ISSF: An Intelligent Security Service Framework for Cloud-native Operations

Yikuan Yan[1], Keman Huang[1,2,*] and Michael Siegel[2]

[1]*School of Information, Renmin University of China, Beijing, China*
[2]*Cybersecurity at MIT Sloan, MIT, Cambridge, Massachusetts, USA*
E-mail: keman@ruc.edu.cn
[*]*Corresponding Author*

Received 29 April 2024; Accepted 14 April 2025

Abstract

The growing system complexity of microservice architectures and the bilateral enhancement of artificial intelligence (AI) for both attackers and defenders present increasing security challenges for cloud-native operations. In particular, cloud-native operators require a holistic view of the dynamic security posture for the microservice-based cloud-native environment from a defense aspect. Additionally, both attackers and defenders can adopt advanced AI technologies. This makes the dynamic interaction and benchmark among different intelligent offense and defense strategies more crucial. Hence, following the multi-agent deep reinforcement learning (RL) paradigm, this research develops an agent-based intelligent security service framework (ISSF) for cloud-native operations. It includes a dynamic attack graph model to represent the cloud-native environment and an action model to represent offense and defense actions. Then we develop an approach to enable the training, publishing, and evaluating of intelligent security services using diverse deep RL algorithms and training strategies, facilitating their

Journal of Web Engineering, Vol. 24_4, 655–686.
doi: 10.13052/jwe1540-9589.2447

systematic development and benchmarking. The experiments demonstrate that our framework can sufficiently model the security posture of a cloud-native system for defenders, effectively develop and quantitatively benchmark different intelligent security services for both attackers and defenders, and guide further optimization.

Keywords: Cloud-native, dynamic attack graph, intelligent security service model, security service training, publishing and evaluating.

1 Introduction

The cloud-native and microservice approach has been increasingly adopted for cloud service design, development, and deployment [9, 13]. As a modern software development and deployment methodology, it designs applications as loosely coupled microservices interacting through API endpoints and uses container technology to provide lightweight runtime environments. While this approach significantly simplifies the updating, scheduling, and scaling of cloud-native systems, the security threat has risen.

First, microservice-based unbundling and container-based lightweight virtualization technology cause an increasingly difficult-to-control attack surface for defense [50]. Cloud-native defenders can easily get lost in complex configurations and interactions without a precise understanding of the cyber threat situations. A recent direction to bridge this gap is to model the cloud-native system from an attack graph aspect to optimize cyber defense strategies in specific security scenarios like man-in-the-middle attack or container escape [18, 54]. However, it is developed from an attacker aspect rather than from a defense operation aspect. This motivates us to develop a model for cloud operators to understand the dynamic surface of the cloud-native environment.

Second, increasingly advanced AI has significantly powered not only the cyber defenders but also attackers, demonstrated by increasing autonomous intelligent cyber defense and offense [45] services. Adapting typical cloud-native techniques like moving target defense(MTD) in autonomous cyber operations(ACO) can be an effective defense strategy [18, 38]. While some recent studies optimize the defense strategies for cloud-native defenders, these strategies are designed with a specific deployment setting, making it challenging to compare different strategies in a systematic way [26]. More importantly, cyber attackers can also adopt similar, if not more advanced, AI techniques. For example, in a cloud-native environment, attackers can

set up a cloud-native environment to pre-train an intelligent offense service [16, 36] and then use it to guide the cyber offense. While existing studies take the perspective from either the attacker [26] or the defender [18], it is essential to consider them as a whole, especially by focusing on their dynamic interactions and comparisons in a systematic and scalable way, to support an effective cloud-native security operation.

Hence, this research proposes an intelligent security service framework, named *ISSF*, to investigate intelligent security services for both attackers and defenders within the cloud-native environment. First, ISSF establishes a dynamic attack graph model from a defense perspective to represent the security surface for a cloud-native environment. Each attacker or defender is defined as an agent-based intelligent security service that can undertake offense or defense actions, including three offense actions (*local attack, remote attack and connect*) and four defense actions (*scan, restore, remediate and decoy*). These can be further combined to form complex tactics. Second, ISSF provides a flexible and extensible approach which can *train* offense or defense intelligent security services using diverse deep reinforcement learning algorithms from scratch or *finetune* from a pre-trained security service, and then *publish* it to the *security service pool*. We further develop an ELO rating-based approach to quantitatively *evaluate* the strength of different offense or defense intelligent security services. In such a way, we can develop intelligent security services and compare their performance in a systematic, quantitative, and scalable way. Finally, we use the CAGE Challenge 2 cloud-native case as our study context, given that it is developed by the Technical Cooperation Program (TTCP) to support the development of AI tactics, techniques and procedures for cyber defence. In particular, it can reflect real cloud-native environments and support the practical demonstrations of ACO, which meets our goals perfectly. The experiments demonstrate the effectiveness of our framework. Additionally, the preliminary results reveal that training a security service using more advanced and diverse adversaries could achieve a better performance, suggesting a promising direction for security service optimization. Overall, our framework contributes:

- An agent-based intelligent security service model that includes a dynamic attack graph model to represent security situations and an action model to represent the defense and offense actions.
- A flexible and extensible approach that can train, publish, and evaluate intelligent security services in a systematic, quantitative, and scalable way.

- A case designed to verify the effectiveness of our framework and provide empirical evidence to guide further security service optimization.

The rest of this paper is structured as follows. Section 2 discusses the related work to position this research. Section 3 describes the agent-based intelligent security service model. We detail our approach for training, publishing, and evaluating security services in Section 4. Section 5 describes the experiment case and the results are reported in Section 6. Section 7 concludes this paper.

2 Related Work

2.1 Security in a Cloud-native Service

Cloud-native computing leverages microservices [3], containerization [7], and automated orchestration [44] to enable elastic scalability [10], rapid DevOps/CI/CD workflows [53], and cross-environment adaptability [25]. Cloud-native services exhibit fundamental differences in security architecture and adversarial dynamics compared to traditional monolithic systems. Traditional monolithic architectures adopt centralized deployment models, prioritizing north–south traffic boundary protections (e.g., firewalls, WAFs) [2,21], with relatively static attack surfaces dominated by application-layer threats like SQL injection and XSS [22, 39]. Their implicit trust in internal components further limits lateral movement risks. In contrast, cloud-native systems leverage distributed microservices and containerization technologies, where dynamic scalability and pervasive east–west communication significantly expand attack surfaces [49]. Adversaries exploit vulnerabilities such as container escape (e.g., Linux kernel privilege escalation) [28], API abuse (e.g., unauthorized service-to-service calls) [29], and novel DDoS vectors like Yo-Yo attacks exploiting auto-scaling mechanisms [19]. Additionally, service mesh complexity introduces misconfiguration risks (e.g., incomplete Istio policy coverage) [23].

Defensively, traditional systems rely on static rule-based controls, whereas cloud-native environments necessitate zero-trust architecture (ZTA) [20] for granular access control, service mesh implementations (e.g., Envoy-based encrypted communication) [23] to secure east–west traffic, and zero-touch service management (ZSM) [41] for real-time vulnerability remediation. This reflects a paradigm shift from "perimeter-centric defense" to "continuous verification" and from "manual response" to "intelligent closed-loop automation" [35]. Empirical studies reveal that 76% of cloud-native

security incidents stem from misconfigurations and overprivileged component trust [29], underscoring their heightened dynamic adversarial complexity and technology-dependent defense mechanisms compared to monolithic systems. To counter these evolving threats, the research community has increasingly turned to artificial intelligence techniques. This shift manifests in two key approaches: (1) AI-enhanced security modeling to better represent cloud-native environments' dynamic attack surfaces (Section 2.2), and (2) intelligent autonomous systems that leverage machine learning for real-time threat response (Section 2.3).

2.2 Cloud-native Security Modeling

Establishing security models for cloud-native environments is crucial for structured analysis of security posture [12,18,26,54], leveraging state-of-the-art machine learning techniques to assist organizations in promptly detecting and mitigating security threats [24,42,48,51].

