electronics

Article

An Optimal Active Defensive Security Framework for the Container-Based Cloud with Deep Reinforcement Learning

Yuanbo Li [1,2,*], Hongchao Hu [1], Wenyan Liu [1,3] and Xiaohan Yang [1]

1 National Digital Switching System Engineering and Technological Research Center, PLA Strategic Support Force Information Engineering University, Zhengzhou 450002, China
2 School of Computer and Information Engineering, Luoyang Institute of Science and Technology, Luoyang 471023, China
3 Purple Mountain Laboratories, Nanjing 211111, China
* Correspondence: 200900501775@lit.edu.cn

Abstract: Due to the complexity of attack scenarios in the container-based cloud environment and the continuous changes in the state of microservices, the effectiveness of active defense strategies decreases with the cloud environment and microservice change. To tackle it, the main focus is how to establish a comprehensive threat model and adaptive active defense deployment strategy. In this study, we present an optimal active defensive security framework (OADSF) for a container-based cloud with deep reinforcement learning. Firstly, based on the characteristics of container clouds and microservices, the security threats and attack paths of attackers are analyzed from the application layer and container layer. Then, we propose a Holistic System Attack Graph to quantitatively analyze the security gain, quality of service (QOS) and defense efficiency in the container-based cloud scenarios. Finally, the optimization of a moving target defense (MTD) strategy is modeled as a Markov decision process. Deep reinforcement learning is proposed to handle the state space explosion under large-scale cloud applications, thus solving the optimal defense configuration strategy for the orchestration platform. We use Kubernetes to build container-based clusters. The algorithm is implemented in Python 3.7 based on Tensorflow 1.14. Simulation results show that the proposed method can quickly converge under large-scale cloud applications and increase defensive efficiency. Compared with DSEOM and SmartSCR, the defense efficiency is increased by 35.19% and 12.09%, respectively.

Keywords: microservice; OADSF; MTD; Markov decision process; deep reinforcement learning

check for updates

Citation: Li, Y.; Hu, H.; Liu, W.; Yang, X. An Optimal Active Defensive Security Framework for the Container-Based Cloud with Deep Reinforcement Learning. *Electronics* **2023**, *12*, 1598. https://doi.org/10.3390/electronics12071598

Academic Editor: Andrei Kelarev

Received: 28 February 2023
Revised: 24 March 2023
Accepted: 25 March 2023
Published: 29 March 2023

1. Introduction

In recent years, container technology has been widely used in the cloud environment. As a lightweight alternative to virtual machines in traditional cloud infrastructure, containers have shorter startup time and lower virtualization overhead [1]. The lightweight features brought by container technology provide an ideal running environment for microservices. In the microservice architecture, applications are decoupled into a group of independent microservices, which can help service providers simplify the service update and scheduling process. At the same time, the microservice architecture has better modularity, enabling application developers to take advantage of services from other providers. Microservice applications in the container-based cloud environment can make full use of the characteristics of cloud computing, such as elasticity, agility and resource pooling, accelerate the development and iteration process of applications and improve the scalability of applications [2,3].

Container technology and microservice architecture have changed the deployment and operation mode of cloud applications, but have also brought new security threats. The splitting of microservices makes the interactive interfaces grow explosively, which makes it difficult to control the attack surface of microservices. For example, the widely used logging framework log4j in Spring Cloud has a remote code execution vulnerability (CVE-2021-44228), which has a significant impact on various industries. Virtualization technology enables multiple containers on the same host to share the operating system kernel, which provides convenience for attackers to carry out lateral attacks in clusters [4]. For example, the CVE-2019-5736 vulnerability can be used to attack the run module in the container to achieve container escape. In traditional network security strategies, protection schemes based on boundary deployment are mainly used, such as firewall and intrusion detection [5]. However, in the container-based cloud environment, the boundary of traditional application software is gradually blurred, and it is difficult to determine the deployment location of protective equipment. Therefore, the traditional protection model cannot fully cope with the security threats in the container-based cloud environment [6].

Moving target defense is a typical active defense mechanism, which constantly changes the attack surface of IT systems to increase the uncertainty and complexity of attacks [7]. MTD technology includes the dynamic execution environment [8], dynamic software implementation [9], dynamic network topology [10], etc. Although MTD technology effectively increases the difficulty of attackers to penetrate the system, it brings new problems. On the one hand, in the MTD defense mechanism, defense effectiveness is particularly prominent because it highly depends on the state of defense target. If the defense strategy does not match the assumed conditions, the active defense mechanism will not work well. On the other hand, although MTD technology can improve the security of cloud services, its impact on service performance cannot be ignored. For example, network-level MTD technology will directly lead to the decline of web service quality and affect the communication between services [11], which may indirectly lead to the performance decline of cloud services. Therefore, the research contents focus on how to optimize the use of active defensive capability, and how to determine the appropriate MTD strategy to balance the performance cost and effectiveness to further improve the efficiency of MTD technology.

In view of the above problems, more and more researchers have begun to study the MTD security strategy for the container-based cloud. In this research, Bardas et al. [6] proposed a platform based on cloud computing, which can capture service dependency in the cloud environment and find the best strategy for service instance replacement to maximize the difficulty of attack. Connell et al. [12] introduced a quantitative analysis model to evaluate the resource availability and MTD performance. However, the above MTD strategies were only considered to solve the optimal defense configuration in a static environment. In order to optimize the active defense capability under the dynamic environment, Jin et al. [13] proposes an MTD strategy based on dynamic security assessment and configuration optimization for the container-based cloud. By dynamically evaluating the critical nodes in the container-based cloud, the strategy can dynamically adjust the protected target. Because the strategy only protects critical nodes, it is difficult to defend against the attack scenario where attackers bypass critical nodes in the microservice architecture. To solve this problem, Zhang et al. [14] proposed the SmartSCR framework to realize the dynamic protection of nodes. However, these strategies only considered the security after MTD deployment, and ignore how to quantify the security gain.

Based on the above analysis, the optimal defense efficiency of the container-based cloud should consider both the comprehensive threat model of the target environment and the adaptive active defense configuration. Unfortunately, the existing optimal active defense security framework cannot solve both problems jointly. For threat models, existing research mainly uses intranet attack graphs to illustrate attack behavior. For the deployment of active defense strategies, existing research mainly focuses on dynamic deployment of important nodes. The limitations of existing methods are as follows: (1) they describe the risks and vulnerabilities of the intranet, while ignoring the specific threats and attack difficulties brought by virtualization technology, and (2) existing methods do not address the constant updating of threats in the container-based cloud. Therefore, we propose an optimal active defensive security framework (OADSF) for a container-based cloud with deep reinforcement learning to solve this problem. Deep reinforcement learning is a learning paradigm that solves the problem of maximizing rewards through learning strategies during the interaction between agents and the environment.

The main contributions of this paper are summarized as follows.

(1) In order to comprehensively analyze the complex attack scenarios in microservices, a Holistic System Attack Graph (HSAG) model is established, and the security gain and defensive efficiency of MTD strategies are described based on the HSAG model.
(2) In order to optimize the defense efficiency, we propose an OADSF. It can dynamically adjust defense configurations by sensing the changes in the microservice state.
(3) We propose an adaptive security configuration algorithm based on Prioritized Dueling Double DQN (P3DQN). It can optimize the defense configuration in real time with the state of the microservice application changes, thereby improving the defense efficiency of the system.

The rest of this paper is presented as follows. Section 2 gives the related work. Section 3 introduces the threat model of microservice in container-based cloud. The problem and related definitions are described in Section 4. Section 5 shows the details of OADSF. Section 6 is the experimental section. Section 7 summarises of the work.

2. Related Work

According to the different defense objects and technology, we introduce the related work from four aspects: container security, microservice security, active defense technology and deep reinforcement learning.

