IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 14, 2019, accepted February 10, 2020, date of publication xxxx 00, 0000,

date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2020.Doi Number

An Intelligent Deployment Policy for Deception Resources Based on Reinforcement Learning

SHUO WANG[1,2], QINGQI PEI[2,3], (Senior Member, IEEE), JIANHUA WANG[1], GUANGMING TANG[1], YUCHEN ZHANG[1], AND XIAOHU LIU[1]

[1] Zhengzhou Information Science and Technology Institute, Zhengzhou 450001 China
[2] State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, 710071, China
[3] Shaanxi Key Laboratory of Blockchain and Secure Computing, Xidian University, Xi'an 710071, China

Corresponding author: Shuo Wang (waltshuo@ 163.com)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFE0126000, in part by the Key Program of NSFC-Tongyong Union Foundation under Grant U1636209, in part by the National Natural Science Foundation of China under Grant 61902292, and in part by the Key Research and Development Programs of Shaanxi under Grant 2019ZDLGY13-04 and 2019ZDLGY13-07.

ABSTRACT Traditional deception-based cyber defenses (DCD) often adopt the static deployment policy that places the deception resources in some fixed positions in the target network. Unfortunately, the effectiveness of these deception resources has been greatly restricted by the static deployment policy, which also causes the deployed deception resources to be easily identified and bypassed by attackers. Moreover, the existing studies on dynamic deployment policy, which make many strict assumptions and constraints, are too idealistic to be practical. To overcome this limitation, an intelligent deployment policy used to dynamically adjust the locations of these deception resources according to the network security state is developed. Starting with formulating the problem of deception resources deployment, we then model the attacker-defender scenario and the attacker's strategy. Next, the preliminary screening method that can derive the effective deployment locations of deception resources based on threat penetration graph (TPG) is proposed. Afterward, we construct the model for finding the optimal policy to deploy the deception resources using reinforcement learning and design the Q-Learning training algorithm with model-free. Finally, we use the real-world network environment for our experiments and conduct in-depth comparisons with state-of-the-art methods. Our evaluations on a large number of attacks show that our method has a high defense success probability of nearly 80%, which is more efficient than existing schemes.

INDEX TERMS Deception resources, deployment policy, attacker-defender scenario, attacker's strategy, reinforcement learning.

I. INTRODUCTION

In recent years, we have witnessed a majority of penetration attacks against the critical server infrastructures within both commercial companies and government organizations [1]. Especially, the advanced persistent threat (APT), which is characterized by its concealment, makes traditional defenses (such as firewall, intrusion detection system, anti-virus software, and so forth) invalid and poses a significant risk to the network security [2] [3].

Consequently, deception-based cyber defenses (DCD) have provided additional selectable measures to complement traditional defense [4]. As an advanced defense concept rather than a specific technology, the core idea of DCD is that the defender deploys a scam in the target network to confuse or mislead the attacker's knowledge of the network system and to thereby cause the attacker to take specific actions that are beneficial to the defender.

Compared to moving target defense (MTD), which can increase the cognitive difficulty of attackers, DCD does not focus on changing the system's attack surface continuously, but instead on misdirecting attackers and feeding them disinformation [5] [6]. Quite evidently, DCD has a farther

target than MTD, and some scholars also call DCD the post-MTD era [7]. In 2016, Springer published "Cyber Deception" [8], the first book dedicated to DCD research, bringing together the latest deception research.

Obviously, creating deception resources is the first step in implementing defense measures that belong to DCD. As is known to all, one of the most common deception resources is honeypot, which has been studied by many scholars. So far, several honeypots have been created in existing literature, which can be divided into three categories: low-interaction honeypot, medium-interaction honeypot, and high-interaction honeypot. Provos *et al.* [9] propose a low-interaction honeypot that mimics the TCP/IP stack of any system, which can be used to deceive the nmap-type stack fingerprint tools. Also, some scholars have created some honeypots for application layer protocols, e.g., Telnet [10] and HTTP [11], as well as some honeypots for special devices, such as smartphones [12], USB devices [13], and data acquisition equipment [14]. Besides these honeypots, there are a variety of forms of deception resources. In [15], Juels *et al.* design the "honey-words" to detect the attacker that is cracking the fake account. The "honey-patches" is proposed in [16] to trick the attacker with the help of fake vulnerability patches. Subsequently, IoT devices [17], URL address [18], and encryption measures [19], [20] are also proposed to lure and deceive attackers. Actually, anything that attackers are interested in can be forged as a deception resource.

Currently, research on DCD focuses on improving the fidelities of the deception resources so that the attacker will fail to recognize the difference between them and the real network environment. However, to our knowledge, little attention has been devoted to the research on how to deploy the deception resources efficiently, which is also an important enabling technique for DCD. The authors of [21] introduce graph theory to model the attacker's behavior and design a decoy chain deployment method to protect against these penetration attacks. A web honeypot based on collaborative mechanism is proposed to make the honeypot-cluster work together as if it is a single honeypot [22]. In addition, Jajodia et al. [23] and Duy et al. [24] obtain the guidance of the honeypot nodes deployment by using game theory. Nevertheless, most of these studies can still be classified as static deployment methods, and they have poor effects.

Understanding the state (intent, ability, and strategy) of the attacker is the key requirement of any successful deception [7]. Meanwhile, in an attacker-defender process, the defender's knowledge of the attacker's state may improve over time. However, the static deployment method considers the attacker's states as the invariant value, which is bound not to support a successful deception. In fact, it is easy to implement a dynamic deployment plan for deception resources by using the software defined network (SDN) technology [25]-[29]. From the above analysis, we can see that it is critical in designing a dynamic deployment method to maximize the effectiveness of the deception resources. Clark *et al.* [30] periodically change the IP address of the honeypot nodes, which will make the honeypot IP identified by the attacker invalid and increase the security of the honeypot nodes. Similarly, Sun *et al.* [31] [32] place the decoy nodes in the target network and then confuse the attacker by IP randomization. By analyzing network characteristics, Shakarian *et al.* [33] combine MTD with decoy nodes and deploy "interference clusters" in the network to reduce the probability of successful intrusion. A model of dynamic array honeypot is proposed to confuse the attacker by coordinating and changing the role pseudo-randomly as a huge dynamic problem [34]. Unfortunately, the above methods suffer from two limitations. First, it has been shown that their approaches tend to prevent attackers from discovering deception resources rather than actively luring them. Second, to infer the attacker's strategy, they have made many strict assumptions and constraints, which cannot be satisfied in practice. As a result, they are too idealistic to be practical.

As one of the most attractive research hotspots of artificial intelligence (AI), reinforcement learning (RL) has made great breakthroughs in many fields, e.g., robotics, autonomous driving, and board games. Especially the emergence of AlphaGo [35] and AlphaGo Zero [36] has greatly shocked the society. Beyond these, in the difficult strategy confrontation games such as StarCraft and Dota2, AI technologies based on RL have also defeated humans. In fact, the essence of network attack and defense is similar to the scene of strategy confrontation games. More importantly, by combining RL and network security data, we can acquire convincing knowledge of the attacker's state, which can help us design more successful deception schemes. Then, inspired by recent success in RL, we are now seeing a renewed interest in DCD. Specifically, we propose an intelligent deployment policy for deception resources based on RL. Our method can dynamically adjust the locations of deception resources according to the network security state and trap the attacker in maximize probability. To our best knowledge, our work is the first one that achieves satisfactory properties without strict assumptions and constraints about the attacker's strategy.

The remainder of this paper is organized as follows. In section II, the problem setting of deception resources deployment is described. In Section III, the method for preliminary screening of effective deployment locations of deception resources based on TPG is given. In Section IV, the details of intelligent deployment policy for deception resources based on RL are presented. Section V describes the experiments and exhibits obtained results. Finally, we conclude the paper in Section VI.

II. PROBLEM SETTING OF DECEPTION RESOURCES DEPLOYMENT

In this section, the problem setting of deception resources deployment is described. We present the assumptions, the attacker-defender scenario, and the attacker's strategy in detail. As mentioned earlier, the core idea of DCD is to induce attackers to stray into the false deception schemes designed by the defender in advance so as to realize the pro-active protection of real resources in the target network. Clearly, the effectiveness of DCD depends on two key factors. One is the fidelity of deception resources, that is, how to make attackers believe them without a shadow of a doubt; the other is the policy of deception resources deployment, namely, how to induce attackers to fall into deception resources. In this paper, we focus on solving the second problem. For simplicity, we assume that deception resources have high fidelity, which means that our deception resources can lure the attacker into investing and depleting all his/her resources.

