SCIENCE CHINA
Information Sciences

July 2022, Vol. 65 170305:1–170305:19
https://doi.org/10.1007/s11432-021-3462-4

• RESEARCH PAPER •
Special Focus on Cyber Security in the Era of Artificial Intelligence

Defensive deception framework against reconnaissance attacks in the cloud with deep reinforcement learning

Huanruo LI[1*], Yunfei GUO[1], Shumin HUO[1], Hongchao HU[1] & Penghao SUN[1,2]

[1]*Department of Computer Science, Information Engineering University, Zhengzhou* 450001*, China;*
[2]*Department of Communication Technologies, Academy of Military Science, Beijing* 100091*, China*

Received 31 July 2021/Revised 19 January 2022/Accepted 25 March 2022/Published online 23 June 2022

Abstract Implementing defensive deception in the cloud is promising to proactively counter reconnaissance attack. This technique presents decoys to camouflage cloud assets and distracts attack resource. However, the major challenge is to develop an effective deception strategy to orchestrate digital decoys. To address this issue, we propose a deep reinforcement learning (DRL)-based defensive deception framework. First, we formulate a utility function, which mathematically models underlying threats associated with common vulnerabilities among virtual machines in the cloud. Then, we customize training interfaces and the neural networks for a DRL agent. The reward function reflects the effectiveness of asset concealment and the waste of attack resources, referring to a comprehensive defense goal. Finally, the well-trained DRL agent generates the optimal defense strategy. It specifies a more granular deception strategy than existing proposals. Simulation results show that the proposed framework leads to a 7.87% average advantage in realizing the comprehensive defense goal. Moreover, it can stably improve the concealment degree of cloud assets up to 20.58%, and increase the attack cost up to 40.40%. This study shows that it is promising to improve cloud security with deception defense and artificial intelligence techniques.

Keywords cyber deception defense, artificial intelligence, cloud security, reconnaissance attack, deep reinforcement learning, deception strategy

Citation Li H R, Guo Y F, Huo S M, et al. Defensive deception framework against reconnaissance attacks in the cloud with deep reinforcement learning. Sci China Inf Sci, 2022, 65(7): 170305, https://doi.org/10.1007/s11432-021-3462-4

1 Introduction

According to the cyber kill chain model, reconnaissance is a fundamental step for a cyberattack [1]. During this stage, the adversary collects essential information about the target system to prepare for subsequent attacks. Protecting the operating systems (OSs)' associated information of each working node is significant, especially for the infrastructure-as-a-service (IaaS) cloud tenant network. The main reasons are as follows. (1) The OS is a critical software infrastructure for the working nodes (i.e., virtual machines (VMs)) in the IaaS cloud. Secure systems could be compromised in a stealthy way by attacks on their OSs, instead of via their application or service vulnerabilities [2]. (2) Off-the-shelf OSs may share security flaws [3]. Hence, the adversary could launch and rapidly spread attacks by exploiting on OS-related common vulnerabilities.

However, preventing reconnaissance attacks is difficult. On the one hand, tenant networks often maintain a similar configuration for VMs concerning the feasibility of operation, development and management [4]. A static and homogeneous configuration provides attackers a good opportunity to study the target environment (i.e., the version and amount of installed OSs) for attack preparation (i.e., acquiring specific tactics to exploit certain vulnerabilities). On the other hand, detecting reconnaissance traffics is complicated. Attackers could employ existing networking tools to perform reconnaissance, and disguise corresponding data packets with benign missions in the cloud [5].

* Corresponding author (email: viaviavialhr@outlook.com)

Existing studies propose defensive deception to counter reconnaissance attacks to reverse the asymmetrical disadvantage for the defender [6]. Different from signature-based countermeasures (i.e., intrusion detection system), defensive deception does not rely on prior knowledge to identify attacks. Instead, it makes use of attackers' incomplete cognition, and presents digital decoys to obfuscate reconnaissance [7]. Therefore, defensive deception is beneficial to not only camouflage system assets (i.e., by presenting decoy fingerprinting), but also waste attack efforts (e.g., preparing tactics for falsified vulnerabilities).

To maximize the effectiveness of deception and fully achieve the defense goal, digital decoys should be configured and placed strategically in the cloud [8]. That is, the defender should develop an optimal deception strategy. Related work in this field is insufficient in the following aspects. (1) There is a lack of adaptability to the multi-tenant cloud network [9]. Existing studies mainly established deception strategies for the Intranet or web servers using game theory or attack graphs [10–12]. However, these strategies are less optimal in configuring and placing decoys for the cloud due to the increasing size of networks. As the size of a network does not only refer to the number of hosts, but also to the number of active services and associated vulnerabilities, solving equilibria or attack graphs will encounter climbing complexity [13,14]. (2) There is an inadequate fulfillment of a comprehensive deception defense goal [15]. A comprehensive defensive deception goal shall comprise multiple aspects, such as obfuscating reconnaissance by concealing system assets and distracting attack resources. However, existing studies often consider a single perspective, such as mere attack prevention, attack detection, or attack mitigation. This condition restraint will be the potential of deception defense in the cloud. (3) The deception strategy has little granularity. Existing studies often present coarse-grained strategies and fail to specify detailed configuration for decoys to a node-level. This limitation hinders the practical implementation of defensive deception in the real world.

As the deception strategy defines the granular configuration and placement of decoys in the cloud, we propose to formulate the solution of an optimal strategy as a security decision problem. The emerging machine learning (ML) and artificial intelligence (AI) technologies bring us a new horizon to solve such problems. For instance, Qu et al. [16] proposed the use of generative adversarial networks to ensure the security of location information of 5G networks. Ning et al. [17] studied a deep reinforcement learning (DRL)-based approach to allocating resource for the mobile blockchain security. DRL is a combination of a deep neural network (DNN) and the reinforcement learning (RL). Accordingly, it is possible to solve the deception strategy. First, the RL framework trains an agent to make an optimal configuration and placement strategy through a trial-and-reward process. During training, the agent completely observes the threat scenario of the cloud (i.e., the OS associated common vulnerabilities of multiple tenant networks) and generates decision actions to earn maximum accumulated reward (i.e., fulfilling defense goals). Second, the DNN serves as a function approximator to map high dimensional input data with optimal output actions. This method guarantees a more granular deception strategy for the cloud.

Therefore, we propose a defensive deception framework against reconnaissance attacks in the cloud with DRL. Our work examined the common vulnerability-based threat scenario for the cloud, and customized a DRL agent to generate the optimal deception strategy. The framework eventually presents a more granular configuration and placement for digital decoys to counter reconnaissance attacks. Our main contributions are as follows.

(1) We propose a defensive deception framework to counter reconnaissance attacks in the cloud tenant network. To generate the optimal deception strategy, the framework considers concealing assets' configuration and increasing the attack cost. The strategy configures and places decoys granularly to fulfill the aforementioned comprehensive defense goal.

(2) We formulate a utility function to model the OS common vulnerability-associated threat scenario for the cloud. Then we customize a DRL agent to solve the optimal deception strategy based on the utility function.

(3) The simulation results show that the proposed defensive deception framework increases the attack cost from 38.33% to 40.40% on average under different proportions and varieties of decoys, and improves the concealment of the system asset up to 20.58%.

The rest of this paper is organized as follows. Section 2 introduces the background and motivation. Section 3 formulates the problem. Section 4 presents an overview of the proposed framework, and Section 5 presents the details of the framework design. Section 6 presents and analyzes the simulation and evaluation. Finally, Section 7 concludes this work.

2 Background and motivation

2.1 Reconnaissance attacks in the cloud tenant network

In this study, we focus on internal reconnaissance attacks discussed by Roy et al. [18]. The adversary usually performs technical reconnaissance (i.e., network scanning and probing) in the target cloud tenant network through a previously compromised node. Although this action does not introduce instant damage, it brings lasting and consequential impacts to the system. In a tenant network, if the adversary maintains continuously stealthy reconnaissance, it will gradually be prepared to discover exploitable common vulnerabilities and launch devastating attacks. This condition will eventually cause a giant loss to the cloud. Countering internal reconnaissance is significant because the cloud tenant network provides legitimate users with production services. Once being attacked, the interruption of multiple processes and services will suffer immense loss [19]. However, it is difficult because of the following reasons.

(1) Reconnaissance attacks have technical advantage [20]. Network scanning techniques and tools, such as the ICMP ping and the NMap, provide the attacker with mature tactics to study information and configuration (i.e., IP blocks, OS fingerprinting, and software vulnerabilities).

(2) Malicious reconnaissance is often overlooked and difficult to identify [5]. On the one hand, the attacker's occasional probings may seem harmless and are sometimes neglected by the security administrator. On the other hand, such traffics are mixed with common legitimate network missions and are hard to identify.

(3) The static and homogeneous virtual infrastructure of the cloud tenant network benefits from reconnaissance attacks [21]. On the one hand, the static environment offers attackers extra time to systematically study the target. On the other hand, the homogeneous configurations allow attacks to be propagated effortlessly due to common vulnerabilities.

2.2 Defensive deception for the cloud

Defensive deception is promising to counter reconnaissance as an active defense. It provides falsified intelligence to confuse attackers (i.e., using decoys to authentic system configuration), which will invalidate information they collected during reconnaissance (i.e., exploitable vulnerabilities associated with system configurations) and affect their following steps (i.e., prepare to exploit specific vulnerabilities). Existing studies in this field could be divided into deception techniques and deception strategies.

The history of deception techniques dates back to the development of the Deception ToolKit [22], where Cohen proposed various tactics including concealment, camouflage, falsified, and planted. Afterward, Spitzner presented the honeypot system, a digital decoy, to lure, log, and learn malicious access [23]. The effectiveness of deception techniques largely hinges on their interaction with attackers. Therefore, related research has focused on improving the enticement of decoys. Nowadays, state-of-the-art techniques, such as ML and natural language processing (NLP), are engaged to evolve deception techniques [8]. Various high-interactive digital decoys have been researched, such as honeytokens [24], honey-patches [25], Honeyfile, decoy services [26], and decoy network devices [27]. These techniques can be installed in the cloud as countermeasures [28].

