Research Article

A Multiphase Dynamic Deployment Mechanism of Virtualized Honeypots Based on Intelligent Attack Path Prediction

Yazhuo Gao , **Guomin Zhang**, and **Changyou Xing**

Command & Control Engineering College, Army Engineering University of PLA, Nanjing 210007, China

Correspondence should be addressed to Changyou Xing; changyouxing@126.com

Received 10 August 2021; Revised 4 October 2021; Accepted 6 October 2021; Published 21 October 2021

Academic Editor: Shahram Babaie

Copyright © 2021 Yazhuo Gao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As an important deception defense method, a honeypot can be used to enhance the network's active defense capability effectively. However, the existing rigid deployment method makes it difficult to deal with the uncertain strategic attack behaviors of the attackers. To solve such a problem, we propose a multiphase dynamic deployment mechanism of virtualized honeypots (MD2VH) based on the intelligent attack path prediction method. MD2VH depicts the attack and defense characteristics of both attackers and defenders through the Bayesian state attack graph, establishes a multiphase dynamic deployment optimization model of the virtualized honeypots based on the extended Markov's decision-making process, and generates the deployment strategies dynamically by combining the online and offline reinforcement learning methods. Besides, we also implement a prototype system based on software-defined network and virtualization container, so as to evaluate the effectiveness of MD2VH. Experiments results show that the capture rate of MD2VH is maintained at about 90% in the case of both simple topology and complex topology. Compared with the simple intelligent deployment strategy, such a metric is increased by 20% to 60%, and the result is more stable under different types of the attacker's strategy.

1. Introduction

With the vigorous development of Internet technology, network attack methods have become more complex and diverse, and network deception [1] has become an important idea to combat network threats. As one of the important methods of network deception technology, honeypots mainly capture and monitor attackers by deploying deception nodes that provide fake services in different topological networks. The existing honeypot system design is mainly divided into two perspectives: improving the intelligent management capability and enhancing the trapping capability. The former aims to make the deployment of the honeypot more flexible, effective, and manageable, while the latter focuses on techniques that can be used to construct a deception scenario with high fidelity.

Generally speaking, honeypot deployment strategies include the static strategy and the dynamic strategy. The traditional honeypot systems are mainly based on physical devices or virtual machines, and the deployment

and configuration processes are very complicated. Therefore, it is more inclined to adopt the static deployment strategies, which are usually combined with game theory [2] or graph theory [3]. Such a deployment strategy is inflexible and does not scale well, which is difficult to accurately reflect the defender's dynamic cognitive process of the attackers. Therefore, with the development of lightweight virtualization technology, dynamic deployment strategies have gradually become the mainstream. Such strategies will adjust the deployment scenario of honeypots in different stages according to the current behaviors of the attackers. However, most of the current dynamic deployment strategies can only be deployed according to the preset scenarios, which is still a quasi-static strategy with poor adaptability. This paper mainly focuses on enhancing the trapping capability of the honeypot by introducing an intelligent deployment strategy. The main purpose is to predict the attackers' intention based on the continuous understand of the attack behaviors and adjust the deployment strategy

dynamically under the constraints of limited honeypot resources, so as to prevent the lateral movement in time while maximizing the benefits of trapping. Based on such an idea, this paper takes a hypothesis that we already have some means to detect the vulnerability exploitation process and to deceive the attackers inside the honeypot. And we mainly pay attention to predicting the attack path accurately and adjusting the deception resources dynamically based on the continuously collected information.

In order to enhance the self-learning ability of deployment strategies and enable it to deal with more complex scenarios, this paper combines the intelligent algorithms with the idea of dynamic deployment and proposes a multiphase dynamic deployment mechanism for virtualized honeypot (MD2VH) based on an intelligent attack path prediction. Based on the Bayesian state attack graph model, MD2VH monitors and analyzes the attackers' attack behaviors, forms the predicted attack path, and uses the machine learning algorithms to learn the attackers' behavior patterns online based on the attack path, so as to form a real-time deployment strategy model of the virtualized honeypots. Finally, by using the containers and SDN technologies, we also establish a loosely coupled and extensible honeypot system, which can adaptively generate and adjust the deployment of honeypots according to the intelligent deployment strategies. Generally speaking, the main contributions of this paper are as follows:

- (1) By combining the attribute attack graph with the Bayesian network graph model, we propose a new type of Bayesian state attack graph model, which can describe the network vulnerabilities as well as the exploitation of these vulnerabilities, and thus provide a basis for constructing the attack path prediction models.
- (2) We proposed MD2VH to solve the intelligent honeypot deployment problem. MD2VH establishes a multiphase dynamic deployment optimization model of the virtualized honeypots based on the extended Markov's decision-making process and generates the deployment strategies dynamically by combining the BP neural network and DQN [4] algorithm strategy.
- (3) By using the SDN and container technology, we also establish a loosely coupled and scalable virtual honeypot system to evaluate the trapping ability of MD2VH, which demonstrates that MD2VH is more competitive when considering the capture rate and resource consumption.

The rest of the paper is arranged as follows: Section 2 introduces the related work, Section 3 models and analyzes the intelligent trapping decision-making problem and then provides a solution method based on BP neural network and Deep Q Network, Section 4 proposes a prototype system to verify the effectiveness of the strategy, and finally, Section 5 summarizes the full-text work.

2. Related Work

Initially, honeypot systems are often built by using real hosts, but with the maturity of virtualization technology, lower-cost virtual machines have gradually become the main components of honeypot systems. HoneyV [5] judges the credibility level of all inbound traffic through IDS and puts attackers into four honeypots of different monitoring levels for analysis based on the results. INTERCEPT+ [6] changes the virtual machine migration code so that the source virtual machine remains running after the migration, ensuring the normal interaction of communication with the source virtual machine.

