Hierarchical Multi-agent Reinforcement Learning for Cyber Network Defense

Anonymous authors
Paper under double-blind review

Keywords: Multi-agent reinforcement learning, Cybersecurity, Deep reinforcement learning, Hierarchical reinforcement learning

Summary

Recent advances in multi-agent reinforcement learning (MARL) have created opportunities to solve complex real-world tasks. Cybersecurity is a notable application area, where defending networks against sophisticated adversaries remains a challenging task typically performed by teams of security operators. In this work, we explore novel MARL strategies for building autonomous cyber network defenses that address challenges such as large policy spaces, partial observability, and stealthy, deceptive adversarial strategies. To facilitate efficient and generalized learning, we propose a hierarchical Proximal Policy Optimization (PPO) architecture that decomposes the cyber defense task into specific sub-tasks like network investigation and host recovery. Our approach involves training sub-policies for each sub-task using PPO enhanced with domain expertise. These sub-policies are then leveraged by a master defense policy that coordinates their selection to solve complex network defense tasks. Furthermore, the sub-policies can be fine-tuned and transferred with minimal cost to defend against shifts in adversarial behavior or changes in network settings. We conduct extensive experiments using CybORG Cage 4, the state-of-the-art MARL environment for cyber defense. Comparisons with multiple baselines across different adversaries show that our hierarchical learning approach achieves top performance in terms of convergence speed, episodic return, and several interpretable metrics relevant to cybersecurity, including the fraction of clean machines on the network, precision, and false positives.

Contribution(s)

1. A scalable hierarchical multi-agent reinforcement learning method for cyber defense that decomposes the complex cyber defense task into multiple sub-tasks.
   Context: Prior work uses hierarchical MARL in other domains such as multi-robot learning, while current RL-based methods in the cyber defense domain are single agent.
2. A design guided by cybersecurity domain expertise to enhance the RL agents' observation space and facilitate learning of better policies.
   Context: Prior work on RL cyber defense uses the observation space provided by a cyber environment such as CybORG, without expanding it.
3. Defensive strategies that transfer either directly or via fine-tuning against a range of deceptive, stealthy adversaries in the CybORG CAGE 4 cyber environment.
   Context: We show that the proposed H-MARL methods generalize to three types of stealthy adversarial agents, besides the default red agent in CybORG CAGE 4, and we also demonstrate transferability to new red agents after fine-tuning.
4. Definition and analysis of multiple interpretable metrics for providing insights to security operators on the developed defenses.
   Context: Prior work in RL for cyber defense mainly analyzes the cumulative return, but does not discuss interpretable metrics, which are very relevant to security operators.

Hierarchical Multi-agent Reinforcement Learning for Cyber Network Defense

Anonymous authors
Paper under double-blind review

Abstract

Recent advances in multi-agent reinforcement learning (MARL) have created opportunities to solve complex real-world tasks. Cybersecurity is a notable application area, where defending networks against sophisticated adversaries remains a challenging task typically performed by teams of security operators. In this work, we explore novel MARL strategies for building autonomous cyber network defenses that address challenges such as large policy spaces, partial observability, and stealthy, deceptive adversarial strategies. To facilitate efficient and generalized learning, we propose a hierarchical Proximal Policy Optimization (PPO) architecture that decomposes the cyber defense task into specific sub-tasks like network investigation and host recovery. Our approach involves training sub-policies for each sub-task using PPO enhanced with cybersecurity domain expertise. These sub-policies are then leveraged by a master defense policy that coordinates their selection to solve complex network defense tasks. Furthermore, the sub-policies can be fine-tuned and transferred with minimal cost to defend against shifts in adversarial behavior or changes in network settings. We conduct extensive experiments using CybORG Cage 4, the state-of-the-art MARL environment for cyber defense. Comparisons with multiple baselines across different adversaries show that our hierarchical learning approach achieves top performance in terms of convergence speed, episodic return, and several interpretable metrics relevant to cybersecurity, including the fraction of clean machines on the network, precision, and false positives.

1 Introduction

Cyber defense is critical in both private and public network infrastructures, which are frequently targeted by increasingly sophisticated external attackers with malicious intentions. In 2024, the number of security breaches has surpassed 10,000 and attackers constantly adapt their tools and strategies to evade existing defenses (Hylender et al., 2024). The implications for national security are significant, as security breaches often lead to theft of intellectual property, compromise of sensitive information, and disruption of critical infrastructures. Currently, organizations employ teams of security professionals who constantly oversee the security of their networks and design the overall security strategy using their domain expertise. While a range of machine learning (ML) tools are available for detecting specific classes of attacks (Antonakakis et al., 2012; Nelms et al., 2013; Antonakakis et al., 2011; Bilge et al., 2012; Yen et al., 2013; Ongun et al., 2021), the advancement of deep reinforcement learning (DRL) presents an opportunity to automate the cyber defense strategy and reduce the burden on security operators.

Towards this goal, the technical cooperation program (TTCP), a collaborative working group including UK, USA, Canada, Australia and New Zealand, developed a series of CAGE challenges for advancing cyber defense (CAGE, 2021). These challenges leverage the Cyber Operations Research Gym (CybORG), a simulated environment that can be used for creating realistic interactions (or games) between attackers and defenders on realistic network topologies. These environments task defenders (blue agents) with monitoring and restoring compromised machines on a simulated enterprise network, to prevent external adversaries (red agents) from accessing critical assets. The first CAGE challenges model a cyber game between a single blue agent and a single red agent, leading to the development of DRL techniques for training a blue agent interacting against a red agent (Vyas et al., 2023; Mcdonald et al., 2024; Hammar et al., 2024). The most recent CAGE 4 challenge (CAGE-4, 2023) models a team of multiple blue agents defending a distributed network, playing against multiple red agents compromising the network. Existing techniques for single defensive agents are either computationally expensive (by training a different agent for each red agent (Vyas et al., 2023; Mcdonald et al., 2024)), do not generalize to new attackers, or require extensive, causal pre-processing which is intractable as the network size scales (Hammar et al., 2024). Thus, they cannot be immediately applied to the multi-agent CAGE 4 environment, which requires new methods. Additional challenges for training multi-agent defenders in this environment include large policy spaces, partial observability of the network, shared rewards among all blue agents, and playing against stealthy, deceptive adversarial strategies.

In this paper, we propose the first scalable multi-agent reinforcement learning (MARL) technique for automating defense in cyber security environments such as CAGE 4. We formulate the problem as a decentralized, partially-observable Markov decision process (Dec-POMDP) (Oliehoek & Amato, 2016) and propose two hierarchical strategies, H-MARL Expert and H-MARL Meta, each with their own advantages. Both methods decompose the complex cyber defense task into smaller subtasks, and train sub-policies for each sub-task using PPO enhanced with domain expertise. The difference between the methods is in the design of the master policy that coordinates the selection of the sub-policies at each time step. H-MARL Expert utilizes security domain expertise to define a top master rule based on well-established practices for cyber defense, and performs best in most of our experiments. However, there are situations when it is difficult to define a deterministic Expert rule. To address this issue, we propose H-MARL Meta, that trains the master policy, and, thus, has the advantage of generalizing to new, unseen adversarial behavior. Another insight in our design is that extended observation spaces including security indicators, such as presence of malicious files and malicious processes on a host, are beneficial in increasing the blue agent's ability to defend the network. We evaluate our methods against multiple baselines at different stages of the design process to motivate our methodology and design decisions. We further propose multiple relevant and interpretable metrics for cyber defense, including ratio of uninfected hosts, false/true positive rates on host recovery, and number of adversarial impacts on hosts. Across these metrics our proposed hierarchical techniques display significant improvements over traditional MARL approaches.

To summarize, our contributions are: (1) scalable hierarchical multi-agent reinforcement learning methods for cyber defense; (2) a design guided by domain expertise to enhance the agents' observation space and decompose the complex cyber defense task into multiple sub-tasks; (3) evaluation in CybORG CAGE 4, a realistic cyber environment with partial observability and deceptive, stealthy adversaries; (4) empirical transferability of trained sub-policies after fine-tuning to new adversarial agents, and (5) multiple interpretable metrics for providing insights to security operators.