Deception strategy is another component in implementing deception defense. A defender tailors specific strategies to orchestrate enticing decoys according to defense goals (i.e., confusing reconnaissance) [29]. Game theory and attack graphs are often studied in developing the deception strategy [30]. In defending cloud networks, Kandoussi et al. [20] used stochastic games to allocate honeypots and detect stealthy attackers. Almohri et al. [31] developed a digraph-based strategy to place fake services in the cloud to delay remote attackers. Other studies, such as [10, 32], leveraged attack graphs to model the virtual network architecture and solve the placement of honeypots with the game theory. Satisfiability modulo theories (SMT) were also employed to search for the optimal deception strategy. Duan et al. [15] proposed CONCEAL to search for the optimal combination of shadow honeypots with the SMT.

2.3 Limitations of existing studies

An optimal deception strategy should be adaptive to certain threat scenarios and defense goals (i.e., against the reconnaissance attacks in the cloud tenant network), considering a comprehensive defense goal (i.e., concealing system assets and increasing the attack cost), and granular for decoys (i.e., specifying

deployable amounts and configurations). With these requirements, we conclude that the main limitations of existing studies are as follows.

(1) Lack of adaptability to the multi-tenant cloud network [9]. Existing studies mainly employ game theory and attack graphs to search for the optimal strategy for the Intranet or web. On the one hand, they focus less on the common vulnerability of critical software components. On the other hand, their approaches are less resilient to concerning the size of multi-tenant cloud networks.

(2) Inadequate fulfillment of the comprehensive defense goal [15]. A comprehensive deception defense goal should comprise multiple perspectives, such as the concealment of system assets and the increase in attack cost. However, existing studies often achieve a single defense goal, such as attack prevention, attack detection, and attack mitigation [8].

(3) Insufficient granularity in configuration [8]. Existing studies mainly employed coarse-grained deception strategies. For example, Ref. [10,33] studied the optimal amount to deploy honeypots, Ref. [31,32] studied the optimal location to place decoys in the obscured cloud and Intranet network topology. These studies provide novel ideas to establish deception strategies. However, they fail to specify the exact type and configuration of corresponding decoys (e.g., versions of OSs and services). This condition hinders defenders from precisely orchestrating credible decoys in the cloud.

2.4 Opportunities

The emerging DRL is promising for building up an effective deception strategy for the cloud. DRL is a combination of the DNN and RL [34]. The RL framework trains an agent to learn from a trial/error interaction under a dynamic environment. It reasonably models the attacker's stealthy reconnaissance on the cloud assets (including the real and decoys). Aligning with the DNN as the function approximator, DRL can map high-dimensional input data (i.e., the threat scenario) with optimal decisions (i.e., granular deception strategy). For example, Sethi et al. [35] applied DRL to detect intrusions in the cloud infrastructure. They trained the DRL agent to classify and respond to attacks with near real-time VM logs.

Another important benefit of solving the deception strategy with DRL is that the corresponding model training does not require a beforehand statistical analysis. That is, the DRL agent will be adaptable to various threat scenarios (e.g., different sizes and vulnerable nodes) of multiple tenant networks in the cloud.

A profound study on the enticement and credibility of digital decoys also empowers the implementation of defensive deception in the cloud. For instance, a mainstream technique, Honeyd [36], allows us to distribute and manage virtual hosts as decoys across the local area network (LAN) with a daemon process. Moreover, Cowrie [37], a mid-interactive decoy system, could be built into the container. Such techniques could be utilized in the cloud without occupying overwhelming computing resources. Moreover, a majority of existing studies on decoys, including Honeyd and Cowrie, accept customization using a configuration file. This function provides a solid foundation to configure and present decoys with the granular deception strategy generated by the DRL agent.

Accordingly, this work focuses on a DRL-enabled deception strategy. We customized neural networks and threat scenario interfaces for the DRL algorithm. After multiple rounds of interaction, the well-trained DRL agent generated a fine-grained strategy for decoys to counter reconnaissance attacks. The defense goal was formulated as the training goal, ensuring that the strategically deployed decoys could maximize obfuscation on reconnaissance and increase the attack cost.

3 Problem formulation

In this section, we propose a threat model for the multi-tenant cloud network and formulate the defense problem. In the threat model, we consider a single attacker,whose ultimate goal is to collect configuration information of the tenant network through limited rounds of probing. The configuration information mainly refers to the security settings, OS fingerprints, and vulnerabilities. The defender's goal is to obfuscate the attacker's reconnaissance and waste attack resources by strategically configuring and presenting decoy hosts. The notations used for problem formulation and threat are summarized in Table 1.

Table 1 Summary of notations

Notation	Description
$\Omega : \mathcal{H}_n^{i,\mathcal{X}} \to \{O_{\mathbb{N}} \times \mathcal{X}\}$	A finite set of configuration information of the target tenant network
n	The amount of hosts in the target tenant network
$\mathcal{H}_n^{i,\mathcal{X}}$	The nth host in the target tenant network with i OS and $\mathcal{X}$ security configuration
$O_{\mathbb{N}} = \{\mathrm{os}_i \vert i = 1, 2, \ldots, m\}$	A subset of m-type OS configuration in the target tenant network
$\mathcal{X} \in \{\mathrm{vul}, \mathrm{sec}, \mathrm{dec}\}$	A finite set of security configuration, denoting vulnerable, secured and decoy hosts, respectively
$\mathcal{U} = \sum_i \sum_{\mathcal{K}} w^{\mathcal{K}} \mathcal{V}_{i,x}^{\mathcal{K}}$	Utility function that maps the possible impacts with OS common vulnerabilities
$\mathcal{K} = \{s, k, a, d\}$	The categories of vulnerabilities: system software, kernel, application, and driver
$\mathcal{E}_d$	Deception entropy that evaluates the concealment degree of system assets
$O_{\mathbb{A}}$	Attack strategy, a subset to $O_{\mathbb{N}}$
$O_{\mathbb{D}}$	The set of OS for defense resources, a subset to $O_{\mathbb{N}}$
$P(O_{\mathbb{D}})$	The defender's strategy, specifying the distribution over OS configuration in $O_{\mathbb{D}}$

3.1 Preliminaries and metrics

Firs, we use n to denote the amount of hosts $\mathcal{H}_n$ in a target tenant network. Let Ω denote a finite set of configuration information of the target, such that $\Omega : \mathcal{H}_n^{i,\mathcal{X}} \to \{O_{\mathbb{N}} \times \mathcal{X} | \mathcal{X} \in \{\mathrm{vul}, \mathrm{sec}, \mathrm{dec}\}\}$. $O_{\mathbb{N}}$ represents the OS configuration, where $O_{\mathbb{N}} = \{\mathrm{os}_i | i = 1, 2, \ldots, m\}$ containing m types of OS. And $\mathcal{X}$ refers to the security configuration of each host, where $\mathrm{vul}, \mathrm{sec}, \mathrm{dec}$ denotes vulnerable, secured, and decoy hosts, respectively. In particular, different Ω with dissimilar variants represents different tenant networks in the cloud platform.

In the target network, $\forall \mathrm{os}_i \in O_{\mathbb{N}}$ shares a number of common vulnerabilities with one another. If a vulnerability could be used to impact one or more components, then it can be used to compromise many or all of software components simultaneously. This kind of vulnerability is defined as the common vulnerability. Garcia et al. [3] classified OS vulnerabilities into the following stacks: system software, kernel, application, and driver. We use $\mathcal{V}_{i,j}^{\mathcal{K}}$ to denote the number of common vulnerabilities between a pair of OS, os_i, and os_j, where $\mathcal{K} = \{s, k, a, d\}$ denote the category, and $i \neq j, \mathrm{os}_i, \mathrm{os}_j \in O_{\mathbb{N}}$. $\mathcal{V}_{i,i}^{\mathcal{K}}$ is a scalar denoting the number of vulnerabilities in a single OS, os_i. Hereby, we define a utility function to calculate the impacts of OS-associated common vulnerabilities:

Definition 1 (Utility function $\mathcal{U}$)**.** $\mathcal{U}$ maps the possible impacts with the amount of common vulnerabilities among OSs. The weights $w^{\mathcal{K}}$ denote the severeness of vulnerabilities being exploited per category [19]. Exploiting on vulnerabilities in each category has dissimilar impacts and difficulty, such that

$$\begin{aligned} \mathcal{U} &= \sum_i \sum_{\mathcal{K}} w^{\mathcal{K}} \mathcal{V}_{i,x}^{\mathcal{K}}, \\ \text{s.t. } & i = 1, 2, \ldots, m, \\ w^{\mathcal{K}} &= \begin{cases} 4, & \mathcal{K} = s, \\ 3, & \mathcal{K} = k, \\ 2, & \mathcal{K} = a, \\ 1, & \mathcal{K} = d. \end{cases} \end{aligned} \tag{1}$$

Definition 2 (Deception entropy $\mathcal{E}_d$)**.** Measures the concealment degree of system configuration and evaluates the confusion of information introduced by deception. The concealment of system configuration is used to evaluate the effectiveness of defensive deception [15]. Ever since Shannon has named missing information by entropy in the information theory, it has become a universally accepted definition of this term. In the network system, existing studies employ different forms of entropy to describe the diversity or richness of configuration [38], whose fundamental measurement is the information presented to potential observers. The insight of deception entropy $\mathcal{E}_d$, is to evaluate the concealment of system configuration and confusion on information when decoys are deployed in the tenant networks.

$$\mathcal{E}_d = -\sum^{i} p_i \ln p_i \quad \text{s.t.} \quad p_i = \frac{\sum_{\mathcal{K}} \mathcal{V}_{i,i}^{\mathcal{K}}}{\sum_i \sum_{\mathcal{K}} \mathcal{V}_{i,i}^{\mathcal{K}}}. \tag{2}$$

