云原生下基于深度强化学习的移动目标防御策略优化方案

张 帅*① 郭云飞① 孙鹏浩② 程国振① 扈红超①
①(战略支援部队信息工程大学信息技术研究所 郑州 450002)
②(军事科学院 北京 100000)

摘 要：针对云原生环境下攻击场景的复杂性导致移动目标防御策略配置困难的问题，该文提出一种基于深度强化学习的移动目标防御策略优化方案(SmartSCR)。首先，针对云原生环境容器化、微服务化等特点，对其安全威胁及攻击者攻击路径进行分析；然后，为了定量分析云原生复杂攻击场景下移动目标防御策略的防御效率，提出微服务攻击图模型并对防御效率进行刻画。最后，将移动目标防御策略的优化问题建模为马尔可夫决策过程，并使用深度强化学习解决云原生应用规模较大时带来的状态空间爆炸问题，对最优移动目标防御配置进行求解。实验结果表明，SmartSCR能够在云原生应用规模较大时快速收敛，并实现逼近最优的防御效率。

关键词：云原生；移动目标防御；强化学习；微服务
中图分类号：TN915.08; TP302 **文献标识码**：A **文章编号**：1009-5896(2023)02-0608-09
DOI: 10.11999/JEIT211589

Moving Target Defense Strategy Optimization Scheme for Cloud Native Environment Based on Deep Reinforcement Learning

ZHANG Shuai① GUO Yunfei① SUN Penghao② CHENG Guozhen①
HU Hongchao①
①(*Institute of Information Technology, Strategic Support Force Information Engineering University, Zhengzhou* 450002, *China*)
②(*PLA Academy of Military Sciences, Beijing* 100000, *China*)

Abstract: To deal with the difficulty of configuring Moving Target Defense (MTD) strategy under complexity attack scenarios in the cloud native environment, a deep reinforcement learning based moving target defense strategy optimization scheme (SmartSCR) is proposed. First, the security threats together with the attack paths are analyzed considering the characteristics of containerization and microservice. Then, in order to evaluate the defense efficiency of moving target defense under complexity attack scenarios in the cloud native environment, the microservice attack graph model is proposed to defense quantify efficiency. Finally, the optimization of moving target defense strategy is modeled as a Markov decision process. A deep reinforcement learning based strategy is proposed to handle the state space explosion under large scale cloud native applications, thus to solve out the optimal configuration for moving target defense strategy. The experiment results show that SmartSCR can quickly converge under large scale cloud native applications, and achieve near optimal defense efficiency.

Key words: Cloud native; Moving Target Defense (MTD); Reinforcement learning; Microservice

1 引言

随着云计算技术的持续发展，已经由“面向云迁移应用”的阶段演进到“面向云构建应用”的阶段，即由“以资源为中心”演进到“以应用为中心”的云原生阶段[1]。在云原生环境下，传统单体式应用按照功能逻辑被拆分为多个微服务，容器技术则为微服务提供了轻量级运行环境[2]。云原生环境下应用可以充分利用云计算弹性、敏捷和资源池等特性，加速应用的开发与迭代过程，提高应用的可扩展性。由于云原生的突出优势，云原生技术生态不断演进，云原生思想也逐渐深入人心[3,4]。

收稿日期：2021-12-29；改回日期：2022-05-19；网络出版：2022-06-13
*通信作者： 张帅 2012301200229@whu.edu.cn
基金项目：国家重点研发计划(2021YFB1006200，2021YFB1006201)，国家自然科学基金(62072467)
Foundation Items: The National Key Research and Development Plan (2021YFB1006200, 2021YFB1006201), The National Natural Science Foundation of China (62072467)

云原生彻底改变了云端应用的设计、开发、部署和运行模式，同时也带来了新的安全威胁。微服务化拆分使得服务间的交互接口爆炸式增长，导致微服务攻击面难以管控。基于容器的轻量级虚拟化技术使得同一宿主机上的多个容器共享操作系统内核，给攻击者在集群中横向移动提供了便利[5]。在传统网络安全策略中，主要使用基于边界部署的防护方案，如防火墙、入侵检测等[6]。然而，云原生环境下传统应用软件的边界逐渐模糊化，防火墙、入侵检测等防护设备的部署位置难以确定。因此，传统基于边界的防护模型无法完全应对云原生环境下的安全威胁[7]。作为一种典型的主动防御技术，移动目标防御(Moving Target Defense, MTD)通过持续改变防御目标的攻击面，以起到阻断攻击链的作用。常见的MTD技术包括执行环境动态化[8]、软件实现动态化[9]、网络拓扑动态化[10]等等。针对云原生环境下的MTD策略的优化设计，文献[11]提出基于动态安全评估与配置优化的MTD策略。该策略通过动态评估容器云环境中的关键节点，实现对MTD策略防护对象的动态调整。由于上述策略仅对关键节点进行防护，难以防范攻击者“绕过”关键节点的攻击场景。

针对以上问题，本文提出一种基于深度强化学习的MTD策略优化方案(SmartSCR)。与文献[11]不同的是，本文考虑采用动态清洗策略对目标应用下所有的微服务进行防护，以应对云原生环境下复杂的攻击场景。同时，使用深度强化学习技术求解最优的动态清洗周期，以实现防御效率的最大化。首先，对云原生环境下微服务化、容器化引入的复杂攻击场景进行分析，建立了微服务攻击图(Microservice Attack Graph, MAG)模型，并基于MAG模型对MTD策略下的防御效率进行刻画；然后，提出基于深度Q网络(Deep Q Network, DQN)的MTD安全配置优化算法，以应对云原生应用规模较大时带来的状态空间爆炸问题。实验结果表明，即使面对较大规模的云原生应用，SmartSCR仍然能够快速实现收敛，并实现逼近最优的防御效率。