2.1. Container Security

Container security has always been an important research field because container technology has been widely used by mainstream cloud platforms. Belair et al. [15] reviewed the research on using Linux kernel features to improve container security performance. Lopes et al. [16] proposed an automatic generation mechanism for container configuration to solve the complex configuration using the Linux process mechanism. Lin et al. [17] analyzed 11 attacks that can realize container escape, and abstracted a general attack model for container escape. Flora et al. [18] proposed a method of deploying the IDS system for anomaly detection to model abnormal behavior of containers. Lim et al. [19] performed an anomaly detection by collecting container logs. Considering that the short container life cycle leads to fewer training samples, Lin et al. [20] proposed an anomaly detection algorithm that does not rely on a large number of samples. Jin et al. [13] proposed a dynamic defense strategy for key nodes to solve the high dynamics of a container-based cloud that may introduce new security threats and lead to the failure of the defense strategy. This strategy computes the critical nodes through betweenness centrality and only protects the critical nodes.

2.2. Microservice Security

According to the problem that microservice splitting leads to the expansion of the attack surface, encryption, authentication and access control mechanisms are widely used in microservices. Almeida et al. [21,22] investigated microservice security and built a security defense model. The paper divides microservice security into four levels: physical environment security, cloud infrastructure security, network security and application layer security. By comparison, Pereira et al. [23] found that most of the literature focuses on the threat disposal of microservices, including the prevention and mitigation of attacks, but there are few studies on the modeling and analysis of microservices. Xu et al. [24] implemented an API gateway based on the project Kong. The gateway uses a web authentication mechanism to authenticate the user's identity and improve the security of the microservice. Sankaran et al. [25] introduced a permission control mechanism suitable for serverless workflow, and verified the effectiveness and performance in a real workflow scenario. Torkura et al. [26] proposed the concept of security control based on the vulnerability to solve the problem of insufficient security testing caused by frequent updates and iterations of microservices, and ensured the security of the microservice by integrating continuous security assessment. Bardas et al. [6] proposed a dynamic cleaning strategy mechanism based on the attack window model for microservice application systems. Their strategy assumes that the dynamic cycle of all microservices is the same, which greatly reduces the computation, and the dynamic cycle can be obtained through traversal.

2.3. Active Defense Technology

Ahmed et al. [27] designed a Byzantine fault-tolerant system of cloud computing based on the idea of redundancy. The system makes use of the characteristics of virtualization and multiple replicas to provide services, which makes sure the system can meet the Byzantine fault-tolerant conditions. Li et al. [28] proposed a traffic distortion mechanism using redundancy in the information physical systems. Yu et al. [29] introduced Mimic Defense Technology in distributed storage architecture, and added data heterogeneous encoding and storage verification technology based on redundant data backup, which greatly improved the security and availability. Wu et al. built a 5G core network with endogenous security characteristics by using the mimic defense technology [30] and network functions virtualization. By making use of the dynamic and redundancy of NFV technology, they achieved the combination of mimic defense and a 5G core network, and effectively reduced the costs caused by mimic defense. Wang et al. [31,32] introduced mimic defense into the scientific workflow, and built a scientific workflow intrusion tolerance system. In this system, the scientific workflow sub-tasks are distributed to multiple virtual machines to improve the reliability. Then, the lag decision mechanism can effectively reduce the execution time of the scientific workflow while ensuring the security of the intermediate data.

2.4. Deep Reinforcement Learning

The complexity and dynamics of network attacks require the adaptability and scalability of protection mechanisms. The emergence of deep reinforcement learning has brought a new perspective for network security. It has a strong ability to solve complex and dynamic network defense, and is very promising in network security [33,34]. Xiao et al. [35] studied attack models in edge cloud computing systems. In this paper, reinforcement learning technology is used to provide a security solution for edge nodes to uninstall safely and resist interference attacks. Lopez et al. [36] studied the application of deep reinforcement learning in intrusion detection systems. The DRL algorithm can be used as a supervised method to classify marked intrusion data. SmartFCT [37] takes advantage of DRL to generate optimal traffic integration policies while ensuring the completion time of streams. It effectively reduces power consumption, but does not significantly reduce the QoS of the network. Zhang et al. [14] depicts the attack difficulty with the attack graph model. They propose the SmartSCR framework to realize the dynamic protection of microservices. The framework is implemented based on the MTD dynamic cleaning strategy.

3. Threat Model

In the container cloud environment, the single application is divided into multiple microservices and runs in the cloud computing cluster. Multiple microservices coordinate and cooperate with each other to achieve specific functions through a call chain. For each microservice, its running environment uses the lightweight virtualization technology to realize the isolation requirements. At the same time, each microservice will adjust its number of replicas to cope with its dynamic concurrent requests. However, splitting the application into the microservice mode also leads to an explosive growth of the attack surface. In Figure 1, we describe security threats from three aspects: attack targets, strategies and capabilities.

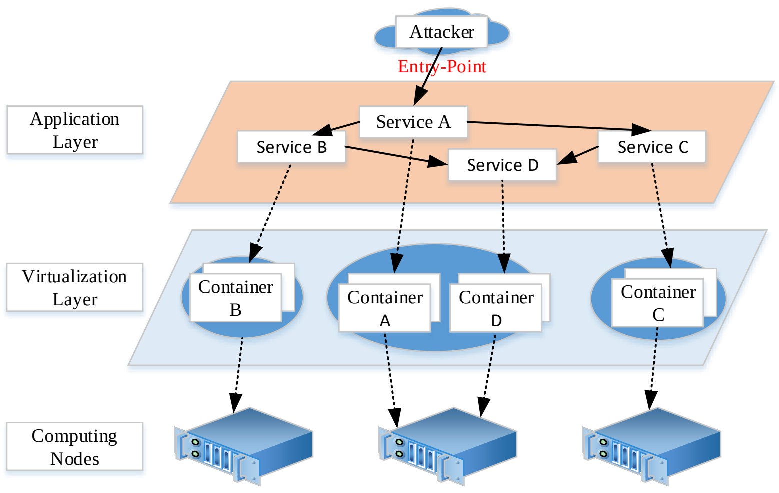

Figure 1. The architecture of a container-based cloud.

(1) Attack targets: In the container cloud environment, all microservices may become the target of attackers. For the i service, its attack surface can be composed of an application layer attack surface and a container layer attack surface, which can be expressed as $ST_i = \{S_i, C_i\}$.The application layer attack surface is composed of service code and dependent libraries, and the container layer attack surface refers to the container operation environment of the i service. As shown in Figure 1, the attack surfaces of microservices lies in the application layer and virtualization layer, respectively. In addition, we assume that some vulnerabilities in these targets can be exploited by attackers.

(2) Attack strategies: We assume that attackers adopt the cyber killing chain (CKC) model to carry out attacks. In this model, the attacker first performs various reconnaissance actions to identify the target's vulnerabilities. Then, the attacker selects an appropriate vulnerability and prepares the corresponding network attack tool against it. Next, attackers can use these tools to execute malicious code to compromise the target.

(3) Attackers' capabilities: We assume that the attacker is outside the cloud platform and attacks the microservices through the Internet. In general, microservice applications open specific service access portals. Therefore, attackers can attack only the application layer attack surface, such as service A. When attackers successfully hijack service A through the application layer vulnerability, attackers have the following two attack modes to expand their foothold in the container-based cloud.

- Application layer: After escaping from the application layer, attackers continue to search for network-reachable microservices and attack the application layer attack surface. It is assumed that the network configuration in the container-based cloud is subject to the network isolation policy on the management node [13]. Only when there is a dependency relationship between microservices, the networks are reachable. As Figure 1 shows, after hijacking service A, attackers can continue to attack services B and C.
- Virtualization layer: After escaping from the application layer, attackers can also attack the container hosting them. If container escape is successful, attackers can escape from the virtual environment to the worker node. Then, they can obtain the permission of the worker node where the container directly hijacks the services running in the container environment. As Figure 1 shows, attackers can directly enter container A after hijacking service A. If attackers successfully escape from container A, they can enter container D and hijack service D.