Although virtual machines can balance the conflict between deployment costs and interaction levels to a certain extent, the number of virtual machines that can be established on a server is still limited, making it difficult to achieve large-scale trapping network deployment. With the development of technologies such as SDN [7], lightweight virtualization technology LXC [8], and Docker [9], many teams began to combine new network and virtualization technologies with the design ideas of honeypot systems, which combine the trapping system structure of SDN [6] and the active defense of containers [10]. Sun et al. [11] proposed a dynamic hybrid honeypot system, and the architecture includes lightweight front-end decoys and high-interaction back-end decoys. The front-end decoys transparently intercept and forward malicious commands to the back-end decoys, which use a low-interaction honeypot to show a high-interaction system view. Honeyproxy [12] can copy one attack traffic to multiple trapping points and then selects the most suitable one from all of the replies to feedback to the attacker. Honeypatches [13] use LXC as a trapping point and transfer the attack session that has touched the decoy patch to a specific trapping point for attack trapping. AHEAD [10] proposed that active defense tools can be encapsulated inside Docker and installed directly into the real system, which forces the attacker to filter real and fake services during the attack, which delays the attacker's attack time. After developing AHEAD, the authors further simplified the deployment and integration of active defense tools and proposed the ADARCH [14] architecture. HADES [15] proposed a shadow honeynet architecture combined with SDN technology. Urias et al. [16, 17] used an introspection program to monitor and configure all virtual machines and can migrate the traffic in the real host after suspicious traffic is detected. HoneyDOC [18, 19] proposed a TCP replay mechanism combined with SDN technology, which can seamlessly migrate the attacker's traffic without reestablishing the connection after the migration.

The dynamics of these systems are mainly reflected in the dynamic migration of traffic and the dynamic generation of decoys, but there are not many discussions on how to dynamically optimize the location of honeypots in the network, and they are basically based on preset settings. The deployment is still essentially a quasi-static deployment strategy.

Therefore, in recent years, many honeypot systems have gradually introduced the idea of intelligent learning and

game. El Kamel et al. [20] proposed an algorithm based on the idea of machine learning clustering to identify the attacker in the trapping point and the result used for the configuration of the later defense strategy. Huang et al. [21] introduced a honeypot mechanism that could not be recognized by attackers based on a random forest algorithm. SMDP [22] proposes applying the Markov decision process method to attack trapping, transforms the continuous-time process into an equivalent discrete decision model, uses reinforcement learning to train the model, and finally gets the optimal strategy. Gill et al. [23] and others designed an independent GTM-CSec model, which combines the static game of complete information with the design of trapping mechanism, and simplifies the game model through inferior strategies; this model can intelligently select appropriate defense modules but reduce system energy consumption and respond to attacks intelligently. Zhou et al. [24] used the Markov to model the penetration testing process and proposed an improved DQN algorithm, which converges faster in larger networks and is more suitable for large-scale networks. Bohara et al. [25] used methods such as Principal Component Analysis and Median Absolute Deviation-based outlier detection to analyze the traffic captured by honeypots to resist lateral movement attacks. Takabi and Jafarian [26] proposed a defense idea that combines deception defense with MTD, which can be used to mitigate internal attacks. Amin et al. [27] combined POMCP and Hidden Markov Model and proposed a new path prediction method to prevent the attacker from moving laterally in the internal network.

In addition, some papers introduce the theory of attack graphs based on the Markov model to optimize the modeling scenario. Zhang et al. [28] established an incomplete information random game model for the attack and defense process after the attackers entered the real system and used an improved Q_Learning algorithm to solve the problem, but the decision in this document focused on the system state transitions with different defense actions, the algorithm requires all network states to be known, but our algorithm does not require all states, there are fewer restrictions on scenario settings, and there is no need to guess the attackers' type in advance, which reduces the difficulty of scene modeling and is more practical. Wang et al. [29] combined the two-layer threat penetration map, Q_Learning, and dynamic deployment to propose a dynamic deployment strategy of deception resources based on reinforcement learning, their strategy can also predict the attackers' next path based on the attackers' current alert paths, but the attackers considered by this method are a single-mode attack, the defender uses offline learning during the entire learning process, and it can only learn a known attack mode for defense based on the collected attack information. In addition, this method considers that the attacker has compromised a certain system node, so, the starting position of the attack path has been determined and cannot be predeployed. But MD2VH combines online learning with offline learning, the initial state can be based on the current strategy model to decide the deployment of the first honeypot, which increases the speed of trapping attackers and

reduces attack losses, and it can also deal with multiple types of attack modes and can conduct online learning and modify models during the process.

However, most of the current intelligent honeypot systems still lack the ability of effective analysis and prediction of the attackers' behaviors, which makes them cannot perform the fine-grained strategic deployment and adjustment of virtual honeypots intelligently. To solve these problems, in this paper, we proposed a Bayesian state attack graph-based method to analyze and predict the attackers' behaviors and designed a dynamic deployment strategy construction algorithm under the constraint of the predicted attack graph and virtual honeypot resources. Compared with these methods, the method proposed in this paper can provide a more accurate description of the dynamic attack threats and increase the capture rate of the honeypot effectively by constructing a more strategic dynamic deception scenario.

3. Methodology

3.1. Overview. In the penetration attack stage, the attackers often firstly establish a stronghold in the internal network and then use the topological connections and vulnerability exploitation relationship to perform the lateral movement continuously and finally achieve the goal of successfully compromising the target node. In such a multiphase process of continuous reconnaissance and movement, different attackers will form different attack paths to reach the attack target based on the information of different vulnerabilities found in the network. As a result, by deploying honeypots adaptively according to the attacker's behaviors, we can prevent the lateral movement of the attacker effectively, making it difficult to reach the target node. Based on such an idea, we give the basic design concept of MD2VH.

