Master Thesis Project

University of Groningen ‘& TNO

Collaborative Reinforcement Learning for Cyber Defense: Analysis of Environments, Strategies, and Policies

First Supervisor:
Dr. F. Turkmen

Second Supervisor:
R. F. Cunha

External Supervisors:
Ir. F. Fransen
P. de Haan

Author:
Davide Rigoni *(s3570215)*

November 21, 2024

rijksuniversiteit
groningen

Abstract

As cybersecurity threats grow in scale and sophistication, traditional defenses increasingly struggle to detect and counter such attacks. Recent advancements leverage reinforcement learning to develop adaptive defensive agents, yet challenges remain, particularly around how agents learn, in what environments, and what strategies they acquire. These complexities intensify in multi-agent scenarios, where coordinating collaborative defenses becomes especially demanding. This study provides a comprehensive analysis of collaborative reinforcement learning for cybersecurity defense, examining key components such as environment models, RL methods, and agent policies. Initially, various open-source environments are assessed for usability and flexibility to identify the most suitable one for experiments. The chosen CybORG environment is then tested with a range of multi-agent reinforcement learning approaches to determine optimal defense configurations. Finally, the study evaluates the policies learned by the agents to assess their applicability to real-world scenarios, highlighting areas where agent strategies may diverge from effective, practical defense.

Contents

List of Figures

List of Tables

1 Introduction

In today's digital landscape, the frequency and sophistication of cyberattacks are rising at an unprecedented rate, putting organizations and individuals alike at constant risk [74]. From data breaches to ransomware, the complexity of these threats often outpaces traditional cybersecurity defenses, leaving systems vulnerable to exploitation. This reflects a steady increase in recorded cyber attack incidents as highlighted by the website *hackmageddon* [67] which reports a 35% increase of recorded cyber attacks in 2023 compared to 2022. The rise in security threats is a widespread issue affecting most countries, leading to a sharp increase in both cybersecurity losses from external attacks and the costs associated with preventive measures aimed at mitigating these threats [95].

As cybersecurity threats continue to grow in both volume and complexity, traditional approaches to detecting and mitigating these attacks are increasingly facing significant challenges. However, advancements in technology, particularly in machine learning, have enabled improvements in various cybersecurity practices.

Techniques such as deep learning and reinforcement learning have been successfully employed in cybersecurity mechanisms [5], reinforcement learning has garnered considerable attention for automating cybersecurity defense mechanisms, a trend reflected in the growing body of scientific research on the subject [4]. In this context, reinforcement learning has proven highly effective in both defending systems and testing cyberdefense strategies.

With the growth in research, it is important to study how reinforcement learning can be applied to cybersecurity, identifying the best methods and practices, and determining whether it is feasible for defense agents to learn policies that are applicable in real-world scenarios. This is particularly relevant for complex environments that closely mimic real-life conditions, involving multiple agents working collaboratively toward the shared goal of defending an environment. Based on these foundational concepts, the following three research questions will be explored in this study:

- **Research Question 1:** Which open-source reinforcement learning environments for cybersecurity offer the best resources for training and deploying diverse agents?
- **Research Question 2:** Which reinforcement learning methods are the most effective for the chosen baseline environment?
- **Research Question 3:** How can collaborative reinforcement learning policies be applied to defend against intrusion in the selected environment?

The first research question involves a literature review of available open-source reinforcement learning environments for cybersecurity, providing researchers with a reference for selecting the most appropriate environment for their use case. This section will also focus on identifying the best environment for a multi-agent scenario and will be covered in Section 3. The second research question will delve into the results of applying various reinforcement learning methods, aiming to identify the most effective approaches in the chosen environment. The third question will evaluate the policies learned by the best-performing agents and examine whether these policies can be adapted for real-world application. Section 4 will detail the methodology used to answer the second and third research questions, with results presented in Section 5 and discussed in Section 6. Additionally, Section 2 will provide background analysis on multi-agent reinforcement learning, while Section 8 will present the study's conclusions.