On the basis of the analysis, it can be seen that microservices and containers in the container-based cloud lead to the explosive growth of attack paths, and security management and control are very difficult. On the basis of MTD, we adopt the method of dynamic cleaning to actively defend against all microservices and increase the difficulty of attackers to escape. However, this strategy will also affect QOS. In order to balance the security gain and service performance and maximize the defense efficiency, it is necessary to dynamically optimize the cleaning cycle of different microservices. The main difficulty is how to quantitatively describe the security gain, the cost and defense efficiency under different cleaning cycles. At the same time, an application may contain a large number of microservices, and the state of microservices changes at any time, so the dynamic optimization method needs to have strong dynamics and scalability.

4. Problem Modeling

In this section, firstly, according to the characteristics of microservice penetration, we propose the HSAG model to describe the attacker's diversified attack paths and the attack difficulty of nodes. Then, based on the HSAG model, the security gain and QOS overhead are defined. Finally, the configuration optimization is transformed into a Markov Decision Process (MDP). MDP is a mathematical model of sequential decision, which is used to simulate the stochastic strategies and rewards that agents can achieve.

4.1. HSAG Model

On the basis of the analysis of the attacker's attack path in the cloud environment, we define HSAG as follows.

Definition 1. *HSAG is a directed graph $G = (N, E)$, where N is a set of nodes in the graph, including all attack surfaces of attackers (I) and microservices. Assuming that the application consists of M microservices, then $N = \{I, S_1, S_2, \ldots, S_M, C_1, C_2, \ldots, C_M\} \cdot \{S_1, S_2, \ldots, S_M\}$ represent the application layer attack surface and $\{C_1, C_2, \ldots, C_M\}$ represent the virtualization layer attack surface. $E \subseteq N \times N$ is the set of all edges in the graph, and each edge represents the attacker's attack path.*

Definition 2. *Given nodes $N_a, N_b \in N, a \neq b$, $E_i = (N_a, N_b)$ indicate that attackers can use the vulnerability of node N_b to expand its foothold to N_b , based on the hijacked node N_a . We define $D(E_i)$ as the weight of edge E_i , and $D(E_i)$ represents the difficulty of successfully attacking node N_b .*

In order to evaluate the difficulty of vulnerability exploitation, we first use the indicators defined in the Common Vulnerability Scoring System (CVSS) to quantify the attack difficulty of the target node. In the CVSS 3.1 specification, the defined metric consists of three parts: base metric, temporal metric and environmental metric. In the CVSS 3.1 specification and literature [14], the exploiting difficulty ED in the node is also estimated through Equation (1):

$$ED = (8.22 \times AV \times AC \times PR \times UI)^{-1}, \tag{1}$$

where 8.22 is a constant value and AV, AC, PR and UI respectively represent the attack vector, attack complexity, privilege requirement and user interaction. For the attack surface of each node, there may be multiple exploitable vulnerabilities. However, due to the unknown ability of the attacker, every vulnerability in the node may be attacked. To solve this problem, we use temporary metrics to evaluate the weight w of each vulnerability being attacked. Further, by the weighted average of the difficulty of vulnerability exploitation on all nodes, the attack difficulty $D(E_i)$ of the node can be expressed as Equation (2).

$$D(E_i) = \frac{\sum\limits_{v \in V} (w \times ED)}{\sum\limits_{v \in V} w} \tag{2}$$

Here, V represents a set of all vulnerabilities of the node and w represents the weight value of exploitable vulnerabilities.

After the dynamic cleaning strategy is deployed on the node, the attack process on the node may be interrupted due to the change of the attack surface. This makes attackers need to spend more money to compromise these nodes. In this case, the node attack difficulty $D'(E_i)$ can be expressed as Equation (3) after the MTD is deployed.

$$D'(E_i) = \frac{T_s}{T_i} \times D(E_i) = \frac{f(D(E_i))}{T_i} \times D(E_i) \tag{3}$$

Here, $T_s = f(D(E_i))$ represents the maximum time required for the node to be successfully attacked in a static environment, and function $f(.)$ represents the mapping of vulnerability exploitability from $D(E_i)$ to T_s. T_i indicates the dynamic cleaning cycle of the node, and T_s/T_i represents the cleaning times of nodes when dynamic cycle is T_i.We assume that the attacker has to restart the attack process after each MTD implementation. Therefore, the difficulty of the attacker is T_s/T_i times more than the original.

Definition 3. *Assuming that the attacker is rational, when attacking the target node, he will choose the easiest attack path to attack the target. Therefore, the shortest path from the attacker to the target node in the HSAG model can be used to characterize the holistic attack difficulty of the target node.*

Assuming that the target of the attacker is N_a, the holistic attack difficulty of node N_a can be expressed as Equation (4).

$$AF(N_a) = \sigma(I, N_a) \tag{4}$$

Here, I indicates the attacker outside the cloud, and he can access the service instance at the entrance. $\sigma(I, N_a)$ indicates the shortest path between two nodes. After deploying the MTD strategy, the total attack difficulty on this path is calculated by Equation (5).

$$AF'(N_a) = \sigma'(I, N_a) \tag{5}$$

Here, $\sigma'(I, N_a)$ indicates the shortest path between two nodes after deploying the MTD strategy. We use $\Delta S(I, N_a)$ to calculate the security gain of node N_a by deploying the MTD strategy. The formula is shown in Equation (6).

$$\Delta S(I, N_a) = AF'(N_a) - AF(N_a) = \sigma'(I, N_a) - \sigma(I, N_a) \tag{6}$$

Based on the analysis, the security gain is related to two factors: (1) the weight of edge $D(E_i)$; (2) dynamic cleaning cycle T_i .

4.2. Problem Description

For the application $S = \{S_1, S_2, \ldots, S_M\}$ with M microservices, it is assumed that each microservice has its dynamic cleaning cycle, and the holistic security configuration can be expressed as $H = \{T_1, T_2, \ldots, T_M\}$.When the number of microservices and the worker nodes are determined, $G = (N, E)$ can be generated based on the HSAG model. Assumption $PN = \{N_1, N_2, \ldots, N_w\}, PN \subset N$ is a set of nodes that need to be protected in the cloud environment. Since all service instances may be the targets of attackers, the defender cannot perceive the target of attackers in advance. Therefore, we adopt the average security gain of all services to evaluate the system security. The calculation formula is shown in Equation (7).

$$S = \frac{1}{W} \sum_{N_a \in PN} \Delta S(I, N_a) \tag{7}$$

Here, W represents the number of nodes that need to be protected. $\Delta S(I, N_a)$ represents the security gain of node N_a. It should be pointed out that the literature [13] also adopts the average idea to measure the system security when the attack target is unknown. In order to improve the system security, it is obvious that the smaller the dynamic cleaning cycle, the more difficult it is for attackers to complete the attack, and the higher the system security is. However, the dynamic cleaning strategy also affects the QOS of applications. Hence, we adopt dynamic cleaning times per unit time as the defense cost (DC). The calculation formula is shown in Equation (8).

$$DC = \sum_{i=1}^{w} \frac{1}{T_i} \tag{8}$$

Here, T_i indicates the dynamic cleaning cycle of the node. To optimize the system defense configuration, we define defense efficiency DE as the ratio of system security to defense cost. At the same time, we take DE as the optimization goal to optimize the defense configuration, which can realize the key defense against the attack surface, and the precise configuration of defense resources. The optimization problem can be expressed as Equation (9).

$$\begin{gathered} \max DE = \frac{S}{DC} = \frac{1}{W} \frac{\sum\limits_{N_a \in PN} \Delta S(I, N_a)}{\sum\limits_{i=1}^{w} \frac{1}{T_i}} \\ s.t. T_i \in H, T_{\min} \le T_i \le T_{\max} \le T_s \end{gathered} \tag{9}$$

Here, $T_{\max}$ and $T_{\min}$ represent the upper and lower limits of the dynamic cleaning cycle, respectively.