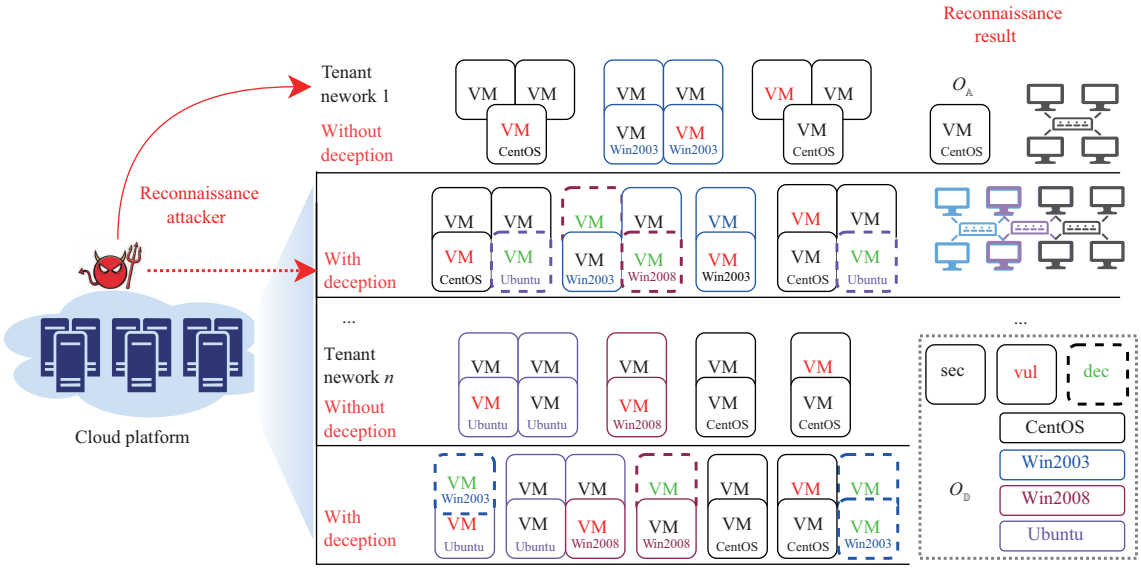

Figure 1 (Color online) Obscured architecture of a multi-tenant cloud platform, where malicious users could conduct reconnaissance on penetrating into benign tenant networks. The red arrow points to the case where no decoy is placed and the attack could gain the real OS and number of VM. The dashed arrow presents a decoy-placed occasion. $\mathcal{X} \in \{\text{vul}, \text{sec}, \text{dec}\}$ are depicted by solid line and black letters, solid line and red letters, and dashed line and green letters, respectively. Examples of different OS configurations are presented in black, blue, burgundy, and purple, respectively. For example, in tenant network 1, if without deception, the attacker could acquire that the production hosts are 7 CentOS VMs and 4 Win2003 VMs. However, the attacker does not know that there are three vulnerable hosts with CentOS and Win2003. With deception, the attacker might find out 6 CentOS VMs, 4 Win2003 VMs, 2 Win2008 VMs, and 2 Ubuntu VMs.

Because the information refers to common vulnerabilities associated with the OS configuration for VMs, the entities measured by $\mathcal{E}_d$ are OS vulnerabilities. As the decoy hosts are seamlessly merged with production nodes from the observer's view (e.g., adversaries or legitimate users), the vulnerabilities include both real and deceptive configurations.

3.2 Threat model and defense objectives

Although tenant networks are isolated on a cloud platform, malicious users could conduct lateral movement by side-channel attacks, co-resident attacks or VM escape. Once penetrated into a benign tenant network, the attacker utilizes network scanning tools to collect the information of active hosts. Afterwards, based on the reconnaissance results, the attacker could weaponize for the maximum attack benefits. A typical threat scenario is depicted in Figure 1.

Attack capability. We assume that the attacker is a malicious user in the cloud platform. It attempts to penetrate into the target tenant network and is capable of performing reconnaissance using scanning tools. The attacker aims to collect the information of infrastructure in the tenant network, especially OS vulnerabilities. Moreover, the attacker's capability is limited. First, we assume that the attacker does not acknowledge the security configuration in the network; that is, it cannot locate and identify vulnerable and decoy hosts. Second, the attacker is a rational entity. It attempts to maximize probing benefits while minimizing attack costs.

Attack strategy. Choosing to weaponize on specific OSs is considered an attacker's strategy [39]. The attacker attempts to earn the highest reward (e.g., exploiting the most common vulnerabilities), and it chooses the OS configuration of a host with equal probability to weaponize. This is because it could not recognize the exact OS distribution of vulnerable hosts, but it would not pass valuable information. Accordingly, we define the attack strategy and costs as follows.

Definition 3 (Attack cost $C_{\mathbb{A}}$)**.** Let $O_{\mathbb{A}} = \{\text{os}_a | \text{os}_a \in O_{\mathbb{N}}\}$ denote a subset of OS configurations formulating the attack strategy. We define the attacker's cost by the possible utility that he could gain in vulnerable hosts $\mathcal{G}_{\mathbb{A}}$, and lose in decoy hosts $\mathcal{L}_{\mathbb{A}}$, such that

$$\begin{aligned}\mathcal{G}_{\mathbb{A}} &= \frac{\mathcal{U}|_{i=i,\mathcal{X}=\text{vul}}}{\mathcal{U}|_{i=O_{\mathbb{A}},\mathcal{X}=\text{vul}}},\\ \mathcal{L}_{\mathbb{A}} &= \frac{\mathcal{U}|_{i=i,\mathcal{X}=\text{dec}}}{\mathcal{U}|_{i=O_{\mathbb{A}},\mathcal{X}=\text{dec}}},\\ C_{\mathbb{A}} &= \mathcal{L}_{\mathbb{A}} - \mathcal{G}_{\mathbb{A}}.\end{aligned} \tag{3}$$

The utility function $\mathcal{U}$ defined in 1 measures the possible impacts if the attacker weaponizes based on a specified OS, i.e., the severeness of damage it could cause to the target environment by common vulnerabilities between this OS and the other working nodes. Because our work is designed to provide seamless decoy hosts, the attacker's determination on the target OS will be accordingly influenced. That is, the attacker will be likely to prepare the following tactics for non-existing common vulnerabilities (derived from decoy hosts). This situation is regarded as a waste of attack resources, denoted by the loss of the attacker, $\mathcal{L}_{\mathbb{A}}$. Contrarily, the situation that the attacker exploits real common vulnerabilities will be regarded as the gain of attacker, denoted by $\mathcal{G}_{\mathbb{A}}$. Therefore, the defense goal to maximize the attack cost could be formulated into maximizing the minus between the gain and loss.

For simplification, we only consider the utility of decoy hosts as the attacker's loss and neglect the reconnaissance costs because both scanning vulnerable hosts and decoys are inevitable costs for reconnaissance attacks. The decoy hosts are a defense resource labeled as $O_{\mathbb{D}}$, and $\mathcal{L}_{\mathbb{A}}$ also represents the defender's benefit.

Figure 1 depicts an example of the threat scenario. In tenant network 1, without defensive deception, the attacker could acquire that production hosts are 7 CentOS VMs and 4 Win2003 VMs. However, the attacker does not know that there are three vulnerable hosts with CentOS and Win2003. With deception, the attacker might find out 6 CentOS VMs, 4 Win2003 VMs, 2 Win2008 VMs, and 2 Ubuntu VMs. For the former case, the attacker may choose to exploit the vulnerabilities of CentOS as its attack strategy $O_{\mathbb{A}}$. Once weaponized based on the CentOS, most production hosts are at risk, especially the vulnerable hosts. However, for the latter case where reconnaissance results are manipulated by the decoy hosts, the attacker will be distracted in the strategy-making. It will be likely to prepare for exploiting on decoy hosts without gains, thus allowing the defender to waste attack efforts.

Defense strategy. The defender's goal is to obfuscate the attacker's reconnaissance and increase attack costs by strategically configuring decoys. We formulate the defender's strategy as installing different OSs on decoy nodes and assigning these nodes with an unused IP address. To implement such a strategy, the defender could utilize a system daemon to generate and support virtual decoy hosts [36]. Therefore, it will not dramatically occupy valuable cloud computing resources. Let $O_{\mathbb{D}}$ denote a set of available OS configurations for decoy hosts and $P(O_{\mathbb{D}})$ denote the defender's strategy, referring to the distribution of different OSs in $O_{\mathbb{D}}$, such that

$$\begin{aligned} &P(O_{\mathbb{D}}) = \{p(\text{os}_d)|\text{os}_d \in O_{\mathbb{D}}, O_{\mathbb{D}} \subset O_{\mathbb{N}}\}\\ &\text{s.t. } p(\text{os}_d) \in [0,1),\\ &\sum^{d} p(\text{os}_d) = 1.\end{aligned} \tag{4}$$

In (4), the maximum probability of a single-decoy OS is less than 1 to avoid all decoy hosts with the same configuration. The defense objective is to optimize the strategy to maximize the deception entropy $\mathcal{E}_d$ in (2), and attack cost $C_{\mathbb{A}}$ in (3).

4 Framework overview

To obfuscate the reconnaissance attack in cloud tenant network and increase the attacker's cost, we propose a configuring digital decoy to conceal system assets deceptively. Our work includes a fine-grained deception strategy, presenting the version and amount of OSs to be configured for decoys. To fully achieve the comprehensive defense goal, we trained a DRL agent to interact with the cloud and generated optimal decision strategies.

Figure 2 presents the overview of our framework. First, the utility function $\mathcal{U}$ is updated based on real-time configurations (i.e., the number of active working nodes and OS vulnerabilities). The utility

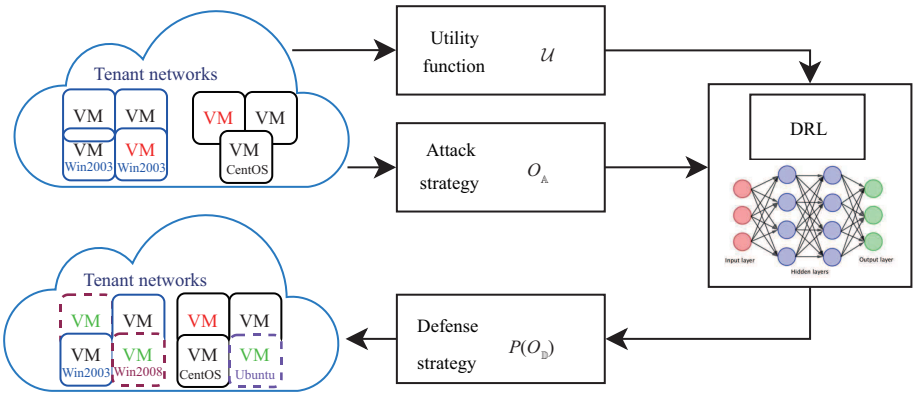

Figure 2 (Color online) Components of the framework.

function $\mathcal{U}$ and attack strategy are sent to the input neurons to train the DRL agent. The attack strategy is generated following [33]. Afterward, through the calculation of the neural network, the DRL agent generates a list of probability distributions as actions. The finite set of actions formulates the defense strategy. Meanwhile, $\mathcal{E}_D$ and $C_{\mathbb{A}}$ are calculated in accordance with $\mathcal{U}$. They are used as rewards for evolving the DRL agent. Back and forth, the DRL agent learns to operate under different tenant networks and attack strategies. Lastly, the defense strategy is formatted and sent to the tenant network as scripts to configure decoys on working nodes for defense.