A. ATTACKER-DEFENDER SCENARIO

In this paper, we go one step beyond by showing that RL can help us obtain the convincing knowledge of the attacker's state and design a successful deception policy solely from network security data, without human data, guidance or domain knowledge. Based on the idea of DCD, we consider a simple and practicable attacker-defender scenario (ADS), which consists mainly of three components: target network, the attacker, and the defender, as shown in Fig. 1.

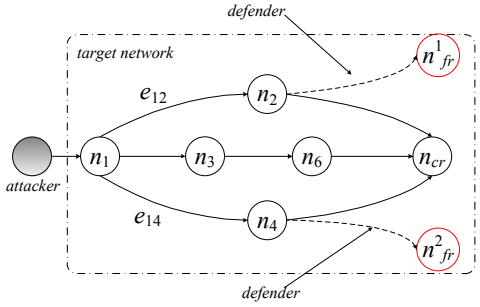


FIGURE 1 Components of attacker-defender scenario.

Definition 1: (*Target Network*) A target network *TN* is a four tuple $TN=(N,E,n_{cr},N_{fr})$ where $N=\{n_j \mid j=1,2...,k\}$ is the set of normal nodes, $E \subset N \times N$ is a set of edges, $n_{cr} \in N$ is the only one of these normal nodes that installs the confidential resources (data or files), and $N_{fr}=\{n^i_{fr} \mid i=1,2...,m\}$ is the set of deception resource nodes, which install the fake resources.

Note, if $e_{ij} \in E$, i.e., there is an edge from node n_i to other node n_j, it means that the users have access to n_j from n_i. In practice, attackers usually have specific goals. In our ADS, we assume that the goal of the attacker is to obtain these confidential resources which are installed in n_{cr}.

In most real-world target networks, there are some nodes named "external facing", which can be accessed by the outside attacker. In Fig.1, the node n_1 belongs to the "external facing". At the beginning of the ADS, the attacker invades and controls these "external facing" (by exploiting some vulnerabilities in a piece of code running on these nodes), and then he/she will launch several penetration attacks by compromising some other nodes in the target network. Clearly, a successful attack on node n_i from the attacker needs to satisfy two indispensable conditions: i) the attacker has access to node n_i, ii) there are some exploitable vulnerabilities on node n_i. Note that the attacker does not know where the confidential resources are installed. Thus, one complete attack will start from one node belonging to "external facing", and keep penetrating until either the attacker invades the node n_{cr} or the attacker strays into one of deception resource nodes. For simplicity, $A_a(n_i)$, as an attack action implemented by the attacker, means that the attacker invades the node n_i.

To deceive and defeat the attacker, the defender creates deception resources, denoted as N_{fr}, which is the set of deception resource nodes. For each $n^i_{fr} \in N_{fr}$, apart from replacing confidential resources with meticulously synthesized fake resources, it is the same as node n_{cr}. Because creating and maintaining the node n^i_{fr} needs to cost a lot of time and money, the number of deception resource nodes is usually smaller. Besides, we assume that the attacker cannot distinguish the difference between confidential resources and fake resources. Then we can define a deception resources deployment action A_d as a mapping from the set N_{fr} to the set N. Thus, for each deception resource node $n^i_{fr} \in N_{fr}$, there must be one corresponding node $n_j \in N$ that satisfies $A_d(n^i_{fr})=n_j$, which means that the defender creates a connection from n_j to n^i_{fr}. For the sake of brevity and readability, when $A_d(n^i_{fr})=n_j$, we call that the defender deploys n^i_{fr} behind n_j or the deployment location of n^i_{fr} is n_j. In this case, the attacker has access to n^i_{fr} from n_j, and therefore, once the attacker invades and controls the node n_j, he will be able to implement the attack action $A_a(n^i_{fr})$ in the next step. In fact, there are three kinds of possible mapping relationships from N_{fr} to N, as shown in Fig. 2.

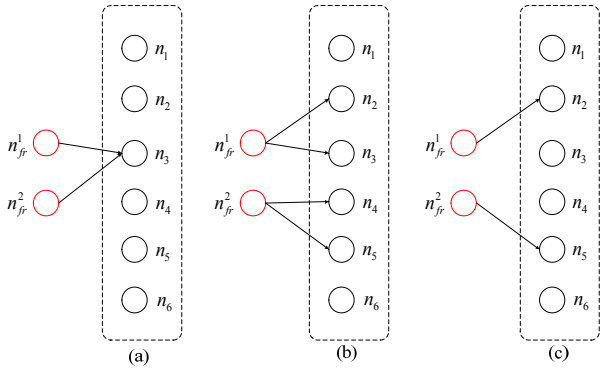


FIGURE 2 Three kinds of possible mapping relationships from N_{fr} to N. a) many-to-one mapping. b) one-to-many mapping. c) one-to-one mapping.

The Fig.2 (a) displays the many-to-one mapping, which means that the defender deploys more than one deception resource node behind the same normal node. As all the deception resource nodes are identical, there is no good except to increase the suspicion of the attacker. The Fig.2 (b) displays the one-to-more mapping, which means that the defender deploys one deception resource node behind more than one normal node at the same time. In this case, according to our observations in the world, the attacker will invade other nodes by using the deceptions resource node as a springboard. The Fig.2 (c) shows the one-to-one mapping, which means that the defender deploys only one deception resource node behind one normal node. Luckily, there are no obvious shortcomings similar to the other two kinds of mapping relationships. From the above analysis, we can see that a reasonable deception resources deployment policy must follow the rules of one-to-one mapping.

After a limited number of attack actions, our ADS will terminate under two cases: (1) when the attacker implements the attack action $A_a(n_{cr})$, he will obtain the confidential resources, which means that the defender failed; (2) when the attacker implements the attack action $A_a(n_{fr}^i)$, he will obtain the fake resources, which means that the defender succeeded.

B. UNCERTAINTY

Understanding the attacker's strategy is critical in designing a successful deception scheme. Thus, in our ADS, accurate prediction about the attacker's next attack action plays a significant role in selecting the effective deception resources deployment policy. Further analysis shows that the attacker's next attack action is mainly determined by two factors: the set of compromised nodes N_{compr} and the attacker's strategy π_a. However, both of these factors are unknown to the defender.

$N_{compr} \subseteq N$ represents the set of nodes that have been already compromised by the attacker. Obviously, N_{compr} will change as the attack progresses, and it can determine the possible attack actions that the attacker is able to implement in the next step. Fortunately, we can understand N_{compr} with the help of a network monitoring system (NMS), which will keep monitoring the health status of each normal node and trigger an alarm of the node when it encounters an attack. Namely, we can infer N_{compr} from these alarms. However, due to the existence of missing and false alarms, there are some uncertainties in this method, as shown in Fig. 3.

In Fig.3, the node with grey solid circle indicates that it has been compromised by the attacker, and the node connected by a red triangular indicates that the NMS raises an alarm. In this example, we set $N_{compr}=\{n_1,n_3\}$, which means the attacker has compromised n_1 and n_3. Based on this, the Fig.3 (a) displays the situation of accurate alarms: the NMS raises the alarms of n_1 and n_3. Fig.3 (b) displays the situation of missing alarms: the NMS only raises the alarm of n_1, but omits the alarm of n_3. Fig.3 (c) displays the situation of false alarms: apart from the alarms of n_1 and n_3, the NMS also raises the alarm of n_2 when no attack occurs. Fig.3 (d) displays the situation of missing and false alarms: the NMS omits alarm of n_3 and raises the extra alarm of n_2. So, it seems that N_{compr} cannot be accurately inferred from these alarms.