2 Related Work and Background

Traditional cyber defenses, such as anti-virus and network intrusion detection tools, leverage specific detection rules for thwarting existing attacks, but they are relatively easy to evade. To address their limitations, organizations employ security operators who perform "threat hunting" to detect novel attacks on their networks. A variety of machine learning (ML) tools are available for threat detection (Antonakakis et al., 2011; 2012; Bilge et al., 2012; Nelms et al., 2013; Yen et al., 2013; Ongun et al., 2021), but the overall defensive strategy in most organizations is still manually designed. The advancement of DRL and MARL provides an opportunity to automate cyber defense strategies and improve the security of cyber infrastructures.

The CAGE-4 challenge (CAGE-4, 2023) is a recent security environment aimed at encouraging research in autonomous cyber defense. It provides a cyber simulation of attacker and defender actions in realistic network topologies. CAGE-4 is a partially observable environment with multiple, decentralized blue agents defending the network by playing against a team of red agents performing various attacks over time. It can be modeled as a decentralized, partially observable Markov decision process (Dec-POMDP). Dec-POMDPs (Oliehoek & Amato, 2016) are a special class of MDP where multiple, independent and decentralized agents with incomplete observations interact to optimize a shared reward signal. Formally, a Dec-POMDPs is defined as the tuple $\mathcal{M} = (\mathcal{I}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R}, b_o)$, where $\mathcal{I} = \{1 \cdots n\}$ is the set of n agents, $\mathcal{S}$ is a finite set of states, $\mathcal{A} = \times_{i \in \mathcal{I}} A_i$ is the set of joint actions composed of individual actions A_i for each agent i, $\mathcal{T}$ is the state transition function, $\Omega = \times_{i \in \mathcal{I}} \Omega_i$ is the set of joint observations, $\mathcal{O} : \mathcal{S} \times \mathcal{A} \rightarrow \Omega$ is the joint observation function, and b_0 is a distribution over initial states.

Several methods have been developed for solving general Dec-POMDPs, including multi-agent PPO (Yu et al., 2022; Schulman et al., 2017), Q-MIX (Rashid et al., 2020), and independent PPO (De Witt et al., 2020). However, these methods are often uninterpretable and struggle to converge in settings with large joint action spaces. Consequently, some works have developed hierarchical or meta-action-based approaches, which combine multiple actions or sequences of actions into a single, meta-action.

Hierarchical and meta-learning approaches in single-agent reinforcement learning have been explored (Hutsebaut-Buysse et al., 2022; Lake et al., 2017; Hospedales et al., 2022; Mishra et al., 2018; Sun et al., 2019; Guan et al., 2024). Several works have developed adaptations of hierarchical approaches in MARL for different domains (Xiao et al., 2020; Fosong et al., 2024; Chang et al., 2023; Frans et al., 2017), though the general problem remains a challenge. In this work, we aim to study the design and capabilities of Hierarchical MARL (H-MARL) approaches specific to the cyber network defense domain. Previous works (Vyas et al., 2023; Mcdonald et al., 2024; Hammar et al., 2024) have looked at single-agent domains, and to the best of our knowledge there is no existing work applying MARL to cyber network defense.

3 Problem Statement

In cyber security, red teams act as attackers who attempt to exploit network vulnerabilities and carry out malicious activities aimed at compromising the system. Blue teams are tasked with defending against red team opponents to secure the networks, while maintaining network operations. In this work, we focus on the CybORG CAGE 4 cybersecurity MARL framework (CAGE-4, 2023), which is a realistic environment that models cyber defense. We discuss several aspects of the environment below. For additional details, please see Section 1 in the supplemental materials, and the CAGE 4 description (CAGE-4, 2023).

Network topology. Cyber networks are often segmented into operational enterprise networks that encompass multiple security zones depending on the proximity to critical resources. This setup leads to a multi-agent competitive environment, where each defender agent is protecting its own security zone(s), with the overarching team goal of defending the entire network. The CAGE 4 network consists of seven security zones (subnets), assigned to five blue agents. To increase robustness of defenses, the number of hosts in each zone and their services are randomized, with each zone having between 4-16 servers and user machines (or hosts). An additional network (Contractor) is completely undefended, so that the red team always maintains a presence in the network.

Threat Model. The two teams are represented by multi-agent systems: defender (the blue team) and attacker (the red team). Defender and attacker have competing goals, while the agents on each team collaborate to achieve their goals. The attacker's goal is to maximize its reward by degrading services available to users, represented by green agents, and compromising the critical Operational Technology (OT) service. The defender's goals are two-fold: maintain the security of the cyber network by reducing the adversarial presence, and minimize the operational impact on users. We face a strong adversary, whose capabilities include stealth, phishing, propagating through the network, and the ability to discover blue agents' deception. Red agents maintain persistent presence on the Contractor network, which is the starting point of the attack and cannot be defended with

Table 1: Challenges of a realistic RL model for cybersecurity, with concrete examples from the CybORG environment.

Environment	
Partially-observable	Blue agents receive incomplete information from the environment. Lacking access to the true state, agents monitor and analyse hosts to discover compromised hosts. Monitoring is noisy, affected by false positives and false negatives (depending on detection rate).
Memoryless	Once reported, alerts are not maintained.
Duration of actions	Actions take between one time step and 5 time steps in the environment.
Large policy space	Blue agents defend between 1 and 3 subnets, of up to 16 hosts each. This maps to an 92-242 action space, and an 82-210 observation space.
Shared reward	Blue agents are rewarded collectively, as a team, for defending the network. However, they only receive local observations.
Adversary	
Stealth	Low chance (0.25) of raising alerts via the Stealthy Service Discovery action. Ability to withdraw from a security zone after impact.
Deception	Can detect and avoid decoy services.
Ease of spreading	Appears via phishing emails, in addition to moving laterally through the network.
Strong foothold	Can not be removed from the Contractor subnet, which is undefended. The red team scores about 60 points on average in this subnet.

blue actions. Red agents scan machines for vulnerable services to exploit and propagate through the network. CAGE 4 implements both an *aggressive service discovery* action, which is faster (1 time step), but has a high chance (0.75) of raising alerts, and a *stealthy service discovery* action, which is slower (3 time steps), but less likely (0.25) to raise alerts. In addition to *moving laterally* through the network by exploiting remote services, red agents can also *spawn* with a given probability when a green user opens a phishing email or accesses a compromised service. Furthermore, red agents can use the *discover deception* action to determine if the blue team has installed decoy services on a specific host, and avoid to infect that host to maintain stealth. CAGE 4 implements a default deceptive red agent, called *FiniteStateRedAgent*, but we also create our own red agents with more aggressive service discovery and even stealthier presence on the network to measure the generality of our defense.

Defensive actions and rewards. The blue team monitors the network for suspicious events, and detects and responds to attacks through the following actions: analyze a host looking for malware information; start a decoy service on a host (blue team is alerted when a red agent attempts to compromise the decoy service); block traffic to and from a specified security zone (at the expense of disrupting the work of green agents); allow traffic to and from a specified security zone; remove malicious processes from a host; restore the host to an earlier secure state (temporarily making its services unavailable). The reward scheme models a zero-sum game where blue agents incur penalties when green agents are impacted due to degraded services becoming inaccessible. In addition, blue agents are penalized when red agents impact the critical OT security service. The specific reward values depend on the *mission phase*. Three active phases are carried out throughout each episode, to reflect the changing criticality of security zones on current operations.

Challenges. A series of features, described in Table 1, make the CybORG environment particularly realistic and challenging for training multi-agent blue defenders. The environment provides partial observability of red presence, as blue agents need to run monitor and analyse actions to discover compromised hosts, and these actions incur false positives and false negatives. The policy space is large, including a set of actions for each host on the network, and the observation space is memoryless. In addition, actions have variable duration, and all blue agents share a common reward, even though each of them protects a different part of the network.