The security configuration optimization in this paper can be transformed into an MDP problem. It is represented by a 5-tuple $M = \langle S, A, P, R, \gamma \rangle$ [38]. Among them, S represents a set of states, A represents a set of actions and $P(S_{t+1}|S_t, a)$ represents the probability that the agent performs the action $a \in A$ to change state from S_t to S_{t+1}. $R(S_t, a)$ is the reward from performing action a in state S_t. $\gamma \in [0, 1]$ is the discount factor, which controls the compromise between future returns and current returns. The purpose of reinforcement learning is to solve the strategy function $\pi^*(a|S_t)$, which maps from state S_t to action a, so as to maximize the cumulative return value. The cumulative return value is given by the Behrman equation, which has two forms: state value function and state-action value function, which are equivalent to each other. The calculation formula is shown in Equations (10) and (11).

$$v_{\pi^*}(S_t) = E_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \tag{10}$$

$$q_{\pi^*}(S_t, a) = E_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A = a\right] \tag{11}$$

In this paper, we sample at a fixed time, ΔT.At t time, the agent needs to obtain the current application state R_s and security configuration H_t as the current state inputs, that is, $S_t = \{RS_t, H_t\}$. When an agent performs actions, it selects a microservice and then increases or decreases its cleaning cycle by ΔT or keeps it unchanged. Therefore, assuming that the application has M microservices, there are 2M + 1 optional actions in the action set. After each action, the agent needs to update the current security configuration. For the current benefit, the defense efficiency DE in the optimization problem can be used to measure it.

5. Detailed Design of the Framework

This section mainly discusses the design of OADSF, including its overall framework design and security configuration optimization algorithm based on P3DQN.

5.1. Design of OADSF

In this paper, we propose OADSF in a container-based cloud, which includes a monitoring module, a security decision module and a control module. The framework is shown in Figure 2.

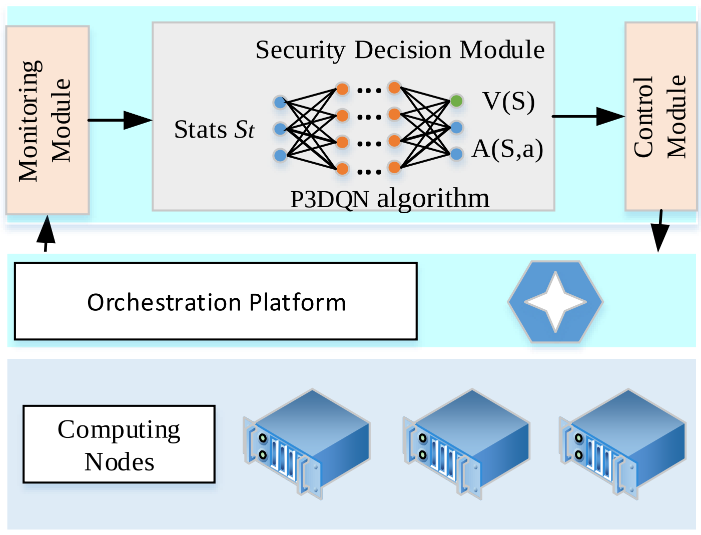

Figure 2. The workflow of the proposed OADSF.

In the container-based cloud cluster, the platform Kubernetes extracts real-time details and sends them to the monitoring module. Kubernetes is used to manage container-based applications in cloud platforms. Its goal is to make deploying container-based applications simple and efficient. When the monitoring module detects that the application state changes, it sends the latest state to the security decision module. The security decision module generates the optimal security configuration according to the running state. The process is as follows: the security decision module preprocesses the running state and current security configuration, and then transmits the data to the input layer of its neural network. After iteratively optimizing the security configuration and achieving convergence, the P3DQN algorithm sends the optimized security configuration to the control module. The control module interacts with Kubernetes to manage microservices according to the current security configuration. When the running time of the microservice replicas reaches the cleaning time, the control module puts the cleaning event for each replica into an independent queue. The cleaning event is to release the microservice container-running environment and regenerate a new container running environment based on Kubernetes. Only after the execution of the cleaning event is complete, the control module takes the cleaning event from the queue and executes the next cleaning event. This mechanism ensures that the dynamic cleaning strategy does not affect the availability of services. The core of OADSF is that the decision module based on P3DQN can quickly optimize the security configuration according to the state of the input. This adaptive security configuration algorithm based on P3DQN will be introduced in detail in Section 5.2.

5.2. Adaptive Security Configuration Algorithm Based on P3DQN

In order to solve the MDP problem, most commonly the DQN algorithm is used. The DQN algorithm combines deep learning and reinforcement learning, taking states as the input of the neural network and actions as the output of the neural network. It interacts with the environment and trains the neural network to make it continuously approach the state-action value function of the current environment. However, the DQN algorithm has overestimation. Instead of using the real action, it selects the largest value action in the target network when calculating the loss function. In this way, the calculated error is not accurate. Based on the analysis, we combined Double DQN, Dueling DQN and priority experience replay to alleviate the overestimation problem, and constructed the P3DQN algorithm. The algorithm process is shown in Figure 3.

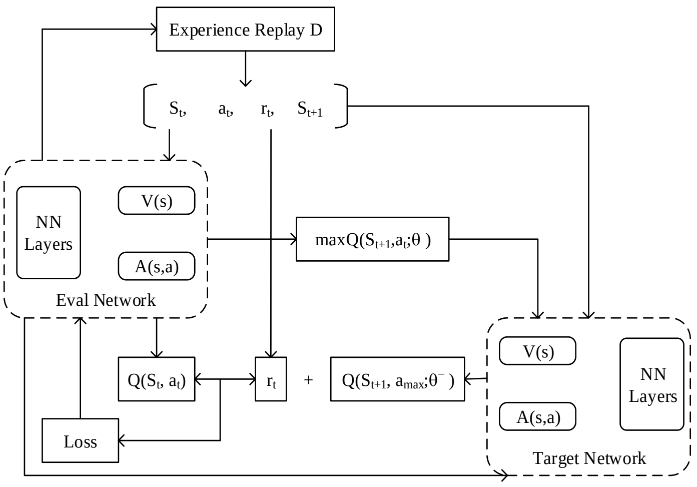

Figure 3. The process of P3DQN algorithm.

P3DQN has two independent neural networks: the evaluation network and target network. The parameters are represented by θ and θ^- , respectively. At t time, the agent executes action a_t in state S_t , observes that the reward is r_t , and the next state is S_{t+1}.The 4-tuple $\langle S_t, a_t, r_t, S_{t+1}\rangle$ is the experience P3DQN acquires from the environment. The parameters of the evaluation network will be updated iteratively based on the acquired experience. At t time, its loss function can be expressed as Equation (12):

$$Loss = E[y_t - Q(S_t, a; \theta)]^2, \tag{12}$$

where $Q(S_t, a; \theta)$ is the output value of action a corresponding to input S_t when the neural network parameter is θ. The output layer of the neural network consists of two outputs, namely, the current state value V and the action advantage function A, which are combined into the Q value. The advantage function shows how good the action value is relative to the average value of the state. $Q(S_t, a; \theta)$ is calculated as shown in Equation (13):

$$Q(S_t, a; \theta) = V(S_t; \theta) + A(S_t, a; \theta) - \underset{a}{mean} A(S_t, a; \theta), \tag{13}$$

where *mean* represents the average value in the n-dimensional space and y_t is the learning target of the evaluation network. The P3DQN algorithm alleviates the overestimation by decoupling action and Q value. When calculating the actual Q value, the evaluation network provides the action in the next state, and the target network provides the Q value of this action, which can be expressed as Equation (14).