The research landscape reveals several predominant modeling paradigms that represent the current state of the art. The first paradigm integrates operational monitoring. For example, Torkura et al. propose a RDFI framework for chaos engineering integration [43], while Theodoropoulos et al. combine security-as-a-service (SECaaS) models with edge/cloud resources to decentralize threat intelligence and compliance monitoring [42]. These demonstrate promising results in real-time threat detection but lack comprehensive defensive action modeling. Another paradigm emphasizes architectural modeling. Zdun et al. [54] established microservice-specific design decision models, incorporating security metrics for architectural trade-off analysis. This was complemented by Ma et al.'s mutation-enabled defense strategies [26] that model man-in-the-middle attack scenarios. While valuable, these architectural models typically operate at design-time and struggle with runtime dynamics.

A more comprehensive approach involves extending classical attack graphs to cloud environments. Jin et al. develop a holistic attack graph, which depicts attack scenarios in container-based cloud environments [18]. Engström et al. present a domain-specific modeling language to construct and traverse attack graphs to assess security in AWS environments [12]. Ibrahim et al. propose utilizing attack graphs in the continuous delivery infrastructure of microservicesbased systems [17]. While Zambianco et al. design a metric to evaluate the decoy effectiveness in luring attacks according to the attack graph structure which models the admissible lateral movements of an attacker

between microservices [51]. Further, they modeling cloud-native security using attack graphs as directed acyclic graphs where vertices represent microservices/decoys and edges represent attacker lateral movement paths weighted by vulnerability exploitability metrics [52]. Also, machine learning techniques like reinforcement learning has been proven to be effective with the attack graph modeling of cloud-native systems [24, 48].

While existing attack graph approaches have advanced cloud-native security modeling, they exhibit three fundamental limitations that hinder comprehensive defense support. First, current models operate primarily at the infrastructure dependency level (e.g., [51]'s service-to-service edges or [18]'s container escape paths), failing to capture higher-order behavioral patterns in microservice interactions. There is also a lack of effective models to support defenders in understanding the security situation as these approaches primarily adopt an attacker's perspective, evidenced by Zambianco's decoy effectiveness metrics [51] and Ibrahim's continuous delivery attack graphs [17].

2.3 Intelligent Cybersecurity Operations and Optimizations

Growing research on machine learning (ML) has integrated security mechanisms into the cloud-native environment. ML has demonstrated significant potential in enhancing cloud-native environments, particularly in anomaly detection by identifying deviations from normal patterns, such as network intrusions or data breaches [8, 32]. Emerging technologies like blockchain combined with ML further improve data integrity and access control reliability [14].

Security automation is another critical area. Several research build their cloud-native threat scenario independently to apply machine learning methods to solve security questions. Wang et al. proposes a cloud boundary network active defense model and decision method based on the reinforcement learning of an intelligent agent [47]. Hu et al. propose a precise greybox penetration testing approach called TAC for third-party services to detect IAM PEs, which apply reinforcement learning with graph neural networks (GNNs) to improve the efficiency by minimizing the interactions with customers [15]. Arulappan et al. applies deep reinforcement learning (DQN/DDQN) to achieve adaptive self-healing of virtual network functions (VNFs) in cloud-native networks to automatically repair failures and optimize quality of service (QoS), thereby enhancing network reliability and reducing operational costs [6].

In recent years, autonomous cyber operations (ACOs) have become a promising field for intelligent cybersecurity operation and optimizations, where deep reinforcement learning has been a effective approach [34] to develop analyse and decision-making processes that can be autonomously optimized and performed to safeguard computer systems and network environments [45]. This term is always described as an adversarial game, i.e., a game between a defender agent (blue team) and an attacker agent (red team) [38]. A direct driver for this field is the proliferation of DRL-based ACO Gyms, which serves as cyber system environments that enable the deployment of autonomous red and blue team agents have been developed, including CyberBattleSim [40], YAWNING-TITAN [4] and CybORG [38]. Research efforts further extended their capabilities through component and functional extensions [46] and algorithmic development [5].

However, the predominant single-agent optimization paradigm, exemplified by Arulappan et al.'s self-healing VNFs [6] and Ma et al.'s MitM attack strategies [26], artificially decouples the co-evolutionary dynamics between attackers and defenders. This fragmentation manifests in evaluation metrics that assess either attack success rates [18] or defense efficacy [47] in isolation, ignoring the emergent properties of their continuous adaptation. Without considering attackers and defenders as a whole, it is challenging to evaluate their true effectiveness and further optimize the attacker or defender strategies.

3 The Agent-based Intelligent Security Service (ISS) Model

We first apply the multi-agent reinforcement learning (MARL) paradigm to model cloud-native scenarios (Figure 1). The model consists of (1) a dynamic attack graph as the *environment*, representing the dynamic surface of a cloud-native system, and (2) an action model as the *agents*, representing the security operation for attackers and defenders.

3.1 Dynamic Attack Graph Model for Cloud-native Systems

A cloud-native system is a dynamic system where service instances, the microservice application running on a container, can be easily deployed, updated, or destroyed. Logically, these service instances can access each other through API endpoints or be controlled through credentials, forming a dynamic attack surface which can be described as a graph.

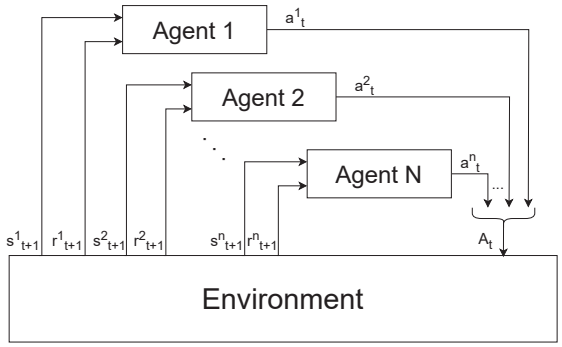


Figure 1 Multi-agent reinforcement learning.

In particular, the attack graph has emerged as a pivotal model in the domain of network security analysis [55]. Typically, these graphs delineate malicious events as nodes interconnected by causal relationships, thereby facilitating the analysis of attacker strategies and pathways. Hence, inspired by this concept, we define a cloud-native environment as a dynamic attack graph as follows.

Definition 1. A cloud-native environment is a directed graph $DG_e = (N, E)$, where N is a set of nodes representing the service instances in the cloud-native environment, and $E \subseteq N \times N$ is a set of edges where each edge is an access path between the ordered pair of nodes (N_i, N_j) that $N_i, N_j \in N$ and $i \neq j$, using API endpoints or credentials.

Note that a node represents a service instance, which can be a microservice, a container, or a physical machine in the cloud-native environment [18]; we have not distinguished them in this research from a security situation aspect.

Definition 2. A node is a quadruplet *N = (AV,STA,CRE,VUL)*, where *asset value* (*AV*) defines the intrinsic value of the digital asset associated with the node and *state* (*STA*) represents the access state from attacker's perspective. *Credential* (*CRE*) defines the required *connection credentials* to access the node. *Vulnerabilities* (*VUL*) can be exploited by the attackers and result in credential or topological information leaks beyond value lost.

Definition 3. An edge indicates an access path *E=(SOU,TAR,CON)*, where *SOU* represents the *source node* and *TAR* represents the *target node*. *CON* describes the *connections* through API endpoints or access credentials.

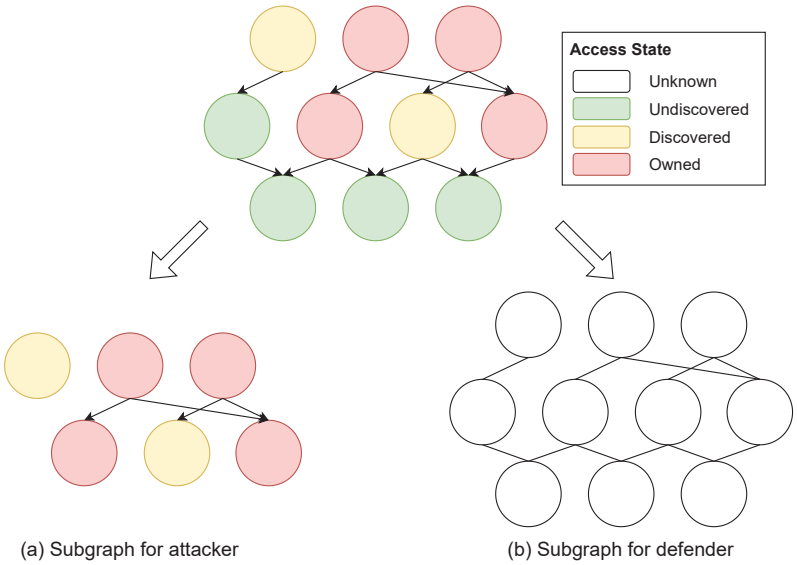


Figure 2 Observation of a dynamic attack graph from attackers and defenders. The color of the nodes represents the attackers' access state (*undiscovered* in green, *discovered* in orange and *owned* in red) to the nodes. Whether the *unknown* nodes (in white) are undiscovered, discovered or owned by the attackers is *unknown* to the defenders. The edges in the subgraph for the attacker represent the attack path while the edges in the subgraph for the defender represent the access path known by defenders.