2　问题分析及挑战

本节以实际场景为例，阐明了云原生环境下存在的主要安全威胁，分析了MTD技术解决上述安全威胁的独特优势。并梳理出云原生场景下应用MTD技术的主要挑战。

2.1　安全威胁

云原生环境下，单体式应用被拆分为多个微服务，运行在云计算集群中。多个微服务间协调配合，通过调用链的方式实现特定功能。对于每个微服务，其运行环境使用容器这种轻量级虚拟化技术，实现了微服务运行环境的隔离需求。同时，每个微服务将调整其副本数量，已应对自身动态变化的并发请求。然而，将单体式应用拆分为微服务模式，同样导致攻击面爆炸式增长。以图1为例，本文从攻击目标、攻击过程和攻击者能力假设3方面描述安全威胁。

(1)攻击目标。在云原生环境下，云计算集群中运行的所有微服务都可能成为攻击者的目标。对于第i个微服务，其攻击面可由应用层攻击面和容器层攻击面组成，可表示为$\mathrm{AS}_i=\{A_i,C_i\}$。其中，应用层攻击面$A_i$包含应用自身代码以及代码所依赖的代码库、框架等。容器层攻击面C_i指的是微服务应用所使用的容器运行环境。

(2)攻击过程。本文采用网络杀伤链(Cyber Kill Chain, CKC)模型来分析攻击者的攻击过程[7]。在该模型中，攻击者首先需要对攻击目标进行侦察探测，获取发起网络攻击所需的信息。然后，攻击者基于收集到的信息(如目标存在的漏洞信息)，并对目标展开攻击。

(3)攻击者能力假设。假设攻击者处于云计算集群外部，通过互联网对微服务应用展开攻击。攻击者首先只能攻击对外开放的微服务的应用层攻击面，如服务A。当攻击者通过应用层漏洞成功挟持服务A时，攻击者有以下两种攻击模式，以实现在云环境的横向移动。

(a) 面向应用层的攻击。攻击者从应用层逃逸后，继续寻找网络可达的微服务，对其应用层攻击面展开攻击。假设云原生环境下网络配置服从最小权限原则[11]，即只有当微服务间有调用关系时，两者网络才是可达的。如图1所示，攻击者在挟持服务A后，可继续对服务B和服务C展开攻击。

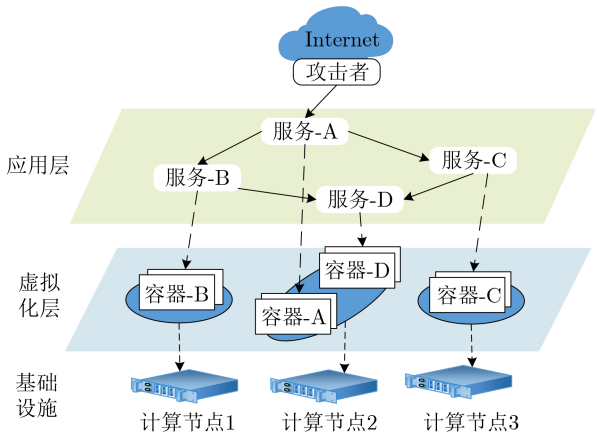


图 1 云原生环境下微服务攻击面示意图

(b) 面向虚拟化层的攻击。攻击者从应用层逃逸后，进入到该微服务的容器层攻击面，寻找容器的漏洞以实现容器逃逸(例如容器配置漏洞CVE-2016-5195)。若能成功实现容器逃逸，则攻击者可获得该容器所在计算节点的权限，移动至该计算节点上的其他容器，直接对容器环境中运行的服务进行挟持。如图1所示，攻击者在挟持服务A之后，可直接进入到容器A。若攻击者成功从容器A逃逸，则可进入同驻的容器D，并挟持容器D所运行的服务D。

基于上述分析可知，云原生环境下微服务化、容器化导致攻击面陡增，攻击路径爆炸增长，安全管控十分困难。

2.2 移动目标防御策略设计

基于MTD的思想，本文考虑通过动态清洗微服务的方式，增加攻击者的攻击逃逸以及横向移动的难度，实现云原生环境下的安全管控。例如，定时删除微服务A的副本，并基于镜像创建新的副本。假设攻击者正在尝试攻击该微服务副本，该清洗策略会使得副本的IP地址发生变化，并清除掉所有攻击者已设法注入的文件，使得攻击者前功尽弃。同样地，假设攻击者已经完成了对被清洗微服务副本的劫持，并设法横向移动，该清洗策略会使得攻击者丧失对该副本的控制。在文献[11]中，防御者仅考虑对应用中的关键目标使用移动目标防御策略。然而，该文献主要关注应用状态变化后对关键目标的实时调整，仍存在以下问题有待于优化：(1)未能给防御者提供关键目标数量选取的理论依据；(2)未考虑移动目标防御策略的参数设定问题。因此，防御者只能盲目地选取上述参数，难以实现安全和开销的折中。与文献[11]思路不同的是，本文考虑对所有的微服务都使用主动防御策略，通过调整的防御策略参数的方式优化防御效率，无需防御者设定参数即可实现安全与开销的折中。

2.3 主要挑战

尽管动态清洗策略能够显著提高攻击者完成攻击与在内网横向移动的难度，但是，该策略也会对服务质量造成影响。为了平衡微服务的安全与性能，最大化防御策略的效率，需要对不同微服务的清洗周期T进行优化设计。该问题主要面临的挑战如下：