5 Details of the defense framework

In this section, we present the design details of the proposed defensive deception framework, which mainly contains two algorithms: (1) update of the utility function $\mathcal{U}$ and (2) DRL algorithm.

5.1 Update of the utility function

A primary component of the framework is the utility function $\mathcal{U}$. $\mathcal{U}$ transfers OS vulnerabilities and security settings to the impacts on the target environment. Such an impact varies from $\mathcal{K}$ categories and $\mathcal{X}$ security settings. Subsequently, the attack cost $C_{\mathbb{A}}$ is calculated according to (3). Let $N_{\text{vul}}, N_{\text{sec}}, N_{\text{dec}}$ denote the amount of hosts under the security settings. The configuration information of the tenant network is transferred into $\Omega = \{\mathcal{H}_n^{i,\mathcal{X}} | i = 1, 2, \ldots, m\}$, where z is the unused IP address in a tenant network. The calculation and update of the utility function are presented in Algorithm 1.

Line 1 initializes $\mathcal{G}_{\mathbb{A}}$, $\mathcal{L}_{\mathbb{A}}$, and $C_{\mathbb{A}}$. Line 2 initializes utility weights $w^{\mathcal{K}}$ for OS vulnerabilities in each category. Line 3 initializes transition list entities. Lines 4–7 calculate the types of OSs for decoy hosts after installing the defense strategy $P(O_{\mathbb{D}})$. Lines 8 and 9 extract decoy and vulnerable configuration to dconf and vconf, respectively. Afterward, lines 10–15 and 16–21 update the attack gain $\mathcal{G}_{\mathbb{A}}$ and loss $\mathcal{L}_{\mathbb{A}}$, respectively. We divide both $\mathcal{G}_{\mathbb{A}}$ and $\mathcal{L}_{\mathbb{A}}$ with the ideal value of maximum utility for normalization.

5.2 Algorithm framework of DRL

We leverage DRL to generate an optimal deception strategy. The types and amounts of OS configurations are optimally decided. In the DRL framework, the agent interacts with a dynamic environment (i.e., the different vulnerability conditions in tenant networks). Through the interaction, it evolves to output optimal decisions under a typical RL framework. The DNN eventually maps the threat model to a series of fine-grained deception configurations.

The DRL agent observes a state s_t of the tenant network at each step t. Then, the agent generates an action a_t based on a policy π. The agent's action a_t changes the environment and gets a reward r_t. That is, the agent changes the configuration and placement of decoys, and examines how the reconnaissance will be obfuscated. Based on r_t, the agent updates its policy. In the RL framework, the learning objective is to maximize the cumulative reward over a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots)$:

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

Algorithm 1 Update of the utility function $\mathcal{U}$

Input: host amount $N_{\text{vul}}, N_{\text{sec}}, N_{\text{dec}}$, tenant network configuration Ω, attack strategy os_x, defense strategy $P(O_{\mathbb{D}})$, vulnerability categories $\mathcal{K}$, and common vulnerability table $\text{Comm}_{\mathcal{V}}$;
Output: attack gain $\mathcal{G}_{\mathbb{A}}$, attack loss $\mathcal{L}_{\mathbb{A}}$, and attack cost $C_{\mathbb{A}}$;
Initialize $\mathcal{G}_{\mathbb{A}} \Leftarrow 0$, $\mathcal{L}_{\mathbb{A}} \Leftarrow 0$, $C_{\mathbb{A}} \Leftarrow 0$;
Initialize $w^{\mathcal{K}} = \{s:4, k:3, a:2, d:1\}$;
Initialize list $N_{O_{\mathbb{D}}}$, lmax, loss, gmax, gain;
while $p(\text{os}_i)$ in $P(O_{\mathbb{D}})$ **do**
    tmp = round$(p(\text{os}_i) \cdot N_{\text{dec}})$;
    $N_{O_{\mathbb{D}}}$.append(tmp);
end while
dconf = dict(zip($O_{\mathbb{D}}$,$N_{O_{\mathbb{D}}}$));
vconf = Counter(list(Ω[0:N_{vul}]));
while os in vconf **do**
    temp = vconf[os] $\cdot \sum^{\mathcal{K}} w^{\mathcal{K}} \cdot \text{Comm}_{\mathcal{V}}[\text{os}_x][\text{os}]$;
    tmax = vconf[os] $\cdot \sum^{\mathcal{K}} w^{\mathcal{K}} \cdot \text{Comm}_{\mathcal{V}}[\text{os}][\text{os}]$;
    gain.append(temp);
    gmax.append(tmax);
end while
while os in dconf **do**
    temp = dconf[os] $\cdot \sum^{\mathcal{K}} w^{\mathcal{K}} \cdot \text{Comm}_{\mathcal{V}}[\text{os}_x][\text{os}]$;
    tmax = dconf[os] $\cdot \sum^{\mathcal{K}} w^{\mathcal{K}} \cdot \text{Comm}_{\mathcal{V}}[\text{os}][\text{os}]$;
    gain.append(temp);
    gmax.append(tmax);
end while
$\mathcal{G}_{\mathbb{A}} \Leftarrow \frac{\text{gain}}{\text{gmax}}$;
$\mathcal{L}_{\mathbb{A}} \Leftarrow \frac{\text{loss}}{\text{lmax}}$;
$C_{\mathbb{A}} \Leftarrow \mathcal{L}_{\mathbb{A}} - \mathcal{G}_{\mathbb{A}}$.

where γ is the discount factor. The DRL agent maximizes $R(\tau)$ by evolving its policy based on a value function $V^{\pi}(s)$. It evaluates the quality of the current policy for the environment starting from state s, such that

$$V^{\pi}(s) = \underset{\tau \sim \pi}{E}[R(\tau)|s_0 = s]. \tag{6}$$

Traditionally, the RL learning algorithm involves a lookup table to search for optimal policies. However, for problems with a continuous or overly large state space, the lookup table will be infeasible in the solution and storage. Therefore, Mnih et al. [34] proposed the use of DNN to approximate value functions. Here, the agent's objective is to optimize the parameterized policy π_θ and value function $V^{\pi}(s)$. The defense framework updates parameters with the actor-critic (AC)-style proximal policy optimization algorithm. The AC architecture includes two neural networks, namely, the actor network and the critic network. θ and ϕ denote the parameters for the actor and critic networks, respectively. The actor network interacts with the dynamic environment and updates the policy to generate optimal actions. Meanwhile the critic network adjusts value function to evaluate actions (i.e., the generated deception strategy).

In the defensive deception framework for the cloud, the dynamic environment is the tenant network. Hence, the state of the environment indicates the configuration information of a set of target tenant networks. Taking massive configuration information as the input data, the DRL agent generates corresponding actions. The output action formulates the deception defense strategy. It includes the probability distribution over available OSs for decoy hosts. Hence, the employed DRL algorithm should be able to handle a large input space and continuous action space. Therefore, we involved the PPO [40] in our framework because it has a good data efficiency and robustness and is compatible with continuous data spaces.

To train the DRL agent, we use multilayer perceptron (MLP), a class of feedforward neural networks, for both the actor and critic networks. Both networks share the same MLP structure, which has two hidden layers and each layer has 64 units. The input layer takes the state s_t composed of configuration details Ω_t as the input data. Meanwhile, Ω_t is sent to the utility function for reward calculation r_t. The activation function for hidden layers is the tanh function, mapping weighted inputs to outputs for each unit. In employing the DRL agent to generate optimal defense actions for the cloud tenant network, the efficiency of sampling data and learning progress are two main considerations. To promote the efficiency of sampling, the actor network utilizes the probability ratio $r_t(\theta)$ to fit the deviation between an old policy and an updated one. To improve the learning efficiency, the critic network leverages generalized advantage estimate (GAE) to generate an advantage surrogate based on the current value function V_{ϕ_k}.

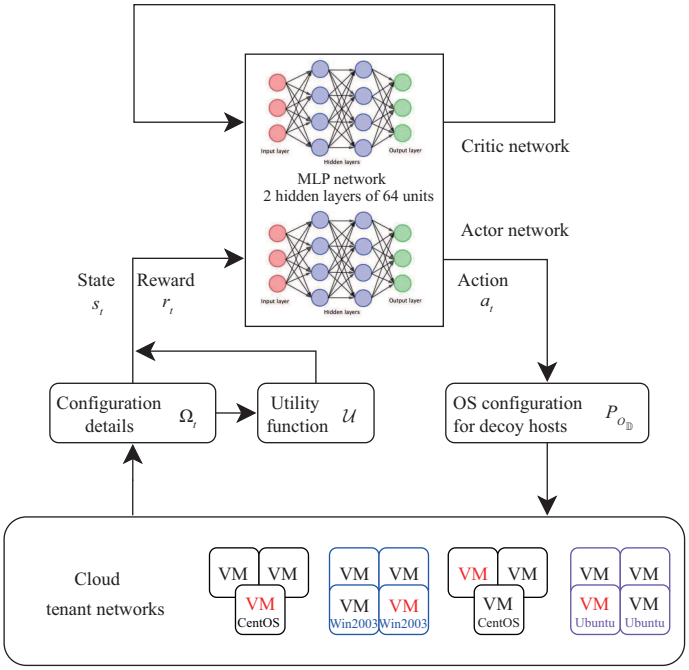

Figure 3 (Color online) Training architecture of the proposed framework.

Table 2 The setting of parameters for the DRL agent

	γ	λ	ϵ	Learning rate	Batch size
Value	0.99	0.95	0.2	0.0003	64

The objective function is formulated as follows:

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

where $r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ and $\hat{A}_t$ is an estimator of the advantage function at time-step t. θ_{old} denotes the vector of policy parameters before the update and $\hat{\mathbb{E}}_t$ indicates the expectation [40]. To control the learning rate for the policy, a hyperparameter ϵ is introduced to clip the stepsize:

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

where $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ modifies the surrogate objective by clipping $r_t(\theta)$. The minimum between the clipped and unclipped objectives is taken as a lower bound of the final objective. This condition avoids local optima or prolonging the training time. For the optimization algorithm, the actor network adjusts and updates based on the Adam algorithm, and the critic network is optimized with mini-batch gradient descent. The MLP network and the training structure used in this DRL framework are shown in Figure 3.