$$y_t = r_t + \gamma Q\left(S_{t+1}, \underset{a}{\max} Q(S_{t+1}, a; \theta); \theta^-\right) \tag{14}$$

The updating process of the neural network is shown in Equation (15).

$$Q(S_t, a; \theta) = Q(S_t, a; \theta) + \left[\alpha\left[r_t + \gamma Q\left(S_{t+1}, \underset{a}{\max} Q(S_{t+1}, a; \theta); \theta^-\right) - Q(S_t, a; \theta)\right]\right] \tag{15}$$

In the DQN training stage, in order to improve the stability and convergence of the algorithm, the samples are randomly selected from the experience replay pool, without considering the priority relationship and ignoring the importance of the samples. In our study, P3DQN stores experience in the priority queue. The difference of experience is δ_t. The larger the absolute value of δ_t is, the larger the gap between the estimated Q value and the target Q value. At this time, the more valuable the experience is, the greater the probability that the experience should be extracted for training, which speeds up the training process. The calculation formula is as shown in (16) and (17), where ϑ is a small value close to 0, which ensures that samples when δ_t is 0 can also be selected.

$$\delta_t = y_t - Q(S_t, a; \theta) \tag{16}$$

$$P(t) = \frac{p_t}{\sum p_t}, p_t = |\delta_t + \vartheta| \tag{17}$$

The solution process of the optimal MTD strategy for P3DQN is shown in algorithm 1. In OADSF, the state is used as the input of the neural network, and the MTD strategy parameters are used as the output of the neural network. In algorithm 1, the first line initializes the neural network parameters; the second line indicates that the output value of the neural network consists of the current state value V and the action advantage function A; lines 4–5 represent randomly generated microservice defense configurations, replica numbers, and scheduling policies; line 6 generates the system input S_t based on the previous defense configuration and state; lines 7–9 generate the next state based on the action, calculate the reward value and store the sample experience in the replay pool D; line 11 calculates the action corresponding to the maximum state value in state S_{t+1}; lines 12–18 determine whether S_{t+1} is the termination state based on the set target value, and update the evaluation network parameters θ and weight of samples; line 19 updates the target network parameters θ^- every C step; and line 21 obtains the optimal microservice defense configuration. The algorithm mainly involves interface, state, action and reward. The specific design is as follows.

Algorithm 1: Security configuration optimization algorithm based on P3DQN.
Input: Call relationship between microservices
Output: P3DQN neural network parametersInitialization: neural network parameters

```
(1)  $\theta$, $\theta^-$ , experience replay pool D, samples L, step C, greedy coefficient $\varepsilon$, discount factor $\gamma$
     and learning rate $\alpha$.
(2)  Set $Q(S_t, a; \pi) = V(S_t; \pi) + A(S_t, a; \pi) - \underset{a}{mean} A(S_t, a; \pi)$ , $\pi \in \{\theta, \theta^-\}$
(3)  For episode in range (STEPS)
(4)      Randomly generate microservice defense configuration $H_t$
(5)      Randomly generate the replicas of each microservice, simulate the scheduling of replicas
     and obtain the running state $RS_t$
(6)      Based on $H_t$ and $RS_t$ , generate the input $S_t = \{RS_t, H_t\}$
(7)      Randomly select an action $a_t$ with $\varepsilon$ probability, otherwise select $a_t = \max_a Q(S_t, a; \theta)$
(8)      Modify $H_t$ based on action $a_t$ , reach the next state $S_{t+1}$ , and calculate the reward $r_t$
     based on the HSAG model
(9)      Store sample $\langle S_t, a_t, r_t, S_{t+1} \rangle$ in experience replay pool D
(10)     Select L samples from D with $P(j) = \frac{p_j}{\sum p_i}$ probability
(11)     Calculate $a_t = \max_a Q(S_{t+1}, a; \theta)$ corresponding to the maximum state value in state $S_{t+1}$
(12)     If $S_{t+1}$ is the termination state,
(13)     let $y_t = r_t$
(14)     else
(15)     let $y_t = r_t + \gamma Q\left(S_{t+1}, \max_a Q(S_{t+1}, a; \theta); \theta^-\right)$
(16)     End if
(17)     Use Equations (12) and (14) to perform gradient descent and update network
     parameters $\theta$
(18)     Update the weight of each sample in D using Equation (16)
(19)     Update target network parameters $\theta^- \leftarrow \theta$ every C step
(20) End for
(21) Obtain the optimal microservice defense configuration
```

(1) State: the state consists of running the state and security configurations. In order to facilitate neural network processing, it is assumed that there are UN computing nodes and M microservices in the cluster. The upper limit of the i-th service replica is UR_i, and the running state of the i-th service is $\mathrm{RMS}_i = \left[\mathrm{ind}_1, \mathrm{ind}_2, \ldots, \mathrm{ind}_{\mathrm{UR}_i}\right]$, where $\mathrm{ind} \in [1, \mathrm{UN}]$ is the compute node serial number. The running state can be composed of the running state of all the microservices, namely $\mathrm{RS} = [\mathrm{RMS}_1, \mathrm{RMS}_2, \ldots, \mathrm{RMS}_M]$. The input state $S_t = \{RS_t, H_t\}$ can be obtained by combining the current running state and security configuration. In order to generate a large number of training data, we randomly generated the replicas and performed simulation scheduling according to the cloud platform strategy, with the results as the current running state. At the same time, the security configuration can also be randomly generated.

(2) Action: in the P3DQN algorithm, the action depends on the output layer. In each iteration, we selected ΔT as the basic unit to increase or decrease the cleaning period, or keep the security configuration unchanged.

(3) Reward: when calculating the current reward, we generated the HSAG model based on the input running state, and calculated the defense efficiency DE as a reward by combining the security configuration.

In algorithm 1, when the number of microservices is M and the number of nodes is N, given a fixed defense configuration, the time complexity of using the Dijkstra algorithm to solve the shortest path is $O\left(N^2\right)$.We assume that there are F options for defense configuration for each microservice. Therefore, for Figure G, the optimal computational complexity is $O\left(F^M \cdot N^2\right)$.

In order to reduce the training time and improve the training accuracy, we proposed a two-stage training strategy for the microservice scenario. In the first stage, we randomly generated the microservice running state and security configuration, and trained the model offline with the P3DQN algorithm until the model converged. We obtained the basic model parameters. In the second stage, when the running state changes, the security configuration is further updated. We used the basic model parameters for online training to find the optimal security configuration. Because online training is based on offline training, it can achieve convergence quickly.

6. Simulation and Evaluation

This section introduces the simulation setup, and then introduces the strategy to compare with OADSF. Finally, we analyze the simulation results to verify its effectiveness and scalability.

6.1. Simulation Setup

In the experiment, we used Kubernetes to build container clustering. The cluster consisted of 11 servers with 40 cores, 64 GB memory, and 2 T disks, where 10 servers were computing nodes, 1 server was the management node, and OADSF ran on the management node. Meanwhile, we deployed a Web application, which was composed of four microservices, with the same call relationship as Figure 1. The specific information and vulnerabilities are shown in Table 1.

Table 1. Vulnerabilities used in the HSAG model.

