


## ORIGINAL RESEARCH

# An optimal defensive deception framework for the container-based cloud with deep reinforcement learning

Huanruo Li<sup>1</sup>  | Yunfei Guo<sup>1</sup> | Penghao Sun<sup>1,2</sup> | Yawen Wang<sup>1</sup> | Shumin Huo<sup>1</sup>
<sup>1</sup>Department of Computer Science, National Digital Switching System Engineering and Technological Research Center, Zhengzhou, Henan, China

<sup>2</sup>Department of Communication Technologies, Academy of Military Science, Beijing, China

## Correspondence

Huanruo Li, Department of Computer Science, National Digital Switching System Engineering and Technological Research Center, Zhengzhou, Henan, China.

Email: [viavialhr@outlook.com](mailto:viavialhr@outlook.com)

## Funding information

The Foundation for Innovative Research Groups of the National Natural Science Foundation of China, Grant/Award Number: 61521003; National Natural Science Foundation of China, Grant/Award Numbers: 62002383, 62072467

## Abstract

Defensive deception is emerging to reveal stealthy attackers by presenting intentionally falsified information. To implement it in the increasing dynamic and complex cloud, major concerns remain about the establishment of precise adversarial model and the adaptive decoy placement strategy. However, existing studies do not fulfil both issues because of (1) the insufficiency on extracting potential threats in virtualisation technique, (2) the inadequate learning on the agility of target environment, and (3) the lack of measurement for placement strategy. In this study, an optimal defensive deception framework is proposed for the container based-cloud. The System Risk Graph (SRG) is formalised to depict an updatable adversarial model with the automatic orchestration platform. Afterwards, a Deep Reinforcement Learning (DRL) model is trained based on SRG. The well-trained DRL agent generates optimal placement strategies for the orchestration platform to distribute decoys and deceptive routings. Lastly, the coefficient of deception,  $\mathbb{C}$ , is defined to evaluate the effectiveness of placement strategy. Simulation results show that the proposed method increases  $\mathbb{C}$  by 30.22%, and increase the detection ratio on the random walker attacker and persistent attacker by 30.69% and 51.10%, respectively.

## KEYWORDS

artificial intelligence, cloud security, computer network security, cyber deception defence, decoy placement strategy, deep reinforcement learning

## 1 | INTRODUCTION

The advent of container-based virtualisation technology and microservice architecture facilitates the rapid development, deployment and automatic scaling of cloud-enabled applications [1]. However, this extends the attack surface of the cloud [2]. On one hand, virtualisation introduces particular threats other than system and software vulnerabilities in the intranet [3, 4]. On the other hand, the frequent update and scaling of microservices increase the dynamism of cloud and complicate its boundaries [5]. More importantly, the evolving attack tactics and evasion techniques enable attackers to bypass entry-point-based countermeasures (i.e. Intrusion Detection System [IDS]) and exploit on internal services [6].

Defensive deception emerges as a distinct countermeasure for the cloud. It reverses the asymmetry between the defender and attacker by presenting intentionally falsified information. Different from signature-based countermeasures, it actively conceals system properties [7], misleads and traps attackers [8]. For example, Almohri et al. [9] propose to conceal target nodes in cloud by organising decoy services. To deter and detect stealthy attackers in the container-based cloud, defensive deception is particularly advantageous from following perspectives: (1) It actively reveals stealthy attackers by presenting exploitable decoys [8]. Decoys are well-designed digital assets with monitors. They reside in specific locations to attract exploits, so as to discover stealthy attackers during their lateral movement; (2) Decoys could be developed from and distributed into any digital asset across the target environment.

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.

© 2021 The Authors. *IET Information Security* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

Different from deploying IDS as the entry-point, defensive deception obfuscates attackers on the reconnaissance stage and increases the probability to trap them [10]; (3) The flexibility of resource management in cloud facilitates the realisation of adaptive deception strategy. When traditional countermeasures are demanding to configure in frequent update internal services, defensive decoys can be rapidly deployed by emerging automatic orchestration platforms (i.e. Kubernetes and Docker Swarm) [6, 11].

Main concerns of implementing defensive deception in the cloud are the establishment of adversarial model [8] and the placement of decoy assets [12]. The precise adversarial model is the foundation to create and evaluate placement strategy. The adaptive decoy placement strategy guarantees the effectiveness of defence, however, system updates. For the adversarial model, existing studies mostly use the attack graph to denote risks of applications [13, 14], while neglecting specific threats of virtualisation (i.e. container escape) in the cloud [11]. For the placement strategy, existing studies mainly focus on game theory model [15–17], probabilistic model [18] and sidecar strategy (i.e. deploy along with valuable assets, sensitive data and risky applications) [19–21]. In conclusion, the major limitations of existing studies on defensive deception for cloud are as follows: (1) Adversarial models are mostly directly shifted from the intranet, which fail to address particular risks in the cloud. Moreover, threat models for the intranet tend to study static system, changing risks in frequent update cloud is not discussed; (2) Existing defensive deception frameworks mainly apply static decoy placement strategies, deriving from equilibriums or attack graph models. These methods do not consider the agility of target environment and do not satisfy the adaptivity issue; (3) The evaluation metric of placement strategy is not comprehensive to include multiple defence characteristics, which hinders the update and re-deployment of strategy in the dynamic cloud.

In this study, we propose An Optimal Defensive Deception Framework for the Container-based Cloud with Deep Reinforcement Learning (DRL) to counter stealthy attackers. In following context, the cloud refers to the container-based cloud. The main contributions of the paper are as follows:

- 1) In the framework, we present a specific adversarial model for container-based cloud, an adaptive decoy placement strategy and a feasible implementation design.
- 2) We propose a precise adversarial model for the cloud, namely, the System Risk Graph (SRG). It comprehensively extracts risks of the cloud applications at  $t$ -time. Also, we define the coefficient of deception,  $\mathbb{C}$ , as a comprehensive quantitative metric to evaluate the deceptive effectiveness of the placement strategy.
- 3) A DRL algorithm is customised for the cloud orchestration platform. This enables not only the dynamic observation on the cloud but also the automatic generation and adaptive execution of decoy placement strategy based on  $\text{SRG}_t$ .
- 4) We run simulations on 1000  $\text{SRG}_t$  with up to 300 replicas and 60 computing nodes. On average, our method leads to

a 30.22% advantage, and a 73.46% decrease in variance of  $\mathbb{C}$  than existing approaches. To evaluate the detection ability, we run simulations on a random-walker attacker and a persistent attacker. The results exhibit our framework's increased detection ratio on both attackers by 30.69% and 51.10%, respectively.

The rest of the paper is organised as follows. Section 2 introduces the research background and motivation. Section 3 formalises the adversarial model for the cloud. Section 4 demonstrates an overview of the framework. Section 5 presents algorithm details and interface designs. Section 6 presents simulation results and analysis. Section 7 studies the related work. Finally, Section 8 concludes the work.

## 2 | BACKGROUND AND MOTIVATION

This section introduces the background and motivation of our work.

### 2.1 | Lateral movement in the container-based cloud

The microservice architecture decouples monolithic applications into 'micro' services according to functions [4]. Credit to the lower virtualisation overhead and shorter start-up time of the container [22], the microservice architecture enables autonomous development and flexible scaling of the cloud [1]. An application may include up to a hundred microservices, and a single microservice can be updated by hundred times a day [23]. However, this brings extra security issues [2]. On one hand, virtualisation technology introduces extra threats [3]. On the other hand, the aforementioned characteristics of microservices increase dynamism of the cloud and complicate its boundaries [5]. Traditionally, countermeasures, such as the IDS, are implemented at the edge of network to detect attacks. Due to the complicated configuration and demanding monitor, IDS will not be placed on every microservice considering resource efficiency. Consequently, the ever-expanding attack surface and increasingly sophisticated evasion techniques provides opportunities to stealthy attackers. Once evading entry-point security mechanism, they are empowered to perform lateral movement across the internal network [24]. For example, remote attackers could leverage vulnerabilities to pivot on open services and then exploit on more internal services.

### 2.2 | Defensive deception for cloud system

As a distinct line, defensive deception defends target system by presenting intentionally falsified information to camouflage system properties and mislead attackers. Cohen [25] firstly proposes to defend information system with deception. He proposed the Deception Toolkit with various tactics including

concealment, camouflage, and, false and planted information. Later, Spitzner propose the Honeypot system, where real computer systems host exploitable vulnerabilities to attract, log and learn unauthorised access [26].

To counter threats in the cloud, defensive deception is studied. Keromytis et al. [27] propose MEERKATS, a deception architecture, where cloud services are constantly changing to create unpredictable targets for the adversary. Brzezko et al. [28] present an active deception model to secure cloud infrastructures by re-routing potential intruders to decoys. Almohri et al. [9] propose to delay remote attackers by strategically placing decoys in the cloud network. Similarly, Pham et al. [29] design a deceptive topology for virtual network from the perspective of stealthy attackers.