Given a dynamic attack graph DG_e for a cloud-native system (Figure 2), attackers and defenders dynamically obtain varying information about this system, leading to the creation of distinct *enhanced subgraphs*. Consequently, these subgraphs can effectively encapsulate an attacker's or a defender's observation of the system, supplemented by the current security posture at a specific timestep t, which is dictated by their respective security actions.

Definition 4. The observation space for the attacker, $OB_{a,t}$, at time t is an *enhanced subgraph* of DG_e, $OB_{a,t} = (N_{a,t}, E_{a,t})$ where $N_{a,t} \in N, E_{a,t} \in E$ and each node is further associated with a access state, the edges represent the attack path until time t.

Definition 5. The observation space for the defender, $OB_{d,t}$, at time t is an *enhanced subgraph* of DG_e, $OB_{d,t} = (N_{d,t}, E_{d,t})$ where $N_{d,t} \in N, E_{d,t} \in E$, while each node is further associated with a security state identified by the defender, each edge represents the connections between nodes until time t.

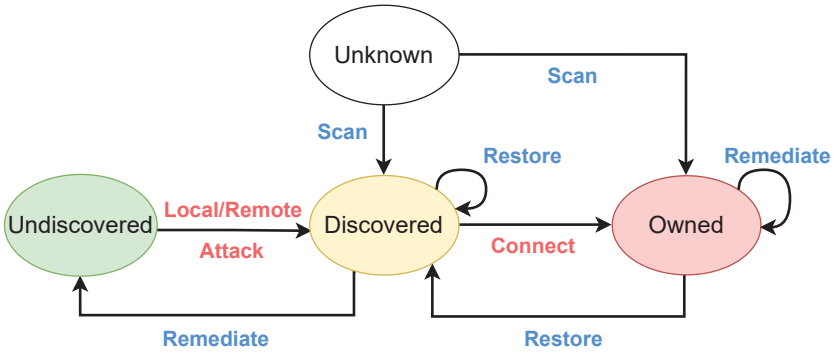


Figure 3 Security state transfer diagram.

3.2 Action Model for Attackers and Defenders

Intuitively, the security situation of a cloud-native system is shaped by the dynamic interactions between attackers and defenders, where they select actions based on their strategy and their observation of the cloud-native environment. Following the reinforcement learning paradigm, we can define the action model to represent the action and reward from it by either an attacker or defender.

Definition 6. The action model is a quadruplet $RLA_t = (OB_{a|d,t}, SS_{a|d}, ACT_{a|d,t}, REW_{a|d,t})$. At time t, the model takes observations from an attacker $OB_{a,t}$ or a defender $OB_{d,t}$ as inputs, uses the embedded security service $SS_{a|d}$ to generate an action $ACT_{a|d,t}$, and finally receives the new observation $OB_{a|d,t+1}$ and reward $REW_{a|d,t+1}$ from the environment by taking this action.

The security services are autonomous, intelligent attacker service SS_a or defender service SS_d published through our security service framework, which will be detailed in Section 4.

In alignment with the design paradigm of the prevailing Autonomous Cyber Operation Gyms (ACO Gyms) [34], we delineate the action space for both attackers and defenders from an abstract or high-level viewpoint. This approach further enhances the model's applicability, allowing us to refine it for any specific scenario.

3.2.1 Action Space for Attackers

As shown by the red words in Figure 3, the attacker's actions are based on their attack vectors, including exploiting vulnerabilities or using credentials.

In particular, vulnerabilities are divided into *local vulnerabilities* and *remote vulnerabilities* based on the access prerequisite of the node.

Definition 7.1. Local attack. The attacker selects a local vulnerability to launch an attack on a node with *owned* state.

Definition 7.2. Remote attack. The attacker selects a remote vulnerability to launch an attack on a node with *discovered* or *owned* state.

We suppose that attackers have the exploit toolkit for all the associated vulnerabilities but without the knowledge of whether a specific node is associated with a specific vulnerability. Hence, only when the selected vulnerability is associated with the selected node will the exploitation succeed. Vulnerability exploitation will lead to various outcomes, such as the leakage of API endpoint or connection credential information. The former will reveal new nodes for the attacker, and the latter will enable the attacker to adopt the credential to connect to a node, which can both enable the attacker to identify an undiscovered node and turn its security state from *undiscovered* into *discovered*.

Definition 7.3. Connect. The attacker utilizes stolen credentials to access and control a node. A successful connection using the matching credentials will compromise the node, subsequently changing its state to *owned*.

These abstracted actions, or *meta actions*, can be instantiated into an attack tactic, and complex tactics can be modeled as the combinations of these actions. For example, taking a *remote attack* action on a *discovered* node N_i to exploit a remote vulnerability to steal the credentials or create an additional account, which identifies a *new discovered* node, N_j and then using the *connect* action to access this new node N_j and increase the privilege using the malicious credential, can represent attackers' lateral movement from one node N_i to the new one N_j within a cloud-native environment.

3.2.2 Action space for defenders

From the defenders' points of view, suppose they have information regarding the API endpoints and credentials. However, defenders don't have details of associated vulnerabilities or the real-time security state for each node. As we focus on the cloud-native environment, we consider three defensive actions at the node level.

Definition 8.1. Scan. Defenders may conduct a comprehensive scan of all nodes to identify *suspicious* ones or perform a targeted scan on a specific node to verify its compromised state.

The proactive *scan* action can help defenders initially identify and mark suspicious activities in the network. However, we assume that extensive scanning cannot accurately determine the attacker's access state of the nodes without a targeted scan. Subsequently, the defender can apply two typical moving target defense techniques [18] to handle the suspicious nodes.

Definition 8.2. Restore. The defender resorts and reimages a selected node to a new one.

Definition 8.3. Remediate. The defender modifies the access information of the node, such as API configurations or credentials, making the node unavailable to attackers.

The *restore* action replaces the current instance with a new one from its image, which is fast and cost-effective in a cloud-native system. This action can transform the *owned* node to a *discovered* one, rather than *undiscovered*, since no API endpoint is changed for maintaining the availability of the system. In other words, attackers still have the related topological information to identify it.

The third action is to *remediate* the leaked information by modifying the node's configuration, such as techniques like IP shuffling [33]. This will nullify the attacker's information collected from previous vulnerability exploitation, which transforms the *discovered* node to *undiscovered*, invalidating the credential.

Definition 8.4. Decoy. The defender can strategically create a deceptive process on designated services by leveraging available and compatible options, making it appear as though a specific attack vector exists.

Decoys can obfuscate the attacker's perception, compelling them to engage in speculative attempts to identify genuine attack pathways, thereby effectively impeding their progress.

Similarly, defenders can take a combination of these actions to implement some complex defense tactics. For example, for an *owned* node, if defenders take a *restore* action, followed by a *remediate* action, which will change the way to access it through those nodes linked to it, the node will become out of the radar for the attacker. In other words, it becomes *undiscovered* by the attacker.

3.2.3 Reward

The reward design is critical for policy optimizations in reinforcement learning. The reward for an action depends on its gain and cost.

Definition 9. Reward. For attackers and defenders, $reward = gain - cost$.

In addition to defining the availability and confidentiality of each node in the network based on prior knowledge of the environment, we recommend the common vulnerability scoring system (CVSS)[1] to quantify the gain and cost of vulnerability exploitation, which has been proven to be effective in previous work [18]. For example, the *impact subscore* measures the gains and the *exploitability subscore* measures the costs.