(1)基于云原生环境下的安全威胁分析可知，攻击者在云环境中攻击路径多样。而为了实现高效的防御，则需要以较高的强度，动态清洗攻击路径上的关键微服务，实现“要地”的重点防御。然而，目前缺乏相应的安全模型，对云原生环境下攻击者的多样化攻击路径进行刻画，并衡量在不同清洗周期T下的防御效果。

(2)在云原生环境下，一个复杂的应用中可能包含大量的微服务。例如，Uber中包含了大约2200个微服务[12]。同时，每个微服务还可能包含多个副本。为了给海量的微服务副本合理配置清洗周期，MTD策略优化算法需要有着较强的可扩展性，以应对云原生应用规模较大的场景。

3 问题建模

针对云原生环境下MTD策略所面临的挑战，本节首先根据攻击者内网渗透的特点，提出MAG模型，刻画攻击者多样化的攻击路径以及MTD效果；然后，基于MAG模型，详细归纳了动态清洗策略所面临的清洗周期配置问题。最后，将MTD配置优化问题转化为马尔可夫决策过程(Markov Decision Process, MDP)。

3.1 微服务攻击图模型

基于对云原生环境下攻击者攻击路径的分析，本文对MAG做如下定义。

定义1 使用有向图$G=(N,E)$来表示MAG，其中，N是图中节点的集合，包括攻击者I与微服务所有的攻击面。假设云环境中应用由M个微服务副本组成，则$N=\{I,A_1,A_2,\cdots,A_M,C_1,C_2,\cdots,C_M\}$。其中，$\{A_1,A_2,\cdots,A_M\}$代表应用层攻击面，$\{C_1,C_2,\cdots,C_M\}$代表虚拟化层攻击面。$E\subseteq N\times N$是图中所有边的集合，每条边代表着攻击者的攻击路径。

基于上述定义，对于节点$N_a,N_b\in N,a\neq b$，边$e=(N_a,N_b)$表示攻击者基于已挟持的节点N_a，利用节点N_b的漏洞横向移动至N_b。本文考虑采用MTD策略下成功抵御攻击的概率来表示图中边的权重。假设节点N_b的动态清洗周期为T_b，则边e的权重与两方向因素相关：(1)节点N_b上存在的漏洞利用的难易程度；(2)节点N_b的动态清洗周期为T_b。

针对节点上漏洞利用的难易程度，考虑基于通用漏洞评分系统(Common Vulnerability Scoring System, CVSS)进行刻画[13]。在最新发布的CVSS 3.1规范中，包含了漏洞的以下指标：基础得分度量、临时得分度量和环境得分度量。为了刻画漏洞利用的难易程度，使用漏洞利用困难度EM进行评估。基于基础得分度量中的可利用性指标，EM具体可表示为

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

其中，AV, AC, PR和UI均为漏洞可利用性指标中的参数，分别代表攻击向量、攻击复杂度、特权需求度和用户交互度。对于每个节点所代表的攻击

面，可能存在多个可利用的漏洞。然而，无法预估攻击者的行为，判断其攻击时会选择的漏洞。针对该问题，本文使用临时得分度量衡量漏洞被攻击者选择的权重W。通过对节点上所有漏洞利用的困难程度进行加权平均，节点漏洞利用的困难程度ND可表示为

$$\mathrm{ND}=\frac{\sum_{v\in V}(W\times \mathrm{EM})}{\sum_{v\in V}W} \tag{2}$$

其中，V表示该节点代表的攻击面存在的所有漏洞的集合。

为了衡量不同清洗周期下攻击者攻击成功的概率，考虑采用S型函数建模攻击成功概率随攻击时间的变化，该模型也被广泛应用于对CKC模型下攻击成功概率的衡量[14]。在该模型下，攻击成功概率$p_{\mathrm{att}}\in(0,1)$随着攻击时间的增加而增加，可分为两个阶段：(1)在第1阶段，攻击成功概率的增长率逐渐增加，代表着攻击者获取到的漏洞利用信息逐渐增加；(2)在第2阶段，攻击成功概率的增长率逐渐减少，代表着攻击者关于攻击目标可获取的漏洞利用信息逐渐饱和。攻击成功概率具体可表示为

$$p_{\mathrm{att}}(t)=p_0+\frac{p_1-p_0}{1+\mathrm{e}^{-\beta(t-\alpha)}} \tag{3}$$

其中，p_0表示攻击成功概率的下限，p_1表示攻击成功概率上限，β表示攻击成功概率的增长率，代表攻击者攻击能力的强弱，$\alpha=f(ND)$表示攻击者达到最大攻击成功概率的增长率所需时间，代表漏洞利用的难易程度。函数$f(\cdot)$为漏洞利用难易程度ND至参数α的映射。

基于上述模型，对于边$e=(N_a,N_b)$，$N_a\neq N_b$，采用防御成功概率表示该边的权重$D(e)$。当N_a与N_b同属于同一个微服务副本时，即N_a与N_b分别属于同一副本的应用层攻击面和虚拟化层攻击面时，攻击者可直接进行横向移动，无需进行漏洞利用，因此此时防御成功概率为0。当N_a与N_b网络不可达时，攻击者无法进行横向移动，因此防御成功概率为1。当网络可达且需要通过漏洞利用才能实现横向移动时，假设攻击者充分利用微服务副本静态的时间开展攻击，则防御成功概率可基于式(3)进行求解。综上，边的权重$D(e)$具体可表示为

$$D(e)=\begin{cases}0，\ N_a\text{与}N_b\text{属于同一个微服务副本}\\ 1，\ N_a\text{与}N_b\text{网络不可达}\\ 1-p_{\mathrm{att}}(T_b)，\ N_a\text{通过漏洞利用移动至}N_b\end{cases} \tag{4}$$