The pseudocode of the DRL algorithm used in this work is shown in Algorithm 2. Line 1 parameterized and initialized the policy and value function. In line 3, the algorithm generates the defense strategy $P(O_\mathbb{D})_t$ to configure decoy hosts based on the policy π_k. Lines 4 and 5 compute rewards and advantage estimates for the agent when interacting with the network with configuration Ω_t by the policy π_k. Line 6 updates the policy with the parameter θ, where $\theta_{k+1} = \arg\max_\theta E_{s,a\sim\pi_{\theta_k}}[L^{\text{CLIP}}(\theta_k)]$. To avoid the overly aggressive or passive updates of the old policy, the hyperparameter ϵ is set as 0.2. Line 7 updates the value function for the critic network.

The main parameters used in training the defense DRL agent include the discount factor γ for the accumulated reward $R(\tau)$, the GAE parameter λ, the parameter for the clipping range ϵ, the earning rate, and the batch size. The detailed settings of each parameter are presented in Table 2.

Algorithm 2 DRL algorithm of the framework

Input: tenant network configuration $\Omega_t = s_t$, attack strategy os_x^t;
Output: defense strategy $P(O_{\mathbb{D}})_t = a_t$, neural network parameters of DRL θ, ϕ;
1: Initialize parameters θ_0 and ϕ_0 for the policy and value function, respectively;
2: **while** $k = 0, 1, 2, \ldots, n$ **do**
3: Collect a set of trajectories $\mathcal{D}_k = \{s_t, a_t\}$ by running policy $\pi_k = \pi(\theta_k)$ to configure the tenant network;
4: Compute rewards-to-go $\hat{R}_t$;
5: Compute advantage estimates $\hat{A}_t$ based on the current value function V_{ϕ_k};
6: Update the policy by maximizing the clipped objective based on ADAM:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A});$$

7: Fit value function with the mini-batch gradient descent:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} (V_{\phi}(s_t) - \hat{R}_t)^2.$$

8: **end while**

5.3 Implementation details of DRL interfaces

According to Algorithm 2, the main interfaces between the DRL agent and cloud tenant network are the state, action, and reward. We present the detailed design of the customized DRL interfaces as follows.

State. The state information is the input of the neural networks. In the proposed framework, the environment to interact with the DRL agent is the cloud tenant network. The state is presented by the OS configuration of each node in the cloud at each orchestration period, denoted by Ω_t. The security status of working nodes is divided into vulnerable, secure, and decoy, denoted by $\mathcal{X} \in \{\mathrm{vul}, \mathrm{sec}, \mathrm{dec}\}$. Hereby, let $\boldsymbol{s}_t = \Omega_t = \{\mathcal{H}_n^{i,\mathcal{X}} | i = 1, 2, \ldots, m\}$ denote the state, where $\boldsymbol{s}_t$ is a vector indicating the OS configuration and security status for each host in the tenant network. To guarantee the scalability of the DRL agent to the changing size of tenant network, the length of $\boldsymbol{s}_t$ is formatted after the largest address space the defense framework could support. For small-size networks, let the zero-vector denote inactive nodes without any loss of the state space. Therefore, let $\mathcal{Z}$ denote the maximum address space, and z denote the number of unused addresses. The dimension of the state is defined as $\dim(\boldsymbol{s}_t) = (\mathcal{Z} - z) \cdot \dim(\Omega_t)$.

Action. The action is the output of neural networks. In the proposed framework, each action for a state is a list of probability distribution over $O_{\mathbb{D}}$, denoting the proportion of specific types of OSs for decoy hosts. We formulate the action according to (4) as $\boldsymbol{a}_t = [a_t^1, a_t^2, \ldots, a_t^d]$, where $a_t^d = p(\mathrm{os}_d) \in [0, 1), \sum_d a_t^d = 1$, refers to the to-be-deployed proportion of each os_d in the defense resource.

Reward. The reward provides a signal to evaluate the agent's decision, which often reflects the training goal. We formulate the training goal with the defender's intention. That is, reconnaissance is obfuscated by concealing real assets and increasing the attack cost. The DRL agent is expected to generate optimal decisions on OS configurations for decoy nodes to reach this goal. Therefore, let $\boldsymbol{r}_t = \mathcal{E}_d + C_{\mathbb{A}}$ denote the reward function at t-step, where both $\mathcal{E}_d$ and $C_{\mathbb{A}}$ are normalized forms.

5.4 Feasibility analysis

In the feasibility analysis, we present a more detailed rationale to show how existing techniques collaborate with the proposed defense framework. The practicability of our work is enabled by the availability of multiple OSs, profound studies on formulating decoy entities, and the automatic orchestration technology.

First, the readily available OSs set a foundation for creating enticing deceptive defense resources (i.e., OS installed on decoy hosts presenting deceptive fingerprints and vulnerabilities from production hosts). On the one hand, they could be directly installed on a decoy host, presenting a high-interactive decoy. On the other hand, their configuration details could be extracted and projected by low-interactive virtual decoy hosts created by deception tools.

Second, formulating decoy entities such as virtual decoy hosts could be supported by existing tools, such as Honeyd. It creates virtual decoy hosts on limited nodes in the target tenant network, and projects virtual hosts with customizable OS configurations for user-required network addresses. This method facilitates the practical implementation of the proposed framework in two aspects. (1) The customizable decoys could receive an optimal deception strategy and execute configurations accordingly through certain

interfaces. This setting is particularly significant for defense resource allocation in the cloud. Because manual settings and updates are inefficient in time and personnel. (2) Decoy OS configuration could be projected with virtual hosts on a limited number of nodes. This method strikes a balance between defense operation and resource saving. As reconnaissance attacks mainly focus on information collection, the adversary will have limited interactions with the target environment to stay stealthy. Providing deceptive OS configurations with low-interactive decoys is considered a reasonable approach to both achieve deception goals and optimize the defense budget for the cloud.

Third, with the development of virtualization technology and the prevalence of the cloud, a large number of automatic VM management tools have been built to facilitate resilient orchestration. With automatic management tools (e.g., Vagrant), the running status could be accessed through interfaces. This method facilitates the calculation of the input state data and reward function.

6 Simulation and evaluation

This section presents simulations and evaluations of the proposed framework. The effectiveness of defensive deception was evaluated and analyzed with the deception entropy $\mathcal{E}_d$ and attack cost $C_{\mathbb{A}}$, including the attack gain $\mathcal{G}_{\mathbb{A}}$ and loss $\mathcal{L}_{\mathbb{A}}$.

6.1 Simulation setup

First, we introduce the simulation setup. The $\mathrm{OS}_{\mathbb{N}}$ includes the following OSs: OpenBSD, NetBSD, FreeBSD, OpenSolaris, Solaris, Debian, Ubuntu, RedHat, Win2000, Win2003, and Win2008. Their common vulnerability information, the $\mathrm{Comm}_{\mathcal{V}}$ table, was extracted from [3]. The insight of the setup for $\mathrm{OS}_{\mathbb{N}}$ will be reasoned from the following perspectives. (1) The popularity of OSs in production scenarios. According to W3Techs' latest survey on the dominance of OSs for websites, Unix and Windows were used by 79% and 21.2% of all the surveyed websites, respectively. And there are less than 0.1% of websites that use macOS. As Linux is known as a Unix-like open-source OS, the survey's statistics on Unix include the utilization of Linux OS. According to the survey, among the two most popular series of OSs, Ubuntu, Debian, CentOS, and RedHat are the top four applied Linux OSs, and the rest of Unix includes Solaris and BSD OSs. We constructed $\mathrm{OS}_{\mathbb{N}}$ considering the popularity of the widely-applied OSs in generalizing the proposed framework for potential changes in the configurations of the tenant network. (2) The vulnerable state and common vulnerabilities statistics of OSs. The proposed defense framework aims at misguiding attackers with decoy configuration, and preventing them from inferring exploitable common vulnerabilities from reconnaissance. OSs with several common vulnerabilities should be paid equal attention.

We simulated 12000 Class-C cloud tenant network configuration samples to evaluate our framework. To ensure the believability of deception, we keep 10% of the address spaces unused. For the $\mathcal{X}$ setting, we set the number of vulnerable hosts vul, randomized from 16 to 64 in each simulated sample. Then, the sum of sec and vul is the remaining 90% of the address space minus the number of decoys. The DRL agent was implemented in Python 3.7 based on Tensorflow 1.14.

Second, the metrics we adopt to evaluate the effectiveness of the defensive deception framework are as follows.

(1) The deception entropy $\mathcal{E}_d$. According to Definition 2, $\mathcal{E}_d$ evaluates the concealment of system assets by decoy entities in the target network. The concealment degree reflects the effectiveness of defensive deception confusing the attacker at reconnaissance.

(2) Attack cost $C_{\mathbb{A}}$. By placing decoy hosts, the attacker will collect unidentifiable false information and influence his following weatherization. Hence, we used $C_{\mathbb{A}}$ to evaluate the increase in attack costs by introducing defensive deception. To better compare $C_{\mathbb{A}}$ with different tenant network configurations, we used the normalized metric $C_{\mathbb{A}}|_{\mathrm{norm}} = \frac{1}{\exp^{-C_{\mathbb{A}}}+1}$.

(3) The joint-defense goal. To evaluate the achievement of the comprehensive defense goal, we integrate $\mathcal{E}_d$ and $C_{\mathbb{A}}$ to formulate a scalar, namely, the joint-defense goal, joint-defense goal $= \mathcal{E}_d| + C_{\mathbb{A}}$, where both terms are normalized.

As surveyed in [8], the fine-grained placement strategy for decoys was understudied, hindering the feasibility of defensive deception in the real-world. To the best of our knowledge, we are the first to study a concrete placement and configuration strategy at the host level for decoys in the cloud. Hence, we compared our framework with a random strategy in reference to [41], and then we compare the

Table 3 The average decision-making time for each compared strategy

Size of the defense resource	Decoy proportion	DRL strategy (s)	Game theory strategy [19] (s)	CONCEAL strategy [15] (s)
$O_{\mathbb{D}} = 6$	0.15	0.785	13.4	238
	0.30	0.783	18.7	397
	0.60	0.779	27.9	570
$O_{\mathbb{D}} = 11$	0.15	0.788	35.4	483
	0.30	0.783	49.7	831
	0.60	0.787	67.3	1005

computing time with a game theory solution [19], and an SMT solution [15]. Furthermore, we compared the scalability with important studies in related field [42, 43].

Lastly, we introduce a set of variants in the simulation to evaluate the performance of defensive deception, including the following.

(1) Proportion of decoy hosts. According to [33], the amount of honeypots will have different impacts on attack costs. The proportion of decoy hosts refers to its amount out of the available address space. In this study, we evaluated the performance of defensive deception on inflection points studied in [33]. Accordingly, we simulated on 15%, 30%, and 60% of decoy hosts.