Generally, we suppose the combat between attacker and defender is zero-sum. Hence, the gain and cost for vulnerability within the *remediate* action and the intrinsic value for the *restore* action are consistent with the attacker actions, whereas the fixed cost is different based on the scenario, especially when considering the availability of the system.

4 Security Service Training, Publishing and Evaluating

Based on the ISS model put forward in Section 3, we further develop a flexible approach to train, publish, and evaluate the security services. As shown in Figure 4, it is a three-layer framework built around a *security service pool*. First, we can easily train a new service from scratch or fine tune a pre-trained one. Then, we publish the new service to the pool for further use. Finally, it's convenient to run simulations by selecting different attack or defense services from the service pool, calculating performance metrics and forming benchmarks to validate their comparative performance.

4.1 Train a Security Service

The training phase can be divided into two steps: instantiating an ISS model, and training the service with a learning approach. The first step to model a cloud-native system includes: (1) configuring the nodes and edges to generate a dynamic attack graph, such as the intrinsic value of nodes, vulnerabilities, and the results of vulnerability exploitation, (2) building action models by determining the parameters, such as the false positive rate of the *scan* action or the downtime of the node of the *reimage* action. Examples of these processes will be provided in Section 5.

With a complete model, we can train a security service using a specific learning algorithm. As most ACO gyms like *CybORG*[2] are implemented with

[1] https://nvd.nist.gov/vuln-metrics/cvss

[2] https://github.com/cage-challenge/CybORG

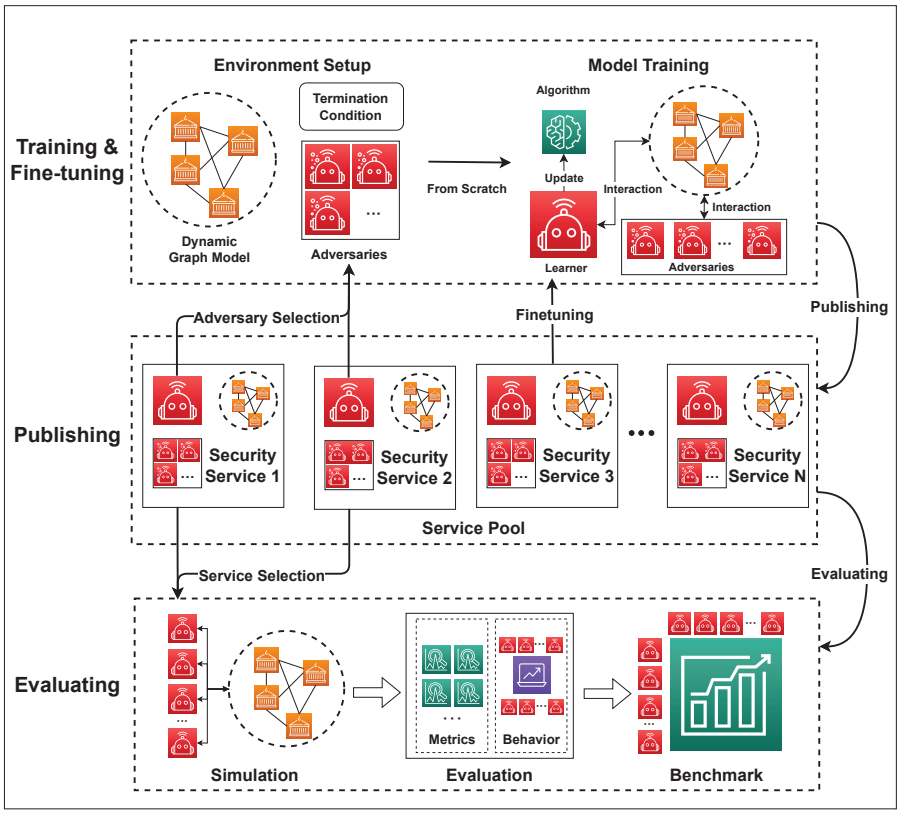


Figure 4 An approach for security service training, publishing and evaluating.

OpenAI Gym[3] are highly suitable for reinforcement learning, training a policy with RL algorithms will be a straightforward choice. More importantly, given the adversarial nature of cloud-native security, selecting an adversary is necessary for training the service, especially for defenders who have nothing to do without an attacker. After determining the learning algorithm and adversary, we can train the service for episodes and find out the best hyper-parameters.

Another way to train a new service is fine tuning a pre-trained one. There are two potential factors to improve a pre-trained service's performance: (1) Fine tuning with different adversaries may strengthen the service's robustness. (2) Fine tuning in different environments can transfer knowledge from

[3]https://github.com/openai/gym

the pre-trained one to another [27]. Similarly, once a pre-trained service is selected, the system can follow the same process above to select an adversary and then fine tune the pre-trained service on the selected environment.

4.2 Publish a Security Service to the Service Pool

Once the training or finetuning process is finished, we can package and publish the service to the *security service pool* for further evaluation. This phase extends the framework's ability from a personal application to a shared one, which can promote the use of different security services and, more importantly, benchmark their performance in a systematic way. Thus, a standard definition of the security service is necessary.

Definition 10. A Security Service is a quintuple *SS=(RO,DAG,RLA,INF)* where *RO* is the *role* which ca n be either attacker or defender, *DAG* is the *dynamic attack graph* representing the cloud-native environment, *RLA* represents the *action model* of the service, and *INF* consists of the *training information* including the adversary SS_{adv}, the pre-trained service SS_{pre}, and terminal condition if applicable.

Notably, the adversary SS_{adv} is also a security service. In the case where the adversary is not used, such as no defender service is implemented to protect a network that an attack service targets, the adversary will be represented as *NA* to denote the absence of an adversary. Pre-trained service SS_{pre} is involved when fine tuning a security service. Additionally, although a cloud-native system appears more like a continuous environment, it is necessary to set a terminal condition such as max timestep to ensure the effectiveness of the training.

4.3 Evaluating Security Services based on ELO Rating

Once published, we can easily run simulations to evaluate the service's performance with any adversary in the pool. As our service pool extends, it's challenging to evaluate the performance of a service in a systematic way, as it may perform well with some adversaries but fail with others. Hence, we apply the ELO rating [11] to calculate their relative performance. This is because the ELO rating is a method used to assess and compare the relative skill levels of players in competitive games or sports. It assigns numerical ratings to each player, and adjusts the ratings after each competition using the outcome and the opponents' ratings. This matches our scenario perfectly. Hence, in this section, we will first introduce the evaluation metrics and then detail the way we calculate the ELO rating.

4.3.1 Episodic metrics

Given an attacker and defender security service $< SS_{att}, SS_{def} >$ for evaluation, we can set up an environment modeled as a *DAG* and run rounds of simulations on it to collect the following metrics:

- **Average episode length**, *AEL* represents the average length of the simulation episodes. The lower *AEL* the better for the attacker as a lower *AEL* indicates the attacker takes a short time to achieve the attack goal.
- **Average episode reward**, *AER* represents the average reward of the simulation episodes. The larger *AER* the better for both attacker and defender services.

4.3.2 ELO rating based performance

Each service is assigned the same initial ELO rating. Take the attacker service as an example, we have each *DAG,* SS_{adv} as a scenario, and compare the performance of two attacker services in each scenario to update their ELO ratings with the following algorithms.

Algorithm 1 Update ELO rating.