4 H-MARL Methodology

For complex real-world tasks, action spaces can grow intractably large as the problem, or task, scales, making task learning difficult for many standard reinforcement learning (RL) approaches (Dulac-Arnold et al., 2015). These issues are compounded by the introduction of high-dimensional, noisy state spaces. Many hierarchical RL methods aim to solve these problems by breaking large action spaces down into smaller sub-tasks. However, learning these partitions online remains a challenging problem (Hutsebaut-Buysse et al., 2022). Therefore, in this section, we introduce our hierarchical framework showing how domain expertise can be leveraged to solve issues of intractability in both the state and action spaces.

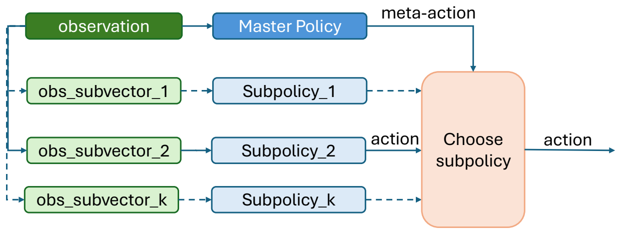

Figure 1: Hierarchical MARL. Upon receiving an observation, the master policy first chooses a sub-policy, which selects the final primitive action.

An overview of our design is shown in Figure 1. The action space $\mathcal{A}$ is split into n smaller subsets, or classes, chosen using domain expertise. For example, the "recover" class refers to all primitive actions for removing processes and restoring machines on the network. Thus, each sub-policy handles one class of primitive actions that will be executed in the network. We define the action space of the master policy as a new set $\mathcal{A}_m$ comprised of meta-actions, where each meta-action corresponds to a different sub-policy. The master policy chooses a meta-action, and the associated sub-policy then samples a primitive action from its subset. In cyber domains, the agent must choose the machine to investigate or restore potentially from a list of thousands. In our design, these additional details, such as what machine to restore, will be abstracted away, under a single meta-action, significantly reducing the action space.

Under this formulation we begin to leverage the hierarchical design. We first establish a master policy π_m whose objective is to choose some meta-action $A_c \in \mathcal{A}_m$ given observation o_t at time step t. We then assign each meta-action A_c to a corresponding sub-policy ψ_c whose goal is to choose the primitive action $a_t \in A_c$ given some input history h_t at time step t. This primitive action a_t is the final action executed in the environment. Under this design the master policy must learn the best policy $\pi_m : \mathcal{H} \to \mathcal{A}_m$ over meta-actions, while each sub-policy $\psi_c : \mathcal{H} \to A_c$ must learn the best policy over all actions in their respective meta-action class. Here $\mathcal{H}$ represents the set of possible observation histories. This reduces the larger, more complex task posed by the base Dec-POMDP into a much manageable set of sub-tasks.

We then use expert domain knowledge to enhance each sub-policy's respective observations with transformation functions $f_c : \mathcal{O} \to \mathcal{O}_c$ for sub-policy observation spaces $\mathcal{O}_c$. In practice, these transformations are applied to observation histories. The transformation function reduces the observation space of each sub-policy by keeping only information relevant to their respective class of actions. For example, the sub-policy responsible for restoring machines only needs to know about the hosts that present clear indicators of compromise, rather than about all the alerts in the system.

4.1 Hierarchical MARL Design

In the previous section, we primarily talked about our hierarchical approach at a high level from a single-agent perspective. In this subsection, we propose our methods H-MARL Expert and H-MARL Meta, which are designed to overcome a multitude of difficulties induced by multi-agent training, such as environment and training instability caused by policy interdependencies. In particular, there are three key interdependencies that make learning in this setting difficult: (i) inter-

Algorithm 1 Sub-policy training (H-MARL Expert)

```
Input Dec-POMDP $\mathcal{M}$, Expert policy $\pi_E$, Transformations $\{f_c\}_{c=1}^{k}$
Initialize Sub-Policies $\{\psi_c\}_{c=1}^{k}$, Replay Memories $\{\mathcal{D}_c\}_{c=1}^{k}$, Transformations $\{f_c\}_{c=1}^{k}$, Episode Length $T$, Iterations $N$
for $i \leftarrow 1, N$ do
    Sample initial observation $o_0 \sim \mathcal{M}$
    for $t \leftarrow 1, T$ do
        Update history $h_t$ given observation $o_{t-1}$
        Sample $c \leftarrow \pi_E(h_t)$ from expert policy
        Sample $o_t,\ r_t \sim \mathcal{M}$ given action $a_t \sim \psi_c(f_c(h_t))$
        Store $(h_t, a_t, r_t)$ in $\mathcal{D}_c$
    for $j \leftarrow 1, k$ do
        Update $\psi_j$ given $\mathcal{D}_j$ using PPO
```

Algorithm 2 Master policy training (H-MARL Meta)

```
Input Dec-POMDP $\mathcal{M}$, Sub-Policies $\{\psi_c\}_{c=1}^{sp}$, Transformations $\{f_c\}_{c=1}^{k}$
Initialize Master Policy $\pi_m$, Replay Memory $\mathcal{D}_m$, Episode Length $T$, Iterations $N$
for $i \leftarrow 1, N$ do
    Sample initial observation $o_0 \sim \mathcal{M}$
    for $t \leftarrow 1, T$ do
        Update history $h_t$ given observation $o_{t-1}$
        Sample $c_t \leftarrow \pi_m(h_t)$ from master policy
        Sample $o_t$, $r_t \sim \mathcal{M}$ given action $a_t \sim \psi_{c_t}(f_c(h_t))$
        Store $(h_t, c_t, r_t)$ in $\mathcal{D}_m$
    Update $\pi_m$ given $\mathcal{D}_m$ using PPO
```

dependence between each agent under a shared reward signal, (ii) between master and sub-policy performance, and (iii) between sub-policies under shared episodic returns.

This first interdependence results in agents receiving rewards that are not related to actions they have taken. This is particularly challenging in our cyber defense setting as each agent interacts with disjoint sub-networks, but receives a shared reward considering the state of the whole network. This leads us to use IPPO (De Witt et al., 2020) as the foundation of our approach, where each agent has a separate critic which only receives observations corresponding to their respective sub-network. Each agent is then trained in parallel along with its respective critic. This setup prevents the critics from being biased by occurrences outside their respective agent's sub-network, resulting in greater stability and less bias in each agent. Secondly, the performance of the master policy depends on the performance of each sub-policy – poorly trained sub-policies can make otherwise optimal meta-actions sub-optimal – resulting in a biased master policy. To overcome this, we utilize a two-phase training approach seen in Algorithm 1 and Algorithm 2.

Algorithm 1: Sub-policy training (H-MARL Expert). Algorithm 1 describes how we train the sub-policies for each agent i in parallel. We first define an expert policy π_E called H-MARL Expert, which uses deterministic rules generated from domain knowledge to select meta-actions, along with a set of observation transformation functions $\{f_c\}_{c=1}^{k}$. At the start of each episode, we receive an initial observation from $\mathcal{M}$ and use it to initialize our history h_t. At each time step π_E then uses h_t to choose the best meta-action A_c. The sub-policy ψ_c subsequently chooses a primitive action a_t given its transformed history $f_c(h_t)$. This action is used to sample the next observation and reward from $\mathcal{M}$ which is stored in the replay memory $\mathcal{D}_c$ of ψ_c. Each policy is then updated with PPO (Schulman et al., 2017) on its respective replay memory $\mathcal{D}_c$. This design allows the sub-policies to address only their related tasks and train to near-optimal while avoiding instability caused by a trained master policy. Additionally, this allows us to solve our third form of interdependence, as the deterministic, static expert policy π_E allows for stability in the training of each sub-policy.