(2) OS diversity for decoy hosts $|O_{\mathbb{D}}|$. According to [3, 15, 19], the diversity of OSs adds to the obfuscation of reconnaissance attacks. We simulated on different sizes of $O_{\mathbb{D}}$ to evaluate the effectiveness of defensive deception. In the simulation, $|O_{\mathbb{D}}|$ is equal to $|O_{\mathbb{N}}|/2$ and $|O_{\mathbb{D}}|$. In particular, when $|O_{\mathbb{D}}| = |O_{\mathbb{N}}|/2$, the subset of $O_{\mathbb{D}}$ contains 6 OSs with the highest vulnerabilities. Defensive deception aims to present maximum falsified information to obfuscate the reconnaissance results.

6.2 Simulation results

Performance of the DRL agent. As aforementioned, we trained a DRL to generate the fine-grained deception strategy for decoys. The deception strategy decides the concrete amount and types to configure OSs for decoy nodes. The ultimate defense goal of this work is to obfuscate reconnaissance attacks using the following actions: (1) maximizing the concealment of system assets and (2) increasing the attack cost. The DRL agent aims to maximize the deception entropy $\mathcal{E}_d$ and the attack cost $C_{\mathbb{A}}$. Based on the simulation setup, we trained the DRL agent for the single defense goal, $\mathcal{E}_d$ or $C_{\mathbb{A}}$, and the joint-defense goal. The training performance is plotted in Figure 4.

Figure 4 shows that the algorithm is well convergent for all cases. However, a small $|O_{\mathbb{D}}|$ converges early, especially between Figures 4(a) and (b). This is because of the limitation of the DRL algorithm, where a small action space will have good efficiency. Moreover, the DRL agent performs better in the joint-defense goal than $C_{\mathbb{A}}$ and $\mathcal{E}_d$. We analyzed that the joint-defense goal has a trade-off between $C_{\mathbb{A}}$ and $\mathcal{E}_d$, which encourages the policy searching for the DRL agent to accelerate the convergence. Moreover, the findings prove the effectiveness of the DRL algorithm in generating optimal decisions for complicated configuration conditions of the cloud.

For the performance of the computation time and scalability, we compared the average strategy-generation-time for each cloud tenant network in Tables 3 and 4, respectively. In reference to Table 3, the decision-making time climbs with the increase of the size of defense resource and decoy proportion. Especially, the increase in $O_{\mathbb{D}}$ has a large impact for solutions proposed by [15, 19]. We reason that both solutions generate an optimal strategy by searching the entire strategy space each time. For a larger state and action space, the computational complexity is correspondingly growing. Moreover, the long time required in solving a large action space is due to many searching rounds under the same state space. Meanwhile for the DRL agent, it takes less than 1 s to output an optimal strategy after the training process. This method greatly improves the feasibility to handle massive cloud tenant network configurations for the cloud platform. However, if the overall defense goals of the cloud platform are altered, then the DRL agent should be re-trained with an updated reward function. The training of another DRL agent will introduce extra model-training time, but we do not discuss it here as the proposed framework is under a fixed defense goal.

The comparison on scalability is presented in Table 4. Our work focuses on deceiving common vulnerabilities for the OS. The results show that it performs well for large sizes of deception resources and cloud networks. However, we have a narrower coverage of vulnerabilities comparing with [42, 43]. The

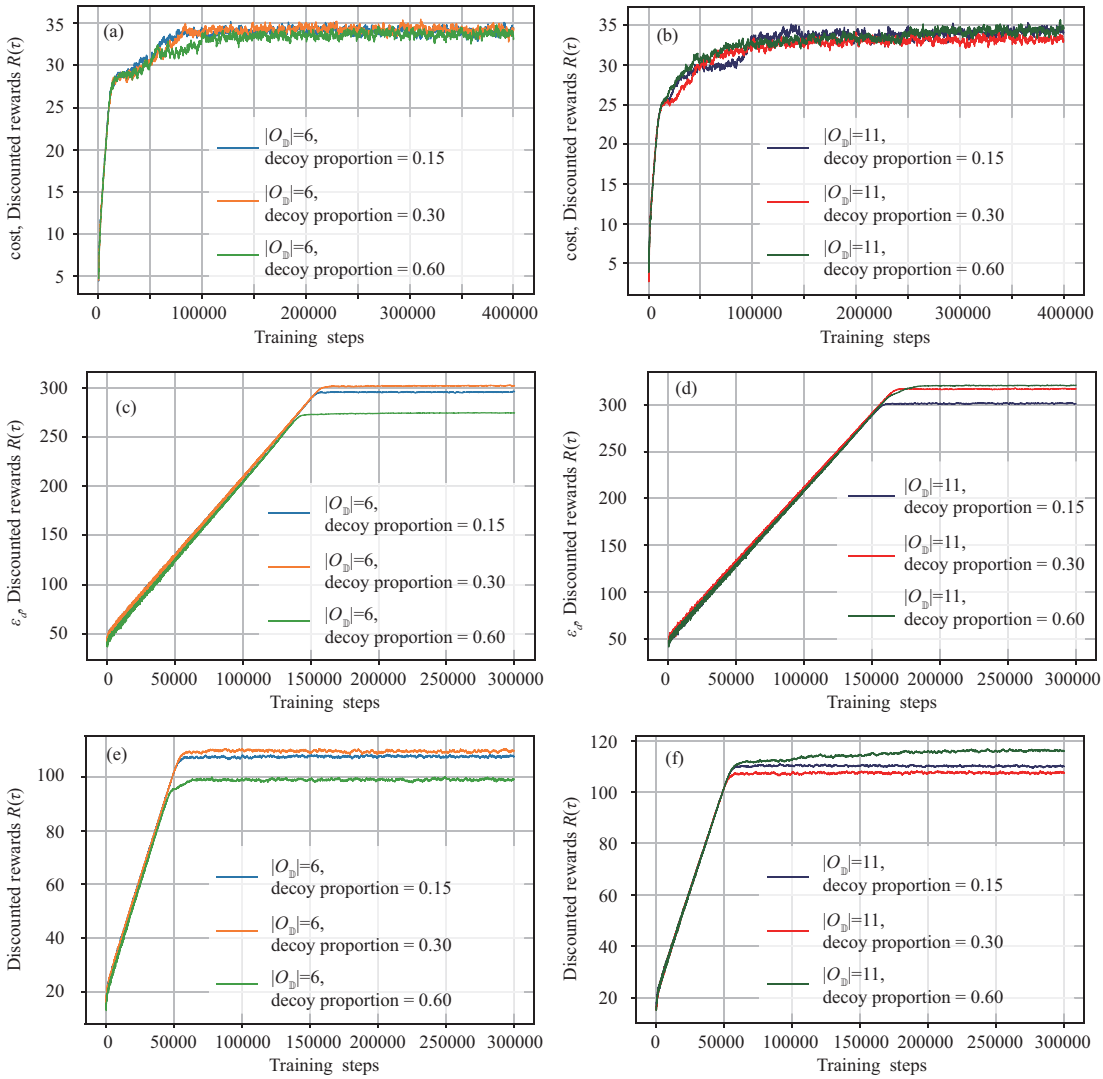

Figure 4 (Color online) Training performance of the DRL agent. (a) and (b) plotting the training progress for DRL agent to increase the attack cost $C_{\mathbb{A}}$; (c) and (d) training defense goal as $\mathcal{E}_d$; (e) and (f) plotting the performance of the DRL agent under the joint-defense goal.

Table 4 The scalability of the proposed strategy and existing studies

The scalability entity	Coverage of vulnerabilities	Deception instances	Number of hosts
MTD strategy [43]	Software vulnerabilities	Not addressed	254
Cyber deception game [42]	Software vulnerabilities	2–10	20
CONCEAL strategy [15]	Not addressed	6	254
Game theory strategy [19]	OS common vulnerabilities	6	100
This work	OS common vulnerabilities	6–11	254

main considerations are the significance of the OS for the cloud infrastructure, and possibly destructive damages for simultaneous exploits on the common vulnerabilities of OSs among working nodes. Even though covering a wide range of vulnerabilities will have a different context with deceiving OS-associated common vulnerabilities, protecting multiple service instances running in the cloud network from reconnaissance is equally urgent. Therefore, we would like to conduct a further study on the strategy for multidimensional decoy configurations in the future.

Deception entropy $\mathcal{E}_d$. This metric evaluates the concealment of system assets and the degree of obfuscation introduced by decoys. We compared the performance of our work and [41] on $\mathcal{E}_d$. Figures 5(a) and (b) illustrate the performance of $\mathcal{E}_d$ under different defense resource $|O_{\mathbb{D}}|$. First, the results exhibit that the DRL-based strategy proposed by this framework has an advantageous and stable performance in

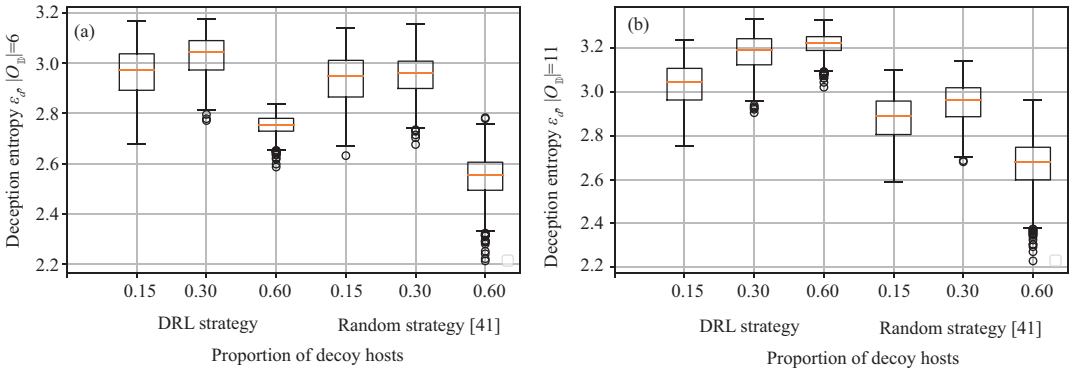

Figure 5 (Color online) Results of the deception entropy $\mathcal{E}_d$. (a) and (b) compare each scheme's performance on $\mathcal{E}_d$ under different sizes of defense resource $|O_{\mathbb{D}}|$.