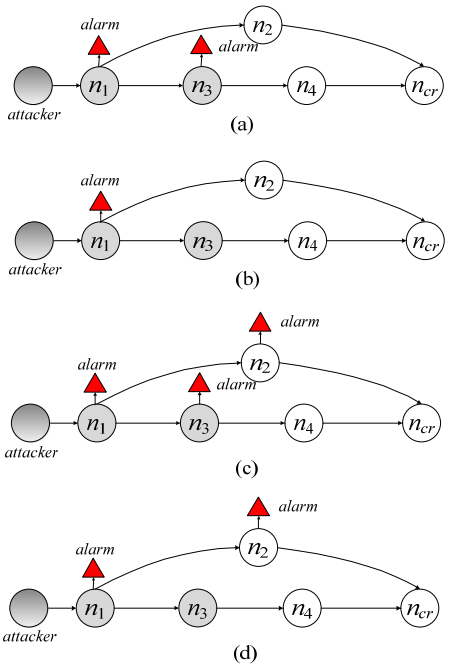


FIGURE 3 Uncertainties coming from missing and false alarms. a) the accurate alarms. b) the missing alarms. c) the false alarms. d) the missing and false alarms

Given the set N_{compr}, let $N_{next} \subseteq N$ denote the set of nodes that the attacker can invade in the next step, and then we can conclude that for each $n_i \in N_{next}$, there is at least one node $n_j \in N_{compr}$, which satisfies $e_{ij} \in E$. Then, we assume that the node that attacker will invade in the next step is n_{next}, hence, $n_{next} \in N_{next}$. By analyzing the actual network attack and defense, n_{next} is mainly determined by the attacker's strategy π_a, which is closely related to the attacker's interestingness distribution I and success probability distribution J.

Given a node $n_i \in N_{next}$, let $I(n_i)$ and p_{n_i} denote the attacker's interestingness in node n_i and the probability that attacker invades n_i in the next step respectively. Hence, the bigger $I(n_i)$ is, the bigger p_{n_i} will be. Analogously, let $J(n_i)$ denote the probability that the attacker invades n_i successfully. The bigger $J(n_i)$ is, the bigger p_{n_i} will be. Here, n_{next} is given by

$$n_{next} = \pi_a(I,J) = \arg\max_{n_i \in N_{next}} \left(\alpha \times I(n_i) + \beta \times J(n_i)\right) \quad (1)$$

where α and β are the weight coefficients ($0<\alpha,\beta<1$) to weigh the importance of the interestingness and the success probability from the perspective of the attacker, respectively. On the one hand, the interestingness distribution I is related to the nature of the nodes, the network topology, and the attacker's preference, which is unknown to the defender. On the other hand, the attacker's success probability distribution J is related to the complexity of vulnerabilities in the nodes, which can be quantified by common vulnerability scoring system (CVSS). Given the deception resource node n_{fr}^{i}, although its location will not affect the attacker's success probability $J(n_{fr}^{i})$, it will affect the attacker's interestingness $I(n_{fr}^{i})$. In this way, the location of n_{fr}^{i} will affect the attacker's next attack action.

Note that the deployment of N_{fr} can also affect the attacker's next attack action. However, from the perspective of the defender, there are two types of uncertainties. First, the missing and false alarms make the inference of the attacker's compromised nodes uncertain, which further leads to the fact that the inference of the attacker's next attack action is uncertain. In this process, the direction of uncertainty transmission is $alarms \rightarrow N_{compr} \rightarrow n_{next}$. Second, the defender is uncertain about the attacker's interestingness distribution I and success probability distribution J, which further also leads to the fact that the inference of the attacker's next attack action is uncertain. In this process, the direction of uncertainty transmission is I, $J \rightarrow \pi_a \rightarrow n_{next}$.

C. ANALYSIS OF DEPLOYMENT POLICY

As discussed above, the key focus of the deployment policy is to accurately predict the attacker's next attack action, which is decided by N_{compr} and π_a. A deployment policy, as a decesion-making rule, defines the defender's way of behaving at given time. In other words, a deployment policy is a mapping from network security states of the target network to deployment actions to be taken when in those states. It can be written as

$$\pi_d(t) = A_d^t \tag{2}$$

Note that A_d is a mapping from N_{fr} to N. Now let AD denote the set of all possible mappings from N_{fr} to N. That is, $A_d \in AD$. According to simple combinatorics, $|AD| = \begin{pmatrix} |N_{fr}| \\ |N| \end{pmatrix}$, where $|N|$ is the total number of normal nodes, $|N_{fr}|$ is the total number of deception resource nodes, and $|AD|$ is the total number of mappings from N_{fr} to N. When the node that attacker will invade in the next step belongs to the set of deception resource nodes, i.e., $n_{next} \in N_{fr}$, it means that the defender traps the attacker, and the deployment policy is successful. Note that there are three kinds of deployment policies that are easy to think of.

First of all, in the traditional network defense, the defender usually places the deception resource nodes in some fixed locations, and we call this method a static deployment policy (denoted as π_d^1). Obviously, the static deployment policy has no relation with time t, and it can be written as

$$\pi_d^1(t) = A_d^t \equiv A_d \tag{3}$$

where A_d represents a fixed element of the set AD. Second, we can randomly place the deception resource nodes in different locations over time, and we call this method the random-dynamic deployment policy (denoted as π_d^2). We can then write π_d^2 as,

$$\pi_d^2(t) = A_d^t = random(AD) \tag{4}$$

where $random(AD)$ means that we randomly select an element from the set AD. Third, deploying deception resources dynamically behind the nodes where the alarms are raised looks like a good idea too, and we call this method the following-alarm deployment policy (denoted as π_d^3). Let AL denote the set of nodes that have encountered an alarm. Then π_d^3 can be written as,

$$\pi_d^3(t) = A_d^t = random(AL) \tag{5}$$

where $random(AL)$ means that we randomly select an element from the set AL.

Although these three kinds of policies are easy to implement, they are more or less flawed. Because of the fixed deployment locations, the static deployment policy can easily be detected and identified by the attacker, which will lead to poorer defense effects. The latter two kinds of policies overcome the disadvantage of being easily identified by the attacker due to their dynamic characteristics. However, the random-dynamic deployment policy can be very much a hit-and-miss affair, and its effect is very unstable from good to bad. Similarly, due to the uncertainty discussed in section II.B, the effect of the following-alarm deployment policy needs to be improved.

For the sake of avoiding the above uncertainties, this paper proposes an intelligent deployment policy for deception resources based on RL. In reality, the target network tends to encounter countless network attacks, which can support our learning model with a large amount of data. Our method regards these uncertainties of all kinds as a black box and uses the alarm data to eliminate them step by step based on RL. Through the learning process, we can build on prior knowledge on predicting the attacker's next attack actions in the ADS. Using our intelligent deployment policy, the locations of these deception resource nodes can be dynamically changed according to the network security state. Let π_d denote our policy, and it can be written as,

$$\pi_d(t) = \pi_d(S_t) = RL(S_t) \tag{6}$$

where S_t represents the network security state at time t, and RL represents the currently trained model.

Compared with the previous three methods, our approach is based on the fact that learning can reduce uncertainties,

which is objective and reasonable. It should be noted that the longer the learning process is, the higher the accuracy of our policy will be. In addition, given the fact that not all nodes can be broken by attackers, we use the TPG model to reduce the deployment location space and improve the efficiency of learning. The full details are in Section III. After that, we describe the complete RL model in Section IV.

III. PRELIMINARY SCREENING OF EFFECTIVE DEPLOYMENT LOCATIONS FOR DECEPTION RESOURCES BASED ON THREAT PENETRATION GRAPH

As mentioned in Section II, not every node in *TN* can be invaded successfully by the attacker. For instance, in Fig.1, if there is no connection from node n_1 to n_4, the attacker will never have access to the node n_4. Hence, he cannot launch an attack on node n_4. In another case, if there is no vulnerability on the node n_2, or if vulnerabilities on this node are too difficult for the attacker to exploit, the attacker cannot successfully invade and control this node even if he has access to it. So, apparently, if the defender deploys the deception resource nodes behind node n_2 or node n_4, the attacker will never be able to invade them. Accordingly, neither node n_2 or node n_4 can be used as an effective deployment location for deception resources. For this reason, we design the preliminary screening method based on TPG to achieve effective deployment locations for deception resources and eliminate these invalid deployment locations simultaneously. As a result, the efficiency of our algorithm for selecting the optimal deployment policy has been enormously improved.

Traditionally, the penetration paths are generated by using the attack graph (AG), which has two significant drawbacks: (1) it has a rather complex generation process, and hence it is not suitable for large-scale networks; (2) it can only describe the connections among the existing vulnerabilities. To solve these problems, we propose a two-layer model (TPG), whose lower layer describes the micro-penetration scenario between each node-pair, and the upper layer shows the macro-penetration relationships between each node-pair in the target network. Let G_{TPG}=(G_{HTPG},G_{NTPG}) denote the TPG of the target network *TN*.