Defensive deception is particularly promising to defend the cloud for following reasons: (1) It actively reveals stealthy attackers by presenting exploitable decoys [8]. Decoys are well-designed digital assets with monitors. They reside in specific locations to attract exploits, so as to discover stealthy attackers during their lateral movement; (2) Decoys could be developed from and distributed into any digital asset across the target environment. Different from deploying IDS as the entry-point, defensive deception obfuscates attackers on the reconnaissance stage and increases the probability to trap them [10]; (3) The flexibility of resource management in cloud facilitates the realisation of adaptive deception strategy. When traditional countermeasures are demanding to configure in frequent update internal services, defensive decoys can be rapidly deployed by emerging automatic orchestration platforms (i.e. Kubernetes and Docker Swarm) [6, 11].

Significantly, the effectiveness of deception defence hinges on the interaction with attackers. Hence, the *adversarial model* [8] and *placement* [12] of decoys are main concerns to deploy deception defence in the cloud:

- 1) *A precise adversarial model.* Lu et al. [8] point out that a precise adversarial model is the key to create and evaluate a deception strategy. As deception defence leverages attackers' cognitive biases [10], detailing adversarial model of target environment is essential for transferring the weakness of adversaries into defence advantage. For container-based cloud, the precision is described in two dimensions: time and range. *The time dimension* requires the adversarial model to extract changing threats in time. Because the container-based cloud updates and scales automatically under busy and massive user requests. Active applications vary per time-slot, which brings along changing risks and vulnerabilities. *The range dimension* refers to the variety of risks. On one hand, the cloud hosts shared vulnerabilities with the intranet. On the other hand, the virtualisation technology introduces unique threats. Thereby, covering overall risks is essential for the construction of the adversarial model.
- 2) *The adaptivity of decoy placement strategy.* As for the dynamic cloud, static placement strategy will be ineffective to various risk scenarios. Given the frequent updating and scaling of microservice, active applications and their

mappings to computing nodes changes constantly [11]. Hence, it is essential for the placement strategy of decoy to adapt to the highly dynamic cloud system [30].

## 2.3 | Limitation of existing works

According to the aforementioned characteristics, an optimal defensive deception for the container-based cloud should consider both the precise adversarial model for target environment and the adaptive decoy placement strategy. Unfortunately, existing studies on defensive deception framework for the cloud fail to address both issues jointly.

*For the adversarial model,* existing studies mainly take intranet attack graph to illustrate attack behaviours [9, 13, 15]. The rest of works include empirical 67 model based on assumed attack difficulty [28], probabilistic model [18] and game theory based interactive model surveyed in Ref. [31]. The limitations of existing works are as follows:

- 1) Existing models do not address the constant update issue of the adversarial model for the dynamic cloud;
- 2) They depict risks and vulnerabilities in the intranet, while neglecting particular threats and their attack difficulty brought by virtualisation technology.

*For the decoy placement strategy,* existing works can be broadly classified as follows: (1) Game theory-based placement strategies [13, 16, 17]. However, Ferguson-Walter et al. [32] point out that major usage of one-shot game is unable to capture the multi-step interaction between the defender and attacker. Afterwards, Horak et al. [15] leverage partially observable stochastic games to optimise honeypot allocation against dynamic lateral movement. (2) Empirical placement strategies. This category indicates the rest of mainstream placement strategies where decoys are positioned alongside valuable assets [9, 19], sensitive data [21] and risky applications [20]. The limitations of existing approaches are that:

- 1) The gap between equilibrium-based decoy placement strategy and real world deployment. Especially, the computational complexity in solving high-dimensional input data space, such as the instant condition and threat model of the dynamic cloud network.
- 2) The empirical strategy does not constantly generate the global optimal result and may not be applicable to a wider range of target environments.
- 3) Due to the lack of measurement metric, existing placement strategies do not address the issue of update and re-deploy per changes of cloud.

## 2.4 | Design considerations

Based on aforementioned analysis, we derive that an optimal defensive deception framework for the container-based cloud should build upon the precise adversarial model and the

adaptive decoy placement strategy. Hence, we consider the following requirements:

- 1) The adversarial model should involve the overall risks and vulnerabilities in the target cloud system, ranging from application layer to virtualisation layer.
- 2) The adversarial model is updatable to extract the real-time risk status of the target system, which lays the foundation for generating adaptive decoys placement strategy.
- 3) The placement strategy for decoys should be adaptive to the dynamic cloud and scalable to the expanding application replicas and computing nodes.

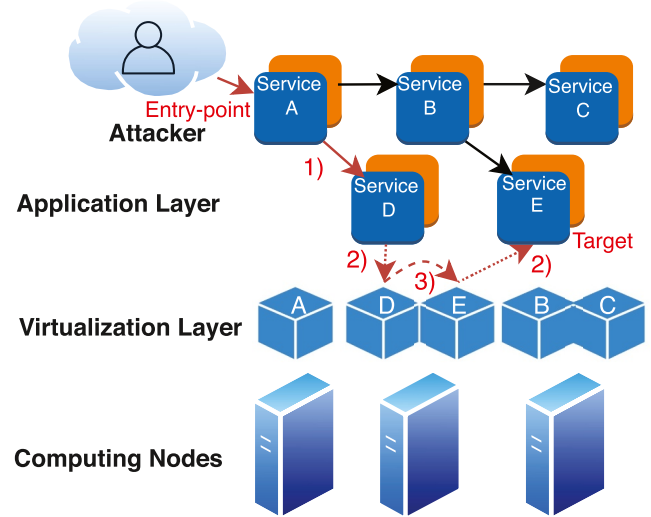
## 2.5 | Opportunities

Emerging deception Machine Learning (ML) techniques facilitate the automation in cybersecurity. For example, Ayoade et al. [33] leverage Deep Learning to automate the generation and update of deception traffic stream without human in the loop. For optimal decision control in complicated environment, the DRL has its specific potential: (1) DRL is a combination of Deep Neural Network (DNN) and Reinforcement Learning (RL). The RL learns from a trial/error interaction in a dynamic environment. Combined with DNN, the DRL performs better to approximate the high-dimension input data with control decisions. For instance, Sethi et al. [34] apply DRL to detect intrusions in the cloud infrastructure. By receiving Virtual Machines' (VMs) logs as input data, the DRL agent classifies attacks and generates optimal intrusion responses. In this study, we leverage DRL to generate adaptive decoy placement strategy by analysing the real-time adversarial model of container-based cloud. (2) The model training of DRL does not require labelled dataset and the input data can be formalised by real-time information extracted from orchestration platforms.

## 3 | ADVERSARIAL MODEL

In the container-based cloud with microservice architecture, some services are available for open access while others are internal services to execute back-end operations. The internal network connects front-end and back-end services. We assume the cloud platform and service provider are trusted. The adversary is a remote attacker from external network. The adversary's goal is to penetrate into the internal network to collect important data or expand its foothold [35]. We present a sample architecture of the container-based cloud in Figure 1. The attack targets, tactics and capability are formulated as follows.

**Attack target:** There are multiple assets in the container-based cloud. The application layer refers to code and dependency of microservices. The virtualisation layer refers to the running environment of these codes, namely, containers. We assume the orchestration tools and computing nodes are secured from attacks. The attack targets are entry-point services, internal microservices, and containers hosting them. For



**FIGURE 1** The architecture of a container-based cloud. If Service A is the entry point and Service E is the target, the arrows and dashed arrows in red represent three types of possible attack paths. The edges' types marked by 1), 2) and 3) refer to three attack scenarios in Def 1

example, in Figure 1, Service A is an entry-point service. The host containers of service replicas are depicted by the labelled cubes.

**Attack tactics:** Generally, the adversaries conduct lateral movement by exploiting on well-known vulnerabilities in open services [36]. It is reasonable to assume in the adversarial model that the attacker pivot through the internal network from a compromised microservice by Remote Code Execution (RCE) on its neighbours.

**Attack capability:** We assume the adversary are from the external network and launches attack from a single entry-point. The attacker is capable of scanning the network, conducting reconnaissance and performing fingerprinting. Based on these, he could extract attack difficulty of targets. The attacker behaves stealthy and aims to stay undetected. He either chooses random attack paths to expand his foothold [18] or persists on specific targets [9]. Hence, on one hand, he uses evasion techniques to bypass internal firewalls. On the other hand, he follows the most exploitable paths to reach the target sooner. But he is constrained from acknowledging the configuration details of the cloud, including the orchestration policy, the map between service replicas to computing nodes and security configurations in each container and computing nodes.

We propose the SRG, an acrylic digraph, to formulate the aforementioned adversarial model. The notations used in the Adversarial Model are summarised in Table 1.

### Def 1 System Risk Graph

Let  $SRG_t = (\lambda, N, E, \mathcal{V}_N)$  denote the SRG of cloud at time- $t$ , all variants in SRG are  $t$ -relevant:

- $\lambda$  denotes a set of computing nodes in the cloud, either physical machines or VMs.