Firstly, we give a brief description of the network attack and defense scenario, as shown in Figure 1, where the attackers' attack means is represented by A_i , and their purposes are to compromise the target node T . The attackers can only attack the nodes connected to the compromised nodes, and they will select the attackable nodes according to their own mode. The defense models (denoted by D_i) are stored in the model pool maintained by the defender, and the defender's purpose is to capture the attackers as early as possible. The defender knows the dependencies between all vulnerable nodes in the system but does not know the probability of the attackers' attack choice. For simplicity of the description, here, we set the defense model corresponding to A_1 as D_1 and then describe each stage as follows:

- (1) Before the attack begins, the defender will conduct preliminary modeling of the overall network topology and known vulnerabilities of the system and use the Bayesian state attack graphs to describe the vulnerability exploitation relationship between all nodes. Besides, the defender also infers the attack path based on the current threat alarms (i.e., as shown in Figure 1, the attack path has been inferred to be $1 \rightarrow 2$, and the attack mode chosen by the attacker at this time is A_1).

FIGURE 1: Network attack and defense scenario design.

- (2) The defender will select the corresponding model from the defense model pool according to the currently learned strategy model and deploy the honeypots according to the model (i.e., if the defender selects model D_1 , it will deploy the honeypot at node 4'. Otherwise, if the defender selects model D_2 , it will deploy a honeypot at node 5').
- (3) The attacker calculates the attack probability of the subsequent nodes of the currently compromised node according to his own preferences and then selects one of the attackable nodes according to the probability distribution (i.e., because it is the subsequent stages of the same attack process, the attacker will continually use mode A_1 to launch an attack on node 4).
- (4) If the defender has deployed the corresponding honeypot of the attacked mode (i.e., when the model selected by the defender corresponds to the attack mode of the attacker, the honeypot deployed in node 4' will capture the attacker), then the defender will capture the attacker and control its behavior. Otherwise, the attack will continue and repeat steps 2~3.
- (5) There are two situations when the network attack and defense process end. One is that the attacker is captured by the honeypot, and the defender will generate a honeynet dynamically, so as to further deceive the attacker and analyze its behaviors. The other is that the attacker successfully compromises the target node T, the defender fails to capture the attacker, and the attack ends.

Based on the attack and defense scenario design, the following part will model and analyze the specific attack and defense process with the extended Markov process and propose an intelligent solution algorithm.

3.2. Network Attack and Defense Process Model. In the traditional Markov decision process, the agents' action space is fixed and will not change during the entire learning process. However, in a real network attack and defense scenario, the defender may not know all possible occurrences before the attack and defense process begins. The attackers' behavior preference is, therefore, considered to introduce a certain degree of dynamics in the Markov decision process. Before the attack and defense process begins, the attackers' attack modes may be greater than the defender's defense models, and after multiple rounds of learning, the defense models of the defender will increase as the collected attack information changes, so the action space of the defender may change during the entire attack and defense process.

Definition 1. Network Attack and defense process: $(G, T, N, S, A, D, \pi, r)$.

$G = (V, E, P)$: Bayesian state attack graph.
 $E = \begin{pmatrix} \varepsilon_1^1 & \cdots & \varepsilon_n^1 \\ \vdots & \ddots & \vdots \\ \varepsilon_1^n & \cdots & \varepsilon_n^n \end{pmatrix}$ denotes edges between nodes;
 $\varepsilon_j^i = 1$ indicates that there is an edge between node i and node j ; otherwise, $\varepsilon_j^i = 0$. $V = \{1, 2, \dots, n\}$ denotes that there are n nodes in the system that can be attacked and deployed honeypots. P is the transition probability matrix between nodes.

$T = \{1, 2, 3, \dots\}$: the number of phases of the attacks, which is an integer with a value range greater than 0.

$N = \{\text{attacker, defender}\}$: the participants.

$S = (s_1, s_2, \dots, s_n)$: the set of network states. n denotes the total number of nodes; $s_i = 1$ indicates that node i sends out an alarm signal; $s_i = 2$ indicates that node i has deployed a honeypot; otherwise, $s_i = 0$. S^t denotes the state of t phase.

$A = \{A_1, A_2, \dots, A_{ka}\}$: attackers' action space. ka represents the total number of actions the attackers can perform, and each action represents a pattern of aggressive behavior; A_i^t indicates that the attacker chooses the action A_i in the t phase.

$D = \{D_1, D_2, \dots, D_{kd}\}$: action space for the defender. kd represents the total number of actions the defender, and each action represents a defensive model. D_i^t denotes that the defender chooses the action A_i in the t phase. kd is a variable that will change as the attack information increases.

π : strategy function. Define $\pi_a: S \rightarrow A$ is the mixed strategy chosen by the attacker in each state; $\pi_a(S^t)$ indicates the mixed strategy of the attacker in the state S^t . $\pi_d: S \rightarrow D$ denotes the mixed strategy chosen for the defender in each state; $\pi_d(S^t)$ indicates the mixed strategy of the defender in the state S^t .

r : rewards for both the attackers and defender. The value of r is related to the actions of both the offensive and defensive parties in the current state. $r_a(S^t, A_k^t, D_k^t)$ denotes the reward of the attacker when the attacker takes the action A_k and the defender takes the action D_k in S^t . $r_d(S^t, A_k^t, D_k^t)$ denotes the reward of the defender in the same case.

3.2.1. Bayesian State Attack Graph. The attack graph can be denoted by $G = (V, E)$, which includes nodes set V and directed edges set E . The attack graph mainly includes the attribute attack graph and the state attack graph. Since the state attack graph has the state explosion problem when the number of nodes is large, the application of the attribute attack graph is more common. There are two types of nodes in the attribute attack graph: vulnerability nodes and condition nodes. Vulnerability nodes generally represent services that can exploit the vulnerabilities, and condition nodes represent conditions that the attacker has. Such conditions generally refer to certain permissions.

A common attribute attack graph is shown in Figure 2(a)), where the rectangle nodes represent the condition nodes, and the oval nodes represent the vulnerability nodes. The arrow pointing from the condition node to the vulnerability node indicates that the condition is a precondition for exploiting the vulnerability, and the arrow by which the vulnerability node points to the condition node indicates the postconditions. Only when all the preconditions are met, the vulnerable node may be successfully exploited.