Definition 2: (*Host Threat Penetration Graph*) As the lower layer of TPG, the host threat penetration graph (HTPG for short) is a directed graph G_{HTPG}=(N_{HTPG},E_{HTPG}), where the N_{HTPG} is the set of nodes and E_{HTPG} is the set of edges. A node of N_{HTPG} can be represented as a tuple (*Host*, *Privilege*), which means the host privileges obtained by the attacker. Specifically, *Host* denotes the invaded host by the attacker, and it can be marked as its IP address. *Privilege* denotes the privilege in the host the attacker has obtained, which is divided into User and Root. An edge of E_{HTPG}, indicating the signal-step penetration attack, can be represented as a triple (*Service*, *Vulnerability*, *Probability*), where *Service* denotes the service used by the attacker to invade the node, *Vulnerability* denotes the vulnerability of the service exploited by the attacker, and *Probability* denotes the probability that attacker invades the node successfully.

Definition 3: (*Network Threat Penetration Graph*) As the upper layer of TPG, the network threat penetration graph (NTPG for short) can be defined as a directed graph G_{NTPG} =(N_{NTPG},E_{NTPG}), where the N_{NTPG} is the set of nodes and E_{NTPG} is the set of edges. Specifically, a node of N_{NTPG} denotes a host in the target network, which can be marked as its IP address. An edge of E_{NTPG}, indicating the probability that attacker obtains privilege from one node to another node, can be represented as a tuple (U_P, R_P), where U_P denotes the user privilege and R_P denotes root privilege. Obviously, both U_P and R_P are real numbers between 0 and 1.

To refine the expression, the specific generation method of TPG is shown in the literature [37]. Fig. 4 shows a simple example of a G_{TPG}.

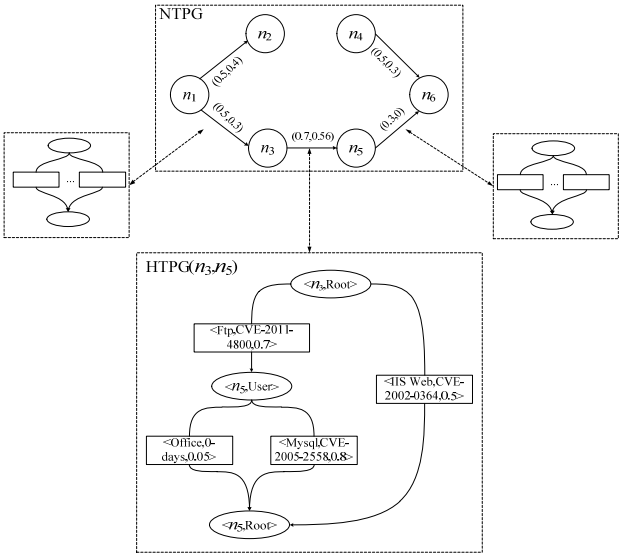


FIGURE 4 A simple example of threat penetration graph.

By using the G_{TPG}, all these normal nodes that cannot be invaded successfully by the attacker will be regarded as the invalid deployment locations for deception resources. For instance, in Fig.4, if only node n_1 can be accessed by the outside attacker initially, the attacker will never be able to invade and control node n_4. Thus, node n_4 is an invalid deployment location for deception resources. Compared with the traditional attack graph, TPG generation algorithm has the advantage of low computational complexity due to the idea of stratification. In addition, the TPG itself, clearly, has a concise visual effect, which can make it easier for the defender to understand the network security state.

IV. INTELLIGENT DEPLOYMENT POLICY FOR DECEPTION RESOURCES BASED ON REINFORCEMENT LEARNING

As analyzed above, in the ADS, the deception resources deployment is an optimization and decision-making

problem with many uncertainties and little prior information. As an important research hotspot of AI, reinforcement learning, which has a strong self-learning ability and can achieve the goal despite uncertainty about the environment it faces, is widely used to solve this kind of problem. In this section, we describe our intelligent deployment policy for deception resources based on RL in detail.

A. MODEL OVERVIEW

According to the classical RL theory, the model to select the optimal deception resources deployment policy based on RL is as shown in Fig. 5.

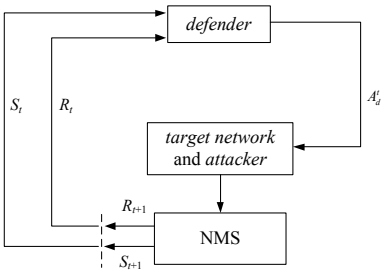


FIGURE 5 Model to select the optimal deception resources deployment policy based on reinforcement learning.

In our model, the defender is equivalent to the agent, while the target network and the attacker together (containing many uncertainties) are equivalent to the environment, and the NMS can be regarded as the sensor of the environment. At time t, NMS integrates several alarms, which can be used to represent the network security state S_t. Generally, it is a fact that the defender will select the deception resources deployment action A_d^t according to the current network security state S_t. Due to the joint action of the attacker, defender, and target network, the network security state will change. At the same time, the defender will be rewarded with environmental feedback R_t. Note that state transitions are considered to be stochastic because the next network security state not only depends on the defender's action, but also on uncertainties that the defender cannot control. In this process, the agent (defender) interacts with the environment (target network and the attacker) continuously and finally tries to select an optimal policy, which can be the guideline for the defender to implement the deception resources deployment actions in any network security state. By implementing the optimal policy, the defender can achieve the purpose of trapping the attacker in maximize probability.

B. MODEL REPRESENTATION

Note that the reward signal R in our model indicates what is good in an immediate sense. Roughly speaking, the better the defender's action is, the higher the reward will be. A reward signal defines the goal in an RL problem. Selecting the optimal policy is to maximize the total reward it receives over the long run. Beyond the defender, the attacker, and target network, one can identify four main elements of an RL model: state, action, reward, and policy.

Definition 4: (*State*). Network security state S, as the name suggests, reflects the security situation of the target network. Also, it can reveal the traces of the attacker for the defender. In our ADS, an alarm may be generated by the NMS once the attacker invades a normal node. For this reason, we can use the alarms generated by NMS to represent the network state S. As mentioned in Definition 1, k is the number of normal nodes in the target network TN. Given the time t, the network security state can be written as

$$S_t = (\psi_1^t, \psi_2^t, ..., \psi_k^t) \tag{7}$$

If the NMS generates an alarm about node i, then $\psi_i^t = 1$. If the NMS generates no alarm about node i, then $\psi_i^t = 0$. Obviously, the size of network security state space is 2^k. Let S_{final}^{fail} and $S_{final}^{success}$ denote the defense final state of failure and success, respectively. In one case, if the deployment policy of the defender cannot trap the attacker, which means that the attacker achieves his/her attack goal and the defense is unsuccessful, then the network security state will change to S_{final}^{fail}; in another case, if the policy traps the attacker, which means that the defense is successful, then the network security state will change to $S_{final}^{success}$.

Definition 5: (*Action*). Let A_d be the deception resources deployment action of the defender. We assume that the number of deception resource nodes available for deployment is m. Given time t, A_d can be written as a matrix of $m \times k$.

$$A_d^t = \begin{pmatrix} \xi_1^{n_{fr}^1} & \xi_2^{n_{fr}^1} & \cdots & \xi_k^{n_{fr}^1} \\ \xi_1^{n_{fr}^2} & \xi_2^{n_{fr}^2} & \cdots & \xi_k^{n_{fr}^2} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_1^{n_{fr}^m} & \xi_2^{n_{fr}^m} & \cdots & \xi_k^{n_{fr}^m} \end{pmatrix} \tag{8}$$

If the defender deploys the deception resource node n_{fr}^j behind the normal node n_i, then $\xi_i^{n_{fr}^j} = 1$. If the defender deploys no deception resource node n_{fr}^j behind the normal node n_i, then $\xi_i^{honeypot_j} = 0$. Obviously, the size of the defender's action space is $\binom{m}{k}$. In addition, for arbitrary $j \in \{1, 2, ..., m\}$, we have

$$\sum_{i=1}^{k} \xi_i^{n_{fr}^j} = 1 \tag{9}$$

Definition 6: (*Reward*). After the defender implements a deployment action, the environment will give him a reward

R. Let us make a human analogy, rewards are somewhat like success (if high) and failure (if low), whereas values correspond to a more farsighted and refined judgment of how successful or unsuccessful the defender's deployment action is in a particular network security state. In our model, at the time *t*, we define

$$R_t = \begin{cases} -1 & if \ \ S_{t+1} = S_{final}^{fail} \\ 1 & if \ \ S_{t+1} = S_{final}^{success} \\ 0 & otherwise \end{cases} \tag{10}$$

The defender implements the deployment action A_d^t according to the network security state S_t, after that, the network security state will transform to S_{t+1}. When $S_{t+1} = S_{final}^{fail}$, which indicates that the defender has failed, we have R_t=-1. Similarly, when $S_{t+1} = S_{final}^{success}$, which indicates that the defender has succeeded, we have R_t=1. In other cases, which indicates that the game is not over yet, we have R_t=0.

