Zeus	4021 (80.4 %)	5
Geodo	3465 (69.3%)	4

V. CONCLUSION AND DISCUSSION

In this paper, we demonstrate a generic black-box attack based on deep reinforcement learning against botnet detection machine learning models, which represents a new direction in automatic evasion research.

The performance of the framework we proposed might be able to improve in some ways. First, the more information about the model we know, the better. For example, if making the state emitted by the environment in the form of feature vector extracted by the model, the agent will learn better and faster, which could be seen from the experiment results for the CNN model. Second, due to the generalization capacity and the delayed reward problem of Q-learning, it is may be better to use other RL algorithms, such as the policy-based methods. Finally, we can attempt to promote the action space, for example modifying the server IP address of the command sent to the bots.

REFERENCES

[1] Song Jinwei, Yang Jin, Li Tao. Research on Domain Flux Botnet Domain Name Detection Method Based on Weighted Support Vector Machine. Netinfo Security, 2018, 18(12): 66-71.

[2] Bilge L, Balzarotti D, Robertson W, et al. Disclosure: detecting botnet command and control servers through large-scale netflow analysis[C]//Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 2012: 129-138.

[3] François J, Wang S, Engel T. BotTrack: tracking botnets using NetFlow and PageRank[C]//International Conference on Research in Networking. Springer, Berlin, Heidelberg, 2011: 1-14.

[4] Gu G, Perdisci R, Zhang J, et al. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection[C]//USENIX security symposium. 2008, 5(2): 139-154.

[5] Wang W, Zhu M, Zeng X, et al. Malware traffic classification using convolutional neural network for representation learning[C]// Information Networking (ICOIN), 2017 International Conference on. IEEE, 2017: 712-717.

[6] Torres P, Catania C, Garcia S, et al. An analysis of recurrent neural networks for botnet detection behavior[C]//Biennial Congress of Argentina (ARGENCON), 2016 IEEE. IEEE, 2016: 1-6.

[7] Huang L, Joseph A D, Nelson B, et al. Adversarial machine learning[C]//Proceedings of the 4th ACM workshop on Security and artificial intelligence. ACM, 2011: 43-58.

[8] Hu W, Tan Y. Generating adversarial malware examples for black-box attacks based on GAN[J]. arXiv preprint arXiv:1702.05983, 2017.

[9] Szegedy C, Zaremba W, Sutskever I, et al. Intriguing properties of neural networks[J]. arXiv preprint arXiv:1312.6199, 2013.

[10] Carlini N, Mishra P, Vaidya T, et al. Hidden Voice Commands[C]//USENIX Security Symposium. 2016: 513-530.\

[11] Xiang C, Binxing F, Lihua Y, et al. Andbot: towards advanced mobile botnets[C]//Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats. USENIX Association, 2011: 11-11.

[12] Zhang J, Saha S, Gu G, et al. Systematic mining of associated server herds for malware campaign discovery[C]//Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on. IEEE, 2015: 630-641.

[13] Liu Chaoling, Zhang Yan, Yang Huiran, et al. Design and Implementation of a DPDK-based Virtual NIPS. Netinfo Security, 2018, 18(5): 41-51.

[14] Grosse K, Papernot N, Manoharan P, et al. Adversarial perturbations against deep neural networks for malware classification[J]. arXiv preprint arXiv:1606.04435, 2016.

[15] Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets[C]//Advances in neural information processing systems. 2014: 2672-2680.

[16] Anderson H S, Woodbridge J, Filar B. DeepDGA: Adversarially-tuned domain generation and detection[C]//Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security. ACM, 2016: 13-21.

[17] Brendel W, Rauber J, Bethge M. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models[J]. arXiv preprint arXiv:1712.04248, 2017.

[18] Xu, Weilin, Yanjun Qi, and David Evans. "Automatically evading classifiers." Proceedings of the 2016 Network and Distributed Systems Symposium. 2016.

[19] Papernot, Nicolas, et al. "Practical black-box attacks against machine learning." Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. ACM, 2017.

[20] Sutton, Richard S., Andrew G. Barto, and Francis Bach. Reinforcement learning: An introduction. MIT press, 1998.

[21] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529.

[22] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

[23] M. Plappert. Keras-rl. https://github.com/keras-rl/keras-rl, 2016.

[24] HADDADI F, PHAN D T, ZINCIR-HEYWOOD A N. How to choose from different botnet detection systems?[C]//Network Operations and Management Symposium (NOMS). 2016: 1079-1084.

[25] ZHAO D, TRAORE I, SAYED B, et al. Botnet detection based on traffic behavior analysis and flow intervals[J]. Computers & Security, 2013, 39: 2-16.

[26] LeCun Y, Jackel L D, Bottou L, et al. Learning algorithms for classification: A comparison on handwritten digit recognition[J]. Neural networks: the statistical mechanics perspective, 1995, 261: 276.

Authorized licensed use limited to: California Polytechnic State University San Luis Obispo. Downloaded on January 23,2023 at 02:55:46 UTC from IEEE Xplore. Restrictions apply.