Input: Metrics *AEL, AER* and ELO rating *R* for services $SS_{att,i}, SS_{att,j}$

1: **if** $(AEL_i \leq AEL_j) \wedge (AER_i > AER_j)$ **then**
2: $\quad S_{SS_{att,i}} = 1$ ▷ $S_{att,i}$ wins.
3: **else if** $(AEL_i = AEL_j) \wedge (AER_i = AER_j)$ **then**
4: $\quad S_{SS_{att,i}} = 0.5$ ▷ Draws.
5: **else**
6: $\quad S_{SS_{att,i}} = 0$ ▷ $S_{att,i}$ loses.
7: **end if**
8: $S_{SS_{att,j}} = 1 - S_{SS_{att,i}}$
9: $E_{SS_{att,i}} = (1 + 10^{\left(R_{SS_{att,i}} - R_{SS_{att,j}}\right)/400})^{-1}$
10: $E_{SS_{att,j}} = (1 + 10^{\left(R_{SS_{att,j}} - R_{SS_{att,i}}\right)/400})^{-1}$
11: $R_{SS_{att,i}} = R_{SS_{att,i}} + K \times \left(S_{SS_{att,i}} - E_{SS_{att,i}}\right)$
12: $R_{SS_{att,j}} = R_{SS_{att,j}} + K \times \left(S_{SS_{att,j}} - E_{SS_{att,j}}\right)$

In particular, for each service pair in Algorithm 1, we run simulations with the same adversary and scenario to collect the AEL and AER metrics. The winner or loser of the comparison is described as S_{SS} (lines 4–8). If and only if $S_{SS_{att,i}}$ has a lower AEL and higher AER, does it win. Then, we calculate the E_{SS} (lines 9, 10) as the change in rating and assign it to the services with predefined factor K (lines 11, 12).

Furthermore, as summarized in Algorithm 2, we continuously update the ELO ratings by iterating each pair of attack services in each scenario.

Eventually, we can compute the overall performance of all the attack services in the pool. Notably, we can run this ELO rating calculation method dynamically. In other words, once a new service is published into the pool, we can run simulations to compare its performance with existing services and update the ELO rating of services within the pool. Thus, the ELO rating benchmark is dynamic and can be almost real-time.

Algorithm 2 ELO rating benchmark from the attacker's perspective.

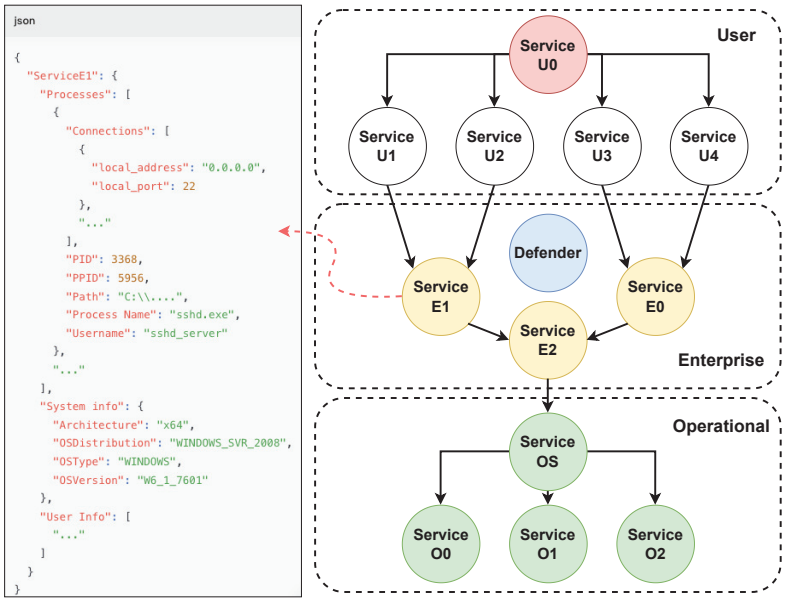
Input: A cloud-native environment *DAG*, Service Pool
for each $(SS_{att,i}, SS_{att,j}) \in$ Service Pool **do**
  for each $SS_{def,k} \in$ Service Pool **do**
    Metric$_{SS_{att,i}}$ = Simulation($SS_{att,i},\ SS_{def,k},\ DAG$)
    Metric$_{SS_{att,j}}$ = Simulation($SS_{att,j},\ SS_{def,k},\ DAG$)
    UpdateELORating(Metric$_{SS_{att,i}}$, Metric$_{SS_{att,j}}$)
  end for
end for

5 Scenario: CAGE Challenge 2

In this section, we present the experimental scenario named CAGE Challenge 2 (CC2) as a motivating case study to illustrate the intelligent security service model. Proposed by the TTCP CAGE Working Group,[4] this scenario is well-suited to the characteristics of cloud-native systems. The system comprises three isolated service subnets, where an attacker starts from a single foothold and progressively compromises the entire infrastructure. The objective of the defender is to prevent the attacker from compromising the availability and confidentiality of the network.

As shown in Figure 5, the CC2 system has been converted into a DAG. In practice, we can organize the properties of a node in a JSON format. Notably, certain microservices contain topology information that may reveal the existence of other service instances. Once a vulnerability is exploited to compromise a microservice, attackers can harvest such topological data, enabling cross-subnet infiltration. For example, service *SU1* includes the API endpoint to service *SE1*, so the successful connect on service *SU1* will enable attackers to discover *SE1* in another subnet. Each microservice contains at least one vulnerability that attackers can exploit through remote or local attack.

[4]https://github.com/cage-challenge/cage-challenge-2



(a) Node Example (b) Dynamic Attack Graph

Figure 5 CAGE Challenge 2 cloud-native system. (a) Configuration for node *Service E1* in JSON format. (b) The dynamic attack graph for the environment, including the nodes and their links through the API endpoint information.

Aligned with CC2's value assumptions and reward computation framework, each microservice is assigned a value tier: *none, low, medium, or high*. The rewards for both attackers and defenders are derived from a weighted combination of confidentiality and availability values. Furthermore, CC2 meticulously configures service-specific parameters and maps abstract actions in the action model to concrete operations. For example, a remote exploit can be translated into real-world attack vectors like EternalBlue or SQL injection – thereby enhancing the realism of cloud-native environment simulation.

We also define the parameters for action models to represent the real-world scenario:

- **False positive rate.** Given the imperfect and high false positive for existing threat detection tools [1], we assign a 5% false positive rate, which means ignoring 5% suspicious behavior in *discovered* or *owned* nodes or mistaking 5% normal nodes as suspicious, to reflect such a reality.

- **Reimage time.** When performing the *reimage* action, it takes time to restore the service from its image. We set it to 2 timesteps, which is shorter compared to traditional computer networks and, during this period, the service becomes inaccessible.
- **Remediate.** For the *remediate* action, if multiple vulnerabilities within a node are already exploited by an attacker, we will reset only one randomly.

Finally, we use the following termination condition:

- **Termination condition.** When an attacker connects to all nodes in the *Enterprise* subnet or reaches the maximum episode length of 100 time steps, the episode terminates.

6 Experiments and Results

In this section, we conduct experiments and analyze the results based on the CC2 system proposed in Section 5. We implement our framework by customizing *CybORG*, a typical ACO gym, to turn our intelligent security service model into a numeric environment based on *OpenAI Gym* to support reinforcement learning. It can be also built with any other ACO gyms [38].

6.1 Security Service Training and Publishing

The training algorithms are implemented based on Stable-Baselines3,[5] a library that integrates a series of reliable reinforcement learning algorithms. Furthermore, we designed an adapter to loosely connect the algorithm library and the gym environment, enabling the selection of algorithms.

To build up the security service pool, as reported in Table 1, we initially train three attacker services (*AA*, *AD*, and *AP*) and three defender services (*DA*, *DD*, and *DP*) using three typical reinforcement learning algorithms, A2C [30], DQN [31], and PPO [37], with fixed-strategy adversaries proposed by the CC2 scenario. Then, under the same environment, each defender security service is trained with one attacker security service as the adversary using one of the three reinforcement learning algorithms, resulting in a total of $3 \times 3 = 9$ defender services. Furthermore, we can fine tune the trained defender service with attacker adversary *AA* or *AD*, resulting in another 9 defender services trained with different conditions. In such a way, we can easily implement and publish 24 security services in the security service pool.