Definition 7: (*Policy*). A policy π_d defines the defender's way of behaving at a given time. Roughly speaking, a policy π_d is a mapping from network security states to actions to be taken by the defender when in those states. That is

$$\pi_d(S_t) = A_d^t \tag{11}$$

As a result, the size of policy space is $|AD|^{|S|}$, where $|AD|$ is the size of the defender action space and $|S|$ is the size of the network security state space. We evaluate the policy by the cumulative rewards it receives over the long run, as shown in Formula (12). Thus, the policy with the highest cumulative reward is the optimal policy, as shown in Formula (13).

$$V^{\pi}(S_t) = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \mathrm{K} = \sum_{i=0}^{\infty} \gamma^i R_{t+i} \tag{12}$$

$$\pi^* = \arg\max V^{\pi}(S), (\forall S) \tag{13}$$

C. MODEL LEARNING

The task of RL is usually described by the Markov decision process (MDP). However, in this problem, because of the missing and false alarms generated by NMS along with the uncertainty of attacker's strategy, the state transition probability of the Markov decision-making process is unknown to the defender. In the real world, the target network tends to encounter countless network attacks, which can support our learning model with a large amount of data. As a result, the expected cumulative reward of the policy can be approximated by calculating the average cumulative reward. Accordingly, this paper uses a Q-learning algorithm with an unknown model to learn the optimal policy. The training rule is written symbolically as

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')] \tag{14}$$

where $\hat{Q}_n(s,a)$ is estimated value of $Q(s,a)$, and

$$\alpha_n = \frac{1}{1+visit_n(s,a)} \tag{15}$$

where *s* and *a* are the state and action updated in the *n*th cycle respectively, and $visit_n(s,a)$ represents the number of state-action pair (*s*,*a*) visited by the training algorithm (including the *n*th cycle). Obviously, as the value of *n* increases, α_n decreases, and then the difference between $Q(s,a)$ and $\hat{Q}_n(s,a)$ decreases. The details of the training algorithm based on Q-Learning are shown in Algorithm 1.

Algorithm 1 Training algorithm based on Q-Learning

Input:

the target network *TN*, initial state s_0

Output:

optimal policy π_d

1: Generate the threat penetration graph *TPG* of *TN*
2: Generate the deployment action space *AD* based on *TPG*
3: $Q(s,a)$=0, $visit(s,a)$=0/*Initialize $Q(s,a)$ and $visit(s,a)$ */;
4: $s=s_0$/* Initialize *s* */;
5: **for** *t*=1,2,…,**do** /* Repeat (for each step of episode)*/;
6: *r*,*s*'=*action*(*s*) /* Obtain the reward *r* and new state *s*' after the defender implements the action *a* */;
7: $Q(s,a) = Q(s,a) + \alpha \times (r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
8: $visit(s,a) = visit(s,a) + 1$
9: $s \leftarrow s'$
10: **end for 5**
11: **return** policy $\pi_d(s) = \arg\max_a Q(s,a)$

Function *action*(*s*)
1: select an action according to

$$\pi^{\varepsilon}(s) = \begin{cases} \arg\max_a Q(s,a) & \text{with the probability } 1-\varepsilon \\ \text{select an action } a \text{ from } AD \text{ randomly} & \text{with the probability } \varepsilon \end{cases}$$

2: receive the reward *r* and the new state *s*'.
3: return (*r*,*s*')

According to the convergence theorem of the Q-learning algorithm, when each state-action pair is infinitely visited, the difference between $Q(s,a)$ and $\hat{Q}_n(s,a)$ will be approximately equal to zero. In addition, the computational complexity of Algorithm 1 is $O(n)$, where *n* is the number of training steps. For the above reasons, this algorithm achieves a fast convergence speed and good stability.

V. EXPERIMENTS AND ANALYSIS

To verify the effectiveness of our method, a real network environment is built for testing. The topology of the experimental network is shown in Fig. 6.

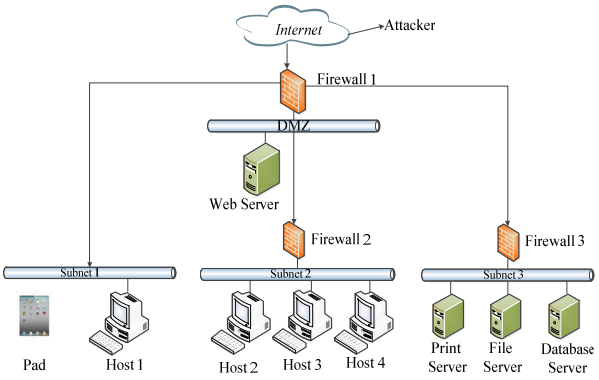


FIGURE 6 Topology of the experimental network.

The experimental network is divided into four regions: DMZ, Subnet 1, Subnet 2, and Subnet 3. The web server is located in the DMZ network. Subnet 1 is composed of a pad and a host. Subnet 2 includes three hosts. Subnet 3 contains a print server, a file server, and a database server. The service access rules in the target network are shown in Table 1. Among them, the attacker controls a host outside the network. Combined with CVSS, we use the Nessus to scan the network and obtain the hosts and vulnerabilities information, which is shown in Table 2. In particular, Pad and Host 1 cannot be accessed directly from Host 2 and Host 3 through the internet, but they can be accessed from these two hosts through USB or other transmission devices due to some improper operations of users. In addition, Host 4, with a higher security level, is a physically isolated entity.

According to the generation method of TPG, HTPG from the attacker to the web server is shown in Fig. 7 and the NTPG of the whole target network is shown in Fig. 8.

TABLE 1. Service access rules of the network.

Source	Destination	Service
Attacker	Web Server	Http
Attacker	Pad 1, Host 1	Ftp,Smtp
Pad	Host 1	Ftp,Smtp
Host 1	Pad	Ftp,Smtp
Pad, Host1	Web Server	Http
Web Server	File Server	Hfs
Web Server	Data Server	Oracle
Host 2	Host 3	FtpShell
Host 3	Host 2	FtpShell
Host 2, Host 3	File Server	Hfs, Rdp
Host 2, Host 3	Data Server	Oracle, Rdp
File Server	Data Server	Oracle

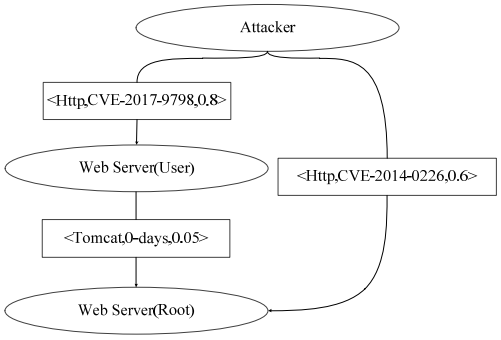


FIGURE 7 Host threat penetration graph from the attacker to Web Server

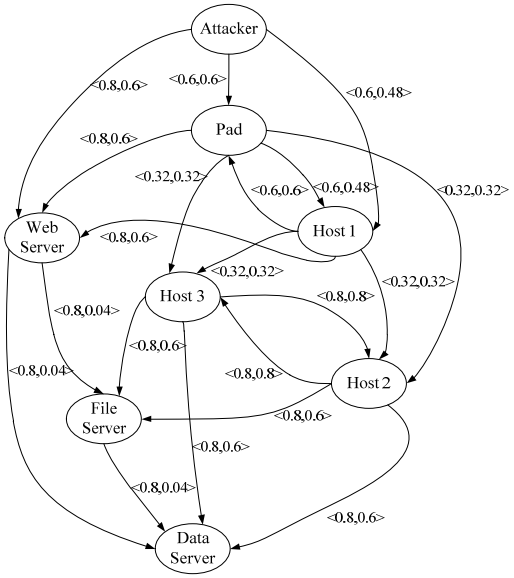


FIGURE 8 Network threat penetration graph of the whole target network.