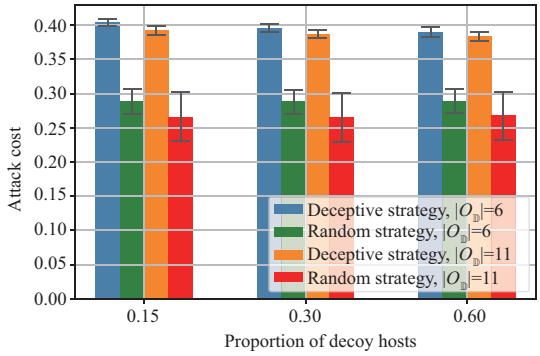

Figure 6 (Color online) The results on attack cost $C_{\mathbb{A}}$.

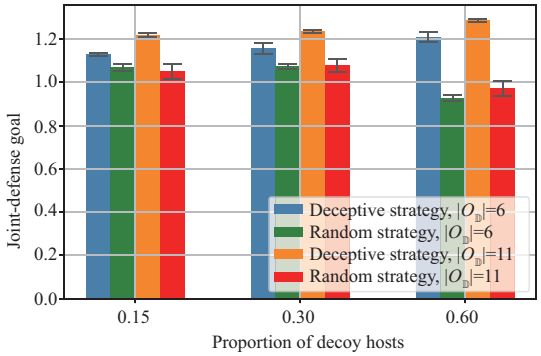

Figure 7 (Color online) The results of joint-defense goal.

$\mathcal{E}_d$. This finding indicates its capability in concealing system assets to obfuscate malicious reconnaissance. In detail, the DRL strategy leads an advantage from 0.89% to 8.02% when $|O_{\mathbb{D}}| = 6$ and 5.36% to 20.58% when $|O_{\mathbb{D}}| = 11$. Furthermore, $\mathcal{E}_d$ generally climbs (1) the proportion of decoy hosts and (2) the diversity of decoy OS, $|O_{\mathbb{D}}|$. These results imply that the diversity of falsified vulnerabilities is significant in concealing real system information instead of randomly presenting many deceptive vulnerabilities.

Moreover, when the decoy hosts dominate an unused address space, i.e., proportion = 0.6, the diversity of defense resources and strategic placement becomes essential. First, Figure 5(a) shows that $\mathcal{E}_d$ drops when proportion = 0.6, $|O_{\mathbb{D}}| = 6$. This is because the diversity of the defense resource is less than that of real system assets, $|O_{\mathbb{D}}| = 11$, which consequently provides less optimal concealment for system information. Contrarily, in Figure 5(b) when proportion = 0.6, $|O_{\mathbb{D}}| = |O_{\mathbb{N}}| = 11$, the DRL strategy has an obvious advantage and stability on improving $\mathcal{E}_d$. Second, the performance of both strategies when proportion = 0.6, $|O_{\mathbb{D}}| = |O_{\mathbb{N}}| = 11$ particularly highlights the significance of the precise deception strategy for the threat scenario. We deduce that the optimal performance of the DRL-based deception strategy is credited to the following. (1) The DRL agent's interaction with target tenant network allows the DRL agent to better understand certain threat scenarios and search for optimal configurations; (2) the approximation of the DNN guarantees the efficiency in handling the high-dimensional input data and the granular optimal deception decisions.

Normalized attack cost $C_{\mathbb{A}}$. It evaluates the additional cost on attack resources by presenting decoys in the tenant network. The simulation results of $C_{\mathbb{A}}$ are shown in Figure 6. Each bar presents the mean and variance value of the normalized $C_{\mathbb{A}}$. Our work increased attack cost from 38.33% to 40.40% as compared to the strategy proposed by [41], i.e., from 26.51% to 28.84%. Evidently, this work is effective in increasing the attack cost for both sizes of $|O_{\mathbb{D}}|$.

First, we analyzed that the trained DRL agent has a better understanding of tenant network configuration after multiple rounds of interactions. Through this process, the DRL agent generated a suitable configuration for decoys to increase the attack cost. Second, for the proportion of decoy hosts, as $C_{\mathbb{A}}$ was normalized by the maximum attack cost in each tenant network, the exact value of the attack cost grew with the increasing amount of decoy hosts. This result proves that the number of decoy hosts is

correlative to $C_{\mathbb{A}}$. Lastly, for the decoy OS diversity, the amount of exploitable vulnerabilities is equally important with the diversity to increase the attack cost. As in the simulation, when $|O_{\mathbb{D}}| = 6$, the average $C_{\mathbb{A}}$ is no less than $|O_{\mathbb{D}}| = 11$. This is because the six OSs have the most vulnerabilities to falsify attackers. Hence, when configuring OSs for decoy hosts, the number of vulnerabilities of a single OS and the diversity of available OSs should be considered.

The joint-defense goal. The goal describes a comprehensive defense consideration other than the single-defense goal. It evaluates the two dimensions of defensive deception: (1) the proactive concealment of system assets and obfuscation of reconnaissance results and (2) presenting decoys to deceive the attacker and waste attack resources. The joint-defense goal value is defined in Subsection 6.1. The attack cost $C_{\mathbb{A}}$ was derived from (3). Our proposal has a 7.87% average higher joint-defense goal value than the compared work.

Figure 7 shows that the DRL strategy outweighs to fulfill the joint-defense goal. Particularly, when the data space expands, the proportion of decoy hosts and $|O_{\mathbb{D}}|$ also grows. This is because the DRL agent is, in particular, proficient at processing complicated input data and generating optimal decisions. In addition, the performance varies according to variants. With a higher proportion and a larger $|O_{\mathbb{D}}|$, defensive deception is more effective. Moreover, the advantage of the DRL strategy is becoming evident when placing more decoy hosts with more diverse defense resources (i.e., proportion = 0.6 and $|O_{\mathbb{D}}| = 11$).

6.3 Complexity analysis

Here, we present a comparative analysis of the complexity of deception strategy generation by the attack graph-based model, game theoretic-based model, and the proposed work.

First, we analyze the potential complexity for the attack graph generation for the cloud tenant networks. A fundamental phrase to build an attack graph is the construction of attack paths [14]. This phrase could be divided into determination and pruning. The latter step is to avoid underlying state explosions in building an attack graph, where massive paths might exceed computational capability. The primary determination has certain impacts on the complexity of attack path pruning. However, the scalability of an attack graph still remains a challenge for networks with increasing size, such as the cloud tenant networks. The challenge is significant because the size not only refers to the number of network hosts within the target environment but also the amount of installed software and associated vulnerabilities. For the cloud tenant network, active working nodes may stay static in the configuration but will maintain a relative scalability in the amount. However, service replicas will be highly dynamic in types and numbers, based on either user requests or resource constraints. Consequently, the generation of an attack graph will be limited, and the provision of near real-time security analysis will be less efficient in the cloud. This will be very probable to degrade the efficiency of the following steps for implementing deception in the cloud (i.e., solving the defense strategy).

Second, we reason the limitations of existing solutions to the deception strategy with game theory. In the attack graph, calculating the centrality degree or associated shortest paths could formulate a placement, nailing the network location for decoys. Despite the scalability issue, this solution might fail to extract the adversarial interaction between the adversary and defender. This situation is very likely to hinder a granular configuration for digital decoys to respond deceptively. Therefore, researchers have employed game theory to solve deception strategy. Ever since John Nash's publication proving that all games have a mixed Nash equilibrium, researchers have sought solutions for computing mixed Nash equilibria. Condon [13] proves that the complexity of a two-player stochastic game is in the class NP$\cap$co-NP. Daskalakis et al. [44] concluded that linear programming can formulate a mixed Nash equilibrium for a zero-sum game, whereas complexity for general-sum games could be worst to exponential time. For simplification, most of game theoretic models were formulated as a zero-sum game, but a gap to fully represent threat scenarios also remains. Their study proposed a polynomial parity argument for directed graphs (PPAD) to define NP and proved that finding ϵ-Nash is a PPAD-complete problem, if ϵ is inversely proportional to an exponential function of the game size. Lipton et al. [45] formulated the running time for a game with a few players (e.g., attacker and defender) as $O(n^{\log n/\epsilon^2})$. However, for the cloud tenant networks, multiple hosts, digital decoys, and their dynamic configurations formulate a particular large policy space. This condition essentially increases the game size. Particularly, when the defender aims to obtain an intelligent deceptive response with a dynamic game, mapping the high-dimensional state space to the optimal deception strategy incurs a cost. For example, Horák et al. [32] placed honeypots with a partially observable stochastic games-based deception strategy. However, this method is limited in terms

of the scalability of configuration specifications for each node in the target system. When the number of honeypots configured for each node exceeds 4 to 6, the calculation is prohibitively expensive. Generating a deception strategy in this way will be very unlikely to adapt to the cloud tenant network because the size of working nodes is scalable, and the orchestration period is restricted to deploying the deception strategy.

The proposal of using the DNN as a function approximator for the RL is promising to map high-dimensional input data to optimal actions. DRL provides a horizon to solve the decision problem with a large space. Although the calculation of neural networks will be no less complicated, DRL takes its advantage to release from the two perspectives. (1) The well-trained DRL model could be directly implemented on the cloud. Existing cloud platforms, such as AWS, provide cloud users with well-trained or customized ML models to cope with production issues. Likewise, well-trained DRL models could be installed to solve security issues. Once training is finished, the packed model could generate an optimal strategy within seconds, which is promising to meet with the efficiency requirements for the cloud. (2) The computing tasks of neural networks could be offloaded to the GPU. Compared with the heuristic algorithm working on the CPU, collaboration computing is essential to accelerate the training and reduce the time complexity. Moreover, the time efficiency of the deception strategy will be more imperative to the cloud tenant network other than the used platform or equipment. Thus, it is reasonable for the defender to accelerate with the GPU.

6.4 Discussion

To summarize the simulation results, we would like to analyze the remaining challenges of the proposed framework, and discuss possible future directions. We focus on expanding the coverage of vulnerabilities and the generalization of the DRL agent.

Primarily, for the joint consideration of vulnerabilities in applications and services, the proposed framework mainly concentrates on the OS configuration for decoy hosts, while not presenting the deceptive configuration for services or applications. Because there are multiple service instances running in the cloud network, preventing attackers from reconnaissance on services and applications is equally urgent. It is indeed necessary to have a further study on integrating the deception of OS and service configurations.