Although the attribute attack graphs can reflect the causal relationship and attack paths between nodes, they cannot well represent the conversion probability between nodes. Therefore, many studies have begun to combine Bayesian graphs with attribute attack graphs to form the Bayesian attribute attack graph [30]. The Bayesian attribute attack graph can be modeled as $G = (V, E, P)$, which extends the probability of successful attack P based on the attribute attack graph. Most Bayesian attribute attack graphs choose

to omit the vulnerable nodes and retain attribute nodes. Such graphs generally have three structures: and, or, and hybrid (both relationships have), as shown in Figure 2(b). Although this type of graph can retain the attack paths of the original attack graph to the utmost extent, the complex connection relationship between the attributes will lead to higher calculation cost of the attack probability, and it is difficult to reflect the influence of different characteristics of vulnerabilities on attackers' preferences.

In order to better solve the above two problems, based on the combination of attribute attack graph and Bayesian graph, MD2VH omits the condition node to generate the Bayesian state attack graph. By default, when an atomic attack is selected, its preconditions are all satisfied; otherwise, the probability of the node i being used is $p_i = 0$. Bayesian state attack graph can highlight the correlation between system vulnerabilities and retain the ability of the attribute attack graph to record the attack paths. As shown in Figure 2(c)), the attack graph starts from the Start node and ends at the Terminate node. The Start node is connected to the initial attack nodes, and the remaining nodes are intermediate nodes. Each intermediate node represents a vulnerability. The characteristics of the vulnerabilities are different, which will affect attackers' selection probability of these nodes. The graph can also be modeled as $G = (V, E, P)$, where V represents the Start node, Terminate nodes, and intermediate nodes. Honeypots can only be deployed at the location of the intermediate node.

3.2.2. The Action Space of the Attackers and the Defender.

The attack action is defined as the behavior pattern of the attacker, which can be expressed as $A = \{A_1, A_2, \dots, A_{ka}\}$. When an attacker enters the system, it will use the compromised node as a springboard to continue to infiltrate the system. Different behavior patterns will produce different preferences and affect the attacker's probability $p(n_i)$ of attacking node n_i ; $p(n_i)$ is related to the benefits $U(v)$ brought by the vulnerability v and the probability of a successful attack $J(v)$. Drawing on the evaluation system of CVSS, we believe that the value of the vulnerability positively correlated with its confidentiality (CI), integrity (II), and availability (AI). $U(v)$ can be calculated as

$$U(v) = \omega_1 \cdot CI + \omega_2 \cdot II + \omega_3 \cdot AI. \quad (1)$$

The probability of a successful attack $J(v)$ is inversely proportional to the complexity of the vulnerability (AC). The probability of the attacker choosing the next attack node is

$$n_{\text{attack}} = \arg \max(p(n_i)) = \arg \max(f(U_{n_i}(v), J_{n_i}(v))). \quad (2)$$

The defense action set is $D = \{D_1, D_2, \dots, D_{kd}\}$. Defense action is defined as a defense model for a specific attack behavior pattern. If the defender selects action D_i , the honeypot node will be calculated as

$$n_{\text{honeypot}} = \arg \max(D_i(n_1, n_2, \dots, n_n)). \quad (3)$$

FIGURE 2: Attack graphs. (a) Attribute attack graph. (b) Bayesian attribute attack graph. (c) Bayesian state attack graph.

It should be noted that kd and ka are not necessarily equal; that is, the number of strategies on both sides of the attack and defense is not necessarily equal, and the attackers may have an attack method unknown to the defender. In addition, as the number of attack rounds increases, MD2VH may learn new defense models. Therefore, during the whole offensive and defensive process, the action model pool of the defender may change; that is, kd is a variable. In the same attack and defense stage, we define n_{target} as the target node of the attacker; then, S^t can be calculated as

$$S^t = \begin{cases} S_{\text{final}}^{\text{success}}, & \text{if } n_{\text{attack}} = n_{\text{honeypot}}, \\ S_{\text{final}}^{\text{fail}}, & \text{if } n_{\text{attack}} \neq n_{\text{honeypot}}. \end{cases} \quad (4)$$

3.2.3. The Attack and Defense Strategies. The attack and defense strategies are the rule for both offensive and defensive parties to choose actions. In the state of S^t , the rules for the attacker or the defender to take actions are called the offensive and defensive strategies in this state. This strategy is a mixed strategy, which means that the probability distribution of each different action is taken by the attacker and the defender in S^t :

$$\pi_a(S^t) = (\sigma_{a1}(S^t, A_1^t), \sigma_{a2}(S^t, A_2^t), \dots, \sigma_{an}(S^t, A_{ka}^t)), \quad (5)$$

where $\sigma_{ai}(S^t, A_i^t)$ represents the probability of the attacker choosing action A_i in S^t ; the same is

$$\pi_d(S^t) = (\sigma_{d1}(S^t, D_1^t), \sigma_{d2}(S^t, D_2^t), \dots, \sigma_{dn}(S^t, D_{kd}^t)). \quad (6)$$

The main purpose of the defender is to learn the attack strategy of all attackers in the system, which is

$$\pi_d(S^t) \longrightarrow \pi_a(S^t) = (\sigma_{a1}(S^t, A_1^t), \sigma_{a2}(S^t, A_2^t), \dots, \sigma_{an}(S^t, A_{ka}^t)). \quad (7)$$

Then, we calculate the best position for honeypots where the attacker is most likely to attack:

$$n_{\text{honeypot}} = \arg \max (D_i(n_1, n_2, \dots, n_n)) \longrightarrow n_{\text{attack}}. \quad (8)$$

3.2.4. Quantitative Strategy Benefits

Definition 2. Let DG denote the profit obtained by the defender after the end of a single round of attack. The profit is related to the value of the attacker ADV . Therefore, DG can be calculated as

$$DG = \begin{cases} \beta \cdot ADV, & \text{if } S^{t+1} = S_{\text{final}}^{\text{success}}, \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