**TABLE 1** Summarisation of notations in the adversarial model

Notation	Description
$SRG_t$	The system risk graph of cloud at time-slot $t$
$A$	A set of computing nodes in the cloud
$N = N_S \cup N_C$	A set of vertex in $SRG_t$ denoting attack targets
$N_S$	A subset of vertex denoting microservices in the cloud
$N_C$	A subset of vertex denoting containers
$E$	A set of weighted-directed edges in $SRG_t$
$\mathcal{V}_N$	A set of vulnerabilities hosted by $N$ in $SRG_t$ . For $n \in N$ , $\mathcal{V}_n$ denotes a finite set of exploitable vulnerabilities on it.
ED	Exploit difficulty
$AP(s, t), s, t \subseteq N$	A set of attack paths
ED'	Observed exploit difficulty
$AP'(s, t)$	Observed attack paths
$\mathbb{C}$	Coefficient of deception

- $N = N_S \cup N_C$ , is a finite set of vertex denoting attack targets.  $N_S = \{n_s | s \in S\}$  is a set of vertex denoting both external and internal services, where  $S$  is a finite set of microservices in the cloud.  $f(n_c^i) : N_C \rightarrow N_S \times \lambda$  denotes containers hosting corresponding service replicas on orchestrated computing nodes.
- $E = \{e(\nu, \mu) | \mu, \nu \in N, \mu \neq \nu\}$  denotes a set of weighted-directed edges in the  $SRG_t$ , which connect attack targets. The edges illustrate following three attack scenarios: (1) Remote attacker exploits on the neighbour vertex of a compromised target, which are connected by system configurations; (2) The attacker exploits on the host container from a compromised microservice; and (3) The attacker exploits other containers locate on the same computing node.
- $\mathcal{V}_N = \{\mathcal{V}_n | n \in N\}$  denotes a set of vulnerabilities hosted by attack targets  $N$  in  $SRG_t$ . For each attack target  $n \in N$ ,  $\mathcal{V}_n$  denotes a finite set of exploitable vulnerabilities on it.

## Def 2 Exploit Difficulty

Exploit Difficulty (ED) denotes the difficulty to compromise an attack target by exploiting on one or several vulnerabilities on it. We use Exploitability Metrics (EM) and Exploit Code Maturity (ECM) in Ref. [37] to calculate the overall ED for different attack scenarios. The EM quantifies the exploitability of each vulnerability in the system and the ECM measures the likelihood of vulnerability being attacked. ECM varies per involvement of attack techniques. In the SRG, ED depicts the weight of weighted-directed edges  $E$ .

First, we calculate the EM by definition [37]:

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

where AV, AC, PR and UI are the components of EM. They formulate the EM and reflect the difficulty and technical means by which the vulnerability can be exploited [37]. In detail, AV, AC, PR and UI refer to Attack Vector, Attack Complexity, Privileges Required, and User Interaction, respectively. *Attack Vector*, AV, reflects the context by which vulnerability exploitation is possible. For example, when the vulnerable components are bound to the network stack, some are remotely exploitable while others are limited to logically adjacent topology. *Attack Complexity*, AC, describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. For example, for some vulnerable components, the attacker can expect repeatable success, while for the other, the successful attack cannot be accomplished at will but requires measurable amount of preparation or execution. *Privileges Required*, PR, refers to the level of privileges an attack must possess before successfully exploiting the vulnerability. For example, the attacker is able to exploit on some vulnerable components and launch attacks without authorised prior (i.e. cross-site request forgery). While for the others, a successful attack may require either users' or overall-control privileges. And the *User Interaction*, UI, represents the requirement for a human user, other than the attacker, to participate in the successful compromise of the vulnerable component. For example, some vulnerable components can be exploited solely at the will of the attacker (without any user), while others require the effort of human users to launch a successful attack.

Afterwards, we obtain the ED of an attack target by calculating the weighted sum EM of a set of  $\mathcal{V}_n$ . To reason this: (1) Generally, a target hosts more than one vulnerability and each is likely to be exploited up to attackers' capability and preference. (2) Engaging ECM into calculation is to consider the involvement of attack tactics. Thereby, we use ECM as the weight to depict ED of an attack target:

$$ED = \frac{\sum_{\mathcal{V}_n} (EM * ECM)}{\sum_{\mathcal{V}_n} ECM} \quad (2)$$

As the aforementioned assumption that attackers follow the most exploitable paths, we use the weighted shortest paths to depict attack behaviours. Let  $AP(s, t) \subseteq E, s, t \subseteq N$  denote a set of shortest paths connecting a set of entry-point vertex  $s$  to a set of targets  $t$ .

*Defender's strategy*: The defender's strategy is to allocate decoys on target vertex  $N$ , so as to impact on the attackers' judgement on ED and  $AP(s, t)$ . The attacker is capable of collecting primary information by scanning, reconnaissance and fingerprinting across the network. When decoys are implemented in the cloud, such information may not always reflect real properties but intentional misguidance. We denote information collected after decoys are positioned as *Observed Properties*. Hence, we define the Observed Exploit Difficulty (ED') and the Observed Attack Paths  $AP'(s, t)$ .

**Def 3** *Observed Exploit Difficulty (ED') and Observed Attack Paths AP'(s, t)*

As decoys attract attackers by showing exploitable vulnerabilities, the Observed Exploit Difficulty (ED') presents a lower value than the real Exploit Difficulty (ED). By this, the Observed Attack Paths AP'(s, t) are different from the real Attack Paths (AP). Such that,

$$\begin{aligned} ED' &\neq ED \\ AP' &\neq AP \\ AP' &= AP \cup AP^+ \cap AP^- \end{aligned} \quad (3)$$

where  $AP^+ \neq AP^- \neq \emptyset$ .  $AP^+$  refers to a set of deceptive attack paths introduced by decoys and  $AP^-$  denotes the subset of real attack paths being concealed. Note that,  $ED' = ED$  only when decoys are not placed.

**Def 4** *Coefficient of deception, (C)*

We define the coefficient of deception, C, to evaluate the implementation of defensive deception quantitatively. C is composed of two terms: (1) the concealment of system properties, and (2) the increase of misleading information. We use  $AP^-$  and  $AP^+$  to denote those terms, respectively. Such that

$$\begin{aligned} C &= \frac{|(AP^+ \cup AP^-) - (AP^+ \cap AP^-)|}{|AP|} \\ &= \frac{|AP^+| + |AP^-|}{|AP|} \end{aligned} \quad (4)$$

where each variant is related to time- $t$ , and  $|AP|$  is divided to normalise the coefficient. According to existing studies [7, 10, 12], the effective defensive deception should consider the concealment of system properties and the presentation of wrong information (to the attacker). However, recent studies mainly apply independent metrics to evaluate single characteristic. We establish C as a comprehensive metric to cover both of the aforementioned characteristics based on Attack Paths AP(s, t).

As AP(s, t) are considered as the quantitative presentation of system properties, indicating a series of exploitable vulnerable components in the system. The Observed Attack Paths

AP'(s, t) and the Attack Paths AP(s, t) are used to differentiate system prosperities with and without implementing defensive deception. Thereby, the concealment of system properties could be presented by the decrease of real Attack Paths,  $AP^-$ , and the increase of misleading information could be presented by the additional Attack Paths introduced by decoys,  $AP^+$ . In this way, C is capable of comprising different defence characteristics and formulating a comprehensive metric to quantitatively evaluate defensive deception.

## 4 | Defender's goal

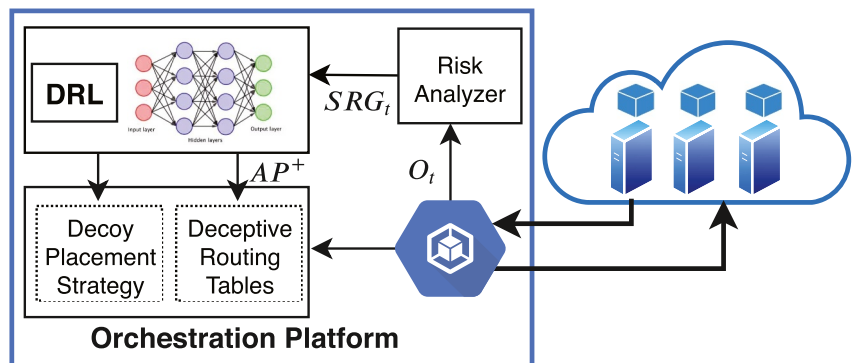
The defender's goal is to generate optimal decoy placement strategies to maximise C, so as to maximise the effectiveness of defensive deception. To reach a max C, the defender should conceal maximum attack paths  $AP^-$  and generate maximum falsified attack paths  $AP^+$ .

## 5 | DESIGN OVERVIEW OF THE FRAMEWORK

In this study, we propose An Optimal Defensive Deception Framework for the Container-based Cloud with DRL. It is intended to be implemented on the orchestration platforms of the cloud. The workflow of the framework is illustrated in Figure 2. The proposed framework generated updatable adversarial model, decoy placement strategy and decoy routing tables for the cloud. The objective is to maximise C in Equation (4).

Primarily, the orchestration platform (i.e. Kubernetes) extracts the real-time orchestration details  $O_t$  and send to *Risk Analyser*, which formulates the System Risk Graph  $SRG_t$ . Generally, for the WEB scenario, once a new request is received, the Orchestration Platform decides serving with available replicas or expanding new containers. Also, it is capable of forwarding and re-routing traffics between replicas.

Afterwards, the  $SRG_t$  is sent to the input neurons of the DRL agent. By the calculation of neural networks, the DRL agent generates a list of actions to formulate the placement strategy. Each action in the list denotes a vertex to place decoys in  $SRG_t$ , corresponding to decoy microservice replicas or



**FIGURE 2** The workflow of the proposed framework. DRL, Deep Reinforcement Learning

decoy containers for the cloud. Meanwhile, by placing decoys, we change the attacker's observation of the cloud and obtain a set of Observed Attack paths  $AP^+$  according to Equation (3). The orchestration platform will formulate  $AP^+$  as deceptive routing tables and plant them into decoys. In addition, the performance of placement strategy is used as the reward data to train and update the DRL agent. This enables the DRL agent to evolve with the dynamic cloud and guarantees the adaptivity of decoy placement.