[5] https://github.com/DLR-RM/stable-baselines3

Table 1 Security service pool with services trained from scratch or pre-trained

Environment: CAGE Challenge 2 cloud-native system					
Phase	Service ID	Role	Algorithm	Adversary ID	Pre-train ID
Baseline	AA	Attacker	A2C	Fixed strategy service	
	AD		DQN		
	AP		PPO		
	DA	Defender	A2C		
	DD		DQN		
	DP		PPO		
From scratch	DAA	Defender	A2C	AA	\
	DAD		DQN		
	DAP		PPO		
	DDA		A2C	AD	
	DDD		DQN		
	DDP		PPO		
	DPA		A2C	AP	
	DPD		DQN		
	DPP		PPO		
Fine tune	DAAA	Defender	A2C	AA	DAA
	DAAD		DQN		DAD
	DAAP		PPO		DAP
	DADA		A2C	AA	DDA
	DADD		DQN		DDD
	DADP		PPO		DDP
	DDDA		A2C	AD	DDA
	DDDD		DQN		DDD
	DDDP		PPO		DDP

In fact, this training approach allows us to implement a series of security services in a systematic way. It can also incorporate complex combinations of strategies, through fine tuning with different adversaries or training algorithms, which can reflect the fact that both attackers and defenders can adopt diverse strategies for security operations. As this work aims at designing an effective framework for developing security services, rather than optimizing algorithms, we used the default parameters without optimization.[6]

6.2 Security Service Evaluation using ELO Rating

We simulated each security service under different scenarios and divided them into three phases: *baseline*, *from-scratch* and *fine tune*, for which we calculated the ELO rating benchmarks respectively.

[6]Training parameters are reported in the Appendix

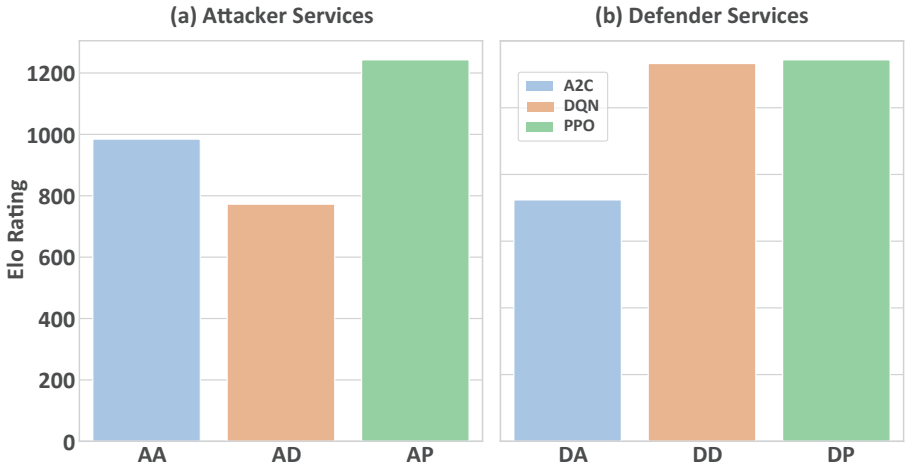


Figure 6 ELO rating benchmark for baseline services.

In each simulation, we constructed scenarios where the each service will face different adversaries from the last phase. Then we run 50 episodes for each scenario to calculate the *AEL* and *AER* metrics. Finally, we paired each security service within each group to calculate the final ELO rating. In particular, for the ELO rating, we followed the method of the International Chess Club (ICC), the world-renowned online chess site, setting the initial rating as 1000 and the K-factor as 32.[7]

6.2.1 Baseline Phase

As illustrated in Figure 6, we compare the performance of three attacker and defender services trained with distinct algorithms against fixed-strategy adversaries. The results demonstrate that *AP* achieves optimal performance in both attacker and defender roles under identical training conditions. By contrast, *AD* exhibits instability, its performance as an attacker lags significantly behind other services, yet approaches *AP*'s efficacy when acting as a defender.

Through the analysis of actions in the episodes, we found that this discrepancy partially stems from the fact that different algorithm-trained agents exhibit distinct behavioral patterns. For instance, in attack scenarios, the entropy values for both action distribution and target selection are 0.35

[7] https://www.chessclub.com/assets/Dasher/Ratings.htm.

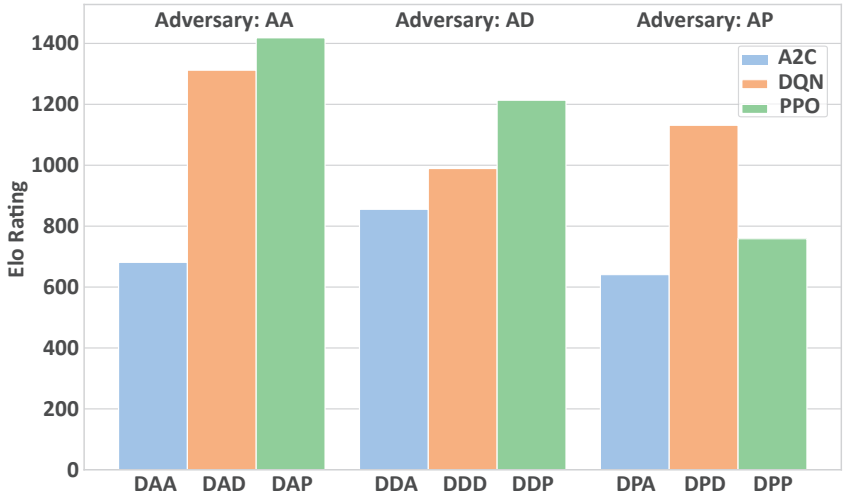


Figure 7 ELO rating benchmark for from-scratch defender services.

for *AD*, whereas *AP* demonstrates significantly higher diversity with corresponding entropy values of 2.21 and 1.86, respectively. That means *AP* can combine diverse methods to achieve network infiltration and lateral movement, whereas the *AD* demonstrates a relatively monotonous attack pattern, constrained by limited reward incentives. Correspondingly, as a defender, the *DD* conducts targeted defense by identifying critical nodes (service E2) in attack paths – despite its relatively simple action space, it achieves effective defensive outcomes.

These observations empirically validate that *different algorithms yield divergent service capabilities across roles and scenarios*, with our proposed evaluation framework quantitatively revealing their disparities.

6.2.2 From-scratch Phase

The performance of the defender services trained in the *from-scratch* phase is illustrated in Figure 7. From the perspective of training algorithms, this scenario still demonstrates the superiority of PPO over DQN and A2C, consistent with the performance observed in the *baseline* phase, indicating a certain degree of algorithmic stability in similar environments. However, compared to the baseline phase, where fixed-strategy adversaries were used, the introduction of dynamically trained adversaries in this stage led to variations in the defender agent's performance. Overall, the *DAP* and *DAD* models trained against the *AA* achieved relatively better results.

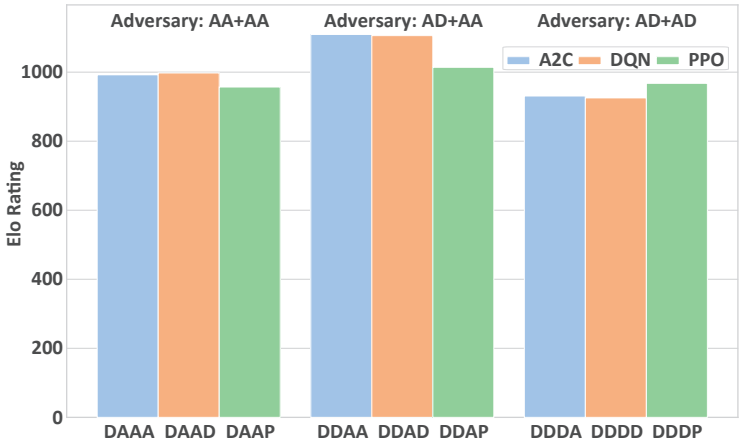


Figure 8 ELO rating benchmark for fine-tuned defender services.

However, an intriguing phenomenon emerges: when the defender and its adversarial opponent are trained using the same algorithm, the defender's overall performance against diverse attacking agents tends to degrade. For instance, when facing *AD*, *DDP* outperforms *DDD*; conversely, when confronting *AP*, *DPD* surpasses *DPP*. Taking a step further, defenders trained against same-algorithm adversaries consistently rank as the weakest within their respective groups. For example, *DDD* underperforms compared to *DAD* and *DPD*, while *DPP* lags behind *DAP* and *DDP*.[8]

Further investigation into these services' performance against different adversaries within episodes reveals that this phenomenon does not stem from individual opponent capability disparities. For example, while *DPP* was specifically trained against *AP*, we would expect it to outperform *DDP* when evaluated against *AP*. However, *DPP* demonstrates inferior performance compared to *DDP* across all three adversarial evaluations, highlighting the defense service's generalization capability when confronted with diverse opponents. This pattern implicitly suggests that although selecting the best-performing algorithm from previous rounds may enhance capability, *the diversity of adversarial opponents warrants deliberate consideration*.