其中，T_b表示节点N_b的清洗周期。

3.2 问题描述

对于包含M个微服务的应用$S=\{s_1,s_2,\cdots,s_M\}$，假设每个微服务有自身的清洗周期，应用整体的安全配置可表示为$H=\{T_1,T_2,\cdots,T_M\}$。当微服务副本数量及每个副本所运行的节点确定后，便可基于MAG模型生成图$G=(N,E)$。假设$PN=\{n_1,n_2,\cdots,n_W\}$，$\mathrm{PN}\subset N$是云环境下需要保护的节点集合，攻击者会选择最容易的攻击路径对目标进行攻击。因此，攻击者达成攻击目标的难易程度可以采用图G中攻击者到攻击目标的最短距离进行衡量。相应地，系统的安全性能也能够由攻击者实现攻击目标的难易程度表示。然而，由于防御方无法感知攻击者的攻击目标，因此考虑采用图G中攻击者到所有目标最短距离的平均值刻画系统的安全性指标η，具体可表示为

$$\eta=\frac{1}{W}\sum_{n\in PN}\sigma_G(I,n) \tag{5}$$

其中，$\sigma_G(I,n)$表示图G中从攻击者I到攻击目标n的最短路径。

为了提高系统的安全性，显然动态清洗策略的周期越小，攻击者越难以完成攻击，系统的安全性也就越高。然而，动态清洗策略也会给系统的性能带来影响，频繁的动态清洗会影响应用的服务质量(Quality of Service, QoS)。因此，考虑采用单位时间动态清洗次数作为防御策略的开销指标cost，具体可表示为

$$\mathrm{cost}=\sum_{k=1}^{i}\frac{1}{T_k} \tag{6}$$

为了最优化系统的防御配置，定义防御效率DE为系统安全性与防御开销的比值。同时，以最优化DE为目标，最优化防御配置H，可实现对云环境下关键攻击面的重点防御，防御资源精准配置。该优化问题可表示为

$$\begin{aligned}&\max\frac{\eta}{\mathrm{cost}}\\ &\text{s.t. }T_i\in H,T_{\min}\le T_i\le T_{\max}\end{aligned} \tag{7}$$

相比于文献[11]，本文的优化问题计算复杂度大幅提高。当图G下微服务数量为M，节点数量为$|N|$时，给定一个固定的防御配置时，使用最短路径算法如Dijkstra算法求解EM的计算复杂度为$O\left(|N|^2\right)$。假设对于每个微服务，防御策略的配置有F种选择。因此，对于图G，遍历求出最优的计算复杂度为$O\left(F^M|N|^2\right)$。

3.3 马尔可夫决策过程

MDP是序贯决策的数学模型，可由5元组

$\mathcal{M}=\langle\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R},\gamma\rangle$表示[15]。其中，$\mathcal{S}$表示有限的状态集合，$\mathcal{A}$表示有限的动作集合，$\mathcal{P}(\mathcal{S}_{t+1}|\mathcal{S}_t,a)$表示在状态$\mathcal{S}_t$下，执行体动作$a\in\mathcal{A}$使得状态变化为$\mathcal{S}_{t+1}$的概率。$\mathcal{R}(\mathcal{S}_t,a)$表示在状态$\mathcal{S}_t$下执行动作$a$的收益值。$\gamma\in[0,1]$是折扣因子，用于控制未来收益与当前收益的折中。强化学习的目的是求解出由状态$\mathcal{S}_t$映射至动作a的策略函数$\pi^*(a|\mathcal{S}_t)$，使得累计回报值最大。其中，累计回报值由贝尔曼方程给出[15]。

本文的安全配置优化问题可转化为MDP问题。将连续时间以固定时间间隔Δt进行抽样，在t时刻，智能体需要获取目前应用状态RS_t以及安全配置H_t，作为当前的状态输入，即$\mathcal{S}_t=\{\mathrm{RS}_t,H_t\}$。对于状态输出，可以对安全配置中的清洗周期以间隔$\Delta T$进行离散化抽样。每次执行动作时，先选取一个微服务，然后将其清洗周期增加或减少ΔT，或是维持配置不变。因此，对于M个微服务，动作集合中共有$(2M+1)$种可选动作。每次执行完动作后，需要对当前安全配置H_t进行更新。对于当前的收益，可使用优化问题中的防御效率DE进行衡量。

4 SmartSCR设计

本节主要论述SmartSCR的设计，包括其总体架构设计以及基于DQN的安全配置优化算法。

4.1 SmartSCR总体架构

SmartSCR基于容器云编排框架Kubernetes实现，其架构如图2所示。SmartSCR包括微服务状态监控模块、安全策略优化模块和微服务安全控制模块。微服务状态监控模块负责获取应用在集群中的运行状态信息。应用的运行状态指的是每个微服务的副本数量以及每个副本所运行的宿主机信息。微服务状态监控模块会将最新的状态发送给安全策略优化模块。安全策略优化模块负责根据应用的运行状态，生成最优的应用安全配置，其处理过程如下：安全策略优化模块将应用的运行状态和当前安全配置进行预处理后，作为DQN算法的输入数据传输到其神经网络的输入层。经过DQN算法对安全配置进行迭代优化并实现收敛后，将优化后的安全配置发送至微服务安全控制模块。微服务安全控制模块通过与Kubernetes进行交互，按照当前的安全配置对微服务进行管理。在微服务副本运行时间达到清洗时间后，对微服务副本执行清洗操作。在对副本进行清洗时，微服务安全控制模块会将每个微服务中副本的清洗事件放入一个独立的队列中。并且，只有当清洗事件执行完成后，才会从队列中取出并执行下一个清洗事件。通过该机制，保证了动态清洗策略不会影响服务的可用性。SmartSCR的核心在于基于DQN的安全策略优化模块能够根据输入的应用状态，迅速对安全配置进行优化。后续将对该基于DQN的自适应安全配置算法进行详细介绍。