Lastly, the orchestration platform places decoys hosting deceptive routing tables into the cloud. As to counterattack stealthy attackers, our framework suggests the following: (1) Placing high interactive decoys when the actions in placement strategy referring to any microservice vertex  $N_S$ . High interactive decoys of a microservice application could be a real application with certain replicas but quarantined with production network. In our design, the incoming requests to a certain application  $N_S$  will be answered by and observed in the high interactive decoy service for a period. If observed any request attempting to access the entities in the deceptive-routing table, such traffic source will be treated by certain intrusion response rules according to the security requirements of the system. For the rest of benign requests, they will be redirect back to production network for the continuing session. (2) Placing a high interactive decoy container when the actions in the placement strategy referring to any container vertex  $N_C$ . In this case, the decoy container is to observe any remote attacker attempts to exploit on virtualisation vulnerabilities of the microservice replicas. Once observed illegal access, the malicious traffics will be treated likewise the previous case.

## 6 | DETAILS OF THE OPTIMAL DEFENSIVE DECEPTION FRAMEWORK

In this section, we will introduce the design details of the optimal framework, which mainly contain two algorithms: the update of SRG and the DRL algorithm.

### 6.1 | The construction of SRG

The Risk Analyser module in the framework processes orchestration details and formulate the instant  $SRG_t$ . In our design, the Risk Analyser collects service topology and orchestration details from the Kubernetes (K8s) platform, where containerised applications are managed and monitored through APIs.  $SRG_t$  is formulated based on Def 1 and Def 2. Let  $k$  denote the type of microservices and  $TP_k$  represent their connections in the cloud. The orchestration details at time- $t$  are depicted by a  $|k| \times \lambda$ -dimension Boolean variant,  $O_t = \{0, 1\}^{|k| \times \lambda}$ . Each  $\lambda$ -bit denotes a subset of active (1) or offline (0) microservice replicas on the  $\lambda$ th computing node. The update of  $SRG_t$  and  $AP$  is shown in Algorithm 1.

Line 1 initialises  $N_t$ ,  $E_t$  and  $AP_t$ . Lines 2–6 and 25–32 transfer the service topology  $TP_k$  into *attack scenario 1* in

Section 3, where the remote attacker exploits on the neighbour vertex of a compromised target. We assume the attacker has compromised an entry-point service, Source. Lines 7–12 calculates edges for *attack scenario 2*, where the attacker escapes to the host-container by compromising a microservice. Lines 13–24 calculates edges for *attack scenario 3*, where the attacker compromise containers from the same computing node from a compromised container. Finally, we obtain  $AP_t$  by calculating shortest paths from Source to  $N_t$  with Dijkstra Algorithm.

---

### Algorithm 1 The update of $SRG_t$

---

**Data:** service types  $k$ , service connections  $TP_k$ , front-end services *Source*  
**Input:** the number of computing nodes  $\lambda$ , current orchestration details  $O_t$   
**Result:** instant vertex  $N_t$  and edges  $E_t$  of  $SRG_t$ , instant Attack Paths  $AP_t$

```

1 Initialize  $N_t \leftarrow \emptyset$ ,  $E_t \leftarrow \emptyset$ ,  $AP_t \leftarrow \emptyset$ ;
2 for  $(i, j)$  in  $(k, TP_k)$  do
3    $N_S \cup i$ ;
4    $j_{weight} = ED(i)$ ;
5    $E_{N_S} \cup j$ ;
6 end
7  $TP_{dict} = dict(TP_k)$ ;
8 for  $i$  in  $index(\mathbf{1}_{O_t})$  do
9    $e = tuple(TP_{dict}[i], i)$ ;
10   $e_{weight} = ED(i)$ ;
11   $E_{S2C} \cup e$ ;
12 end
13  $temp = split(O_t, \lambda)$ ;
14 for  $i$  in  $len(\lambda)$  do
15    $temp[i] = temp[i]^T * temp[i]$ ;
16    $diag(temp[i]) = 0$ ;
17    $s, t = O[temp[i]]$ ;
18   for  $(x, y, z)$  in  $(i * len(k) + s, i * len(k) + t, ED(t))$  do
19      $N_C \cup index(\mathbf{1}_{O_t})$ ;
20      $e = tuple(x, y)$ ;
21      $e_{weight} = ED(t)$ ;
22      $E_{N_C} \cup e$ ;
23   end
24 end
25 for  $i, j$  in  $product(index(\mathbf{1}_{O_t}), repeat=2)$  do
26   if  $i // len(k) \neq j // len(k)$  then
27      $e = tuple(i, j)$ ;
28      $e_{weight} = ED(j)$ ;
29     for  $e \in TP^k$  do
30        $E_{C2C} \cup e$ ;
31     end
32   end
33 end
34  $N_t \leftarrow N_S \cup N_C$ ;
35  $E_t \leftarrow E_{N_S} \cup E_{N_C} \cup E_{S2C} \cup E_{C2C}$ ;
36  $AP_t \leftarrow Dijkstra(Source, N_t, E_t)$ .
```

---

## 6.2 | The DRL algorithm

We leverage DRL to generate optimal strategies to place decoys in cloud. In DRL framework, the agent learns from its interactions with a dynamic environment (i.e. the dynamic cloud) under the typical RL framework. The combination of DNN provides neural network function approximators.

At each step  $t$  of the interaction, the agent observes a state  $s_t$  of the cloud and generates an action  $a_t$  based on a policy  $\pi$  (the policy could be deterministic or stochastic). We denote a sequence of states and actions with a trajectory  $\tau = (s_0, a_0, s_1, a_1, \dots)$ . The agent's action  $a_t$  leads changes on the environment and gets a reward  $r_t$ . Based on  $r_t$ , the agent updates its policy. The agent's objective is to maximise the cumulative reward over a trajectory:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t, \quad (5)$$

where  $\gamma$  is the discount factor [38]. To achieve the goal, the agent consistently optimises its policy. Generally, a value function is used to evaluate the quality of current policy for environment starting from state  $s$ :

$$V^\pi(s) = E_{\tau \sim \pi} [R(\tau) | s_0 = s] \quad (6)$$

In the RL framework, the agent searches for optimal policies in a table where massive value functions are stored. However, for problems with large input space, the storage and looking-up of table will be rather complicated. Mnih et al. [38] propose to use neural network as a function approximator. Such that, the DRL agent's objective becomes optimising the parameterised policy  $\pi_\theta$  and learning an approximator function  $V_\phi(s)$  for  $V^\pi(s)$ , where  $\theta$  and  $\phi$  are corresponding neural network parameters. The objective is denoted as  $J(\pi_\theta)$ , which could be optimised by gradient ascent or maximising local approximation.

As our environment is the dynamic container-based cloud with massive computing nodes, the DRL agent should be able to handle high-dimensional input data and guarantee stable performance. We leverage the Proximal Policy Optimisation (PPO) [39] in our DRL framework. Because PPO has a better data efficiency and robustness than vanilla policy gradient [40], a more compatible architecture than Trust Region Policy Optimisation [41] and is adaptive and stable to a wider range of problems [42]. PPO optimises the policy by maximising a surrogate objective function for  $J(\pi_\theta)$ :

$$L(\theta) = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t] \quad (7)$$

where  $r_t(\theta) = \frac{\pi_\theta(a_t|s)}{\pi_{\theta_{old}}(a_t|s)}$  and  $\hat{A}_t$  is an estimator of the advantage function at time-step  $t$ .  $\theta_{old}$  denotes the vector of policy parameters before the update and  $\hat{\mathbb{E}}_t$  indicates the expectation

[39]. To avoid dramatic updates in the policy, a hyperparameter  $\epsilon$  is introduced to clip the stepsize:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)], \quad (8)$$

where  $\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)$  modifies the surrogate objective by clipping  $r_t(\theta)$ . The minimum of the clipped and the unclipped objective is taken to make final objective a lower bound of the unclipped objective. This is to avoid local optima from aggressive improvement. By clipping, the agent obtains suboptimal changes on policy and constantly exploits rewards. The DRL uses Stochastic Gradient Descent to update Equation (8) with the Multilayer Perceptron (MLP) network and the training structure used in this DRL framework is shown in Figure 3.

The pseudocode of DRL algorithm used in this work is shown in Algorithm 2. In Line 1, the policy and value function are parameterised and initialised. Line 3 generates the decoy placement strategy and deceptive routing tables based on policy  $\pi_k$ . Lines 4 and 5 compute rewards and advantage estimates for agent when interacting with  $SRG_t$  by policy  $\pi_k$ . Line 6 updates the policy with parameter  $\theta$ , where  $\theta_{k+1} = \arg\max_{\theta} E_{s,a \sim \pi_{\theta_k}} [L^{CLIP}(\theta_k)]$ . To avoid aggressive or passive updates of old policy, the hyperparameter  $\epsilon$  is usually 0.1 or 0.2. Line 7 fits the loss of value function to encourage policy entropy.