6.2.3 Fine-tune Phase

Finally, we examine the effectiveness of the *fine tune* phase. Experimental results demonstrate that fine tuned services achieve performance

[8] *DAA* demonstrates only marginal superiority over *DPA*.

improvements over their pre-trained counterparts from the previous stage, for instance, *DAAA* exhibits a 3% increase in episode reward compared to *DAA*. However, our primary focus lies in whether the relative performance of these services shifts after training with different adversarial combinations.

As shown in Figure 8, defender services trained with mixed adversaries consistently outperform those trained against homogeneous adversaries. Notably, the group employing *AD+AA* as adversaries achieves optimal performance across all three algorithms, occupying the central position in the figure. This outcome validates our framework's capability in training services through complex strategic interactions, while simultaneously demonstrating that *fine tuning defender services against diverse attacker variants can effectively enhance their capabilities*.

7 Conclusion

Using the multi-agent reinforcement learning paradigm, we develop an agent-based intelligent security service framework (ISSF) for cloud-native security operations. It includes a dynamic attack graph model representing the environment, an action model representing agent actions, and a flexible approach to train, publish and evaluate the attacker and defender security services. The experiment on a CAGE Challenge 2 system confirms that our framework can effectively develop and quantitatively evaluate diverse, from simple to complex, intelligent security services and generate a benchmark to analysis their relative capability in a systematic way.

While the preliminary result confirms the effectiveness of our framework, our observations also open a gateway for future strategies optimization, such as incorporating advanced and diverse adversaries for training and fine-tuning. Additionally, further studies to investigate different parameters' influence on the stability of the ELO rating are also valuable. Finally, while the current performance evaluation is based on a simulated cloud-native environment, extending the evaluation on the emulated system would also be essential.

8 Appendix: Training Parameters

Please refer to https://stable-baselines3.readthedocs.io/en/master for more details.

A2C		**DQN**		PPO	
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
learning_rate	7e-4	learning_rate	1e-4	learning_rate	3e-4
n_steps	5	buffer_size	1e6	n_steps	2048
gamma	0.99	learning_starts	10000	batch_size	64
gae_lambda	1.0	batch_size	32	n_epochs	10
ent_coef	0.0	tau	1.0	gamma	0.99
vf_coef	0.5	gamma	0.99	gae_lambda	0.95
max_grad_norm	0.5	train_freq	4	clip_range	0.2
rms_prop_eps	1e-5	gradient_steps	1	ent_coef	0.0
use_rms_prop	True	target_update_interval	10000	vf_coef	0.5
use_sde	False	exploration_fraction	0.1	max_grad_norm	0.5
normalize_advantage	False	exploration_initial_eps	1.0	normalize_advantage	True
		exploration_final_eps	0.05		
		max_grad_norm	10		

Acknowledgement

The work was supported by the National Natural Science Foundation of China (62172425), the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China (22XNKJ04), and Cybersecurity at MIT Sloan, which is funded by a consortium of organizations.

References

[1] Bushra A Alahmadi, Louise Axon, and Ivan Martinovic. 99% false positives: A qualitative study of SOC analysts' perspectives on security alarms. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2783–2800, 2022.

[2] Mazhar Ali, Samee U Khan, and Athanasios V Vasilakos. Security in cloud computing: Opportunities and challenges. *Information sciences*, 305:357–383, 2015.

[3] Nuha Alshuqayran, Nour Ali, and Roger Evans. A systematic mapping study in microservice architecture. In *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*, pages 44–51. IEEE, 2016.

[4] Alex Andrew, Sam Spillard, Joshua Collyer, and Neil Dhir. Developing optimal causal cyber-defence agents via cyber security simulation. *arXiv preprint arXiv:2207.12355*, 2022.

[5] Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, Adrian Webster, et al. Bridging automated to autonomous cyber defense: Foundational analysis of tabular q-learning. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security*, pages 149–159, 2022.

[6] Arunkumar Arulappan, Aniket Mahanti, Kalpdrum Passi, Thiruvenkadam Srinivasan, Ranesh Naha, and Gunasekaran Raja. Dqn approach for adaptive self-healing of vnfs in cloud-native network. *IEEE Access*, 12:34489–34504, 2024.

[7] Shmulik Barlev, Z Basil, S Kohanim, Ron Peleg, S Regev, and Alexandra Shulman-Peleg. Secure yet usable: Protecting servers and linux containers. *IBM Journal of Research and Development*, 60(4):12–1, 2016.

[8] Mohamad Mulham Belal and Divya Meena Sundaram. Comprehensive review on intelligent security defences in cloud: Taxonomy, security issues, ml/dl techniques, challenges and future trends. *Journal of King Saud University-Computer and Information Sciences*, 34(10):9102–9131, 2022.

[9] André Carrusca, Maria Cecília Gomes, and João Leitão. Microservices management on cloud/edge environments. In *Service-Oriented Computing–ICSOC 2019 Workshops: WESOACS, ASOCA, ISYCC, TBCE, and STRAPS, Toulouse, France, October 28–31, 2019, Revised Selected Papers 17*, pages 95–108. Springer, 2020.

[10] Shuiguang Deng, Hailiang Zhao, Binbin Huang, Cheng Zhang, Feiyi Chen, Yinuo Deng, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Cloud-native computing: A survey from the perspective of services. *Proceedings of the IEEE*, 112(1):12–46, 2024.

[11] Arpad E Elo and Sam Sloan. The rating of chessplayers: Past and present. *(No Title)*, 1978.

[12] Viktor Engström, Pontus Johnson, Robert Lagerström, Erik Ringdahl, and Max Wällstedt. Automated security assessments of amazon web services environments. *ACM Transactions on Privacy and Security*, 26(2):1–31, 2023.

[13] Xiang He, Zhiying Tu, Xiaofei Xu, and Zhongjie Wang. Re-deploying microservices in edge and cloud environment for the optimization of user-perceived service quality. In *Service-Oriented Computing: 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019, Proceedings 17*, pages 555–560. Springer, 2019.

[14] Arash Heidari, Nima Jafari Navimipour, and Mehmet Unal. A secure intrusion detection platform using blockchain and radial basis function neural networks for internet of drones. *IEEE Internet of Things Journal*, 10(10):8445–8454, 2023.

[15] Yang Hu, Wenxi Wang, and Mohit Tiwari. Greybox penetration testing on cloud access control with iam modeling and deep reinforcement learning. *arXiv preprint arXiv:2304.14540*, 2023.

[16] Keman Huang, Michael Siegel, and Stuart Madnick. Systematically understanding the cyber attack business: A survey. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018.

[17] Amjad Ibrahim, Stevica Bozhinoski, and Alexander Pretschner. Attack graph generation for microservice architecture. In *Proceedings of the 34th ACM/SIGAPP symposium on applied computing*, pages 1235–1242, 2019.

[18] Hai Jin, Zhi Li, Deqing Zou, and Bin Yuan. Dseom: A framework for dynamic security evaluation and optimization of mtd in container-based cloud. *IEEE Transactions on Dependable and Secure Computing*, 18(3):1125–1136, 2019.

[19] Meraj Mostam Kashi, Anis Yazidi, and Hårek Haugerud. Mitigating yo-yo attacks on cloud auto-scaling. In *2022 14th IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 46–53. IEEE, 2022.

[20] Alper Kerman, Oliver Borchert, Scott Rose, Allen Tan, et al. Implementing a zero trust architecture. *National Institute of Standards and Technology (NIST)*, 75, 2020.

[21] Issa M Khalil, Abdallah Khreishah, and Muhammad Azeem. Cloud computing security: A survey. *Computers*, 3(1):1–35, 2014.

[22] Minhaj Ahmad Khan. A survey of security issues for cloud computing. *Journal of network and computer applications*, 71:11–29, 2016.

[23] Eunji Kim, Jungsu Han, and JongWon Kim. Visualizing cloud-native ai+ x applications employing service mesh. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1566–1569. IEEE, 2020.