Definition 3. The loss L of the defender after being attacked denotes the loss suffered by the defender when the attacker attacks the normal node. m represents the number of compromised nodes, and l_i represents the loss of the defender after the attacker successfully attacks node i . L can be calculated as

$$L = \sum_{i=1}^m l_i. \quad (10)$$

Definition 4. Let r_d denote the total reward obtained by the defender after the end of the process. According to the previous paper, it can be calculated as

$$r_d = DG - L. \quad (11)$$

Definition 5. Let AG denote the profit of the attacker after successfully attacking the real node. Let AG^t represent the profit of the attacker at stage t . Let v_i indicate that there is a vulnerability v on node i . AG^t is related to the gains $G(v_i)$ from compromised nodes i and the probability of successful attack $J(v)$. If the attacker chooses to attack node i at stage t , the profit is

$$AG^t = G(v_i) \cdot J(v_i). \quad (12)$$

Definition 6. The attacker cost AC^t refers to the cost of the attackers' attack in stage t , and the resources consumed by different means of attack are different.

Definition 7. The attacker's reward R_a refers to the total reward obtained by the attacker after the end of the attack and defense. According to the previous paper, it can be calculated as

$$r_a = \sum_{t=1}^{\text{final}} (AG^t - AC^t). \quad (13)$$

Figure 3 is a tree based on the offensive and defensive process. In the state S^{t-1} , the defender will select an action D_i with a certain probability according to its own strategy. The attacker does not know the defender, it will choose a certain action A_i according to personal preference. After the actions of both sides, both offensive and defensive parties will get their own staged rewards and then enter the next state.

3.3. Algorithm. The solution MD2VH is mainly divided into two parts: defense model prediction and deployment strategy generation. The main purpose of the defense model prediction is to use the BP neural network to learn the defense models against attackers with different preferences based on the existing attack information. These defense models will be used in the defense model prediction as the action space of the defender. The deployment strategy generation part mainly uses the DQN algorithm framework. The main purpose of this part is to predict the overall strategy of the attackers based on their behavior pattern collected in the current system and to predict the behavior pattern of this round of attackers according to the current alarm and at the same time makes adaptive adjustments to the defense models. The flowchart of the algorithm is shown in Figure 4.

The neural network learning-based model prediction part is an offline learning process. This part of the work is performed before the offensive and the defensive process starts. The training dataset of the neural network is the attack information collected in other systems. The learning algorithm will first classify the attack information and then generate the corresponding defense models, which form the action space D of the defender.

The deployment strategy prediction part combines the online learning ideas with the DQN framework. The action space D of the defender will be used as the input of the strategic prediction algorithm. To simplify our discussion, we suppose the attack strategies of all the attackers form an attack strategy pool, and the defender will learn the corresponding defense strategy π_d according to the specific attack strategy. In order to enhance the adaptability of the mechanism, in the strategy prediction part, the collected attack information will be transmitted to the BP neural network. After the BP neural network processes the information, a new defense model pool is obtained, and D is updated synchronously. The main process of the deployment strategy prediction part is shown in Algorithm 1.

4. Experiments

4.1. Implementation. In order to verify the performance of MD2VH, this paper implements a corresponding prototype

system based on SDN and container technology and establishes a network topology similar to Poolsappasit et al. [31]. As shown in Figure 5, the proxy server is the server of the decision-making layer and is mainly responsible for traffic monitoring, deployment decision, and flow table item management. The flow table of the SDN switch mainly forwards traffic according to the commands issued by the proxy server. Different servers on the internal network will use containers to build different services and honeypots corresponding to vulnerabilities. The container will be established based on the images stored on each server, and these images can be added and changed as needed to improve the scalability of the system.

In addition, when the proxy server transmits the honeypot deployment commands to a specific node, in order to solve the problem of inconsistent communication format and insufficient standardization caused by the complex interface requirements of Socket communication, the system uses the SOAP protocol of Web Service to write multiple API standardized interface. They are used to transfer policy files in XML format between the proxy server and the node. The combination of SOAP and XML language forms a loosely coupled communication architecture, which solves the heterogeneous problems between systems and provides a good foundation for the scalability of the overall system.

The proxy server uses the Centos system, the CPU is 4 cores and 4 threads, i5-3470, 3.2 GHz, the memory is 4G, and the hard disk is 512G. The rest of the servers use the Centos system, the CPU is 10 cores and 20 threads, i9-10900X, 3.7 GHz, the memory is 64G, and the hard disk is 1T. The honeypots and the containers of the real system are, respectively, deployed on two servers inside the trapping environment, and the SDN switch adopts the Pica8 P3297 switch.

The vulnerabilities corresponding table is shown in Table 1 [31]. The Bayesian state attack diagram established based on the table is shown in Figure 6.

4.2. Results and Discussion. The experiment abstracts all attack actions into an attacker's strategic behavior. The attacker will choose different attack paths with different probabilities based on several different preferences. In an attack round, the attacker's preferences are fixed. The defender will make assumptions about the attacker's preferences based on the attack path obtained in real time and select the corresponding defense model to deploy honeypots based on the assumptions made.

This paper selects multiple types of attackers and sets them to be inclined to carry out SSH attacks, SMTP attacks, and BOF attacks, respectively. In each attack mode, the attacker will choose different attack paths according to different preferences and choose the corresponding attack method to attack according to the type of vulnerabilities encountered in the chosen attack path. Their attack preferences are achieved by setting different weight parameters. Attackers will choose different vulnerabilities in the attack path to exploit based on their own characteristics, and their ultimate target is the Administrative Server.

FIGURE 3: The offensive and defensive process tree.

FIGURE 4: The flowchart of MD2VH.