---

### Algorithm 2 DRL Algorithm of the Framework

---

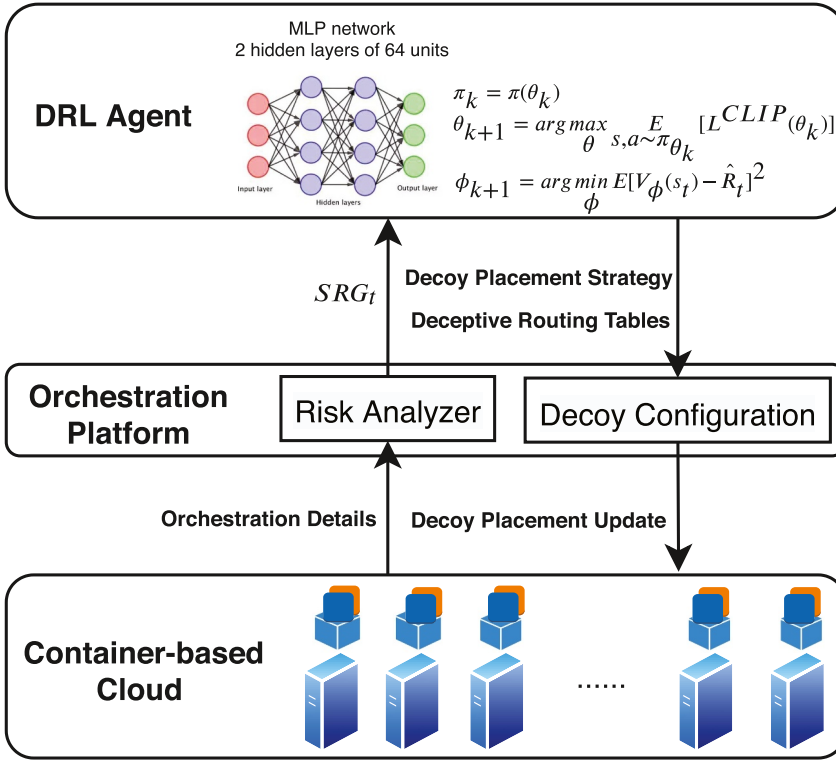
**Data:** service types  $k$ , service connections  $TP_k$ , front-end services  $Source$

**Input:** current System Risk Graph  $SRG_t$

**Result:** Action list: decoy placement strategy, neural network parameters of DRL  $\theta, \phi$

- 1 Initialize parameters  $\theta_0$  and  $\phi_0$  for policy and value function, respectively;
  - 2 **for**  $k=0,1,2,\dots$ , **do**
  - 3     Collect set of trajectories  $\mathcal{D}_k = \{s_t, a_t\}$  by running policy  $\pi_k = \pi(\theta_k)$  on the  $SRG_t$  of the cloud;
  - 4     Compute rewards-to-go  $\hat{R}_t$ ;
  - 5     Compute advantage estimates,  $\hat{A}_t$  based on the current value function  $V_{\phi_k}$ ;
  - 6     Update the policy by maximizing the Clipped objective:
 
$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)$$
  - 7     Fit value function by regression on mean-squared error:
 
$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T (V_{\phi}(s_t) - \hat{R}_t)^2.$$
  - 8 **end**
-





**FIGURE 3** The training architecture of the proposed framework. DRL, Deep Reinforcement Learning

### 6.3 | Implementation detail of DRL interfaces

According to Algorithm 2, main interfaces between the DRL agent and the cloud environment are state, action and reward. We formalise these interfaces in our framework as follows:

#### 6.3.1 | State

We use the SRG defined in Def 1 to denote the state information. The adjacency matrix of  $SRG_t$  is flattened into  $|k \times \lambda|$ -dimension vectors to formalised the input data. Note that a  $SRG_t$  denotes instant active microservices, orchestration information, and potential risks of the cloud at time- $t$ . Therefore, vertices in the adjacency matrix are denoted by their Exploit Difficulty, ED, value. Let  $S$  denote a set of states,  $S = \{s_t \in \{0, ED\}^{|k \times \lambda|} | t \in [0, T]\}$ . Moreover, in real-word deployment, each microservice should have at least one active replica on whichever computing node. This constraint is expressed by  $\sum s_t^k \geq 0$ .

#### 6.3.2 | Action

The DRL generates a lists of actions according to the optimal policy approximated by the neural network. In our design, the action denotes the location of decoys in the cloud, which refers to placing decoy on microservices,  $N_S$ , or containers,  $N_C$ .

According to Def 3, the placement of decoys will decrease the Observed Exploit Difficulty,  $ED'$ , of vertices  $N_t$  in the  $SRG_t$  and change the Observed Attack Paths  $AP'$ . Hence, the action value denotes which vertex to decrease  $ED'$  in  $SRG_t$  and a subset attack paths as deceptive routing tables  $AP^+$ . The action set is denoted as  $A = \{a_t | a_t \in N\}$ .

#### 6.3.3 | Reward

In DRL, the reward generally reflects the learning goals. The goal of this work is to maximise deception by (1) concealing maximum system details ( $AP^-$ ), and (2) revealing maximum falsified Attack Paths ( $AP^+$ ). Therefore, we use the defence objective in Equation (4) to formalise the reward function for DRL. Here, the reward at  $t$ -time is  $r_t = \frac{|AP^+| + |AP^-|}{|AP|}$ .

For the implementation of the neural network in our DRL algorithm, we use a MLP. MLP is a class of feedforward neural network. The construction of MLP in our framework has two hidden layers, where each one is consisted of 64 units. The input layer takes the  $|k \times \lambda|$ -dim state vector  $s_t \in \{0, ED\}^{|k \times \lambda|}$  as input data. The hidden layers utilise tanh activation function to map the weighted inputs to the outputs of each neuron unit. By changing the connection weights according to the error between the output and expected result, the MLP processes  $s_t$  and generates  $a_t$  at the output layer. Afterwards, the output layer generates the action  $a_t$  with the tanh function.

When DRL determines the placement strategy and deceptive routing for the decoy, they are sent to the orchestration platform. Corresponding decoys will be placed in

following ways: (1) For the action value representing  $N_S$ , decoys will be started-up with the all replicas of that specific microservice at  $t$ -time; (2) For the action value pointing to  $N_C$ , decoy containers will be started-up with the decided containers; (3) Lastly, deceptive routings created by  $AP^+$  will be projected in the decoys with crafted traffics. Any exploit on decoys' vulnerabilities or connections to the decoy routings will be classified as the malicious.

## 6.4 | Overhead analysis

Generally, the IDS is configured at the entry-point for all container-based microservices in cloud. Although avoiding wasting computing resource on heavy security configurations of back-end services, this provides stealthy attackers opportunities. Once the adversary evades IDS on the entry-point, he could pivot on internal services for lateral movement [36].

In our work, we counter stealthy lateral movement while controlling security overhead by implementing AI-assisted defensive deception. Firstly, we develop a DRL-based placement strategy to decide optimal topologically locations and types on digital decoys assets. The Orchestration Platform will start decoy containers or multiple decoy replicas for a service on different risk conditions extracted by the instant  $SRG_t$ . The strategical deployment of digital assets will be more resource saving than overall deployment around the cloud. Secondly, credit to the lower virtualisation overhead of container technology and the advent of automatic orchestration platforms, updating service replicas demands controllable resources. Still, creating multiple decoy replicas for a service requires more containers than implementing a single decoy container. However, the collaboration of orchestration platform and container technology indeed decrease resource consumption than tradition high interactive honeypot technique [11]. Lastly, the main cost will be at traffic re-direction for decoy routings. However, as the orchestration platform monitors and decides the re-routing of incoming traffics, re-directions could be effectively handled in conjunction with its automatic restore mechanism without sharp decline in service quality [5]. Hence, the overhead on implementing defensive deception in the cloud is overall favourable.

## 7 | SIMULATION AND EVALUATION

In this section, we present examples of SRG to illustrate the adversarial model. We conduct simulations to evaluate the effectiveness of our framework and existing works.

### 7.1 | Simulation setup

The microservice architecture we experimented in the simulation follows service topology in Ref. [11]. Among all microservices, *Apache* and *Tomcat* are specified as entry-

points, that is, the source attack targets. We collect most recent Common Vulnerabilities & Exposures (CVEs) to calculate the EM and ED with Equation (1) and Equation (2), respectively. The requirements on simulated CVEs are as follows:

- 1) CVEs for each target are from five recent years (2016–2020). It is assumed that most previous published vulnerabilities are patched by the update of software versions or manually by the system security teams.
- 2) CVEs are remotely exploitable in the container-based cloud. This fulfils the assumption of attack tactics, where the attacker pivots on the entry-point target and continues lateral movement in the cloud by RCE on its neighbours.
- 3) CVEs are verified exploitable. From the perspective of the attacker, exploiting verified vulnerabilities with applications or codes are reasonably effortless and beneficial than the unverified. This is because, on one hand, the attack tools are mature and available. On the other hand, the attack result is expectable. Hence, we filtered the verified vulnerabilities in the *Exploit Database*.