[24] Yuanbo Li, Hongchao Hu, Wenyan Liu, and Xiaohan Yang. An optimal active defensive security framework for the container-based cloud with deep reinforcement learning. *Electronics*, 12(7):1598, 2023.

[25] Fang Liu, Guoming Tang, Youhuizi Li, Zhiping Cai, Xingzhou Zhang, and Tongqing Zhou. A survey on edge computing systems and tools. *Proceedings of the IEEE*, 107(8):1537–1562, 2019.

[26] Tengchao Ma, Changqiao Xu, Shujie Yang, Yiting Huang, Qingzhao An, Xiaohui Kuang, and Luigi Alfredo Grieco. A mutation-enabled proactive defense against service-oriented man-in-the-middle attack in kubernetes. *IEEE Transactions on Computers*, 2023.

[27] Zhao Mandi, Pieter Abbeel, and Stephen James. On the effectiveness of fine-tuning versus meta-reinforcement learning. *arXiv preprint arXiv:2206.03271*, 2022.

[28] Antony Martin, Simone Raponi, Théo Combe, and Roberto Di Pietro. Docker ecosystem–vulnerability analysis. *Computer Communications*, 122:30–43, 2018.

[29] Nuno Mateus-Coelho, Manuela Cruz-Cunha, and Luis Gonzaga Ferreira. Security in microservices architectures. *Procedia Computer Science*, 181:1225–1236, 2021.

[30] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[32] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

[33] Jargalsaikhan Narantuya, Seunghyun Yoon, Hyuk Lim, Jin-Hee Cho, Dong Seong Kim, Terrence Moore, and Frederica Nelson. Sdn-based ip shuffling moving target defense with multiple sdn controllers. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks–Supplemental Volume (DSN-S)*, pages 15–16. IEEE, 2019.

[34] Gregory Palmer, Chris Parry, Daniel JB Harrold, and Chris Willis. Deep reinforcement learning for autonomous cyber operations: A survey. *arXiv preprint arXiv:2310.07745*, 2023.

[35] Neeraj Kumar Pandey, Krishna Kumar, Gaurav Saini, and Amit Kumar Mishra. Security issues and challenges in cloud of things-based applications for industrial automation. *Annals of Operations Research*, 342(1):565–584, 2024.

[36] Keri Pearlson and Keman Huang. Design for cybersecurity from the start. *MIT Sloan Management Review*, 63(2):73–77, 2022.

[37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[38] Maxwell Standen, Martin Lucas, David Bowman, Toby J Richer, Junae Kim, and Damian Marriott. Cyborg: A gym for the development of autonomous cyber agents. *arXiv preprint arXiv:2108.09118*, 2021.

[39] Hamed Tabrizchi and Marjan Kuchaki Rafsanjani. A survey on security challenges in cloud computing: issues, threats, and solutions. *The journal of supercomputing*, 76(12):9493–9532, 2020.

[40] Microsoft Defender Research Team. Cyberbattlesim. https://github.com/microsoft/cyberbattlesim, 2021. Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei.

[41] Theodoros Theodoropoulos, Antonios Makris, Ioannis Kontopoulos, John Violos, Przemysław Tarkowski, Zbyszek Ledwoń, Patrizio Dazzi, and Konstantinos Tserpes. Graph neural networks for representing multivariate resource usage: A multiplayer mobile gaming case-study. *International Journal of Information Management Data Insights*, 3(1):100158, 2023.

[42] Theodoros Theodoropoulos, Luis Rosa, Chafika Benzaid, Peter Gray, Eduard Marin, Antonios Makris, Luis Cordeiro, Ferran Diego, Pavel Sorokin, Marco Di Girolamo, et al. Security in cloud-native services: A survey. *Journal of Cybersecurity and Privacy*, 3(4):758–793, 2023.

[43] Kennedy A Torkura, Muhammad IH Sukmana, Anne VDM Kayem, Feng Cheng, and Christoph Meinel. A cyber risk based moving target defense mechanism for microservice architectures. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pages 932–939. IEEE, 2018.

[44] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the tenth european conference on computer systems*, pages 1–17, 2015.

[45] Sanyam Vyas, John Hannay, Andrew Bolton, and Professor Pete Burnap. Automated cyber defence: A review. *arXiv preprint arXiv:2303.04926*, 2023.

[46] Erich Walter, Kimberly Ferguson-Walter, and Ahmad Ridley. Incorporating deception into cyberbattlesim for autonomous defense. *arXiv preprint arXiv:2108.13980*, 2021.

[47] Huan Wang, Yunlong Tang, Yan Wang, Ning Wei, Junyi Deng, Zhiyan Bin, and Weilong Li. Research on active defense decision-making method for cloud boundary networks based on reinforcement learning of intelligent agent. *High-Confidence Computing*, page 100145, 2023.

[48] Hanyi Xu, Guozhen Cheng, Xiaohan Yang, Wenyan Liu, Dacheng Zhou, and Wei Guo. Multi-dimensional moving target defense method based on adaptive simulated annealing genetic algorithm. *Electronics*, 13(3):487, 2024.

[49] George OM Yee. Modeling and reducing the attack surface in software systems. In *2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*, pages 55–62. IEEE, 2019.

[50] Dongjin Yu, Yike Jin, Yuqun Zhang, and Xi Zheng. A survey on security issues in services communication of microservices-enabled fog applications. *Concurrency and Computation: Practice and Experience*, 31(22):e4436, 2019.

[51] Marco Zambianco, Claudio Facchinetti, Roberto Doriguzzi-Corin, and Domenico Siracusa. Resource-aware cyber deception in cloud-native environments. *arXiv preprint arXiv:2303.03151*, 2023.

[52] Marco Zambianco, Claudio Facchinetti, Roberto Doriguzzi-Corin, and Domenico Siracusa. Resource-aware cyber deception for microservice-based applications. *IEEE Transactions on Services Computing*, 2024.

[53] Fiorella Zampetti, Salvatore Geremia, Gabriele Bavota, and Massimiliano Di Penta. Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 471–482. IEEE, 2021.

[54] Uwe Zdun, Pierre-Jean Queval, Georg Simhandl, Riccardo Scandariato, Somik Chakravarty, Marjan Jelic, and Aleksandar Jovanovic. Microservice security metrics for secure communication, identity management, and observability. *ACM Transactions on Software Engineering and Methodology*, 32(1):1–34, 2023.

[55] Kengo Zenitani. Attack graph analysis: an explanatory guide. *Computers & Security*, 126:103081, 2023.

Biographies

Yikuan Yan received his Bachelor degree in information security in 2023 from the Central University of Finance and Economics, China. He is currently pursuing his Master's degree at the School of Information, Remin University of China, China. His research interests include autonomous cyber operation and multi-agents system.

Keman Huang is an Associate Professor at the Renmin University of China and a Research Affiliate at the MIT Sloan School of Management. He uses data-driven empirical study and simulation to work on cybersecurity behavior, policy and strategy, service innovation ecosystems for cutting edge technologies including AI and Blockchain. He has published more than 70 articles in top journals, conferences and magazines, including Harvard Business Review, MIT Sloan Management Review, ACM computing surveys, ACM Conference on Computer Supported Cooperative Work, and International Conference on Web Services.

Michael Siegelis a Principal Research Scientist at the MIT Sloan School of Management and also the Director of Cybersecurity at MIT Sloan (CAMS). Siegel's research focuses on the management, strategy, technology, and organizational issues related to cybersecurity with specific interest in vulnerability markets, cyber risk management, dark web business models, IoT endpoint security, vulnerability management, cybersecurity workforce development, and educating management in cybersecurity. He also has done research in the intelligent integration of information systems, risk management, insurgency and state stability, data analytics, healthcare systems, and systems modeling. Siegel has published articles on such topics as simulation modeling for cyber resilience, cyber vulnerability markets, AI and cybersecurity, data management strategy, architecture for practical metadata integration, heterogeneous database systems, and managing and valuing a corporate IT portfolio using dynamic modeling of software development and maintenance processes. His research at MIT has continued for over 35 years and includes a wide range of publications, patents and teaching accomplishments.