Furthermore, the proposed DRL agent is limited in generalization when confronted with changing settings of the cloud (e.g., unknown environment parameters or undisclosed vulnerabilities). To generalize the defensive deception framework, we intend to investigate in two directions: automatic RL and division on a complicated problem. On the one hand, automatic RL is an emerging area because existing studies discover the potential of RL in solving complicated problems merely in specific areas and are limited to particular design choices [46]. This limits the full potential of applying RL in a wider range. Because automatic ML has exhibited its potential to fit in changing design settings, it is very likely to yield such an advantage with DRL. Therefore, a further investigation on automatic RL will be promising to address the generalization of the DRL agent, particularly addressing different security issues by the changing of the tenant network. On the other hand, we consider dividing a complicated issue into sub-scenarios as another way to enhance the generalization. For example, the changes in the target environment may have different impacts on the cloud tenant network. In this way, we could study the collaboration of multiple agents into a defense framework. Each agent dynamically optimizes a perspective of the complicated issue to reach a joint goal, which eventually enables the defense framework to fit into more general security scenarios for the cloud.

7 Conclusion

Defensive deception is an emerging proactive mechanism to reverse the asymmetry between the attacker and defender. We propose a defensive deception framework to counter reconnaissance attacks in the cloud. A fine-grained deception strategy was mainly studied to adapt to a multi-tenant cloud and fulfill a comprehensive defense goal.

First, we formulated the utility function $\mathcal{U}$ for the cloud to model the possible impacts of OS-associated common vulnerabilities. This could generalize to multiple tenant networks and lay a foundation for developing adaptive deception strategies. Second, to measure the achievement of the comprehensive defense goal, we set a joint-defense goal. It evaluates the obfuscation of reconnaissance through the concealment degree of system assets and quantifies the increase in the attack cost. Third, we customized

a DRL agent to compute the optimal defense strategy. The fine-grained deception strategy points out the detailed type and amount of OS for decoy hosts.

We conducted simulations to evaluate our proposal on 12000 simulated Class-c cloud tenant networks. The results show that our work has a 7.87% mean improvement in realizing the joint-defense goal. For the concealment of system assets, the defensive deception framework leads an advantage from 0.89% to 8.02% when $|O_{\mathbb{D}}| = 6$ and 5.36% to 20.58% when $|O_{\mathbb{D}}| = 11$. The framework increased the attack cost from 38.33% to 40.40% on average with minor variance. In addition, the framework was simulated on different proportions of decoys and dissimilar diversity on the defense resource $|O_{\mathbb{D}}|$. The results show the following. (1) The concealment of system assets largely depends on the diversity of defense resources. When decoys are deployed in a large number, a strategic configuration is especially essential to ensure defense effectiveness. (2) The increase in attack cost has a positive correlation with both the proportion of decoys and the diversity of defense resources. The simulation shows the effectiveness and stability of our framework to counter reconnaissance attacks in the cloud. Moreover, this framework provides a new insight to establish the deception strategy in the cloud. It also reflects the potential of solving cybersecurity problems with emerging AI techniques.

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

References

1 Hutchins E M, Cloppert M J, Amin R M, et al. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Leading Issues Inform Warfare Secur Res, 2011, 1: 80
2 Compastié M, Badonnel R, Festor O, et al. From virtualization security issues to cloud protection opportunities: An in-depth analysis of system virtualization models. Comput Secur, 2020, 97: 101905
3 Garcia M, Bessani A, Gashi I, et al. OS diversity for intrusion tolerance: myth or reality? In: Proceedings of 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks (DSN), 2011. 383–394
4 Achleitner S, La Porta T F, McDaniel P, et al. Deceiving network reconnaissance using SDN-based virtual topologies. IEEE Trans Netw Serv Manage, 2017, 14: 1098–1112
5 Carasik-Henmi A, Shinder T W, Amon C, et al. Chapter 4—Introduction to intrusion detection systems. In: The Best Damn Firewall Book Period. Burlington: Syngress, 2003. 111–124
6 Virvilis N, Vanautgaerden B, Serrano O S. Changing the game: the art of deceiving sophisticated attackers. In: Proceedings of the 6th International Conference On Cyber Conflict (CyCon 2014), 2014. 87–97
7 Fraunholz D, Anton S D, Lipps C, et al. Demystifying deception technology: a survey. 2018. ArXiv:1804.06196
8 Han X, Kheir N, Balzarotti D. Deception techniques in computer security. ACM Comput Surv, 2018, 51: 1–36
9 Lu Z, Wang C, Zhao S, et al. Cyber deception for computer and network security: survey and challenges. 2020. ArXiv:2007.14497
10 Durkota K, Lisy V, Bosansky B, et al. Optimal network security hardening using attack graph games. In: Proceedings of the 24th International Conference on Artificial Intelligence, 2015. 526–532
11 Fraunholz D, Schotten H D. Defending web servers with feints, distraction and obfuscation. In: Proceedings of 2018 International Conference on Computing, Networking and Communications (ICNC), 2018. 21–25
12 Pawlick J, Colbert E, Zhu Q. A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy. ACM Comput Surv, 2019, 52: 1–28
13 Condon A. The complexity of stochastic games. Inf Comput, 1992, 96: 203–224
14 Kaynar K. A taxonomy for attack graph generation and usage in network security. J Inf Secur Appl, 2016, 29: 27–56
15 Duan Q, Al-Shaer E, Islam M, et al. CONCEAL: a strategy composition for resilient cyber deception-framework, metrics and deployment. In: Proceedings of IEEE Conference on Communications and Network Security (CNS), 2018. 1–9
16 Qu Y Y, Zhang J W, Li R D, et al. Generative adversarial networks enhanced location privacy in 5G networks. Sci China Inf Sci, 2020, 63: 220303
17 Ning Z L, Sun S M, Wang X J, et al. Intelligent resource allocation in mobile blockchain for privacy and security transactions: a deep reinforcement learning based approach. Sci China Inf Sci, 2021, 64: 162303
18 Roy S, Sharmin N, Acosta J C, et al. Survey and taxonomy of adversarial reconnaissance techniques. 2021. ArXiv:2105.04749
19 Wang Y, Guo Y, Guo Z, et al. CLOSURE: a cloud scientific workflow scheduling algorithm based on attack-defense game model. Future Generation Comput Syst, 2020, 111: 460–474
20 Kandoussi E M, Hanini M, El Mir I, et al. Toward an integrated dynamic defense system for strategic detecting attacks in cloud networks using stochastic game. Telecommun Syst, 2020, 73: 397–417
21 Zhan J, Fan X, Han J, et al. CIADL: cloud insider attack detector and locator on multi-tenant network isolation: an OpenStack case study. J Ambient Intell Hum Comput, 2020, 11: 3473–3495
22 Cohen F. A note on the role of deception in information protection. Comput Secur, 1998, 17: 483–506
23 Spitzner L. The Honeynet Project: trapping the hackers. IEEE Secur Privacy, 2003, 1: 15–23
24 Petrunic A R. Honeytokens as active defense. In: Proceedings of the 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015. 1313–1317
25 Araujo F, Hamlen K W, Biedermann S, et al. From patches to honey-patches. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2014. 942–953
26 Shu Z, Yan G. Ensuring deception consistency for FTP services hardened against advanced persistent threats. In: Proceedings of the 5th ACM Workshop on Moving Target Defense. New York: ACM, 2018. 69–79
27 Rrushi J L. NIC displays to thwart malware attacks mounted from within the OS. Comput Secur, 2016, 61: 59–71
28 Kyriakou A, Sklavos N. Container-based honeypot deployment for the analysis of malicious activity. In: Proceedings of Global Information Infrastructure and Networking Symposium, 2019
29 Rowe N C, Rrushi J. Introduction to Cyberdeception. Cham: Springer, 2016

30 Zhu Q. Game theory for cyber deception: a tutorial. In: Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security, 2019
31 Almohri H M J, Watson L T, Evans D. Misery digraphs: delaying intrusion attacks in obscure clouds. IEEE Trans Inform Forensic Secur, 2018, 13: 1361–1375
32 Horák K, Bošanský B, Tomášek P, et al. Optimizing honeypot strategies against dynamic lateral movement using partially observable stochastic games. Comput Secur, 2019, 87: 101579
33 Crouse M, Prosser B, Fulp E W. Probabilistic performance analysis of moving target and deception reconnaissance defenses. In: Proceedings of the 2nd ACM Workshop on Moving Target Defense. New York: ACM, 2015. 21–29
34 Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature, 2015, 518: 529–533
35 Sethi K, Kumar R, Prajapati N, et al. Deep reinforcement learning based intrusion detection system for cloud infrastructure. In: Proceedings of International Conference on Communication Systems & NETworkS (COMSNETS), 2020. 1–6
36 Provos N. Honeyd—a virtual honeypot daemon. In: Proceedings of the 10th DFN-CERT Workshop, Hamburg, 2003. 4
37 Cabral W, Valli C, Sikos L, et al. Review and analysis of cowrie artefacts and their potential to be used deceptively. In: Proceedings of International Conference on Computational Science and Computational Intelligence (CSCI), 2019. 166–171
38 Zhang M, Wang L, Jajodia S, et al. Network diversity: a security metric for evaluating the resilience of networks against zero-day attacks. IEEE Trans Inform Forensic Secur, 2016, 11: 1071–1086
39 Guo M, Bhattacharya P. Diverse virtual replicas for improving intrusion tolerance in cloud. In: Proceedings of the 9th Annual Cyber and Information Security Research Conference. New York: ACM Press, 2014. 41–44
40 Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. 2017. ArXiv:1707.06347
41 Gutierrez M, Kiekintveld C. Online learning methods for controlling dynamic cyber deception strategies. In: Adaptive Autonomous Secure Cyber Systems. Cham: Springer, 2020. 231–251
42 Schlenker A, Thakoor O, Xu H, et al. Deceiving cyber adversaries: a game theoretic approach. In: Proceedings of International Conference on Autonomous Agents and Multiagent Systems, 2018
43 Sengupta S, Chowdhary A, Huang D, et al. Moving target defense for the placement of intrusion detection systems in the cloud. In: Proceedings of International Conference on Decision and Game Theory for Security, 2018. 326–345
44 Daskalakis C, Goldberg P W, Papadimitriou C H. The complexity of computing a Nash equilibrium. Commun ACM, 2009, 52: 89–97
45 Lipton R J, Markakis E, Mehta A. Playing large games using simple strategies. In: Proceedings of the 4th ACM Conference on Electronic Commerce, 2003. 36–41
46 Chen J Y, Zhang Y, Wang X, et al. A survey of attack, defense and related security analysis for deep reinforcement learning. Acta Autom Sin, 2022, 48: 1–19