Algorithm 2: Master policy training (H-MARL Meta). Algorithm 2 provides the second phase of training for the master policy. Here sub-policies $\{\psi\}_1^k$ trained with Algorithm 1 are kept frozen and just used to generate primitive actions. We define a master policy π_m called H-MARL Meta whose action space size is the number of sub-policies k. Similar to before, at each time step we sample an action $c_t \sim \pi_m(h_t)$ given updated history h_t. The sub-policy ψ_{c_t} is invoked to sample action a_t to take a step in $\mathcal{M}$ given transformed history $f_c(h_t)$. The current history h_t, action c_t and reward from $\mathcal{M}$ given action a_t are then stored in the replay memory $\mathcal{D}_m$. Finally, the master policy π_m is updated with PPO given trajectories stored in $\mathcal{D}_m$.

Instantiating H-MARL in cybersecurity. Figure 2 illustrates the sub-task partition in a cyber environment such as CybORG CAGE 4. We identify three types of sub-tasks: investigate host, recover host, and control traffic between zones. The state-level abstraction used to partition the tasks refers to the presence of indicators of compromise (IOCs) within an agent's security zone(s). Thus, if IOCs such as malicious files are identified on a host, the agent will choose to recover, either by removing the malware or by restoring the machine to a clean state. Furthermore, if network IOCs are detected, such as incoming scanning activity captured by decoy services, then the sub-policy handling traffic control can choose to block traffic originating from the compromised subnet. Each of these sub-policies will receive the relevant subspace of the observation space and the subset of relevant actions. Note that our sub-policies are agnostic to the particular attacker strategy, being motivated by well-established cyber defense practices.

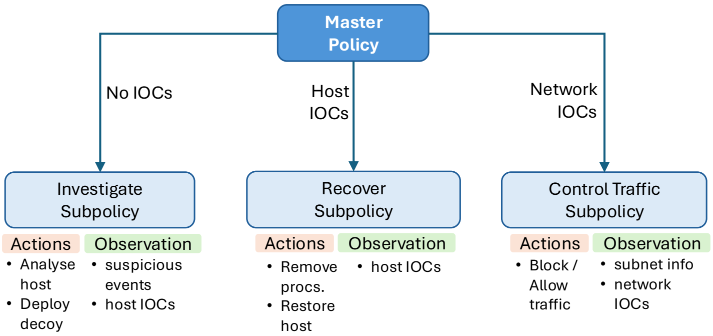

Figure 2: Instantiating H-MARL in cybersecurity.

In the CAGE 4 setup, blocking the traffic between neighboring subnets is not particularly useful for two reasons: (i) it prevents green agents from accessing remote services, which penalizes the blue team, and (ii) red agents are active mostly within their own assigned subnet, and rarely reach remotely into other subnets, a setup that ensures scaling with large networks. In the rest of the paper, with the exception of Section 5.4, we discuss defenses that do not use block or allow traffic actions, focusing on the Investigate and Recover sub-policies.

4.2 Observation Space Design

This section presents our enhancements to the observation space of blue agents and evaluate their ability to facilitate learning of a better defense strategy. In cybersecurity environments, the state holds a lot of data, such as information on system configuration, processes, active sessions, etc. The observation is presented to the agent as a data structure with various fields, which may vary depending on the output of the action. This observation structure needs to be filtered and converted to a consistent vector representation to enable the use of deep reinforcement learning techniques.

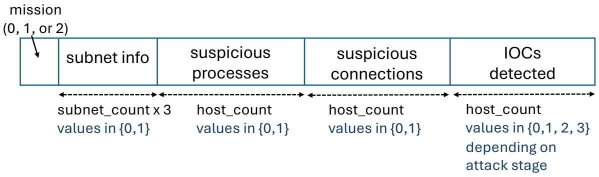

Figure 3: Observation space components. The basic CybORG observation is enhanced with indicators of compromise.

The basic observation vector of CybORG blue agents consists of the first four components from Figure 3. The first bit represents the current mission phase. It is followed by a one-hot encoded vector with subnet-related information: what subnet(s) is the agent protecting, whether the traffic to/from other subnets is blocked, and which interfaces should be blocked based on the current mission phase restrictions. Next, the observation vector contains information on alerts detected with the Monitor action, using a 1/0 binary encoding to denote whether suspicious processes or connection events occurred on a host.

Enhancing observation with memory. Given that a single defense action can be taken per round, blue agents need a persistence mechanism to store alerts that have not been addressed yet. In CybORG, the Monitor action runs automatically at the end of each step but only reports *new* events that have been raised on the current step. This requires us to maintain an updated observation history h_t that keeps track of past events. Therefore, we add new events to the agent's history h_t at each time step given a new observation o_t, and only remove these events from h_t when they have been handled by a respective recover action.

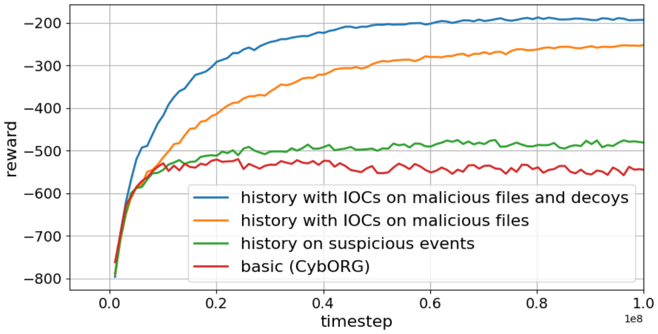

Figure 4: Blue team reward for different observation space designs. We incrementally add each of our enhancements to the observation space, to show their individual contribution. Incorporating history and IOCs provides a high performance boost. (MARL Decentralized training)

Enhancing observation with IOCs. Indicators of compromise (IOCs) (Asiri et al., 2023) are signs or evidence of a cyber threat being present in the network. IOCs are generally classified in three categories: atomic (IP addresses, malware names, registry keys, process names, URLs, etc.), computed (hash of a malicious file), and behavioral (Hutchins et al., 2011). MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) (Strom et al., 2018) is a comprehensive database of adversarial behaviors observed in real-world attacks.

One of our main insights for automating cyber defense is to extend the observation vector with IOC-related information per host, as illustrated in Figure 3. We use two types of atomic IOCs: malicious file names that are placed on the victim machine and the IP address of the compromised host that issues service requests to a decoy service. We also capture adversarial behaviors by prioritizing IOCs based on the attack phase. A value of zero in the observation vector denotes that no IOC has been detected on the corresponding host. The other values differentiate between attack phases: priority 1 for IOCs detected during privilege escalation attempts, such as malicious files with root access; priority 2 for IOCs due to attacker's exploit actions (namely malicious files with user-level access); and priority 3 for IOCs due to the attacker's scanning activity (decoy accesses).

Observation space evaluation. Figure 4 shows the contribution of each of our enhancements to the observation space design for a blue agent trained with a decentralized actor-critic PPO architecture, using the same hyper-parameters from Section 5. Compared to the basic CybORG observation, keeping track of history on suspicious events offers a small performance boost (12% increase in reward). The biggest gain, however, comes from incorporating indicators of compromise related to malicious files (an additional increase in reward of 42%). These files are detected when blue agents perform Analyse actions on hosts in their assigned subnet. Access to decoys, another clear indicator of adversarial behavior, further improves the defense strategy of the blue team by 11%.

5 Experimental Evaluation

We now evaluate our proposed hierarchical MARL architecture in the CybORG CAGE 4 environment against different baselines, aiming to answer the following research questions: (i) How effective is our H-MARL approach in protecting the network compared to other methods, against different adversaries (Sections 5.1 and 5.2)? (ii) Is it feasible to transfer previously trained sub-policies to learn new defense strategies (Section 5.3)? (iii) Can we provide some interpretable insights to security operators related to the performance of our defenses (Section 5.5)?