2 Background

In this section, we will provide an overview of the main topics relevant to this study, with a particular focus on multi-agent reinforcement learning and an analysis of the current state of the art in applying collaborative reinforcement learning within cybersecurity.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning where an agent interacts with an environment, aiming to maximize cumulative reward through a process of trial and error [91]. As illustrated in Figure 1, the agent takes an actionA_t in the environment, which leads to a transition, resulting in a new state S_{t+1} and a reward R_{t+1}.

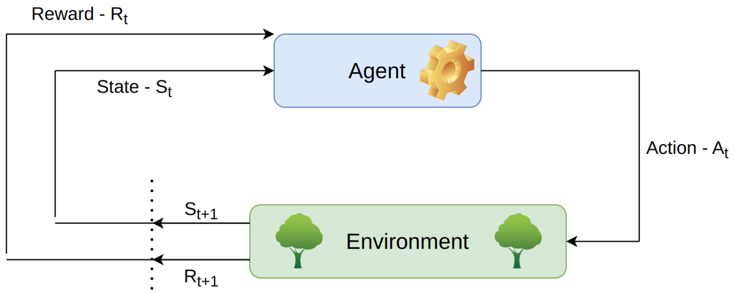


Figure 1: Connection between agent and environment in reinforcement learning

This concept of reward maximization is central to reinforcement learning and is formalized by the *reward hypothesis*, which states that all objectives can be framed as the maximization of expected cumulative reward. Consequently, for an agent to learn optimal behavior in a given environment, it must attempt to maximize its cumulative reward over time. A key challenge in reinforcement learning is the *exploration vs. exploitation* dilemma. Exploration involves seeking additional information about the environment to discover the best actions in specific states, while exploitation focuses on using known information to maximize cumulative reward. Striking a balance between these two strategies is fundamental to effective learning in RL. Several core concepts in reinforcement learning guide the agent's learning process:

- **Policy**: The policy defines a mapping from states to actions, guiding the agent on how to act at each step.
- **Value Function**: This function estimates the expected future rewards, enabling the agent to compare actions and select the one that yields the highest expected reward.
- **Model**: The model provides a formalized description of the environment, aiding the agent in planning and prediction. The two primary models in reinforcement learning are the transition model, which predicts the next state, and the reward model, which estimates the next reward.

These components work together to help the agent learn from its interactions and optimize its decision-making in pursuit of its objective.

2.1.1 Markov Decision Process (MDP)

The Markov Decision Process (MDP) is a foundational concept in reinforcement learning, providing a formal framework for environments in which each state satisfies the Markov Property. This property implies that the agent only requires information from the current state to make a decision, without needing the full history. An MDP is described by the tuple $< S, A, P, R, \gamma >$, where:

- S is a finite set of states
- A is a finite set of actions
- P is a state transition probability matrix, defined in the following way: $P^a_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a]$, which describes the probability to end up in state s' from state s if the agent does action a.
- R is the reward function defined in the following way: $R^a_s = E[R_{t+1} | S_t = s, A_t = a]$, which tells us what is the reward from doing action a in state s.
- γ is the discount faction, $\gamma \in [0, 1]$. This tells the agent how much it should care about future rewards compared to immediate ones. With a discount factor of 0, the agent does not take into consideration future rewards, instead the more γ is close to 1 the more future rewards are taken into account.

This tuple represents a fully observable and finite MDP, where the agent has access to all states and operates within a finite set of actions and states. However, some MDPs may be either infinite or only partially observable.

In the case of an infinite MDP, the model must account for an infinite set of states and action spaces. When dealing with partial observability, we use a Partially Observable MDP (POMDP), where certain states are hidden from the agent.

A POMDP is represented by the tuple $< S, A, P, R, O, Z, \gamma >$, which expands upon the fully observable MDP by including O, a finite set of observations, and Z, an observation function that relates observations to hidden states.

2.1.2 Deep Reinforcement Learning

Deep Learning is a branch of machine learning characterized by the use of multiple deep layers of neural networks [37], enabling the model to learn complex features and patterns from the input data. Neural networks consist of two exposed layers, the input layer, which receives the data, and the output layer, which produces the final output, along with several hidden layers in between. Each layer contains neurons, with each neuron having its own *weight* and *bias* terms. During training, the weights are adjusted to guide the model toward the desired output. The output of one neuron becomes the input for the next, allowing information to flow through the network. A simple representation of a neural network is shown in Figure 2.