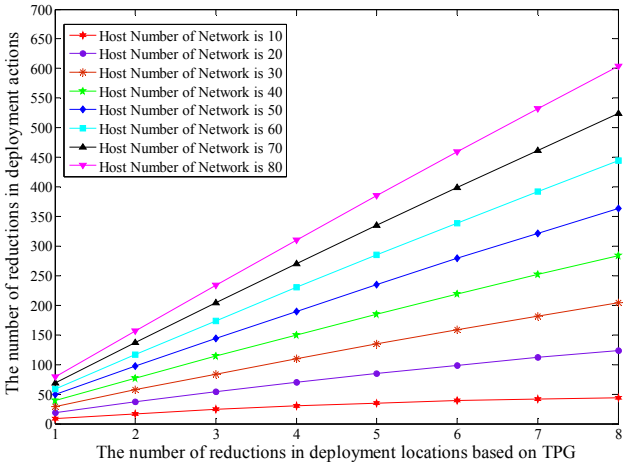


FIGURE 9 Number of actions that decrease with the decrease of the locations of honeypots in target networks of different scales.

TABLE 2. Information of Hosts and Vulnerabilities.

Host Name Number	Host Information	CVE Number	Attack Type	Required Privilege of Source Host	Obtained Privilege of Objective Host	Complexities of Attack
H1	Web Server: Apache, Tomcat	CVE-2014-0226	Remote	User	Root	Medium
		CVE-2017-9798	Remote	User	User	Low
H2	Pad: IOS12	CVE-2016-4729	Remote	User	Root	Medium
H3	Host 1: Windows 7	CVE-2018-8174	Remote	User	Root	High
		CVE-2017-0161	Remote	User	User	Medium
		CVE-2018-8120	Local	User	Root	Low
H4	Host 2: Windows 8 FTPShell	CVE-2015-1769	Remote	User	Root	Low
		CVE-2018-7573	Remote	User	Root	Low
H5	Host 3: Windows 8 FTPShell	CVE-2015-1769	Remote	User	Root	Low
		CVE-2018-7573	Remote	User	Root	Low
H6	File Server: Windows ,HFS	CVE-2014-6287	Remote	User	User	Low
		CVE-2012-0002	Remote	User	Root	Medium
H7	Data Server: Windows ,Oracle	CVE-2016-5555	Remote	User	User	Low
		CVE-2012-0002	Remote	User	Root	Medium

NTPG describes all the possible attack actions in the target network that can be implemented by the attacker. We can see that since there are no vulnerabilities on Host 4 and print server, they are absent from this NTPG. Now, if we deploy the deception resource nodes on these two nodes, the attacker will never invade them. In this case, the deployment is meaningless. On the contrary, Pad, Web Server, Host 1, Host 2, Host 3, File Server, and Data Server belong to this NTPG, which means that they may be invaded by attackers at some time. Hence, these nodes can be used as the deployment locations of deception resources. In short, the NTPG can help us to distinguish which nodes of the target network are suitable for deployment of deception resources and which are not.

Furthermore, we find that the number of reductions in deployment locations based on NTPG is related to the number of reductions in deployment actions by the curve shown in Fig. 9, which can verify that the preliminary screening of effective deployment locations of deception resource nodes based on TPG can reduce the action space of deception resources deployment policy, and ultimately improves the efficiency of RL.

Fig. 8 reveals that there are seven effective deployment locations for deception resource nodes. Then, we assume that there are two deception resource nodes available for deployment, namely, $N_{fr} = \{n_{fr}^{i} \mid i = 1,2\}$. Hence, the size of the network security state space is 128, and the size of the action space is 21. Let the initial state be [0 0 0 0 0 0 0]. The attacker launches a penetration attack against the target network from the external network, then continues to carry out several single-step attacks, and finally realizes the invasion of the attack goal (database server). When the attacker intrudes into the database server, the state transits to S_{final}^{fail} , whereas when the attacker intrudes into any node belonging to N_{fr} , the state transits to $S_{final}^{success}$. In the simulation experiments, the attacker's strategy is expressed by the attacker's interestingness distribution. Our algorithm is trained solely by network security state transition data (i.e., alarms), without any other data. It should be noted that our simulation conditions are not only easy to implement, but also consistent with the actual characteristics of ADS in real networks.

To quantitatively evaluate the quality of our policy, this paper proposed an evaluation index, which is defined as follows.

Definition 9: (Defense Success Probability). Let us assume that the defender implements the policy π_d to deploy the deception resources, then, the ratio of the number of successful defense experiments to the total number of experiments is defined as the defense success probability (*dsp*) of policy π_d , labeled as $dsp(\pi_d)$. That is

$$dsp(\pi_d) = \frac{num}{sum} \times 100\% \tag{16}$$

where *num* denotes the number of successful defense experiments and *sum* denotes the total number of experiments.

Table 3 summarizes the parameters used in our experiments. It should be noted that we consider the false negative rate (FNR) and false positive rate (FPR) of the NMS as variables. To obtain the comparisons, we implement experiments in four different settings of FNR and FPR. Moreover, comparisons of our method with the other three methods that have been described in section II.C are made, and the results are shown in Fig. 10-13.

TABLE 3. List of parameters.

Parameter	Value
target network	Fig. 6
the number of effective deployment locations	seven
the number of deception resource nodes available for deployment	two
initial state	[0 0 0 0 0 0 0]
FNR	Variable
FPR	Variable
total number of experiments	1000

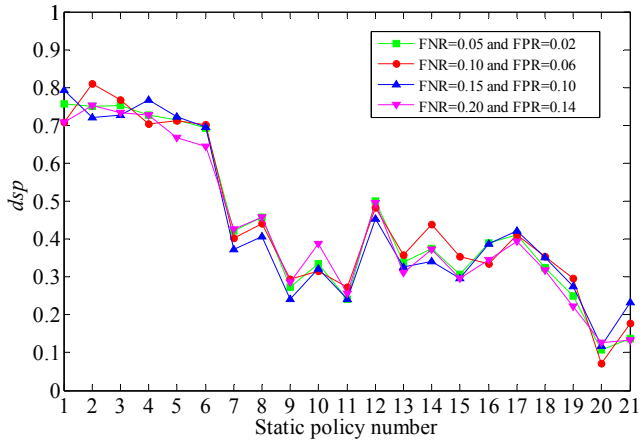


FIGURE 10 Defense success probability of static deployment policy.

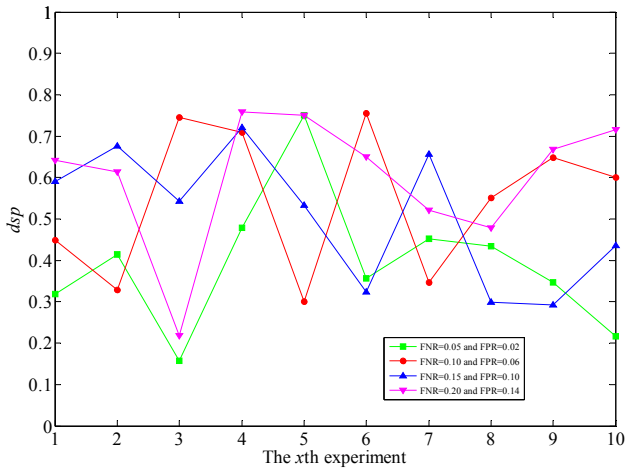


FIGURE 11 Defense success probability of random-dynamic deployment policy.

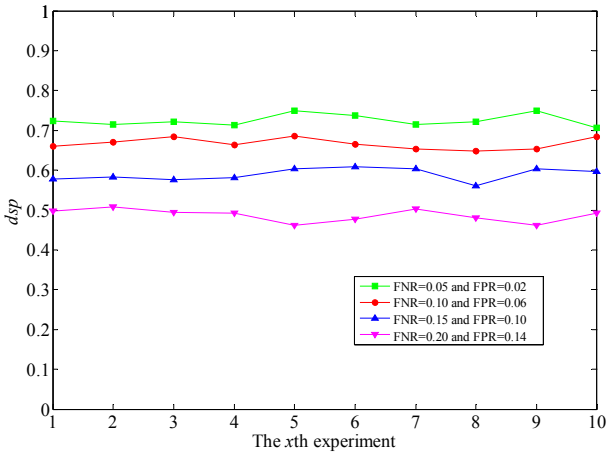


FIGURE 12 Defense success probability of the following-alarm deployment policy.