Training configuration. Our experiments use the state-of-the-art actor-critic PPO algorithm (Schulman et al., 2017). The actor and critic are represented by two feedforward neural networks with two hidden layers and 256 neurons per layer. The training hyperparameters have been tuned to the following values: a learning rate of 5×10^{-5}, a discount factor of 0.99, and a train buffer size of 1 million samples. The SGD algorithm uses the Generalized Advantage Estimation (GAE) function, a

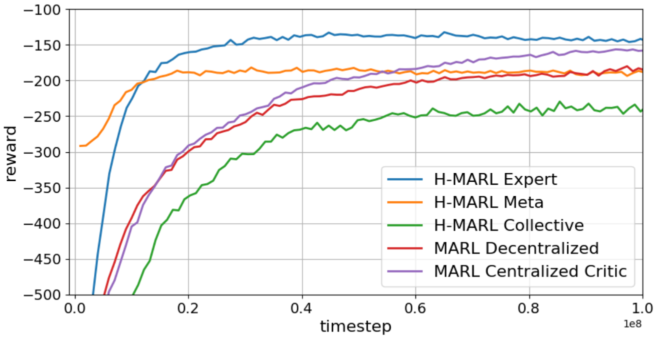

Figure 5: Average training return for all algorithms. H-MARL Expert is guided by a rule-based master policy and performs best. H-MARL Meta converges $3-5\times$ faster than MARL Decentralized.

minibatch size of 32,768 within each epoch, with 30 SGD iterations in each outer loop. Evaluation results are averaged across 100 randomized episodes, where each episode is 500 time steps long, and are accompanied by standard deviation information. The network topology in the CybORG environment is randomized and the models are trained on topologies with varying configurations, which ensures that they generalize to different network environments. The size of the network at initialization is randomly chosen between 32 and 128 hosts across 8 subnets, and up to 10 services are selected randomly to be placed on each host. All our models have been trained on 30 versions of the network, with separate workers collecting the experiences in their own network.

Duration of actions. For increased realism, actions take more than one step to execute. Some of the longer actions are Exploit Remote Services (4 steps) and Restore host (5 steps). In our implementation, while an action is in progress in the environment, we associate the "in-progress" state with a special observation and mask out other actions except sleep to guide PPO training.

5.1 H-MARL Performance

MARL Baselines. We compare the hierarchical architecture with two single-policy MARL paradigms (Lyu et al., 2021): Decentralized Training Decentralized Execution (DTDE), and Centralized Training Decentralized Execution (CTDE). The DTDE baseline uses an actor-critic architecture that learns a decentralized policy and critic for each of the agents locally (*MARL Decentralized*). The CTDE baseline uses the centralized critic approach presented in MADDPG (Lowe et al., 2017). Since each learning blue agent is required to guide its strategy based on the joint team reward, we augment the critic with state and actions of all blue agents on the team, while the actor only has access to local observations. Note that our *MARL Centralized Critic* baseline uses the global state instead of incomplete agent observations to compute the joint value function. This is a reasonable assumption during training, in an effort to provide an unbiased and up-to-date critic for a strong baseline (Lyu et al., 2021).

H-MARL Methods. We evaluate two hierarchical methods, H-MARL Expert and H-MARL Meta. *H-MARL Expert* implements Algorithm 1, where the master policy is replaced with a rule informed by domain knowledge: *If IOCs are detected, Recover; otherwise, Investigate.* The goal is to recover right away to minimize the attacker's damage. *H-MARL Meta* implements Algorithms 1 and 2, following a curriculum style approach (Bengio et al., 2009): the Recover and Investigate sub-policies pre-trained until convergence, are maintained fixed while training the master policy. *H-MARL Collective* is an additional baseline that attempts to learn both the master and sub-policies from scratch, simultaneously. All the hierarchical variants use IPPO, in the decentralized actor-critic framework.

The training results for the blue agents protecting against the default red agent in CAGE 4 are presented in Figure 5. The H-MARL Collective method performs the worst, in line with previous work (Frans et al., 2017) that also observed the sub-optimal performance of updating both sub-policies and the master policy at the same time. Both single-policy methods MARL Decentralized

Table 2: Mean reward against different adversaries, under different learning frameworks. Evaluation is carried out after training has converged. Results are averaged over 100 evaluation episodes, with each episode consisting of 500 time steps.

Opponent	**Default Red**	**Stealthy Red**	**Aggresive Red**	**Impact Red**
MARL Decentralized	-179.8 ± 92.98	-165.8 ± 53.45	-227.93 ± 87.98	-247.15 ± 71.53
MARL Centralized Critic	-245.66 ± 132.98	-217.52 ± 115.06	-255.98 ± 138.70	-332.96 ± 108.13
H-MARL Collective	-237.61 ± 102.6	-204.18 ± 83.24	-282.5 ± 120.50	-350.21 ± 115.25
H-MARL Expert	$\mathbf{-129.53 \pm 44.60}$	$\mathbf{-99.54 \pm 38.28}$	$\mathbf{-118.16 \pm 37.74}$	$\mathbf{-173.17 \pm 64.04}$
H-MARL Meta	-181.62 ± 65.85	-184.76 ± 91.40	-207.66 ± 86.48 &	-278.78 ± 97.76
H-MARL Meta with fine-tuning	-186.24 ± 81.28	-162.51 ± 57.97	n/a	-264.96 ± 81.53

and MARL Centralized Critic converge to a high reward, with a clear advantage for the shared critic method that estimates the return based on joint information.

As expected, the H-MARL Expert performs best (-129.53 reward), given that recovery actions are carried out promptly, before the attack amplifies. H-MARL Meta reaches a similar reward to MARL Decentralized (-181.62). However, training a master policy is significantly faster than training a single policy from scratch (about 3-5 times faster), as we are only tasked with choosing the correct sub-policy, rather than also learning the primitive sub-tasks, i.e., how to recover or investigate hosts. H-MARL Meta invokes the fixed sub-policies that have been pre-trained with H-MARL Expert to take a step though the environment. The ability to *learn* how to combine sub-tasks to solve higher-level tasks effectively with H-MARL Meta is particularly important in situations where defining an expert rule is difficult. We discuss such a scenario in Section 5.4.

5.2 Evaluation against different adversaries

We evaluated so far the performance of blue agents against the default red agent in CAGE 4, but we are interested in how the policy generalizes against other red attacks. We simulate red agent attackers that vary in their ability to circumvent the defense by employing more aggressive scanning, stealthier behavior, or a stronger focus on impacting critical services. In the CybORG environment, red agents use finite state machine transitions to determine what actions to take for each known host. For a detailed description of the default finite state red agent and the transition matrix please see CybORG. We consider four different finite state adversaries: (1) Default Red – equal choice between the two available service discovery actions (stealthy and aggressive), and equal split between the two attack objectives (Impact and Degrade Service); (2) Aggressive Red – always performs aggressive service discovery, a short duration action (1 time step) that has a high chance of being detected by blue agents (0.75); (3) Stealthy Red – stealthy service discovery, characterized by low detection rate (0.25) and long duration (3 time steps); (4) Impact Red – fully committed to impacting the critical OT service, without attempting to degrade other services.

The evaluation results against these four different adversaries are presented in Table 2. Impact Red is the strongest attacker collecting the highest reward against the blue team, in this zero-sum cyber game. Successful impact actions receive the highest reward of -10 during the second and third mission phases, more than any other network compromise. For comparison, users' failed access to degraded services only costs the blue team between -1 and -3. Aggressive Red is the second strongest opponent: the higher chance of being detected by blue agents is offset by the short duration of service discovery actions (1 time step), which enable this attacker to carry out more frequent attempts. Conversely, and somewhat non-intuitively, Stealthy Red poses less of a threat to blue agents due to the longer duration spent in covert scanning, before attempting to exploit.

H-MARL Expert is the best strategy against all four red agents and H-MARL Meta attains similar reward to the single policy baseline method (MARL Decentralized), but training converges significantly faster, as shown in the previous section. We confirm that H-MARL Meta performs better than H-MARL Collective across all adversaries. Interestingly, it is also more effective than the MARL Centralized Critic method. Even though the centralized critic was trained jointly, using the global state, it is unable to utilize the shared information effectively on the individual actors during evaluation. This limitation of centralized critic policies has been previously investigated in the literature Lyu et al. (2021).