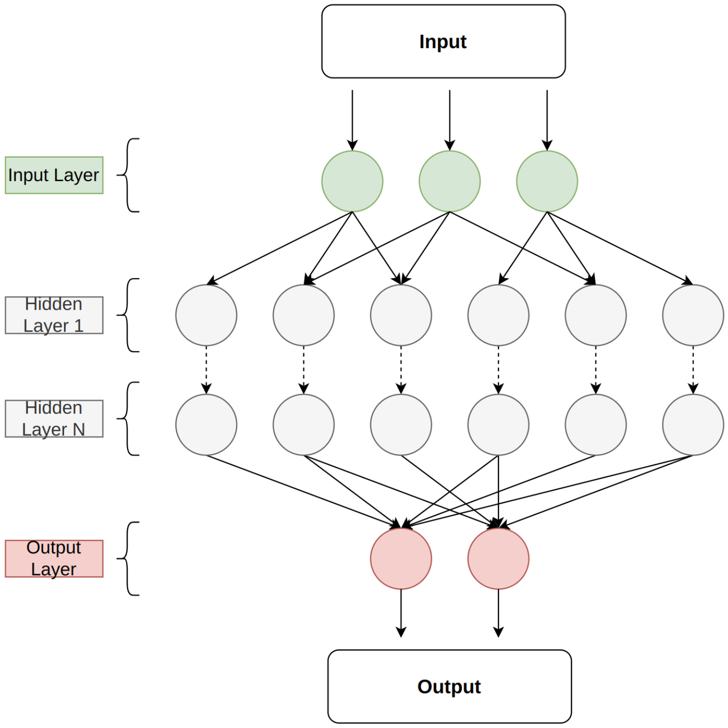


Figure 2: Neural Network example representation

Some more key components of neural networks include the *activation function*, which calculates the node's output based on previous inputs and weights, and the *cost function*, which evaluates the network's performance after training. Deep learning techniques are particularly useful in addressing the "curse of dimensionality," which describes the challenges machine learning models face when processing and understanding high-dimensional data [12].

In the context of reinforcement learning different deep neural networks can be used to approximate many different components, such as the policy, the model, or the value function [53]. The most commonly used neural network in this field consists in **Autoencoders**, mostly used for dimensionality reduction and feature extraction [32], **Recurrent Neural Networks**, characterized by a recursive implementation of the network, and **Convolutional Neural Networks**, characterized by convolutional layers used to process image data.

2.2 Multi-Agent Reinforcement Learning

This section explores Multi-Agent Reinforcement Learning (MARL) in contrast to the single-agent case, drawing primarily from Albrecht et al. [7]. A significant advantage of a multi-agent system is its robustness and scalability: the system can continue functioning even if one agent fails, and new agents can be added as needed without disrupting the overall structure [20]. For intrusion prevention scenarios, a multi-agent setup is especially relevant, as it introduces dynamic behavior across all actors in the system, since intrusion prevention typically involves multiple attackers and defenders, having multiple agents in a network simulation enhances its realism. In a fully **cooperative** configuration, agents work together to maximize a shared reward, collectively striving toward the system's goals. Conversely, in a **competitive** configuration, agents compete against each other. Many systems incorporate a mixed approach, where groups of agents cooperate internally but compete against other groups or individual agents. Figure 3 illustrates an example of such a setting, with multiple attacker and defender agents attempting to access a specific server's data.

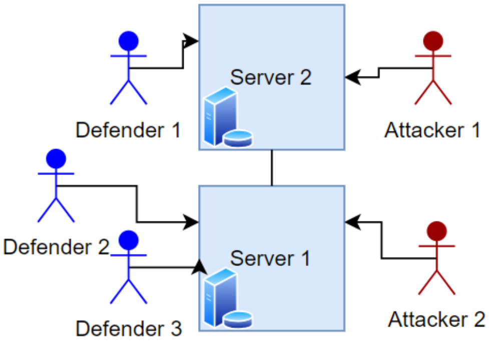


Figure 3: Mixed setting in multi-agent reinforcement learning for the defense and attack of a server

This example also indirectly poses the question of one of the main paradigms in multi-agent reinforcement learning: the training approach for these agents. In a setup where each defender agent learns its policy based on its own local observations, actions, and rewards—ignoring other agents' existence—we have **independent learning**. Alternatively, **centralized learning** involves a single, central policy that receives each agent's local observations, selects joint actions, and allocates actions to each agent. Each of these approaches has specific advantages and challenges that MARL systems share to varying degrees:

- **Non-Stationarity**: Since each agent's policy continuously changes, the environment is highly non-stationary, which makes learning more unstable and variable. This challenge is especially significant in independent learning settings where there is minimal coordination.