Microservice	Name	CVE ID	ED	W	D (Ei)
A	Tomcat	CVE-2019-14768	2.8	7.4	3.1497
		CVE-2019-10104	3.9	8.6	
		CVE-2019-0232	2.2	7.9	
		CVE-2020-26510	3.9	8.3	
		CVE-2020-17388	2.8	7.4	
B	Memcached	CVE-2016-8704	3.9	8.3	3.406
		CVE-2016-8705	3.9	8.3	
		CVE-2016-8706	2.2	6.8	
C	ImageMagick	CVE-2017-14650	2.2	6.8	2.2561
		CVE-2017-14224	2.8	8.3	
		CVE-2019-11832	1.6	6.3	
D	Mysql	CVE-2020-11974	3.9	8.3	3.1192
		CVE-2016-6663	1	6.3	
		CVE-2016-6662	3.9	8.8	
-	Container	CVE-2020-7606	0.5	5.4	2.5599
		CVE-2020-35197	3.9	8.3	

Based on $D(E_i)$, we can obtain the attack difficulty and security gain under different cleaning cycles. In the P3DQN algorithm, the hidden layer of the neural network adopts a three-layer fully connective structure, in which the last layer is divided into two outputs. The learning rate of the neural network is α = 0.0005, the discount factor is γ = 0.9, the greedy coefficient is ε = 0.1 and ϑ = 0.001, the minimum batch data extracted for each training is L = 32 and the update step of the target network is C = 500.

The performance parameters of the P3DQN algorithm include the neural network learning rate, experience replay pool, step size, greedy coefficient and discount factor. From Formula (15), it can be seen that the learning rate affects the update speed of the neural network. The discount factor indicates the degree to which previous experience is valued. The greed coefficient represents the ability of the agent to explore the unknown motion space. $Q\left(S_{t+1}, \max_a Q(S_{t+1}, a; \theta); \theta^-\right)$ indicates that the evaluation network generates the maximum action at the next moment, and the target network generates the Q value. This method reduces training errors. The smaller the experience replay pool and step size, the faster the convergence is, but it is easy to fall into a local optimal solution.

Hence, the selected parameters can ensure the fast convergence of the algorithm and obtain higher rewards. Fast convergence indicates a short time to calculate the optimal defense configuration, and higher rewards indicate a high defense efficiency. This can quickly generate optimal defense configurations for each microservice in a dynamic and complex cloud environment, further improving the security of cloud computing.

6.2. Comparison Strategy

In the experiment, we compared OADSF with the unified configuration strategy [6], DSEOM [13], SmartSCR [14] and optimal strategy, respectively. The details of the comparison strategies are as follows.

(1) The unified configuration strategy simplifies dynamic cycle configuration. It is assumed that the dynamic cycle of all microservices is the same, which greatly reduces the computation, and the dynamic cycle can be obtained through traversal. In the reference [6], this strategy is used to simplify the problem of implementing dynamic cleaning strategy.

(2) The optimal strategy is to find out the optimal defense configuration by brute-force search, which provides a reference for each algorithm.

(3) DSEOM depicts the attack difficulty by the attack graph model. The strategy computes the critical nodes through betweenness centrality and only protects the critical nodes.

The betweenness centrality calculates by the ratio of the number of shortest paths passing through node N to the total number of shortest paths.

(4) SmartSCR also depicts the attack difficulty with the attack graph model and protects all nodes. However, this strategy uses the S-function to calculate the probability to determine the attack difficulty of nodes. It only takes the security after MTD deployment into consideration, and the optimization algorithm overestimates defense efficiency.

6.3. Simulation Results

To verify the performance and effectiveness of OADSF, we first randomly generated the replicas of each microservice, and created the application in the experimental environment. Based on the replicas and Kubernetes scheduling results, DQN and P3DQN were used for training, respectively.

In Figure 4, we compare the convergence speed and average rewards of DQN and P3DQN algorithms. Among them, SmartSCR uses the DQN algorithm, while OADSF uses the P3DQN algorithm. Figure 4 shows the change of the average reward with the training steps. With the increase of training steps, the average reward shows a gradual upward trend and reaches a stable trend after certain steps. However, in terms of the convergence and stability, P3DQN algorithm can obtain a higher average reward with the same training steps. Because it decouples the calculation of the target *Q* value, which avoids overestimation. When the system updates the weight parameters of the neural network every time, it will preferentially select the experience samples with larger time difference to ensure the learning effect. Compared with DQN, P3DQN has higher learning efficiency and can achieve better results.

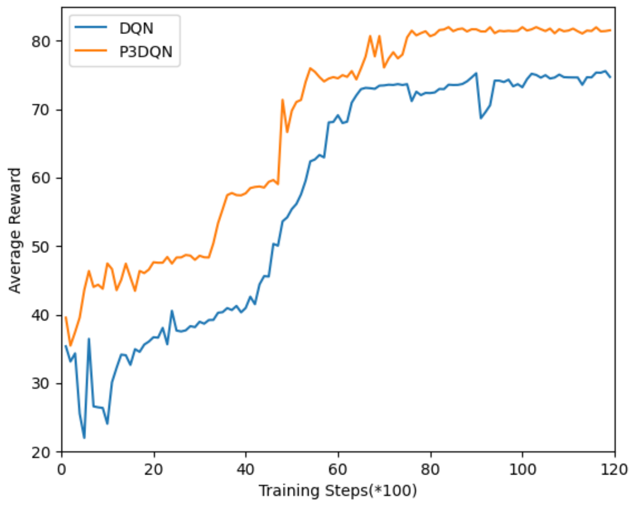

Figure 4. Reward with the training steps.

Figure 5 illustrates the average loss with the variation of training steps during offline and online training. As the figure shows, when its offline training and the experience replay pool is not filled, the P3DQN algorithm will adopt a random strategy to modify the defense configuration. Thus, the average loss also fluctuates all the time. Thanks to the offline training process, the neural network model can converge quickly in online training. Therefore, we can find the optimal security configuration in time after the microservice state changes, and realize the adaptive security optimization configuration.

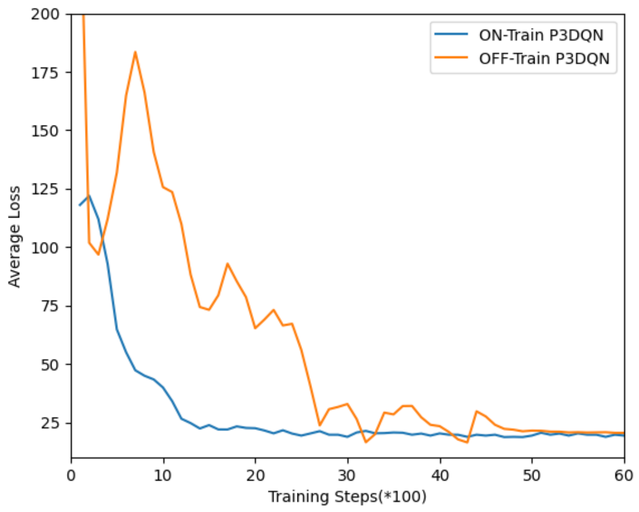

Figure 5. Loss with the training steps.

Figure 6 shows the comparison of defense efficiency between DSEOM, unified strategy, SmartSCR, OFF-OADSF and ON-OADSF. The abscissa represents different algorithms and the ordinate represents the normalized value of defense efficiency.

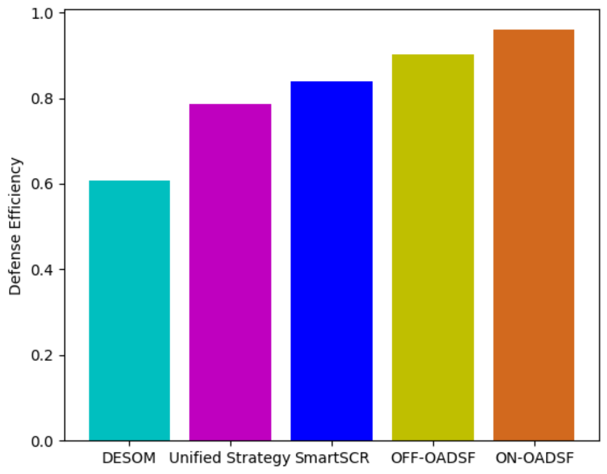

Figure 6. Defense Efficiency of different algorithms.