Note that the value of ECM are extracted up to date from the *Exploit Database*. The upper-bound of ECM is 10, and a larger ECM value indicates a higher probability of being exploited. Also, a larger value of EM and ED refer to greater difficulty in exploiting each vulnerability and target, respectively. Vulnerabilities and the results of EM and ED are presented in Table 2. The size of simulated SRGs scale up to 60 working nodes and 300 vertex. The DRL agent is implemented in Python 3.7 based on Tensorflow 1.14. We run simulations on 1000 SRG modelling different running conditions of the container-based cloud. Examples of the SRG are shown in Figure 5.

To compare our framework with existing strategies, we use following metrics:

- 1) The coefficient of deception,  $\mathbb{C}$ , is used to evaluate the effectiveness of decoy placement strategy. As defined in Def 4,  $\mathbb{C}$  evaluates the concealment of system properties and the increase on misguiding information.
- 2) The detection rate represents the likelihood of detecting the adversary with decoys. This metric evaluates the effectiveness on detection of defensive defence framework.

We compared our framework with following mainstream decoy placement styles:

- 1) Betweenness centrality-based placement strategy [11]. Betweenness centrality denotes the likelihood of a vertex being in between others. Namely, if an attack target is with high betweenness centrality degree in the attack graph, it is in many shortest attack paths. Studies following this style aim to deploy limited amount of security resources to cover as many shortest attack paths as possible. However, this method may not be effective to bring an optimal amount of deceptive information to satisfy the second term of  $\mathbb{C}$ .

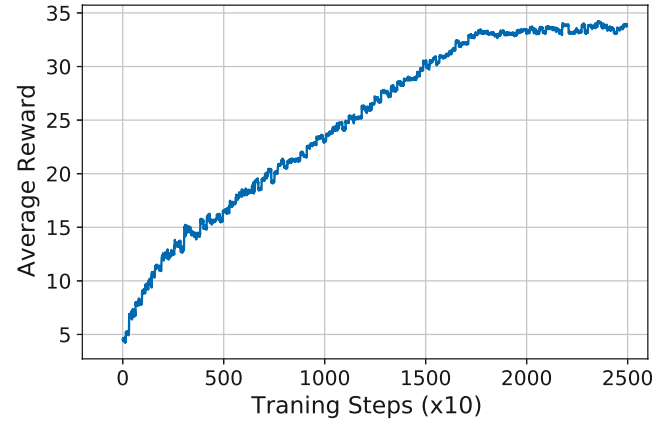
**TABLE 2** Vulnerabilities used in the adversarial model

Attack target	CVE ID	EM	ECM	ED
Apache	CVE-2018-1335	2.2	7.9	2.82
	CVE-2018-11776	2.2	7.9	
	CVE-2019-0230	3.9	9.2	
	CVE-2019-17558	1.6	7.3	
	CVE-2019-17530	3.9	8.3	
Tomcat	CVE-2019-0232	2.2	7.9	3.3
	CVE-2019-10104	3.9	8.6	
	CVE-2019-14768	2.8	7.4	
	CVE-2020-26510	3.9	8.3	
	CVE-2020-1938	3.9	9.6	
Memcached	CVE-2020-17388	2.8	7.4	
	CVE-2016-8706	2.2	6.8	3.41
	CVE-2016-8705	3.9	8.3	
	CVE-2016-8704	3.9	8.3	
ImageMagick	CVE-2019-11832	1.6	6.3	2.23
	CVE-2017-14650	2.2	6.8	
	CVE-2017-14224	2.8	8.3	
MySql	CVE-2020-11974	3.9	8.3	3.12
	CVE-2016-6663	1	6.3	
	CVE-2016-6662	3.9	8.8	
Replicas	CVE-2020-35197	3.9	8.3	2.56
	CVE-2020-7606	0.5	5.4	
	CVE-2016-9962	3.9	8.3	

Abbreviation: CVE, Common Vulnerabilities & Exposures.

- 2) Sidecar placement strategy [19–21]. We summarise strategies placing decoys along with selected objectives. Mostly, objective targets are valuable digital assets or vulnerable services. This placement style distracts the adversary from selected targets but does not address the changing risks and vulnerabilities in the dynamic cloud.
- 3) Random strategy [43] and round Robin strategy. These are regarded as the baseline to simulate dynamic empirical placement strategy in most of the rest studies [12].

To evaluate the effectiveness of detection, we simulate on a random-walker attacker [18] and a persistent attacker [9] as described in the adversarial model. The random-walker attacker takes a random attack path in  $AP'$  to reach target vertex. By deploying decoys, we conceal a set of real attack paths  $AP^-$  and introduce a set of deceptive attack paths  $AP^+$  to trap the attackers at time- $t$ . We use the probability of choosing deceptive attack paths to denote the detection rate on random-walker attacker, that is,  $d_{RW} = 1 - \frac{|AP^- - AP|}{|AP'|}$ . For the persistent attacker, he attempts to reach a certain target  $\nu$  by trying all possible attack paths. Hence we denote the detection of

**FIGURE 4** Training process

persistent attackers with the probability that decoys  $\nu$  are on  $AP^+$ , that is,  $d_{PS} = \frac{\nu_{AP^+}(\nu)}{N}$ .

## 7.2 | Simulation results

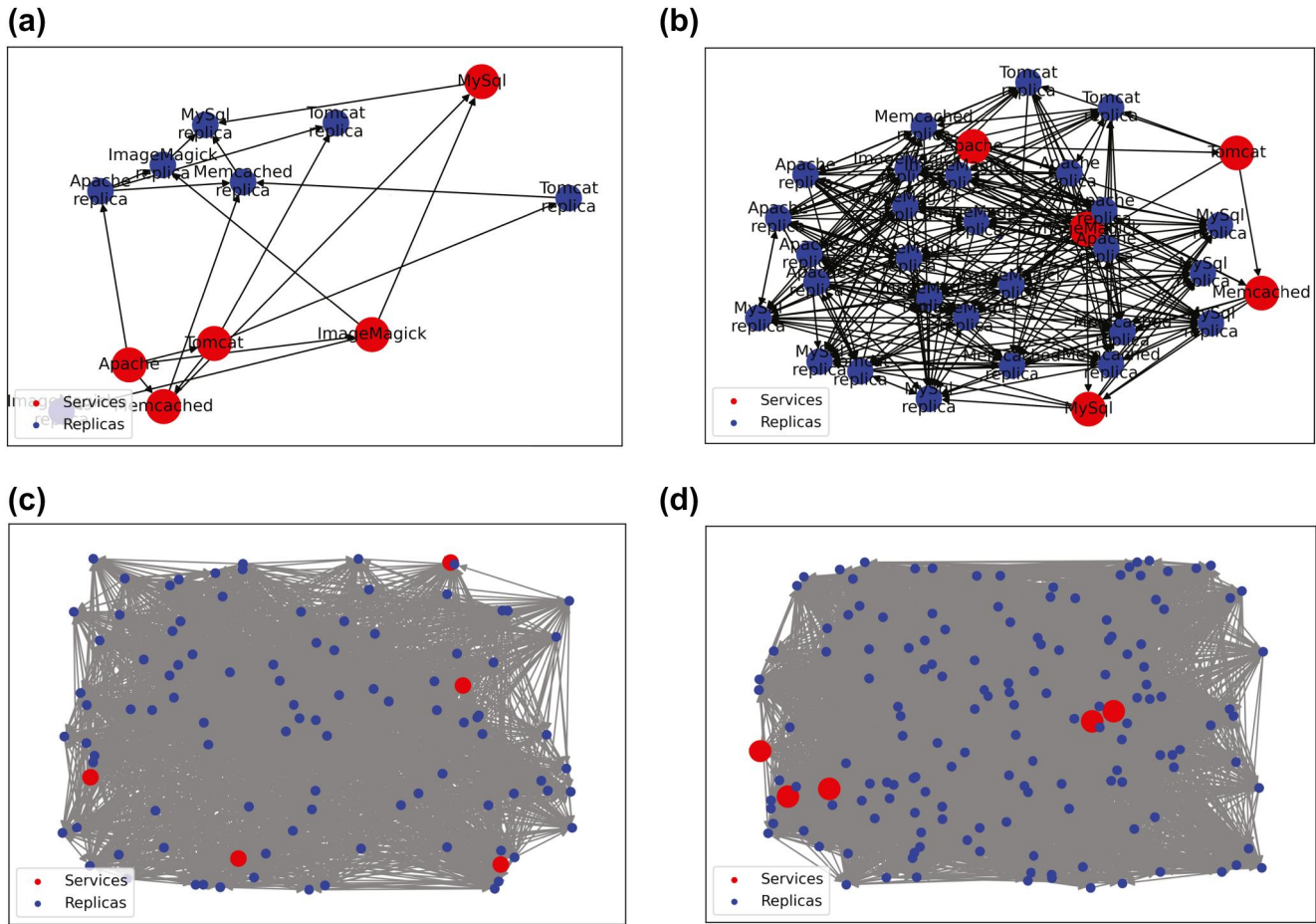
### 7.2.1 | The performance of DRL agent and the scalability of adversarial model

As mentioned above, our framework leverages a DRL algorithm to generate adaptive decoy placement strategy. The training performance of the DRL agent is plotted in Figure 4. The algorithm performs well in convergence time, which starts to converge at 16k training steps.