Firstly, the learning ability of the system is evaluated. The evaluation is mainly divided into two parts. In the model prediction part, the BP neural network is used to learn the preferences of the three aggressive behavior patterns; as shown in Figure 7(a), after 300 rounds of learning, the loss

curve has been basically flat, which verified the learning ability of the defense model. In the deployment strategy prediction stage, three attack modes were also used, and the selection probabilities of the three attack modes were set to 0.7, 0.2, and 0.1, respectively. After 200 rounds of learning,

Input: attack graph model and the defender's action D
Output: D^t and model π_d

- (1) Initialize attack graph model G
- (2) Initialize the attacker's strategy model π_a and action A , and the defender's action D
- (3) Initialize replay Memory A to capacity C
- (4) Initialize DQN parameters $\#Q$ (random weights ω), target Q^* (weights $\omega^- = \omega$) and replay Memory K to capacity N
- (5) For episode = 1, M :
- (6) $S = S^0$
- (7) $D^0 = \text{argmax}_D Q(S^0, D)$
- (8) While S^t is not S_{final} :
- (9) Get the alarm
- (10) Output: $D^t = \varepsilon - \text{greedy}(D)$ or $D^t = \text{argmax}_D Q(S^t, D)$
- (11) $r_{td} = r_d(S^t, D^t)$
- (12) $S = S^{t+1}$
- (13) Store transition $(S^t, D^t, r_{td}, S^{t+1})$ in K
- (14) Sample random minibatch of transitions (S^i, D^i, r_i, S^{i+1}) from K
- (15) Set $y_i: y_i = \begin{cases} r_i, & S^{i+1} \text{ is } S_{\text{final}}, \\ r_i + \gamma \cdot \max_{D'} Q(S^i, D'; \omega^-), & S^{i+1} \text{ is not } S_{\text{final}}. \end{cases}$
- (16) Perform a gradient descent step on $(y_i - Q(S^i, D^i; \omega))^2$ with respect to the network parameters ω ;
- (17) Every C step reset $Q^* = Q$
- (18) End while
- (19) Store attack information in A After the attack information is full, it will be sent to the BP neural network to update models
- (20) If Memory $A = C$: Memory $A \rightarrow \text{BP Network}$
- (21) Update the defender's action D from BP Network
- (22) End for
- (23) Output: $\pi_d = Q^*$

ALGORITHM 1: Deployment strategy prediction algorithm.

FIGURE 5: Prototype system topology.

TABLE 1: Vulnerability exploitation relationship table.

Host	Vulnerability	CVE#
Local desktops	Remote login	CVE-2020-7868
	Buffer overflow (BOF)	CVE-2021-38166
Admin machine	Stack BOF	CVE-2021-38207
	Stack BOF	CVE-2021-33909
Gateway server	Information leakage	CVE-2021-20485
	Heap-based buffer overflow	CVE-2016-0778
SQL server	Failed cookie generation	CVE-2016-1908
	SQL injection	CVE-2020-18746
Mail server	Remote code execution	CVE-2020-28020
	Buffer underwrite	CVE-2020-28024
DNS server	Information leakage	CVE-2020-28025
	DNS cache poisoning	CVE-2020-25685
Web server	Execute code with the privileges of the parent process	CVE-2019-0211

FIGURE 6: Bayesian state attack diagram.

the final convergence result is shown in Figure 7(b). It can be seen that the final strategy basically converges to the attack strategy setting.

In order to further evaluate the dynamic learning ability of the strategy model, the distribution of the attack strategy is adjusted to 0.2, 0.7, and 0.1, and the second learning is performed on the basis of the model after the first learning. It can be seen from Figure 7(c) that after 200 rounds of learning, the final policy distribution results tend to be 0.1, 0.8, and 0.1. This is because, in some cases, the honeypot deployment nodes selected by different models are the same, which leads to a relatively larger probability of model 2. But the final result is not much different from the initial set value, which can prove the dynamic learning ability of the strategy.

Next, the capturing ability of the deployment strategy was evaluated and compared with the other four strategies: static deployment, dynamic deployment following alarms, the Q_Learning method proposed in the literature [29], and the N2-DQN method proposed in the literature [24]. The experiment set up a total of 3 situations (0.7/0.2/0.1, 0.2/0.7/0.1, and 0.3/0.3/0.4), and 200 attacks were carried out in each situation. The number of honeypots of the static deployment strategy is set to 4. The dynamic deployment following the alarm is to randomly select one for deployment after the compromised node while receiving the alarm signal, so it is also called random deployment, and the random deployment prediction will deploy honeypots before the attack starts. According to the Q_Learning method, learning is only performed in the deployment strategy prediction stage and

FIGURE 7: Evaluation of learning ability. (a) Model prediction learning loss. (b) Learning for the first time. (c) Learning for the second time.

adapts N2-DQN's scene to the scene of this paper. Finally, since the end point of the attack graph is unique in the scenario set in this paper, the penultimate stage of the attack is regarded as the final stage. Table 2 shows the capturing rate in the case of 3 attack modes.

Figure 8(a) shows the capture rate statistics of the four algorithms in different situations. It can be seen that MD2VH has the best result, and the capture rate is between 89% and 94%, and it is still 89% when the attack strategy is relatively uniform. The method of Q_Learning has a better result when the distribution probability is quite different. It is basically maintained at about 74%, which is about 20% lower than MD2VH. However, it is difficult for Q_Learning to learn when the distribution probability is relatively even, whose capture rate is greatly reduced, which is about 40% lower than MD2VH. N2-DQN is better than the Q-Learning

method, but the results of the two are not much different. This shows that the learning ability of the strategy prediction algorithm can be significantly improved after the strategy learning is divided into two parts for learning. And the random deployment and static deployment are hardly affected by attack strategies and are basically around 35%.

Figure 8(b) shows the cumulative distributions of the attack capture rate in different stages while the probability distribution of the attack model is (0.7/0.2/0.1). The higher the capture rate at the same stage, the faster the attacker was captured and the higher the effectiveness of the strategy. The X-axis represents the attack phase, and the Y-axis is the cumulative capture rate. Because MD2VH will be pre-deployed according to the current model, and the model becomes more suitable as the number of online learning rounds increases, so about 50% of the attackers can be

TABLE 2: The capturing rate in the case of 3 attack modes.