SmartSCR and OADSF select the defense efficiency after the algorithm converges. It can be seen from the figure that the defense efficiency of DSEOM is the lowest. The main reason is that DSEOM uses betweenness centrality to calculate key targets. Once the attacker finds a path to bypass the key target, he can easily realize the attack target. At the same time, after DSEOM selects the key target, the defender needs to configure the parameters of the MTD strategy independently, which makes the defense efficiency unstable. The unified strategy sets the cleaning cycle of each target to be the same. In this way, the optimal configuration can be selected through traversal. Compared with DSEOM, it can improve the defense efficiency and stability. SmartSCR provides a dynamic defense for all nodes to improve the holistic defense efficiency. However, this strategy uses the S-function to calculate the probability to determine the attack difficulty of target. It only takes the security after MTD deployment into consideration, and the DQN optimization algorithm overestimates defense efficiency. In this study, OADSF describes the holistic security by using the security gain, and further optimizes defense efficiency using the P3DQN algorithm. After convergence, the average defense efficiency is very close to the optimal strategy, whichh can achieve a stable security defense effect. Through offline and online training, the algorithm can converge quickly and efficiently. Compared with

DSEOM and SmartSCR, the defense efficiency of OADSF was improved by 35.19% and 12.09%, respectively.

Figure 7 shows the time consumption of different algorithms at different scales, which is used to measure the scalability of the strategy. The abscissa represents the number of replicas in the application, and the ordinate represents the time consumption to solve the active defense configuration. In this experiment, we changed the scale of the entire application by increasing the replicas of the microservice. At the same time, it was assumed that the upper limit of each replica was 200, to train the model. As the figure shows, the time consumption increases greatly with the growth of the application scale through the unified configuration strategy, which is suitable for small-scale application scenarios. For DSEOM, the time consumption is the shortest, and it does not show exponential growth with the increase of the application scale. The reason is that DSEOM requires less computation and optimizes the solution of the shortest path. SmartSCR actively defends all nodes. Thus, as the scale increases, the time consumption will increase compared with DSEOM. For OADFS, it can train the model in advance offline, so it takes less time to configure the active defense strategy online, which can cope with large-scale application scenarios.

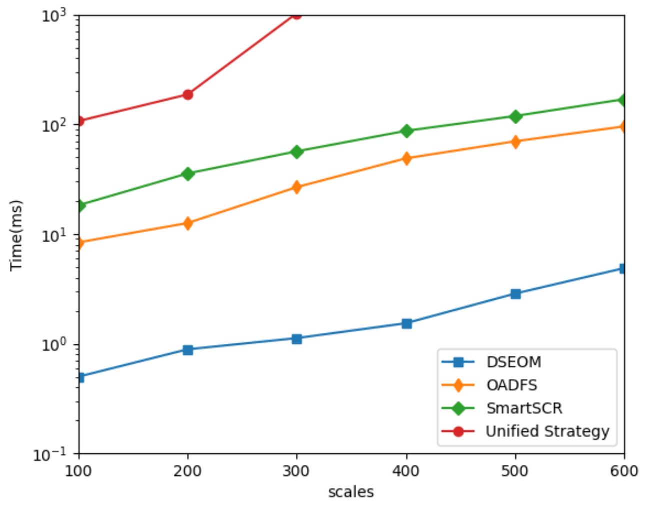

Figure 7. Time consumption under different scales.

Through the analysis of the experimental results in Section 6.3, it can be concluded that the HSAG model we proposed includes threats from the application layer and the container layer, improving the previous attack graph model. The OADSF can extract system threats and the running state of microservices in real time. The P3DQN algorithm can solve the optimal defense configuration in a short time and dynamically improve the security defense capability of cloud computing.

7. Conclusions

This paper studied the optimal configuration of the MTD strategy in the container cloud environment. First, we analyzed the security threats under the container cloud and formalized various complex attack scenarios. Then, we proposed the HSAG model to extract specific risks and threats in the cloud, including application layer and virtualization layer vulnerabilities. The HSAG model was updated according to the system running status. On the basis of this model, the security gain and QOS were quantitatively analyzed in complex attack scenarios. Due to the scale and high dynamics of cloud applications, it is very difficult to directly solve the optimal configuration. To solve this problem, we modeled the optimization of the MTD strategy as a Markov decision process, and used deep reinforcement learning to generate the optimal defense configuration for the orchestration platform. In this method, we described the defense efficiency based on the HSAG model, and proposed the P3DQN algorithm to cope with the scale and high dynamic of cloud applications. Combining offline training and online training, we solved the optimal defense configuration and realized adaptive optimization. The simulation results show that the proposed method can improve the defense efficiency. Compared with DSEOM and SmartSCR, the defense efficiency was increased by 35.19% and 12.09%, respectively.

However, the framework still has the following problems: (1) considering the security configuration under different cleaning cycles, it will inevitably introduce time overhead; (2) the randomly generated scheduling policies of microservices may aggravate the security threats of the system, then affect the defense efficiency. For future work, we consider the following: (1) for different microservice types, different MTD policies will be used for the security configuration to further improve security and reduce time overhead; (2) based on the HSAG model, we will analyze the dependency between microservices and consider the security factor during scheduling; (3) the visualization HSAG model.

Author Contributions: Conceptualization, Y.L. and H.H.; methodology, Y.L. and W.L.; software, Y.L. and X.Y.; validation, Y.L. and X.Y.; formal analysis, Y.L. and H.H.; investigation, Y.L. and W.L; data curation, W.L; writing—original draft preparation, Y.L.; writing—review and editing, H.H. and W.L.; supervision, H.H., W.L. and X.Y.; funding acquisition, X.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partly supported by the National Natural Science Foundation of China (Grant Nos. 62072467 and 62002383), National Key Research and Development Plan of China (Grant Nos. 2021YFB1006200 and 2021YFB1006201).

Data Availability Statement: Not applicable.

Acknowledgments: The authors thank the reviewers for their valuable comments and suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhou, X.; Peng, X.; Xie, T.; Sun, J.; Ji, C.; Li, W.; Ding, D. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. *IIEEE Trans. Softw. Eng.* **2021**, *47*, 243–260. [CrossRef]
2. Gokan Khan, M.; Taheri, J.; Al-dulaimy, A.; Kassler, A. PerfSim: A Performance Simulator for Cloud Native Microservice Chains. *IEEE Trans. Cloud Comput.* **2021**, *1*, 1–18. [CrossRef]
3. Arouk, O.; Nikaein, N. Kube5G: A Cloud-Native 5G Service Platform. In Proceedings of the GLOBECOM 2020–2020 IEEE Global Communications Conference, Taipei, Taiwan, 11–13 December; 2020; pp. 1–6.
4. Gao, X.; Steenkamer, B.; Gu, Z.; Kayaalp, M.; Pendarakis, D.; Wang, H. A Study on the Security Implications of Information Leakages in Container Clouds. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 174–191. [CrossRef]
5. Nife, F.N.; Kotulski, Z. Application-Aware Firewall Mechanism for Software Defined Networks. *J. Netw. Syst. Manag.* **2020**, *28*, 605–626. [CrossRef]
6. Bardas, A.G.; Sundaramurthy, S.C.; Ou, X.; DeLoach, S.A. MTD CBITS: Moving Target Defense for Cloud-Based IT Systems. In Proceedings of the 22nd European Symposium on Research in Computer Security, Oslo, Norway, 1–3 December; 2017; pp. 167–186.
7. Zhuang, R.; DeLoach, S.A.; Ou, X. Towards a Theory of Moving Target Defense. In Proceedings of the First ACM Workshop on Moving Target Defense, Scottsdale, AZ, USA, 3 November 2014; pp. 31–40.
8. Lu, K.; Song, C.; Lee, B.; Chung, S.P.; Kim, T.; Lee, W. ASLR-Guard: Stopping Address Space Leakage for Code Reuse Attacks. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver Colorado, CO, USA, 12 October 2015; pp. 280–291.
9. Larsen, P.; Homescu, A.; Brunthaler, S.; Franz, M. SoK: Automated Software Diversity. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 7–9 May 2014; pp. 276–291.
10. Meier, R.; Tsankov, P.; Lenders, V.; Vanbever, L.; Vechev, M. NetHide: Secure and Practical Network Topology Obfuscation. In Proceedings of the 27th USENIX Security Symposium, Baltimore, MD, USA, 15–18 August 2018; pp. 1–18.
11. Debroy, S.; Calyam, P.; Nguyen, M.; Stage, A.; Georgiev, V. Frequency-Minimal Moving Target Defense Using Software-Defined Networking. In Proceedings of the 2016 International Conference on Computing, Networking and Communications (ICNC), Kauai, HI, USA, 2–5 February 2016; pp. 1–6.
12. Carroll, T.E.; Crouse, M.; Fulp, E.W.; Berenhaut, K.S. Analysis of Network Address Shuffling as a Moving Target Defense. In Proceedings of the 2014 IEEE International Conference on Communications (ICC), Sydney, NSW, Australia, 10–14 June 2014; pp. 701–706.
13. Jin, H.; Li, Z.; Zou, D.; Yuan, B. DSEOM: A Framework for Dynamic Security Evaluation and Optimization of MTD in Container-Based Cloud. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 1125–1136. [CrossRef]
14. Zhang, S.; Guo, Y.; Sun, P.; Cheng, G.; Hu, H. Deep reinforcement learning based moving target defense strategy optimization scheme for cloud native environment. *J. Electron. Inf. Technol.* **2022**, *44*, 608–616. [CrossRef]