The simulation result shows that the framework is scalable to produce SRG for up to 60 working nodes. Examples of SRG are depicted in Figure 5. Blue dots and red dots stand for microservices in application layer and replicas in virtualisation layer, respectively. Figure 5a–d illustrate SRG with 2, 10, 40, and 60 working nodes, respectively. By the increase of computing nodes, the size of SRG expands beyond linear. By leveraging Dijkstra algorithm to calculate AP, the **Risk Analyser** proves its stability and scalability to 60 computing nodes and 300 vertices.

### 7.2.2 | The coefficient of deception

We use the coefficient of deception,  $\mathbb{C}$ , to evaluate the effectiveness of placement strategy.  $\mathbb{C}$  evaluates from two perspectives: the concealment of real attack paths and the increase of deceptive attack paths. The simulation result of  $\mathbb{C}$  on our proposed framework and existing methods are drawn in Figure 6. Generally, our work leads to an average 30.22% advantage in  $\mathbb{C}$  and an average 73.46% decrease in the variance of  $\mathbb{C}$  than existing approaches. It could be analysed from the boxplot that the DRL-based placement strategy proposed in this work has an overall optimal and stable performance. The sidecar deployment mode gains least  $\mathbb{C}$  during the simulation. The random and round Robin strategy perform negative in stability. When comparing with the

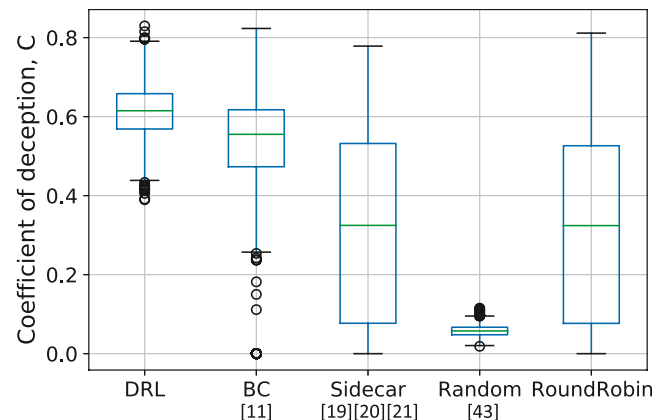


**FIGURE 5** SRG's scalability for different size of computing nodes. Lines represent the finite set of edges  $E$  in SRG. The coloured dots depicting different vertex  $N$ . Given  $N = N_S \cup N_C$ , we differentiate  $N_S$  and  $N_C$  by the size and colour. Larger dots in red and smaller dots in blue depict  $N_S$ , Service and  $N_C$ , Replicas, respectively. SRG for container-based cloud with (a) 2, (b) 10, (c) 40 and (d) 60 working nodes

mainstream between centrality-based placement strategy, our proposed method outweighs in both mean performance and stability on  $\mathbb{C}$ . This is because (1) the betweenness centrality-based method emphasises the concealment of  $AP_-$  but fails to jointly consider the promotion of deceptive attack paths,  $AP_+$ , (2) the sidecar strategy deploys decoys alongside risky services. This consideration is reasonable to protect particular objectives, but fails to satisfy the overall security when the defender is not able to locate stealthy attacker, and (3) either random strategy or round Robin is unstable in performance. This is because they do not fully adapt to the target environment. It is indicated that statistically random placement of decoy will occasionally inactivate defensive deception.

### 7.2.3 | The detection rate

We simulate a random-walker attacker and a persistent attacker to evaluate the effectiveness of detecting stealthy attackers. The simulation results on detection ratio are illustrated in Figure 7.



**FIGURE 6** The results of coefficient of deception,  $\mathbb{C}$

The detect ability in the context refers to the increase on probability to trap the adversary when deploying decoys in target system, while not considering the effect of additional security mechanisms. The results indicate that the proposed



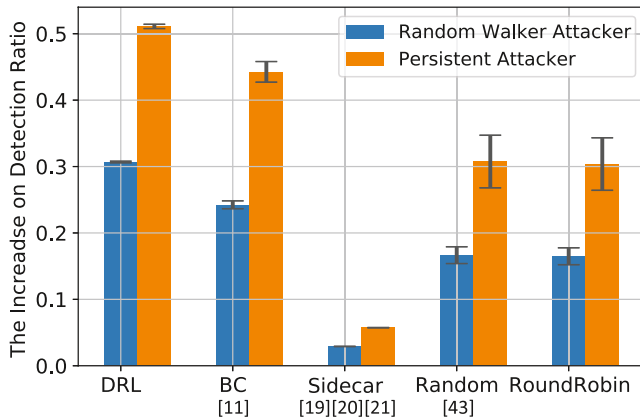


FIGURE 7 The simulation of attacker detection

placement strategy and deceptive routings enable the framework an overall optimal results in both types of attackers, including the detection ability and stability. The sidecar deployment has the least increase on detection ration. The betweenness centrality-based strategy, the random strategy and the round Robin strategy show increasing variance. For the increase of detection ratio, our work leads an average 30.69% on random-walker attacker and 51.10% on persistent attacker. Furthermore, the proposed DRL-based placement strategy leads a higher detection ratio than the rest of the approaches from 4.44% and 4.84% to 27.78% and 45.36% on both types of attackers. Our work improves not only the detection ratio but also the stability on detection. This is because the DRL-based strategy has better adaption with the dynamic cloud environment and thus generate overall optimal placement decisions.

Lastly, we find that the existence of decoys in the target system enable a better performance to detect persistent attackers than the random-walker attacker. This is because the persistent attacker spears no effort to reach the target by making attempts to enumerate attack paths. Since we conceal real attack paths and present deceptive paths, such attack behaviours will be more likely to trigger deceptive decoys and being detected.

## 8 | RELATED WORK

The advent of Artificial Intelligence (AI) and ML brings new horizon to cyberspace. Emerging techniques such as the DRL and Natural Language Processing (NLP) are promising for cybersecurity. For example, NLP is applied to improve the enticement and believability of decoy text. Whitham [21] and Chakraborty et al. [44] generate high interactive Honeyfile to deceive data theft. Such high-simulative fake text is effective in detecting and delaying malicious acquisition on sensitive documents. Malialis and Kudenko [45] use multiagent RL to realise throttle or rate-limit on victim servers, which realises a decentralised solution to counter DDoS in small-scale network topologies.

DRL is also promising to solve complex controlling problems in networking. Recently, CFR-RL [46] and DATE [47] reduce negative impacts on the network for traffic engineering and balances link utilisation by rerouting a number of critical flows selected by RL. SmartEntry [48] effectively mitigates rerouting update overhead by utilising RL to reduce forwarding entries. Apart from QoS promotion, existing studies also leverage DRL to improve the power efficiency of data centre networks. SmartFCT [49] takes the advantage of DRL to generate optimal traffic consolidation strategy while guaranteeing flow completion time. It effectively reduces power consumption without a sharp decrease on network QoS.

## 9 | CONCLUSION

Cyber deception defence is an emerging proactive countermeasure to reverse the asymmetry between the attacker and defender. Especially for the container-based cloud, defensive deception is promising to deter and detect stealthy attackers with limited resource. In this work, we propose an optimal defensive deception framework for the container-based cloud with DRL. The framework addresses two main issues to implement defensive deception in the cloud, including the precise adversarial adversary model and the adaptive placement strategy.

Firstly, we propose a precise adversarial model, SRG, to extract particular risks and treats in the cloud, including vulnerabilities in the application layer and the virtualisation layer. The adversarial model updates per system running status. On the basis of SRG, we propose the coefficient of deception,  $\mathbb{C}$ , a quantitative metric to comprehensively evaluate the effectiveness of placement strategy. Secondly, we modify a DRL algorithm to generate adaptive decoy placement strategy for the highly dynamic cloud. Lastly, we conduct simulations to evaluate the proposed framework with existing studies. For the scalability, the simulation results show that both SRG and the decoy placement strategy scale up to 60 computing nodes. For the performance on placement strategy, we evaluate the coefficient of deception,  $\mathbb{C}$ , and detection ratio. The results show that our method leads to an average 30.22% advantage in  $\mathbb{C}$  and an average 73.46% decrease in the variance of  $\mathbb{C}$  than existing approaches. Last but not least, the proposed framework increases detection ratio on the random-walker attack and the persistent attack in average by 30.69% and 51.10% with minor variance, respectively.

Conclusively, we address the implementation of defensive deception in cloud by establishing a precise and updatable adversarial model, SRG, for the container-based cloud. Based on the SRG, we define coefficient of deception,  $\mathbb{C}$ , to evaluate the decoy placement strategy. Moreover, we address the adaptivity and re-deployment of placement strategy by modifying a DRL algorithm. Our method fully interacts with the dynamic environment to optimise overall implementation. The simulation results show that our framework is more effective and stable to misguide and detect the adversary in the cloud.

## ACKNOWLEDGEMENTS

This work has been partly supported by the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant No. 61521003 and the National Natural Science Foundation of China under Grant Nos 62072467 and 62002383.

## CONFLICT OF INTEREST

None.

## PERMISSION TO REPRODUCE MATERIALS FROM OTHER SOURCES

None.