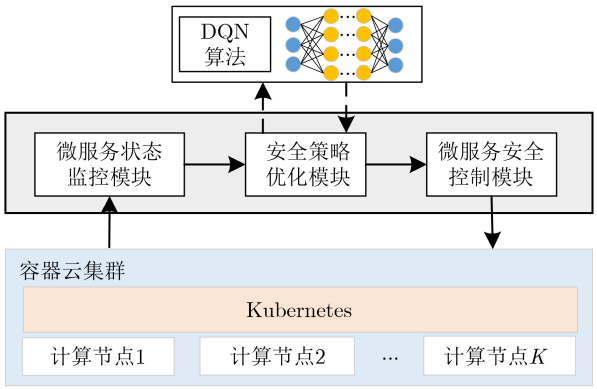


图 2 SmartSCR总体架构图

4.2 基于DQN的安全配置优化算法

为了解决上述MDP问题，传统研究中均采用基于值函数迭代的方式进行求解，具体包括基于蒙特卡罗的强化学习和基于时间差分的强化学习等方法[15]。在这些方法中，需要记录所有状态和动作对应的值函数，作为学习到的“经验”。当状态空间较大时，状态-动作值函数的记录、查询与迭代更新都会产生较大开销。为了实现在大规模状态空间进行强化学习，DQN算法将强化学习与深度学习结合，将状态作为神经网络的输入，将动作作为神经网络的输出，将在该状态下对应动作的输出值作为状态-动作值函数。通过不断与环境交互产生训练数据，训练神经网络，使其不断逼近当前环境的状态-动作值函数。

在DQN中，存在两个相互独立的神经网络：评估网络与目标网络，参数分别用θ与θ'表示。在t时刻，智能体在状态$\mathcal{S}_t$下执行了动作a_t，观察到收益为r_t且下一个状态为$\mathcal{S}_{t+1}$。4元组$\langle\mathcal{S}_t,a_t,r_t,\mathcal{S}_{t+1}\rangle$便是DQN从环境中学习到的“经验”。评估网络的参数会基于学习到的“经验”不断迭代更新，在第j次迭代时，其损失函数可表示为

$$L_j(\theta_j)=\mathrm{E}[y_j-Q(\mathcal{S}_j,a;\theta_j)]^2 \tag{8}$$

其中，$Q(\mathcal{S}_j,a;\theta_j)$表示神经网络参数为$\theta_j$时，输入$\mathcal{S}_j$下动作$a$对应是输出值；$y_j$是评估网络的学习目标，由当前收益与评估网络计算可得，具体可表示为

$$y_j=r_j+\gamma\max_{a'}Q(\mathcal{S}_{j+1},a;\theta') \tag{9}$$

为了提高DQN的稳定性与收敛速度，经验回放机制与定期更新机制被应用于DQN的训练。在经验回放机制中，会将DQN所学到的“经验”存到某

个存储空间中。假设经验复用池容量为D，当经验池中存储满时，才会开始对神经网络进行训练。每次训练迭代中，随机从存储中选取特定数量的“经验”用于神经网络的训练，以减少训练数据之间的相关性。在定期更新机制中，每经过固定的迭代步数，目标网络会拷贝评估网络的参数θ作为自身参数。

最优MTD策略求解过程如算法1所示。在SmartSCR中，DQN中神经网络采用前馈神经网络，使用应用的状态作为神经网络的输入，离散化后的MTD策略参数作为神经网络的输出。算法中主要涉及DQN与环境交互的接口，状态、动作和收益值。具体设计如下：

(1)状态。状态数据由应用的运行状态与安全防护配置组成。为了便于神经网络处理，假设集群中有UN个计算节点，共有M个微服务，第i个微服务的副本数为UR_i，则第i个微服务的运行状态$\mathrm{RMS}_i=[\mathrm{ind}_1,\mathrm{ind}_2,\cdots,\mathrm{ind}_{\mathrm{UR}_i}]$。其中$\mathrm{ind}\in[1,\mathrm{UN}]$且为整数，指的是该副本所在的计算节点的编号。应用的运行状态可由所有微服务的运行状态组成，即$\mathrm{RS}=[\mathrm{RMS}_1,\mathrm{RMS}_2,\cdots,\mathrm{RMS}_M]$。组合当前应用的运行状态和安全配置，便可得到输入的状态数据，即$\mathcal{S}_t=\{\mathrm{RS}_t,H_t\}$。为了产生大量训练数据，可随机生成每个微服务的副本数量，并按照容器云平台调度策略模拟每个副本所调度到的计算节点，由此作为当前迭代的应用状态。同时，安全防护策略也可通过随机的方式生成。