	Static	Random	N2-DQN	Q-learning	MD2VH
0.7/0.2/0.1	0.375	0.32	0.75	0.73	0.945
0.2/0.7/0.1	0.365	0.35	0.73	0.71	0.956
0.3/0.3/0.4	0.35	0.34	0.51	0.42	0.895

FIGURE 8: Evaluation of the capture capacity. (a) Probability of attacks being captured. (b) The cumulative distribution function.

captured by the predeployed honeypots. It can be seen from the figure that MD2VH has captured 80% of the attackers in the first stage, which indicates that the capture speed of MD2VH is the fastest among several strategies.

On the basis of the previous real system, this paper uses virtual nodes to expand the network topology and uses containers to simulate 150 vulnerable nodes. This attack graph can carry out up to 9 stages of attack and defense and expands the attackers' behavior patterns to 5. There are three attack strategies (0.2/0.2/0.2/0.2/0.2, 0.1/0.1/0.6/0.1/0.1, and 0.05/0.05/0.3/0.3/0.3), and 5000 rounds of attacks are carried out in each case. A situation where the unknown attacker mode is added is as follows. In this scenario, the defender only knows the corresponding defense models of the first three attack modes before the attack and defense begin and gradually learns the complete defense models as the number of attack rounds increases. Table 3 shows the capturing rate in the case of 5 different attack modes. This situation is represented by UMA2VH, and the final result is shown in Figure 8.

In Figure 9(a), it can be seen that when the attack graph becomes more complex, the capture rate of the Q_Learning algorithm appears to be less than 40%, while the capture rate of MD2VH is still around 94%, which is even higher than that of the simple network topology. The main reason is that as the number of attack rounds increases, the strategy learned is more accurate. Even if the uncertainty of the attackers' behaviors is increased, the

capture rate of UMA2VH is maintained at about 80%. Compared with the results in Q_Learning, it has increased by more than 36%~61%. Compared with the results in N2-DQN, it has increased by more than 30%~55%. These results reflect the adaptability and capture ability of MD2VH, while the capture rates of the random strategy are lower than 25%, and the static strategy is lower than 7%. It can be seen that the nonintelligent deployment strategies perform poorly in the complicated network topology.

Figures 9(b)~9(d), respectively, show the cumulative distributions of the attack capture rate of the three strategies. It can be seen that the MD2VH's curve is relatively smooth as a whole, which reflects its good learning ability. After increasing the uncertainty of the attacker's behavior, the gradient of UMA2VH's curve increases slightly, which indicates that after adding an unknown attack model, MD2VH needs some learning stages to correctly predict the attack behavior. In the three cases, the curves of Q_Learning and N2-DQN have a sudden change, which indicates that only after a certain learning stage, the simple intelligent deployment strategy can make a more accurate prediction of the current attack behavior, so it is more sensitive to the environmental influences, and its adaptability is not strong. On the whole, regardless of whether the uncertainty of the attacker is increased, the cumulative capture rate of MD2VH at any stage is the highest, which fully illustrates its trapping effect and learning ability.

TABLE 3: The capturing rate in the case of 5 attack modes.

	Static	Random	N2-DQN	Q-learning	MD2VH	UMD2VH
0.2/0.2/0.2/0.2/0.2	0.0496	0.248	0.5172	0.4172	0.9442	0.7728
0.1/0.1/0.6/0.1/0.1	0.0644	0.2426	0.645	0.5842	0.941	0.8216
0.05/0.05/0.3/0.3/0.3E	0.0338	0.1486	0.3952	0.3356	0.9438	0.852

FIGURE 9: Simulation of trapping capacity evaluation. (a) Probability of attacks being captured. (b) The cumulative distribution function in case 1. (c) The cumulative distribution function in case 2. (d) The cumulative distribution function in case 3.

5. Conclusion

As a means of deception defense, honeypots can effectively increase the difficulty of attackers and enhance the protection capability of the network assets. However, the focus of current researches is mainly on how to improve the fidelity and interaction capability of the honeypots, and there

is a lack of in-depth analysis on the honeypot deployment strategies. In fact, the deployment strategy will directly affect the protection capability of the honeypot system. In this regard, based on the Bayesian state attack graph, this paper proposed an intelligent honeypot dynamic deployment mechanism MD2VH, established an intelligent decision-making deployment model based on the multiphase attack

and defense process, and implemented the optimal deployment strategy based on the BP neural network and DQN algorithm. In addition, in order to enhance the flexibility of the honeypot deployment, a prototype system model based on SDN and virtualized containers is also implemented to support the efficient adjustment of deployment scenarios. The experimental results show that MD2VH can form a dynamic deployment strategy on the basis of continuously learning the behavior of attackers, improve the capture rate of the attackers, and reduce the deployment costs.

In the future studies, we will consider the interaction process inside the honeypot after the attacker is captured, so as to increase the deception efficiency further, and consider how to use intelligent algorithms to further improve the deception capability inside the honeypot system. Besides, we also consider introducing the ideas such as MTD (Moving Target Defense) to the honeypot deployment process, so as to increase the deception capability of the system further.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this study.

Acknowledgments

This work was supported by the National Natural Science Foundations of China (62172432 and 61772271) and the Natural Science Foundation of Jiangsu Province (SBK2020043435).