15. Belair, M.; Laniepce, S.; Menaud, J.-M. Leveraging Kernel Security Mechanismsto Improve Container Security: A Survey. In Proceedings of the 14th International Conference on Availability, Reliability and Security, New York, NY, USA, 11–13 October 2019; pp. 1–6.
16. Lopes, N.; Martins, R.; Correia, M.E.; Serrano, S.; Nunes, F. Container Hardening Through Automated Seccomp Profiling. In Proceedings of the 2020 6th International Workshop on Container Technologies and Container Clouds, Delft, The Netherlands, 7 December 2020; pp. 31–36.
17. Lin, X.; Lei, L.; Wang, Y.; Jing, J.; Sun, K.; Zhou, Q. A Measurement Study on Linux Container Security: Attacks and Countermeasures. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3 December 2018; pp. 418–429.
18. Flora, J.; Antunes, N. Studying the Applicability of Intrusion Detection to Multi-Tenant Container Environments. In Proceedings of the 2019 15th European Dependable Computing Conference (EDCC), Naples, Italy, 4 September 2019; pp. 133–136.
19. Lim, S.Y.; Stelea, B.; Han, X.; Pasquier, T. Secure Namespaced Kernel Audit for Containers. In Proceedings of the ACM Symposium on Cloud Computing (SoCC), New York, NY, USA, 15–17 December 2021; pp. 518–532.
20. Lin, Y.; Tunde-Onadele, O.; Gu, X. CDL: Classified Distributed Learning for Detecting Security Attacks in Containerized Applications. In Proceedings of the Annual Computer Security Applications Conference, Austin, TX, USA, 7 December 2020; 2020; pp. 179–188.
21. Almeida, W.H.C.; de Aguiar Monteiro, L.; Hazin, R.R.; de Lima, C.; Ferraz, F.S. Survey on Microservice Architecture—Security, Privacy and Standardization on Cloud Computing Environment. In Proceedings of the 12th International Conference on Software Engineering Advance, Athenas, Greek, 20 December 2017; pp. 1–7.
22. Zhao, P.; Wu, L.; Hong, Z.; Sun, H. Research on Multicloud Access Control Policy Integration Framework. *China Commun.* **2019**, *16*, 222–234. [CrossRef]
23. Pereira-Vale, A.; Fernandez, E.B.; Monge, R.; Astudillo, H.; Márquez, G. Security in Microservice-Based Systems: A Multivocal Literature Review. *Comput. Secur.* **2021**, *103*, 102200. [CrossRef]
24. Xu, R.; Jin, W.; Kim, D. Microservice Security Agent Based On API Gateway in Edge Computing. *Sensors* **2019**, *19*, 4905. [CrossRef] [PubMed]
25. Sankaran, A.; Datta, P.; Bates, A. Workflow Integration Alleviates Identity and Access Management in Serverless Computing. In Proceedings of the Annual Computer Security Applications Conference, Austin, TX, USA, 7 December 2020; pp. 496–509.
26. Torkura, K.A.; Sukmana, M.I.H.; Meinel, C. Integrating Continuous Security Assessments in Microservices and Cloud Native Applications. In Proceedings of the 10th International Conference on Utility and Cloud Computing, Austin, TX, USA, 5 December 2017; 2017; pp. 171–180.
27. Ahmed, N.O.; Bhargava, B. From Byzantine Fault-Tolerance to Fault-Avoidance: An Architectural Transformation to Attack and Failure Resiliency. *IEEE Trans. Cloud Comput.* **2018**, *8*, 847–860. [CrossRef]
28. Li, Y.; Dai, R.; Zhang, J. Morphing Communications of Cyber-Physical Systems towards Moving-Target Defense. In Proceedings of the 2014 IEEE International Conference on Communications (ICC), Sydney, NSW, Australia, 10–14 June 2014; pp. 592–598.
29. Yu, H.; Li, H.; Yang, X.; Ma, H. On Distributed Object Storage Architecture Based on Mimic Defense. *China Commun.* **2021**, *18*, 109–120. [CrossRef]
30. Qiang, W.; Chunming, W.; Xincheng, Y.; Qiumei, C. Intrinsic Security and Self-Adaptive Cooperative Protection Enabling Cloud Native Network Slicing. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 1287–1304. [CrossRef]
31. Wang, Y.; Guo, Y.; Guo, Z.; Liu, W.; Yang, C. Protecting Scientific Workflows in Clouds with an Intrusion Tolerant System. *IET Inf. Secur.* **2020**, *14*, 157–165. [CrossRef]
32. Wang, Y.; Guo, Y.; Wang, W.; Liang, H.; Huo, S. INHIBITOR: An Intrusion Tolerant Scheduling Algorithm in Cloud-Based Scientific Workflow System. *Future Gener. Comput. Syst.* **2021**, *114*, 272–284. [CrossRef]
33. Nguyen, T.T.; Reddi, V.J. Deep Reinforcement Learning for Cyber Security. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, 1–17. [CrossRef] [PubMed]
34. Uprety, A.; Rawat, D.B. Reinforcement Learning for IoT Security: A Comprehensive Survey. *IEEE Internet Things J.* **2021**, *8*, 8693–8706. [CrossRef]
35. Xiao, L.; Wan, X.; Dai, C.; Du, X.; Chen, X.; Guizani, M. Security in Mobile Edge Caching with Reinforcement Learning. *IEEE Wirel. Commun.* **2018**, *3*, 116–122. [CrossRef]
36. Lopez-Martin, M.; Carro, B.; Sanchez-Esguevillas, A. Application of Deep Reinforcement Learning to Intrusion Detection for Supervised Problems. *Expert Syst. Appl.* **2020**, *141*, 112963. [CrossRef]
37. Sun, P.; Guo, Z.; Liu, S.; Lan, J.; Wang, J.; Hu, Y. SmartFCT: Improving power-efficiency for data center networks with deep reinforcement learning. *Comput. Netw.* **2020**, *179*, 107255. [CrossRef]
38. Li, H.; Guo, Y.; Sun, P.; Wang, Y.; Huo, S. An Optimal Defensive Deception Framework for the Container-based Cloud with Deep Reinforcement Learning. *IET Inf. Secur.* **2022**, *16*, 178–192. [CrossRef]