(2)动作。DQN算法的动作取决于输出层取值。在每次迭代中，可以选择以ΔT为基本单位，增加或是减小某个微服务的跳变周期，或是保持安全配置不变。

(3)收益值。在计算当前收益时，可以基于输入的运行状态，生成MAG模型，并结合安全防护配置，计算出防御效率DE作为收益。

算法1　基于DQN的安全配置优化算法训练

输入：微服务间调用关系

输出：DQN神经网络参数θ_j

(1) 初始化经验复用池的容量D，最小批量经验数量L，网络更新步长W

(2) for episode in range(STEPS):

(3)　随机生成微服务防御配置H

(4)　随机生成每个微服务副本数量，并模拟调度器对副本进行调度；

(5)　基于防御配置与应用状态，生成输入$\mathcal{S}_t$

(6)　以ε的概率随机选择一个动作a_t，否则选择$a_t=\max_a Q(\mathcal{S}_t,a;\theta)$

(7)　基于动作a_t修改防御配置，得出下一个状态$\mathcal{S}_{t+1}$，并基于MAG模型计算对应的奖励r_t

(8)　在经验复用池中存储样本$\langle\mathcal{S}_t,a_t,r_t,\mathcal{S}_{t+1}\rangle$

(9)　从经验复用池中随机抽L个样本

(10)　使用式(10)和式(11)执行梯度下降

(11)　每W步更新目标网络参数θ

(12) End

(13) 获取最优微服务防御配置

5　实验与评估

5.1　实验环境与参数设置

本文在实际容器云环境中验证SmartSCR的有效性。本文采用容器云编排平台Kubernetes搭建容器云集群。集群共包括11台服务器，配置均为40核，64 GB内存，2 TB 磁盘。其中，10台服务器作为集群计算节点，1台服务器作为集群管理节点。同时，SmartSCR也运行在集群管理节点上。在该容器云集群上，部署了一个Web应用，该应用由4个微服务组成，相互间调用关系与图1相同。该应用的具体信息以及存在漏洞情况如表1所示。基于微服务攻击困难度，可以求解出不同清洗周期下的攻击成功概率，该映射需要对攻击者能力进行合理假设。本文参照文献[14]中对攻击者能力的假设，在不同清洗周期下，求出不同微服务的成功攻击概率。

同时，假设每个微服务最大副本数为50，并以此设计DQN算法的输入；当微服务副本数量不足50个时，DQN算法输入未使用到的部分则置为0。在实际环境中，微服务会按照请求强度调整自身副本数量。为了在实验中模拟实际生产环境，考虑微服务规模较大的场景，假设所有微服务都具有50个副本，并通过Kubernetes调度器将这些副本调度到计算节点上运行。假设攻击者的目标为修改该应用中Mysql的数据。在MAG模型生成时，Tomcat, Memcached和ImageMagick的每个副本，都在MAG模型中有着相对应的应用层节点和虚拟化层节点。而对于Mysql，由于其是有状态的微服务，多个Mysql副本并不会独立提供服务，而是会通过Mysql集群的方式提供服务。在该模式下，任意一个Mysql副本的数据被改动，都等价于攻击者实现了攻击目标。因此，本文考虑将多个Mysql副本调度到在同一个计算节点上，以最小化攻击面。同时，在MAG模型中，多个Mysql副本组成的Mysql集群仅等价于1个应用层节点以及1个虚拟化层节点。

对于DQN，神经网络的隐藏层采用2层全连接结构，隐藏层神经元数量分别设置为64和32。神经网络的学习率为5×10^{-4}，折扣因子$\gamma=0.9$，每次

训练抽取的最小批次数据$L=32$，目标网络的更新步长为500。

5.2 对比策略

在实验中，将本文所提出的SmartSCR分别与统一配置策略[7]、DSEOM[11]和最优策略进行对比，突出SmartSCR的优势。对比策略的详细信息如下：

(1) 统一配置策略对微服务动态周期的配置的问题进行了简化，假设所有微服务的动态周期相同，从而大大减小了问题求解的计算量，并可通过遍历的方式对动态周期进行求解；文献[7]中在实现动态清洗策略时便采用该策略对问题进行简化；

(2) 最优策略通过暴力搜索的方式，求出最优的防御策略配置，为各算法提供参考。

(3) DSEOM同样通过攻击图模型刻画不同微服务的攻击难度；然而，该策略重点是计算攻击图中的关键节点，并仅对关键节点进行防护。

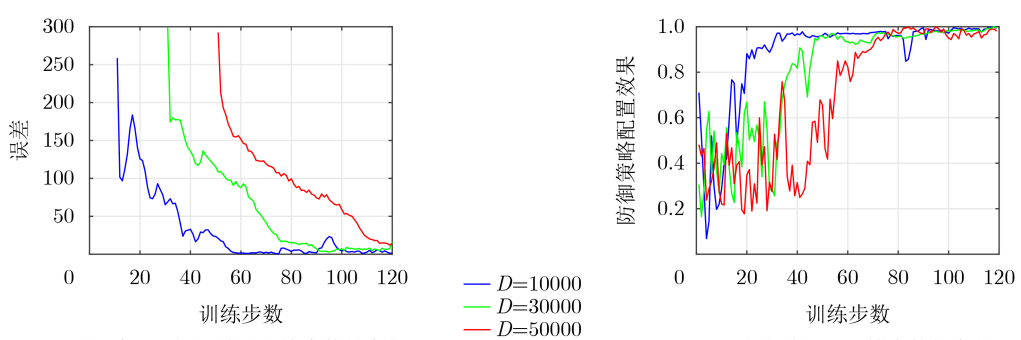
5.3 结果分析

为了验证SmartSCR的性能和有效性，本文首先随机生成了各个微服务的副本数量，并在实验环境中创建了该应用。基于微服务副本数量和Kubernetes调度结果，在不同的经验复用池容量下对DQN进行训练。