5.3 H-MARL Transferability

In this section, we explore the possibility of knowledge transfer (Weinshall et al., 2018; WEI et al., 2018; Zhuang et al., 2020; Nekoei et al., 2021) – given a defense strategy that has already been trained, can we use it to accelerate the learning of a new defense that protects against a different adversary? We adapt the pre-trained sub-policy models to new attacks via fine-tuning, a well-established powerful transfer method for deep models (Zhuang et al., 2020; Sun et al., 2019) that is significantly less costly than training from scratch. In our current design, we only fine-tune the sub-policies for a few iterations. Next, we fully train a new master policy, using the tuned sub-policies.

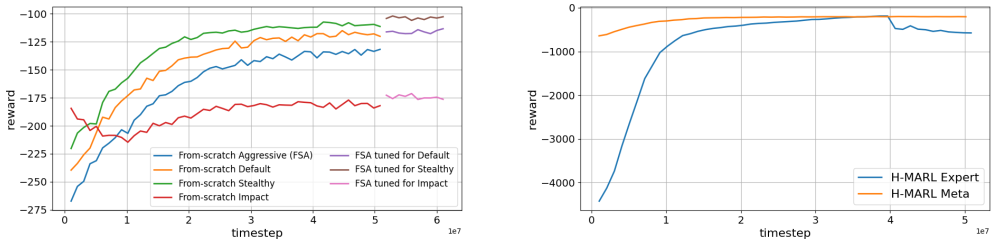

Figure 6: (Left) Investigate sub-policy pre-trained against Aggressive Red is fine-tuned separately against other red agents, performing similarly to the sub-policies trained from scratch against the same red agent.
(Right) H-MARL with 3 sub-policies: Investigate, Recover, and Control Traffic. Due to the probabilistic expert rule, H-MARL Expert method is unstable, while H-MARL Meta performs well, converging fast to a stable performance.

Figure 6 (left) presents the fine-tuning results of the Investigate sub-policie, pre-trained against Aggressive Red, the average-performing attacker. A short fine-tuning is enough to adapt to a new adversary, resulting in learning curves similar to those obtained when training from scratch. The fine-tuning results for the Recover sub-policy are included in Section 2 of the Supplemental Material. We note that Investigate learns different strategies for each adversary (due to different patterns of suspicious events), while Recover is relatively agnostic to the attack (due to working on clear indicators of compromise) and can be directly reused against other red agents.

Last row in Table 2 shows the evaluation results of the master policy that was trained with the fine-tuned sub-policy models. In this case, the master policy performs comparably to (and sometimes even better than) H-MARL Meta that uses sub-policies trained from scratch. For instance, against the Stealthy Red agent, the fine-tuned H-MARL Meta policy attains an average reward of -162.51, compared to an average -184.76 reward when training from scratch. Our empirical results are supported by previous research, which has shown that pre-trained deep learning models have been proven to generalize better than randomly initialized ones (Erhan et al., 2010; Sun et al., 2019).

5.4 What method to use: Expert or Meta?

In our previous experiments, we have seen that H-MARL Expert performs best, as it is guided by a deterministic master policy generated from domain knowledge, specifically: *If host IOCs are present, invoke the Recover sub-policy; otherwise Investigate.* To show that such expert rules do not generalize in all situations, we consider a scenario with a third sub-policy, Control Traffic, in which a defender may choose to block traffic between subnets. However, block actions incur penalties for preventing the remote activity of green users, rendering a deterministic blocking rule unfavorable. We instead experimented with a probabilistic master policy: *If host IOCs are present, invoke the Recover sub-policy; otherwise split 75%-25% between Investigate and Control Traffic.* With this probabilistic rule, the H-MARL Expert method becomes unstable, while H-MARL Meta performs

Table 3: Interpretable metrics for various blue strategies against Default Red, 100-episode averages.

Blue Strategy	**Clean Hosts (ratio)**	**Non-Escalated Hosts (ratio)**	**Mean Time to Recover**	**Useful Recoveries (true positives)**	**Wasted Recoveries (false positives)**	**Recovery Precision**	**Recovery Error**	**Red Impact Count**	**Reward**
H-MARL Expert	0.81	0.99	44.76	6.07	2.2	0.73	0.27	0.88	**-129.53**
H-MARL Meta	0.77	0.97	62.63	5.79	3.74	0.61	0.39	1.14	-181.62
H-MARL Collective	0.85	0.97	35.52	3.41	5.79	0.37	0.63	1.93	-255.56
MARL Decentralized	0.77	0.97	56.08	10.45	27.99	0.27	0.73	1.54	-180

better and converges quickly to a stable performance (Figure 6). This demonstrates the importance of learning the master policy and is discussed in more detail in Section 4 of the Supplemental Material.

5.5 Interpretable Metrics

In cybersecurity applications, the defender has two main objectives: keep the network secure and maintain operational workflows. Total discounted rewards is a natural metric for training and measuring performance in reinforcement learning, however, a single-valued reward provides little interpretable information to security operators. To address this problem, we propose a set of interpretable metrics and analyze their relationship to the existing reward computation. The new metrics provide insight on defense performance from three perspectives – network security, effectiveness of recoveries, and impact on operations – as follows:

- Network Security Posture: *Clean Hosts*: Fraction of hosts with no red presence (from total hosts in the network); *Non-Escalated Hosts*: Fraction of hosts with no red root sessions
- Recovery Metrics: *Mean Time to Recover*: Mean number of consecutive steps spent in a compromised state; *Useful Recoveries*: Recoveries performed on infected machines (true positives, TP); *Wasted Recoveries*: Recoveries performed on clean machines (false positives, FP); *Recovery Error*: Err = FP / (TP + FP); *Recovery Precision*: TP / (TP + FP) = 1-Err;
- Operational Metrics: *Red Impact Count*: Number of times the OT service is impacted becoming unavailable.

We evaluate our blue agent strategies according to these metrics in Table 3. We observe a number of insights that are not evident by comparing policies using solely the reward metric. For instance, MARL Decentralized performs more recoveries than other policies, but its recovery precision is only 0.27. While H-MARL Meta and MARL Decentralized have similar rewards, H-MARL Meta has a much better recovery precision of 0.61. Moreover, red agents are less successful at impacting the critical services with H-MARL Meta, compared to MARL Decentralized. These indicators show that H-MARL Meta is a more effective defense strategy than the single-policy approach (MARL Decentralized), despite having similar reward. H-MARL Expert has the highest recovery precision across all policies, as its expert master policy selects the Recovery sub-policy when IOCs are present on hosts, a strong indication of host compromise. Our analysis demonstrates the need of using the reward signal in conjunction with other metrics that are relevant in the cyber domain. To better align the reward with the defender's goals, one can also incorporate these new metrics in the reward.

6 Conclusion

We propose novel hierarchical multi-agent reinforcement learning (MARL) strategies to train multiple blue agents tasked with protecting a network against red agents. Our H-MARL strategy decomposes cyber defense into multiple sub-tasks, trains sub-policies for each sub-task guided by domain expertise, and finally trains a master policy to coordinate sub-policy selection. We evaluated our proposed hierarchical methods (Expert and Meta) and compared them against standard decentralized and centralized MARL in a realistic cyber security environment, CybORG CAGE 4. We demonstrated that our hierarchical method converges faster than a single PPO policy and generalizes across various red agent behavior, while H-MARL Expert consistently performs better than the baselines.

References

Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, II, and David Dagon. Detecting malware domains at the upper DNS hierarchy. In *Proceedings of the 20th USENIX Conf. on Security*, pp. 27–27. USENIX Association, 2011.

Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: Detecting the rise of DGA-based malware. In *Presented as part of the 21st USENIX Security Symp. (USENIX Security 12)*, pp. 491–506. USENIX Association, 2012.

Mohammed Asiri, Neetesh Saxena, Rigel Gjomemo, and Pete Burnap. Understanding indicators of compromise against cyber-attacks in industrial control systems: A security perspective. *ACM Trans. Cyber-Phys. Syst.*, 7(2), April 2023.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. DOI: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. EXPOSURE: Finding malicious domains using passive dns analysis. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS, 2012.

CAGE. Cage challenge 1. *IJCAI-21 1st International Workshop on Adaptive Cyber Defense.*, 2021.