References

- [1] C. Stoll, *The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage*, pp. 1–3, Doubleday, New York, US, 1989.
- [2] A. B. Sarr, A. H. Anwar, C. Kamhoua, N. Leslie, and J. Acosta, "Software diversity for cyber deception," in *Proceedings of the IEEE Global Communications Conference*, pp. 1–6, Taipei, Taiwan, 11 December 2020.
- [3] 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, 23 pages, Article ID e0189095, 2017.
- [4] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing Atari with deep reinforcement learning," in *Proceedings of the NIPS Deep Learning Workshop*, December 2013.
- [5] B. Rashidi, C. Fung, K. W. Hamlen, and A. Kaminski, "HoneyV: a virtualized honeynet system based on network softwarization," in *Proceedings of the NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–5, Taipei Taiwan, 27 April 2018.
- [6] A. Hirata, D. Miyamoto, M. Nakayama, and H. Esaki, "Intercept+: SDN support for live migration-based honeypots," in *Proceedings of the 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pp. 16–24, Lisbon, Portugal, 19 March 2014.
- [7] N. Mckeown, T. Anderson, H. Balakrishnan et al., "OpenFlow," *ACM SIGCOMM - Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
- [8] S. Soltesz, H. Pötl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization," *ACM SIGOPS - Operating Systems Review*, vol. 41, no. 3, pp. 275–287, 2007.
- [9] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, 2014.
- [10] F. D. Gaspari, S. Jajodia, L. V. Mancini, and A. Panico, "Ahead: a new architecture for active defense," in *Proceedings of the ACM Workshop on Automated Decision Making for Active Cyber Defense*, Vienna, Austria, October 2016.
- [11] J. Sun, S. Liu, and K. Sun, "A scalable high fidelity decoy framework against sophisticated cyber attacks," in *Proceedings of the 6th ACM Workshop on Moving Target Defense*, pp. 37–46, London, UK, November 2019.
- [12] S. Kyung, W. Han, N. Tiwari et al., "Honeyproxy: Design and Implementation of Next-Generation Honeynet via SDN," in *Proceedings of the IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, Las Vegas, NV, USA, Oct 6 2021.
- [13] F. Araujo, K. W. Hamlen, S. Biedermann, and S. Katzenbeisser, "From patches to honey-patches: lightweight attacker misdirection, deception, and disinformation," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pp. 942–953, Scottsdale, Arizona, USA, 03 November 2014.
- [14] E. Al-Shaer, J. Wei, K. W. Hamlen, and C. Wang, "Correction to: autonomous cyber deception," *Autonomous Cyber Deception*, pp. C1–C2, 2019.
- [15] W. Stout and V. Urias, *Hades: High-Fidelity Adaptive Deception & Emulation System*, Sandia National Lab, Albuquerque, NM, US, 2018.
- [16] W. Stout, V. Urias, C. Loverro, and B. Anthony, *Now You See Me Now You Don't: Advancing Network Defense through Network Deception*, Sandia National Lab, Albuquerque, NM, US, 2017.
- [17] V. E. Urias, W. Stout, and H. W. Lin, "Gathering threat intelligence through computer network deception," in *Proceedings of the IEEE Symposium on Technologies for Homeland Security (HST)*, pp. 1–6, Waltham, MA, USA, 11 May 2016.
- [18] W. Fan, Z. Du, M. Smith-creasey, and D. Fernandez, "Honeydoc: an efficient honeypot architecture enabling all-round design," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 683–697, 2019.
- [19] W. Fan and D. Fernández, "A novel sdn based stealthy tcp connection handover mechanism for hybrid honeypot systems," in *Proceedings of the IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–9, Bologna, Italy, 3 July 2017.
- [20] N. El Kamel, M. Eddabbah, Y. Lmoumen, and R. Touahni, "A smart agent design for cyber security based on honeypot and machine learning," *Security and Communication Networks*, vol. 2020, Article ID 8865474, 9 pages, 2020.
- [21] C. Huang, J. Han, X. Zhang, and J. Liu, "Automatic identification of honeypot server using machine learning techniques," *Security and Communication Networks*, vol. 2019, Article ID 2627608, 8 pages, 2019.
- [22] L. Huang and Q. Zhu, "Adaptive honeypot engagement through reinforcement learning of semi-Markov decision processes," in *International Conference on Decision and Game Theory for Security*, pp. 196–216, Springer, Sweden, 2019.

- [23] K. S. Gill, S. Saxena, and A. Sharma, "Gtm-csec: game theoretic model for cloud security based on ids and honeypot," *Computers & Security*, vol. 92, Article ID 101732, 2020.
- [24] S.-C. Zhou, J.-J. Liu, X.-F. Zhong, and C.-J. Lu, "Intelligent penetration testing path discovery based on deep reinforcement learning," *Computer Science*, vol. 48, no. 7, pp. 40–46, 2021.
- [25] A. Bohara, M. A. Nouredine, A. Fawaz, and W. H. Sanders, "An unsupervised multi-detector approach for identifying malicious lateral movement," in *Proceedings of the IEEE 36th Symposium on Reliable Distributed Systems*, pp. 224–233, Hong Kong, September 2017.
- [26] H. Takabi and J. H. Jafarian, "Insider threat mitigation using moving target defense and deception," in *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*, pp. 93–96, Dallas, TX, USA, October 2017.
- [27] M. A. R. A. Amin, S. Shetty, L. Njilla, D. K. Tosh, and C. Kamhoua, "Hidden Markov model and cyber deception for the prevention of adversarial lateral movement," *IEEE Access*, vol. 9, pp. 49662–49682, 2021.
- [28] H. Zhang, J. Yang, and C. Zhang, "Defense decision-making method based on incomplete information stochastic game and Q-learning," *Journal on Communications*, vol. 39, no. 8, pp. 56–68, 2018.
- [29] S. Wang, Q. Pei, J. Wang, G. Tang, Y. Zhang, and X. Liu, "An intelligent deployment policy for deception resources based on reinforcement learning," *IEEE Access*, vol. 8, pp. 35792–35804, 2020.
- [30] Z. Ye, Y. Guo, C. Wang, and A. K. Ju, "Survey on application of attack graph technology," *Journal on Communications*, vol. 38, no. 11, pp. 121–132, 2017.
- [31] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using Bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 34, no. 9, pp. 61–74, 2011.