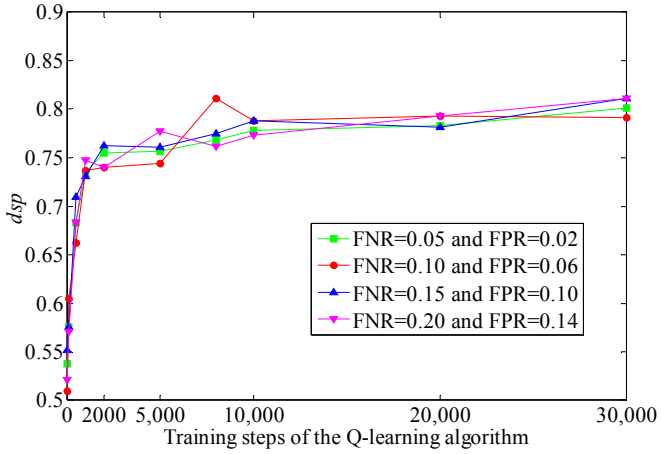


FIGURE 13 Defense success probability of our method.

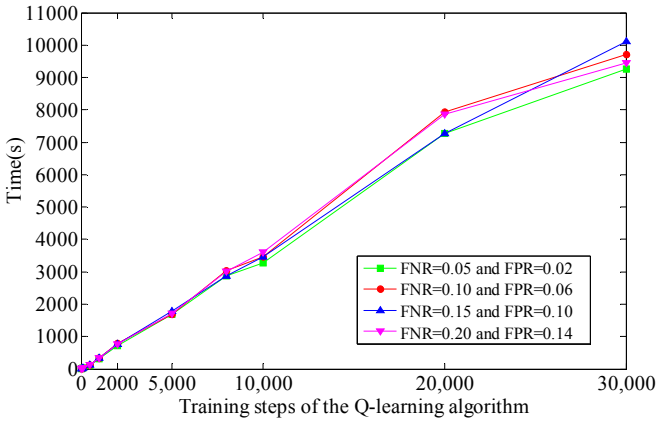


FIGURE 14 Training time needed for our method.

TABLE 4. Comparison Summary.

Method	Detailed Scenario	Loose Assumptions	Support Dynamic	Active Deception	Easy Deployment
[23]	Yes	No	No	No	Yes
[24]	Yes	No	No	Yes	Yes
[25]	Yes	No	Yes	Yes	No
[32]	Yes	Yes	Yes	No	Yes
Ours	Yes	Yes	Yes	Yes	Yes

Fig. 10 shows the results of static deployment policy. According to simple combinatorics, there are 21 static deployment policies due to seven effective deployment locations and two deception resource nodes. The simulation results show that the *dsp* of static deployment policy is very unstable. Among them, the static deployment policy in which the two deception resource nodes are deployed at first and second deployment locations can achieve the best defense effect, and its *dsp* is close to 75.6%. Besides, it should be noted that the results in different settings of FNR and FPR show very similar change rules. However, in the real world, the effect of this policy will be greatly affected by the attacker's strategy. Meanwhile, it is a hard problem to obtain the optimal static deployment policy, which has the biggest *dsp*. Even worse, the static deployment policy can easily be detected and identified by the attacker. Fig. 11 shows the results of the random-dynamic deployment policy. Because of the randomness of this policy, its *dsp* will vary greatly from experiment to experiment. For this reason, the test for this policy is repeated ten times, in which the maximum of *dsp* is close to 75%, and the minimum value is 15.8%. Although the random-dynamic deployment policy can improve the attacker's difficulty in identifying the location of deception resources, it is not a good choice because of its poor effect. Fig. 12 shows the results of the following-alarm deployment policy. We can immediately see that its effectiveness is stable in specific settings of FNR and FPR. Nonetheless, as FNR and FPR increase, its effectiveness is gradually decreased. Due to the high FNR and FPR values of NMS in the real network environment, this policy cannot be widely used in practical applications. Fig. 13 shows the results of our method. When the training steps of the Q-learning algorithm are less than 2000, the *dsp* of our method increases exponentially. And when the training steps are more than 2000, it is maintained at a high level and keeps improving steadily. Finally, with the continuous increase of the training steps, it improves to nearly 80%. Like to the static deployment policy, the settings of FNR and FPR have no effect on the effectiveness of our method. To sum up, our method is superior to the other three deployment policies. Moreover, we record the training time of our method, and the result is shown in Fig. 14, which visibly reveals that the training time of our method linearly increases w.r.t the number of training steps without being affected by the settings of FNR and FPR.

Furthermore, we present a comparison with existing deception resources deployment methods. The comparison results are summarized in Table 4. First, the method with a detailed scenario is convincing and valuable. Second, loose assumptions about the attacker's strategy usually contribute to perfect practicability. Third, static deployment policy for deception resources is easy to be detected and identified by the attacker, so it is significant to support dynamic deployment. Then, active deception can improve the probability of trapping the attacker. Finally, only if these methods are easy-to-deploy can they be widely used.

From the above comparisons, we can conclude that our method is the only one that has not strict assumptions and constraints about the attacker's strategy but achieves satisfactory properties.

VI. CONCLUSION

Nowadays, the deception-based cyber defenses (DCD) has played a significant role in network security. However, the current research on DCD pays less attention to select the optimal deployment policy for deception resources, which limits in some measure the effectiveness of DCD. To solve this problem, we propose an intelligent deployment policy for deception resources based on RL. Our method can dynamically place deception resources as the network security state changes. Starting with describing the attacker-defender scenario and the attacker's strategy, we then analyze the uncertainties and several deployment policies in detail with formal methods. Next, we propose the preliminary screening method that can derive the effective deployment locations for deception resources based on threat penetration graph (TPG). Finally, we design a Q-learning training algorithm for finding optimal deployment policy for deception resources. The experimental results show that our method improves the effectiveness of deception resources deployment. From comparisons with other similar work, we can conclude that our method is the only one that has not strict assumptions and constraints about the attacker's strategy but achieves satisfactory properties. For the future, we plan to extend our work to support large-scale networks and study the relationship between the number of deception resource nodes and the effectiveness of DCD.

REFENERCES

[1] CNCERT. (2019). *2018 Annual Report of Chinese Internet Security.* [Online]. Available: https://www.cert.org.cn/publish/main/upload/File/2018annual.pdf

[2] I. Ghafir, K. G. Kyriakopoulos, S. Lambotharan, F. J.Aparicio-Navarro, B. Assadhan, H. Binsalleeh, and D. M. Dian, "Hidden markov models and alert correlations for the prediction of advanced persistent threats," *IEEE Access*, vol. 7, pp. 99508-99520, 2019.

[3] Z. Shu, G. Yan, "Ensuring deception consistency for FTP services hardened against advanced persistent threats," in *Proc. 5th ACM Workshop Moving Target Defense. (MTD)*, Toronto, Canada, Oct. 2018, pp. 69-79.

[4] D. Fraunholz *et al*., "Demystifying deception technology: a survey," 2018, *arXiv: 1804.06196v1*. [Online]. Available: https://arxiv.org/pdf/1804.06196.pdf

[5] S. Jajodia *et al*., *Moving target defense: creating asymmetric uncertainty for cyber threats*, Berlin, Germany: Springer, 2011.

[6] R. Zhuang, S.A. Deloach and X. Ou, "Toward a theory of moving target defense," in *Proc. Fist ACM Workshop Moving Target Defense. (MTD)*, Scottsdale, AZ, USA, Nov. 2014, pp. 31-44.

[7] C. Wang and Z. lu, "Cyber deception: overview and the road ahead," *IEEE Computer and Reliability Societies*, 2018, pp. 80-85, Mar. 2018.

[8] S. Jajodia, V. S. Subrahmanlan, V. Swarup, and C. Wang, *Cyber deception: building the scientific foundation*. Berlin, Germany: Springer, 2016.

[9] N. Provos, "A virtual honeypot framework," in Proc. 13th *USENIX Security Symp*, San Diego, CA, USA, Aug. 2004.