CAGE-4. TTCP CAGE Working Group. CAGE Challenge 4. https://github.com/cage-challenge/cage-challenge-4, 2023.

Lu Chang, Liang Shan, Weilong Zhang, and Yuewei Dai. Hierarchical multi-robot navigation and formation in unknown environments via deep reinforcement learning and distributed optimization. *Robotics and Computer-Integrated Manufacturing*, 83:102570, 2023.

CybORG. Red overview. URL https://github.com/cage-challenge/cage-challenge-4/blob/main/documentation/docs/pages/reference/agents/red_overview.md.

Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11: 625–660, March 2010. ISSN 1532-4435.

Elliot Fosong, Arrasy Rahman, Ignacio Carlucho, and Stefano V. Albrecht. Learning complex teamwork tasks using a given sub-task decomposition. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, pp. 598–606, Richland, SC, 2024. International Foundation for Autonomous Agents and Multiagent Systems.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies, 2017.

Cong Guan, Ruiqi Xue, Ziqian Zhang, Lihe Li, Yi-Chen Li, Lei Yuan, and Yang Yu. Cost-aware offline safe meta reinforcement learning with robust in-distribution online task adaptation. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, pp. 743–751, Richland, SC, 2024. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9798400704864.

Kim Hammar, Neil Dhir, and Rolf Stadler. Optimal defender strategies for cage-2 using causal modeling and tree search, 07 2024.

T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis; Machine Intelligence*, 44(09):5149–5169, sep 2022. ISSN 1939-3539. DOI: 10.1109/TPAMI.2021.3079209.

Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1:80, 2011.

Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022. ISSN 2504-4990. DOI: 10.3390/make4010009. URL https://www.mdpi.com/2504-4990/4/1/9.

C. David Hylender, Philippe Langlois, Alex Pinto, and Suzanne Widup. Verizon 2024 data breach investigations report. https://www.verizon.com/business/resources/reports/dbir/, 2024.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6382–6393, Red Hook, NY, USA, 2017. Curran Associates Inc.

Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning, 2021.

Garrett Mcdonald, Li Li, and Ranwa Al Mallah. Finding the optimal security policies for autonomous cyber operations with competitive reinforcement learning. *IEEE Access*, 12:120292–120305, 2024. DOI: 10.1109/ACCESS.2024.3446310.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1DmUzWAW.

Hadi Nekoei, Akilesh Badrinaaraayanan, Aaron C. Courville, and Sarath Chandar. Continuous coordination as a realistic scenario for lifelong learning. In Marina Meila and Tong Zhang (eds.), *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8016–8024. PMLR, 2021.

Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. ExecScent: Mining for new c&c domains in live networks with adaptive control protocol templates. In *Proceedings of the 22nd USENIX Conf. on Security*, pp. 589–604, USA, 2013. USENIX Association.

Frans A. Oliehoek and Chris Amato. A concise introduction to decentralized pomdps. In *SpringerBriefs in Intelligent Systems*, 2016. URL https://api.semanticscholar.org/CorpusID:3263887.

Talha Ongun, Oliver Spohngellert, Benjamin A. Miller, Simona Boboila, Alina Oprea, Tina Eliassi-Rad, Jason Hiser, Alastair Nottingham, Jack W. Davidson, and Malathi Veeraraghavan. PORTFILER: port-level network profiling for self-propagating malware detection. In *IEEE Conference on Communications and Network Security, CNS 2021, Tempe, AZ, USA, October 4-6, 2021*, pp. 182–190. IEEE, 2021. DOI: 10.1109/CNS53000.2021.9705045. URL https://doi.org/10.1109/CNS53000.2021.9705045.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

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

Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck: Design and philosophy. In *Technical report*. The MITRE Corporation, 2018.

Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 403–412, 2019.

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

Ying WEI, Yu Zhang, Junzhou Huang, and Qiang Yang. Transfer learning via learning to transfer. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5085–5094. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/wei18a.html.

Daphna Weinshall, Gad Cohen, and Dan Amir. Curriculum learning by transfer learning: Theory and experiments with deep networks. *ICML*, 2018.

Yuchen Xiao, Joshua Hoffman, Tian Xia, and Christopher Amato. Learning multi-robot decentralized macro-action-based policies via a centralized q-net. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10695–10701, 2020. DOI: 10.1109/ICRA40945.2020.9196684.

Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *ACSAC*, pp. 199–208, 2013.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020. URL https://arxiv.org/abs/1911.02685.

Supplementary Materials

The following content was not necessarily subject to peer review.

7 The Cyber Game

This section presents additional details on the cyber game simulated in CybORG CAGE 4 [1]. For increased realism, users on enterprise machines are represented by green agents, which are present on every host. Green agents randomly access local and remote services, such as SSHD, MySQL, FTP, etc. Upon compromise, red agents are able to *degrade* these services, preventing users from completing their work or slowing them down. The red team can attack the network through several actions: scan the network to discover new hosts; scan a host to discover active services; exploit a vulnerability to compromise a host; escalate privileges on a compromised node to gain root access; degrade user experience of green agents; impact (stop) the critical Operational technology (OT) service; discover deception (i.e., probe a host to determine if it is running decoy services).

The action sets for green agents (users), blue agents (defenders) and red agents (attackers) are presented in Table 4. Both blue and red agent actions have a specified time duration, ranging from one time step (block/allow traffic) to five time steps for longer actions (restore host). Agents are not allowed to choose another action before the previous action had completed.

The red agent in CAGE 4 is rather sophisticated and randomized, not deterministic, and is modeled by:

- State transition diagram: The adversary follows a state transition diagram with eight states and nine transitions (that is, actions) that defines the allowed subset of actions based on the state of the host.
- State transition probability matrix: The adversary's actions are randomized, using a state transition probability matrix to choose among the subset of possible actions in each state. For example, in State S (i.e., a Service was discovered on a host), the Red agent can choose to Discover Remote Systems with 0.25 probability, Discover Deception with 0.25 probability, or Exploit Remote Service with 0.5 probability.

We vary the probabilities in the state transition probability matrix to create additional red agent variants and cover a wider range of Red behaviors (fully Aggressive, Stealthy, or Impact). These new attack vectors explore how two important characteristics of any cyber attack, namely the speed/stealth of discovering vulnerabilities, and the attacker's objective affect the attack success.

8 Communication and Cooperation

In this section, we explore other possible extensions that can help the defense strategy, focusing on communication among agents. In the CybORG CAGE 4 environment, blue agents are facing a challenging adversary, who can move through the network in two ways: (1) phishing emails, and (2) active scanning of hosts and services. Each red agent conducts scanning activity mostly within its own assigned subnet(s), and rarely reaches remotely into other subnets. This partitioning is useful from a scaling perspective, to limit the observation and action spaces. However, it also leads to limited compromise attempts that cross subnet boundaries. Thus, each defender can focus its efforts on its own assigned subnet(s), requiring little communication or coordination with other blue agents. Still, communication can be useful in other game settings, to send information about network-level indicators of compromise, such as malicious file names, the hash of a malicious file, or a compromised IP.

As a case study, we implemented a red agent that chooses external scanning in 50% of the time (once it becomes aware of another subnet), and a blue agent that relies heavily on decoys (90% of blue

[1] https://github.com/cage-challenge/cage-challenge-4/tree/main

Table 4: Action sets for green agents (users), blue agents (defenders), and red agents (attackers).