图3展示了神经网络误差和防御效率随训练次数的变化。其中，对于每个训练步长，DQN会与环境进行1000次交互。如图3(a)所示，在不同的经验复用池容量D下，神经网络都能够迅速实现收敛。当D取值较大时，可以存储较多探索阶段所收集的环境信息，使得后续对神经网络的训练更容易收敛，且使其不陷入局部最优。同时，由于DQN会首先填充满经验复用池后，才会从中随机选取经验数据进行训练，因此较大的D对DQN来说可能需要更长时间才能收敛，例如，当$D=10000$时，在DQN运行60步后实现了收敛；当$D=30000$时，在DQN运行90步后实现了收敛。同时，D取值同样不宜过小，因为较小的经验复用池容量可能导致DQN丢弃某些重要的“经验”，造成DQN收敛的不稳定。例如当$D=10000$时，在训练至90步时模型收敛发生了波动。图3(b)展示了所取得的防御效果随着训练步数的变化，其中，纵坐标为当前步数防御效率的平均值与最优防御效率的比值。在经

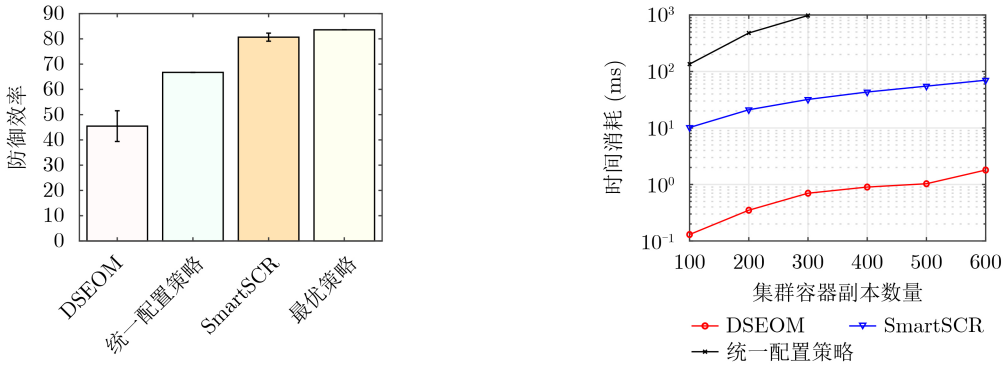
表 1 应用漏洞信息表

微服务	名称	漏洞编号	漏洞利用困难度	漏洞权重	微服务攻击困难度
A	Tomcat	CVE-2021-42340	0.2564	6.8	0.3518
		CVE-2021-30640	0.4545	6.0	
		CVE-2019-0221	0.3571	67.1	
B	Memcached	CVE-2016-8704	0.2564	8.3	0.314
		CVE-2016-8705	0.2564	8.3	
		CVE-2016-8706	0.4545	6.8	
C	ImageMagick	CVE-2017-14650	0.4545	6.8	0.4010
		CVE-2017-14224	0.3571	8.3	
D	Mysql	CVE-2020-11974	0.2564	7.3	0.4179
		CVE-2016-6663	1.0000	4.3	
		CVE-2016-6662	0.2564	8.2	
–	Container	CVE-2021-437847	0.6250	7.6	0.4483
		CVE-2020-35197	0.2564	7.0	



(a) 神经网络误差随训练步数的变化 (b) 防御效果随训练步数的变化

图 3 DQN学习过程示意图



(a) 不同算法防御效率对比　　(b) 不同算法时间消耗对比

图 4 不同算法下的对比

验复用池未填充满时，DQN会采用随机的策略对防御配置进行修改，因此防御效率也一直进行波动。当基于积累的“经验”开始对DQN进行训练后，防御效率会迅速提升，最终收敛至最优防御效率附近。DQN网络在收敛后防御效率仍会有小幅度的波动，这是因为在DQN中为了实现探索-利用的平衡，会采用ε-贪婪策略，即会以概率ε随机选择动作，以概率$1-\varepsilon$按照神经网络的输出选择最优动作。

图4将SmartSCR与不同算法进行对比。其中，图4(a)展示了SmartSCR将与统一配置策略、DSEOM和最优防御策略所取得的防御效率的对比。其中，SmartSCR的防御效率选取$D=30000$时训练收敛后所得的防御效率。如图4所示，DSEOM所取得的平均防御效率最低，且其防御效率方差最大。其主要原因在于DSEOM仅对云环境下的关键目标进行防护，而一旦攻击者找到路径绕过该关键目标，便可以轻松地实现攻击目标，云原生环境下爆炸增长的攻击面也为攻击者绕过重点布防的节点提供了遍历；同时，DSEOM在确定关键目标之后，MTD策略的参数配置需要由防御者自主选择，参数配置的盲目性使得该策略的防御效果不稳定。统一配置策略通过简化配置，使得能够通过遍历选取简化后最优的配置，相对于DSEOM，能够有效提升防御效率，并具有稳定性。本文所提SmartSCR在训练收敛后，取得的平均防御效率已经十分接近最优值，且方差较小，能够实现稳定的安全防御效果。图4(b)展示了SmartSCR与统一配置策略和DSEOM在不同应用规模下时间消耗的对比，用于衡量策略的可扩展性。其中，横坐标表示该应用中所有容器副本的数量，纵坐标表示求解出MTD配置的时间消耗。在该实验中，通过改变前文Web应用微服务副本的数量，从而改变整个应用的规模。同时，假设每个微服务副本的上限为200，对SmartSCR进行训练。由图可知，DSEOM所消耗的时间最短，且随着应用规模的增长时间消耗也未呈现指数级增长，其原因在于DSEOM需求解的计算量较小，且对最短路径的求解进行了优化。对于SmartSCR，由于其可在线下提前对DQN模型进行训练，因此在线上对MTD策略进行配置的时间消耗也较短，并且能够应对较大规模应用的场景。而对于统一配置策略，由于应用规模的增长大幅增加了求解最短路径的时间消耗，时间消耗随着应用规模的增大也大幅增长，适合应用于小规模应用的场景。