## DATA AVAILABILITY STATEMENT

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

## ORCID

Huanruo Li  <https://orcid.org/0000-0001-5737-7309>

## REFERENCES

- Jamshidi, P., et al.: Microservices: the journey so far and challenges ahead. *IEEE Softw.* 35(3), 24–35 (2018)
- Sultan, S., Ahmad, I., Dimitriou, T.: Container security: issues, challenges, and the road ahead. *IEEE Access.* 7, 52976–52996 (2019)
- Nehme, A., et al.: Securing microservices. *IT Prof.* 21, 42–49 (2019)
- Yarygina, T., Bagge, A.H.: Overcoming security challenges in micro-service architectures. In: 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), pp. 11–20. IEEE (2018)
- Indrasiri, K., Siriwardena, P.: *Microservices for the Enterprise*. Apress, Berkeley (2018)
- Osman, A., et al.: Sandnet: towards high quality of deception in container-based microservice architectures. In: ICC 2019 – 2019 IEEE International Conference on Communications (ICC), vol. 2019-May, pp. 1–7. IEEE (2019)
- Duan, Q., et al.: CONCEAL: a strategy composition for resilient cyber deception-framework, metrics and deployment. In: 2018 IEEE Conference on Communications and Network Security (CNS), pp. 1–9. IEEE (2018)
- Lu, Z., Wang, C., Zhao, S.: Cyber Deception for Computer and Network Security: Survey and Challenges, pp. 1–7. arXiv preprint arXiv:2007.14497 (2020)
- Almohri, H.M.J., Watson, L.T., Evans, D.: Misery digraphs: delaying intrusion attacks in obscure clouds. *IEEE Trans. Inf. Forensics Secur.* 13, 1361–1375 (2018)
- Wang, C., Lu, Z.: Cyber deception: overview and the road ahead. *IEEE Secur. Priv.* 16, 80–85 (2018)
- Jin, H., et al.: DSEOM: a framework for dynamic security evaluation and optimization of MTD in container-based cloud. *IEEE Trans. Dependable Secure Comput.* PP(c), 1 (2019)
- Han, X., Kheir, N., Balzarotti, D.: Deception techniques in computer security. *ACM Comput. Surv.* 51, 1–36 (2018)
- Durkota, K., et al.: Optimal network security hardening using attack graph games. In: IJCAI International Joint Conference on Artificial Intelligence, vol. 2015-Janua, pp. 526–532. (2015)
- Hong, J., Kim, D.S.: HARMS: Hierarchical attack representation models for network security analysis. In: Proceedings of the 10th Australian Information Security Management Conference, Novotel Langley Hotel, Perth, pp. 74–81. 3–5 December 2012. <https://ro.ecu.edu.au/ism/146>
- Horák, K., et al.: Optimizing honeypot strategies against dynamic lateral movement using partially observable stochastic games. *Comput. Secur.* 87, 101579 (2019)
- Pawlick, J., Colbert, E., Zhu, Q.: Modeling and analysis of leaky deception using signaling games with evidence. *IEEE Trans. Inf. Forensics Secur.* 14, 1871–1886 (2019)
- Schlenker, A., et al.: Deceiving cyber adversaries: a game theoretic approach. In: International Conference on Autonomous Agents and Multiagent Systems (2018)
- Gavrilis, D., Chatzis, I., Dermatas, E.: Flash crowd detection using decoy hyperlinks. In: 2007 IEEE International Conference on Networking, Sensing and Control, May, pp. 466–470. IEEE (2007)
- Bojinov, H., et al.: Kamouflage: Loss-Resistant Password Management. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds). *Computer Security – ESORICS 2010. ESORICS 2010. Lecture Notes in Computer Science*, vol. 6345. Springer, Berlin (2010). [https://doi.org/10.1007/978-3-642-15497-3\\_18](https://doi.org/10.1007/978-3-642-15497-3_18)
- Kontaxis, G., Polychronakis, M., Keromytis, A.D.: Computational decoys for cloud security. In: *Secure Cloud Computing*, pp. 261–270. Springer New York, New York (2014)
- Whitham, B.: Automating the generation of enticing text content for high-interaction honeypots. In: Proceedings of the 50th Hawaii International Conference on System Sciences, pp. 6069–6078. (2017)
- Soltész, S., et al.: Container-based operating system virtualization. *ACM SIGOPS Oper. Syst. Rev.* 41, 275–287 (2007)
- Heorhiadi, V., et al.: Gremlin: systematic resilience testing of micro-services. In: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), pp. 57–66. IEEE (2016)
- Modi, C., et al.: A survey of intrusion detection techniques in cloud. *J. Netw. Comput. Appl.* 36, 42–57 (2013)
- Cohen, F.: A note on the role of deception in information protection. *Comput. Secur.* 17, 483–506 (1998)
- Spitzner, L.: The HoneyNet project: trapping the hackers. *IEEE Secur. Priv.* 1, 15–23 (2003)
- Keromytis, A.D., et al.: The MEERKATS cloud security architecture. In: 2012 32nd International Conference on Distributed Computing Systems Workshops, pp. 446–450. IEEE (2012)
- Brzezczko, A., et al.: Active deception model for securing cloud infrastructure. In: 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 535–540. IEEE (2014)
- Pham, L.H., et al.: A quantitative framework to model reconnaissance by stealthy attackers and support deception-based defenses. In: 2020 IEEE Conference on Communications and Network Security (CNS), no. i, pp. 1–9. IEEE (2020)
- Kahlhofer, M., Hölzl, M., Berger, A.: Towards reconstructing multi-step cyber attacks in modern cloud environments with tripwires. In: Proceedings of the European Interdisciplinary Cybersecurity Conference, pp. 1–2. ACM (2020)
- Pawlick, J., Colbert, E., Zhu, Q.: A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy. *ACM Comput. Surv.* 52, 1–28 (2019)
- Ferguson-Walter, K., et al.: Game theory for adaptive defensive cyber deception. In: Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security – HotSoS 19, no. April, New York, pp. 1–8. ACM Press (2019)
- Ayoade, G., et al.: Automating cyberdeception evaluation with deep learning. In: HICSS, Vol. 3, pp. 1–10. (2020)
- Sethi, K., et al.: Deep reinforcement learning based intrusion detection system for cloud infrastructure. In: 2020 International Conference on COMMunication Systems & NETWORKS (COMSNETS), pp. 1–6. (2020). <https://doi.org/10.1109/COMSNETS48256.2020.9027452>
- Coppolino, L., et al.: Cloud security: emerging threats and current solutions. *Comput. Electr. Eng.* 59, 126–140 (2017)
- Alshamrani, A., et al.: A survey on advanced persistent threats: techniques, solutions, challenges, and research opportunities. *IEEE Commun. Surv. Tutorials.* 21(2), 1851–1877 (2019)
- FIRST: Common Vulnerability Scoring System Version 3.1 User Guide Revision 1, pp. 1–24. <https://www.first.org/cvss/v3-1/> (2019)
- Mnih, V., et al.: Human-level control through deep reinforcement learning. *Nature* 518(7540), 529–533 (2015)

39. Schulman, J., et al.: Proximal policy optimization algorithms. arXiv, 1–12 (2017)
40. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of the 33rd International Conference on Machine Learning, vol. 48, New York, 20–22 Jun 2016, pp. 1928–1937. PMLR (2016)
41. Schulman, J., et al.: Trust region policy optimization. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning, vol. 37, Lille, France, 07–09 Jul 2015, pp. 1889–1897. PMLR (2015)
42. Duan, Y., et al.: Benchmarking deep reinforcement learning for continuous control. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of the 33rd International Conference on Machine Learning, vol. 48, New York, 20–22 Jun 2016, pp. 1329–1338. PMLR (2016)
43. Gutierrez, M., Kiekintveld, C.: Online learning methods for controlling dynamic cyber deception strategies. In: Jajodia, S., Cybenko, G., Subrahmanian, V., Swarup, V., Wang, C., Wellman, M. (eds.) Adaptive Autonomous Secure Cyber Systems. pp. 231–251. Springer International Publishing, Cham (2020)
44. Chakraborty, T., et al.: A fake online repository generation engine for cyber deception. IEEE Trans. Dependable Secure Comput. 18, 518–533 (2021)
45. Malialis, K., Kudenko, D.: Distributed response to network intrusions using multiagent reinforcement learning. Eng. Appl. Artif. Intell. 41, 270–284 (2015)
46. Zhang, J., et al.: CFR-RL: traffic engineering with reinforcement learning in SDN. IEEE J. Sel. Areas Commun. 38, 2249–2259 (2020)
47. Ye, M., et al.: DATE: disturbance-aware traffic engineering with reinforcement learning in software-defined networks. In: 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS), pp. 1–10. (2021). <https://doi.org/10.1109/IWQOS52092.2021.9521343>
48. Zhang, J., et al.: SmartEntry: mitigating routing update overhead with reinforcement learning for traffic engineering, pp. 1–7. Association for Computing Machinery (2020). <https://doi.org/10.1145/3405671.3405809>
49. Sun, P., et al.: SmartFCT: improving power-efficiency for data center networks with deep reinforcement learning. Comput. Netw. 179, 107255 (2020). <https://doi.org/10.1016/j.comnet.2020.107255>

**How to cite this article:** Li, H., et al.: An optimal defensive deception framework for the container-based cloud with deep reinforcement learning. IET Inf. Secur. 16(3), 178–192 (2022). <https://doi.org/10.1049/ise2.12050>