[10] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama and C. Rossow, "IoTPOT: analysing the rise of IoT compromises," in *Proc. 9th USENIX Conf. Offensive Technol.,* Washington, D.C., USA, Aug. 2015, pp. 9-17.

[11] D. Fraunholz and H. D. Schotten, "Defending web servers with feints, distraction and obfuscation," in *Proc. Int. Conf. Compu. Netw. Commun.(ICNC)*, Maui, HI, USA, Mar. 2018, pp. 21-25.

[12] H. M. Ahmed, N. F. Hassan and A. A. Fahad, "Designing a smartphone honeypot system using performance counters," *Karbala Int. J., Modern Sci.*, vol.3, no. 1, pp. 46-52, Mar. 2017.

[13] R.B. Yehuda, D. Kevorkian, G. L. Zamir, M. Y. Walter, L. Levy, "Virtual USB honeypot," in Proc. 12nd ACM int. Conf. Syst. Storage, Haifa, Israel, Jun. 2019, pp. 181-181.

[14] A. Jicha, M. Patton and H. Chen, "SCADA honeypots: An in-depth analysis of Conpot," in *Proc. IEEE Conf. Intelligence Security Informatics*, Tucson, AZ, USA, Sept. 2016, pp. 196-198.

[15] A. Juels, and R. L. Rivest, "Honeywords: making password-cracking detectable," in *Proc. ACM SIGSAC Conf. Compu Commun Security (CCS)*, Berlin, Germany, Nov. 2013, pp. 145-160.

[16] F. Araujo, K. W. Hamlen, S. Biedermann and S. Katzenbeisser, "From patches to honey-patches: lightweight attacker misdirection, deception, and disinformation," in *Proc. ACM SIGSAC Conf. Compu. Commun. Security (CCS)*, Scottsdale, Arizona, USA, Nov. 2014, pp. 942-953.

[17] F. Dang, Z. Li, Y. Liu, E. Zhai, Q. Chen, T. Xu, Y. Chen and J. Yang, "Understanging fileless attacks, on linux-based IoT Devices with honeycloud," in *Proc. 17th Annual Int. Conf. Mobile Syst. Appl, Services*, Seoul, Korea, Jun. 2019, pp. 482-493.

[18] M. Lazarov, J. Onaolapo and G. Stringhini, "Honey Sheets: What Happens to Leaked Google Spreadsheets," *9th USENIX Workshop Cyber Security Experimentation and Test (CSET 16)*, Austin, TX, USA, Aug. 2016, pp. 1-8.

[19] J. W. Yoon, H. Kim, H. J. Jo, H. Lee and K. Lee, "Visual honey encryption: application to steganography," in *Proc. 3rd ACM Workshop Information Hiding Multimedia Security*, Portland, Oregon, USA, Jun. 2015, 65-74.

[20] A. E. Omolara, A. Jantan, O. S. Abiodun, K. V. Dada, H. Arshad and E. Emmanuel, "A deception model robust to eavesdropping over communication for social network systems," *IEEE Access*, vol. 7, pp. 100881-10898, 2019.

[21] Q. Zhao, C. Zhang and Z. Zhao, "A decoy chain deployment method based on SDN and NFV against penetration attack," *Plos One*, vol. 12, no. 12, pp. 1-23, Dec. 2017.

[22] Z. Jia, B. Fang, X. Cui and Q. Liu, "ArkHoney: a web honeypot based on collaborative mechanisms," *Chin. Compu J.*, vol. 41, no. 2, pp. 413-425, Mar. 2018.

[23] S. Jajodia, N. Park, E. Serra and V.S. Subrahmanian, "SHARE: A stackelberg honey-based adversarial reasoning engine," *ACM Trans Internet Technol*, vol. 18, no. 3, pp. 1-41, Mar. 2018.

[24] Q. Duy La, T. Q. S. Quek, J. Lee, S. Jin and H. Zhu, "Deceptive attack and defense game in honeypot-enabled networks for the Internet of Tings," *IEEE Internet Things J.*, vol. 3, No. 6, pp. 1025-1035, Dec. 2016.

[25] S. Venkatesan, M. Albanese, A. Shah, R. Ganesan, and S. Jajodia, "Detecting stealthy botnets in a resource-constrained environment using reinforcement learning," in Proc. *4th ACM Workshop Moving Target Defense (MTD)*, Dallas, TX, USA, Oct, 2017, pp. 75-85.

[26] T. Shimanaka, R. Masuoka, and B. Hay, "Cyber deception architecture: cover attack reconnaissance using a safe SDN approach," *In Proc. 52nd Int. Conf. Syst. Sci*., Maui, Hawaii, USA, Jan. 2019: 7292-7301.

[27] S. Achleitner, T. L. Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, R. Chadha, "Cyber Deception: Virtual Networks to Defend Insider Reconnaissance," in *Proc. Int. Workshop Manag. Insider Security Threats*, Vienna, Austria, Oct, 2016, pp. 57-68.

[28] S. Achleitner, T. L. Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Deceiving network reconnaissance using SDN-based virtual topologies," *IEEE Trans. Netw. Service Manag.*, vol 14, no.4, pp.1098-1112, Jul. 2017.

[29] J. Kim and S. Shin, "Software-defined honeyNet: towards mitigating link flooding attacks," in *Proc. 47th IEEE/IFP Int. Conf. Dependable Syst. Netw. Workshop (DSN-W)*, Denver, Co, USA, Jun. 2017, pp. 99-100.

[30] A. Clark, K. Sun, and R. Poovendran, "Effectiveness of IP address randomization in decoy-based moving target defense," in *Proc. 52nd IEEE Annual Conf. Decis. Control (CDC)*, Florence, Italy, Dec. 2013, pp. 678-685.

[31] J. Sun, and K. Sun, "DESIR: Decoy-enhanced seamless IP randomization," in *Proc. IEEE Int. Conf. Compu. Commun (INFOCOM)*, San Francisco, CA, USA, Apr. 2016, pp. 1-9.

[32] J. Sun, K. Sun, and Q. Li, "CyberMoat: camouflaging critical server infrastructures with large scale decoy farms," in *Proc. IEEE Conf. Commun. Netw. Security. (CNS)*, Las Vegas, NV, USA, Oct. 2017, pp. 1-9.

[33] P. Shakarian, N. Kulkarni, M. Albanese, "Keeping intruders at bay: a graph-theoretic approach to reducing the probability of successful network intrusions", In *Proc*. Int. *Conf. E-business Telecommun (ICETE)*, Vienna, Austria, Aug. 2014, pp. 191-211.

[34] L. Shi, J. Li, X. Liu, "Research on dynamic array honeypot for collaborative network defense strategy," *J. Commun*, vol. 11, pp. 59-164, Nov. 2012.

[35] D. Silver, *et al*., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484-489, Jan. 2016.

[36] D. Silver, *et al*., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354-359, Oct. 2017.

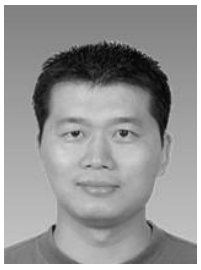
[37] S. Wang, J. Wang, G. Tang, Q. Ping, Y. Zhang, and X. Liu, "Intelligent and Efficient Method for Optimal Penetration Path Generation," *J. Compu Res. Dev.*, vol. 56, no. 5, pp. 929-941, May. 2019.

SHUO WANG is currently pursuing the Ph.D. degree with Zhengzhou Information Science and Technology Institute. His research interests include network security, machine learning and security computing.

QINGQI PEI received the B.S., M.S., and Ph.D. degrees in computer science and cryptography from Xidian University, in 1998, 2005, and 2008, respectively. He is currently a Professor and member of Shaanxi Key Laboratory of Blockchain and Secure Computing. His research interests include digital contents' protection and wireless networks and security. He is a professional member of ACM and a Senior Member of the Chinese Institute of Electronics and the China Computer Federation.

JIANHUA WANG is currently a Professor with the Zhengzhou Information Science and Technology Institute. His main research interests include information security and security management.

GUANGMING TANG is currently a Professor with the Zhengzhou Information Science and Technology Institute. Her contributions encompass the aspects of information theory and security, network architecture, and signal processing.



YUCHEN ZHANG is currently an associate Professor with the Zhengzhou Information Science and Technology Institute. His main research interests include network security situational awareness.

XIAOHU LIU is currently pursuing the Ph.D. degree with Zhengzhou Information Science and Technology Institute. His research interests include network security, moving target defense, and security computing.