6　结束语

本文针对云原生环境下移动目标防御策略的最优配置展开研究。首先，针对云原生环境下复杂的攻击场景进行建模，并对MTD策略下的防御策略进行定量衡量。然后，以最优化防御效率为目标，提出了MTD策略的最优配置问题。由于云原生应用的规模以及较大的安全配置空间，直接求解最优配置十分困难。针对该问题，本文将其转化为MDP问题，并提出SmartSCR方案对该问题进行求解。SmartSCR方案中，基于MAG模型对防御效率进行求解，并使用DQN算法应对状态空间爆炸的问题，求解最优防御配置。实验表明，SmartSCR能够有效应对云原生场景下的安全配置优化问题。

参 考 文 献

[1] 中国信息通信研究院. 云计算白皮书[R]. 中国信息通信研究院, 2021.
China Academy of Information and Communications Technology. Cloud computing white paper[R]. China Academy of Information and Communications Technology, 2021.

[2] ZHOU Xiang, PENG Xin, XIE Tao, *et al*. Fault analysis and debugging of microservice systems: Industrial survey,

benchmark system, and empirical study[J]. *IEEE Transactions on Software Engineering*, 2021, 47(2): 243–260. doi: 10.1109/TSE.2018.2887384.

[3] KHAN M G, TAHERI J, Al-DULAIMY A, *et al*. PerfSim: A performance simulator for cloud native microservice chains[J]. *IEEE Transactions on Cloud Computing*, To be published. doi: 10.1109/TCC.2021.3135757.

[4] AROUK O and NIKAEIN N. Kube5G: A cloud-native 5G service platform[C]. 2020 IEEE Global Communications Conference, Taipei, China, 2020: 1–6. doi: 10.1109/GLOBECOM42002.2020.9348073.

[5] GAO Xing, STEENKAMER B, GU Zhongshu, *et al*. A study on the security implications of information leakages in container clouds[J]. *IEEE Transactions on Dependable and Secure Computing*, 2021, 18(1): 174–191. doi: 10.1109/TDSC.2018.2879605.

[6] NIFE F N and KOTULSKI Z. Application-aware firewall mechanism for software defined networks[J]. *Journal of Network and Systems Management*, 2020, 28(3): 605–626. doi: 10.1007/s10922-020-09518-z.

[7] BARDAS A G, SUNDARAMURTHY S C, OU Xinming, *et al*. MTD CBITS: Moving target defense for cloud-based IT systems[C]. The 22nd European Symposium on Research in Computer Security, Oslo, Norway, 2017: 167–186. doi: 10.1007/978-3-319-66402-6_11.

[8] LU Kangjie, SONG Chengyu, LEE B, *et al*. ASLR-guard: Stopping address space leakage for code reuse attacks[C]. The 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, USA, 2015: 280–291. doi: 10.1145/2810103.2813694.

[9] LARSEN P, BRUNTHALER S, DAVI L, *et al*. Automated Software Diversity[M]. Morgan & Claypool, 2015: 1–8. doi: 10.2200/S00686ED1V01Y201512SPT014.

[10] MEIER R, TSANKOV P, LENDERS V, *et al*. NetHide: Secure and practical network topology obfuscation[C]. The 27th USENIX Security Symposium, Baltimore, USA, 2018: 1–18. doi: 10.5555/3277203.3277256.

[11] JIN Hai, LI Zhi, ZOU Deqing, *et al*. DSEOM: A framework for dynamic security evaluation and optimization of MTD in container-based cloud[J]. *IEEE Transactions on Dependable and Secure Computing*, 2021, 18(3): 1125–1136. doi: 10.1109/TDSC.2019.2916666.

[12] GLUCK A. Introducing domain-oriented microservice architecture[EB/OL].https://eng.uber.com/microservice-architecture, 2021.

[13] NIST. National vulnerability database[EB/OL]. https://nvd.nist.gov/vuln, 2021.

[14] PENG Wei, LI Feng, HUANG C T, *et al*. A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces[C]. 2014 IEEE International Conference on Communications, Sydney, Australia, 2014: 804–809. doi: 10.1109/ICC.2014.6883418.

[15] 邱航, 汤红波, 游伟. 基于深度Q网络的在线服务功能链部署方法[J]. 电子与信息学报, 2021, 43(11): 3122–3130. doi: 10.11999/JEIT201009.

QIU Hang, TANG Hongbo, and YOU Wei. Online service function chain deployment method based on deep Q network[J]. *Journal of Electronics & Information Technology*, 2021, 43(11): 3122–3130. doi: 10.11999/JEIT201009.

张 帅：男，博士生，主要研究方向为云计算、网络安全.

郭云飞：男，教授，博士生导师，主要研究方向为新型网络体系结构、电信网安全、云计算.

孙鹏浩：男，博士，讲师，主要研究方向为新型网络体系结构、强化学习.

程国振：男，博士，硕士生导师，主要研究方向为新型网络体系结构、网络安全.

扈红超：男，研究员，博士生导师，主要研究方向为云计算、网络安全.

责任编辑：马秀强