Green actions	
Green Access Service	Communicate with a server in the local zone or another zone. It has a small chance of being flagged as malicious by Monitor.
Green Local Work	Do work on the local host without communicating. It has a small chance that this action results in red gaining a foothold on the host (e.g. as a result of a Phishing email). Also, it has a small chance of being flagged as malicious by Monitor.
Blue actions	
Monitor	Collection of information about flagged malicious activity on the network.
Analyse	Collection of further information relating to malware files, on a specific host.
Deploy Decoy	Setup of a decoy service (type randomly assigned) on a specified host. Blue is alerted if Red accesses this service.
Remove	Attempts to stop all processes identified as malicious by the monitor action.
Restore	Restoring a system to a known good state.
Block Traffic	Block traffic to and from the specified zone.
Allow Traffic	Allow traffic to and from the specified zone.
Red actions	
Discover Remote Systems	Discovers new hosts/IP addresses in the network through active scanning using tools such as ping.
Aggressive Service Discovery	Discovers responsive services on a selected host by initiating a connection with that host.
Stealth Service Discovery	Same as Aggressive Service Discovery but slower and quieter (i.e., lower chance of raising alerts).
Exploit Network Services	Attempts to exploit a specified service on a remote system.
Privilege Escalate	This action escalates the agent's privilege on the host.
Impact	This action disrupts the performance of the network and fulfils red's objective of denying the operational service.
Degrade Services	If red has root privileges on a host, it may degrade the user experience for a green agent.
Discover Deception	Probe a host to determine if it is running decoy services.
Withdraw	Remove red presence from target host.

actions) to detect the adversary during the scanning phase of the attack. Blue agents broadcast 8-bit messages encoding which remote host is accessing their decoys to warn other agents of potential attackers. We uniquely identify the compromised hosts with 3 bits for the subnet number (1-7), and 4 bits for the host index (0-15). Each blue agent decodes the message to check if it refers to hosts from its own subnet. If so, the message provides an indicator of compromise that will be added to the observation vector. Figure 7 shows that there is some benefit of using this method of communication. However, the benefit is small, due to other factors, such as phishing, stealth, and false positives of the Monitor actions.

9 Traffic Control

Our next case study explores the use of Block and Allow Traffic actions to control the access between security zones. The H-MARL architecture consist of a Master policy and three sub-policies: Investigate, Recover, and Control Traffic. We extended the observation space with network-level indicators of compromise – blue agents communicate whether their assigned subnet(s) contain any IOCs – enabling each agent to have a global view on the network, and facilitating the training of the Control Traffic sub-policy.

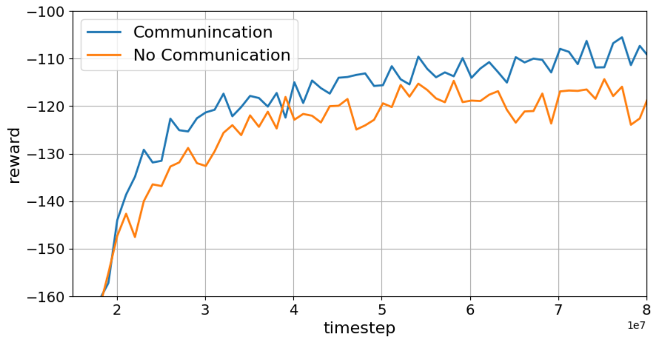

Figure 7: Blue agents use 8-bit messages to warn other team members of potential compromised hosts. This communication strategy shows some benefit over the case when no communication is used. (MARL Decentralized training)

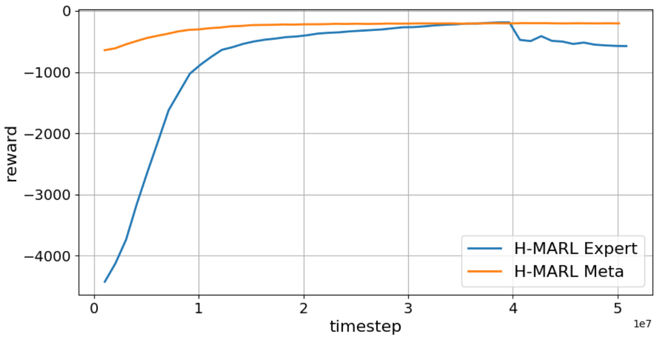

Figure 8: H-MARL with 3 sub-policies: Investigate, Recover, and Control Traffic. Due to the probabilistic expert rule, H-MARL Expert method is unstable, while H-MARL Meta performs well, converging fast to a stable performance.

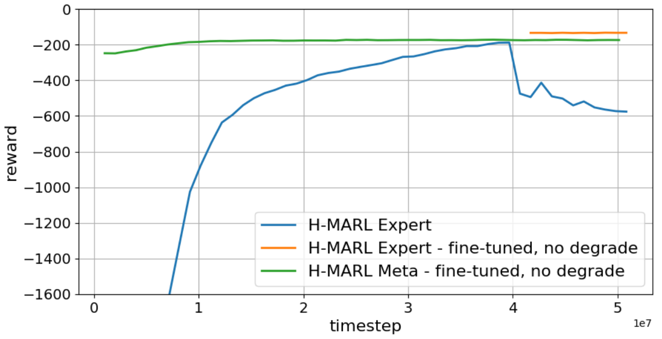

Figure 9: H-MARL with 3 sub-policies: Investigate, Recover, and Control Traffic, after removing the penalty on failed user access (no degrade). All sub-policies are fine-tuned in this new context, against a new red that attempts frequent remote exploits into other subnets. The H-MARL Expert has regained a stable high performance and H-MARL Meta has similar performance.

Defining an expert knowledge to guide the training of a Control Traffic sub-policy to near-optimal is particularly difficult, due to the conflicting outcomes of using Block actions – stop red agents from moving through the network, but at the expense of preventing user agents from completing their work.

For our current experiments, we use the following expert rule to train the sub-policies (Algorithm 1), and leave further research into other expert rules for future work: *If indicators of compromise are present on hosts, call the Recover sub-policy; otherwise, randomly choose Investigate for 75% of the time, and Control Traffic for the remaining 25%.* Thus, the Investigate sub-policy is assigned more weight, as we expect it to be useful more frequently. In fact, for best performance, we use (keep fixed) the Investigate and Recover sub-policies trained previously (see paper), since they have been learned with well-defined expert knowledge, and only train the Control Traffic sub-policy, using the rule specified above.

Next, we follow Algorithm 2 to train the Master policy using the three pre-trained sub-policies. Figure 8 shows the reward as training progresses during Algorithm 1 (using the expert rule), and Algorithm 2 (train the master). We observe the instability of the expert, which has been trained with a probability-based rule, reinforcing the importance of learning the master policy. The H-MARL Meta algorithm is more stable, as the master policy learns how to combine the sub-policies to solve the meta-task, and is not restricted by a fixed, deterministic rule.

Turning off Degrade Service. In our next set of experiments illustrated in Figure 9, we fine-tuned the sub-policies after turning off the reward penalty of green agents being affected by failed service access. We also used the modified red agent introduced in Section 8 of the supplemental material, which performs remote scanning into other subnets more often, and can still collect rewards by impacting the critical OT service. With block actions being now useful at preventing red agents from spreading, without incurring penalties, H-MARL Expert regains a stable, high performance, as expected. H-MARL Meta achieves similar performance with H-MARL Expert in this setting. The blue agents are using $4\times$ more block actions when the failed user access penalty has been removed.

10 H-MARL Transferability

Figure 10 presents the fine-tuning results of the Recover sub-policy, pre-trained against Aggressive Red, the average-performing attacker. The Recover sub-policy is trained on an observation space consisting of indicators of compromise within a subnet. This sub-policy learns a strong strategy regardless of the attack, and can be directly reused against other red agents.

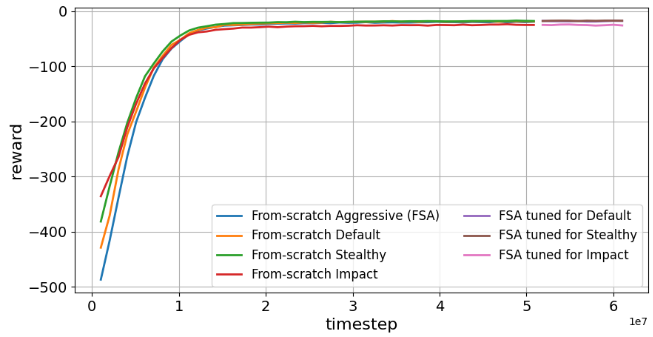

Figure 10: The Recover sub-task is rather agnostic to the attack type and can be re-used against other adversaries.