- **Action-Reward Attribution**: When using a global reward schema, it can be difficult to identify which agent's actions led to a particular reward. This issue is amplified in systems with independent agents where action-reward attribution is unclear.

- **Scalability**: In centralized settings, as more agents join, the communication and the joint action-reward space expand, which can make learning increasingly complex.

- **Policy Optimality**: Unlike in single-agent reinforcement learning, where a policy is optimal if it maximizes reward, in MARL, an agent's reward depends on others' actions. Equilibrium theory, including Nash Equilibrium for competitive, zero-sum games, aims to address this by finding balance points. However, computing these equilibrium solutions can be computationally intensive.

Each of these challenges illustrates the complexities and trade-offs involved in designing effective MARL systems.

2.2.1 Dec-POMDP

In previous sections, we introduced the concepts of MDP and POMDP as models to address single-agent reinforcement learning problems. For multi-agent reinforcement learning, however, we use the Dec-POMDP framework, defined by the following tuple, $< N, S_i, A_i, P, R, O, Z, \gamma >$ which includes additional elements specifically for multi-agent settings:

- N: the set of all agents
- A_i is the joint action set, the set of all possible combined actions for the agents
- S_i is the joint observation set, the set of all possible combined observations for the agents

This paper will adopt the Dec-POMDP framework to approach the reinforcement learning problem in a multi-agent context. Further details on this framework and its application to our study are discussed in Section 3.5.

2.3 Reinforcement Learning Methods

In this section, we will outline the reinforcement learning methods used in this project, detailing their specifications and our implementation.

2.3.1 PPO

Proximal Policy Optimization (PPO), introduced by Schulman et al. [78], is a reinforcement learning method developed as an alternative to Trust Region Policy Optimization (TRPO) with improved sample efficiency. PPO is a policy-based method, meaning the agent's policy is expressed as a probability distribution over all possible actions. It uses an Actor-Critic structure, where the *Actor Network* selects actions and the *Critic Network* evaluates the returns of those actions. The most commonly used variant of PPO, which is also employed in this paper, is `Clipped-PPO`. In this approach, the policy update is constrained within a clipped region during each update. This clipping mechanism ensures stable training by preventing large changes in the policy during updates, which in turn enhances sample efficiency. The formula for the policy update, as defined in the OpenAI PPO implementation, is shown in Equation 1.

$$\theta_{k+1} = argmax_{\theta} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi\theta_k}(s_t, a_t), g(\epsilon, A^{\pi\theta_k}(s_t, a_t))) \tag{1}$$

In this formula, the term $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$ represents the ratio of the log probabilities between the action selected during training and the action evaluated under the current policy. The final update is determined by taking the smaller value between the product of this ratio and the advantage estimates, and the clipped ratio (with clipping threshold ϵ). Moreover, Generalized Advantage Estimation (GAE), proposed by Schulman et al. [77], is used to compute the advantage A, with the formula shown in Equation 2.

$$A_t^{\gamma,\lambda} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{2}$$

In this equation, λ is a parameter that adjusts the weighting of future rewards. Higher values of λ emphasize long-term rewards, while lower values prioritize immediate rewards. Additionally, an entropy bonus is added to the actor loss to encourage exploration. This bonus is computed from the policy's current distribution over visited states, and its average value is included in the policy loss. For the Critic Network, the update equation is shown in Equation 3, where the difference between the estimated returns from the Critic and the actual rewards obtained from the environment is used to update the network.

$$\phi_{k+1} = argmin_{\phi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - R_t)^2 \tag{3}$$

Both the actor and critic losses are propagated through their respective networks using the Adam optimizer [51], with a learning rate that is annealed throughout training to stabilize the process. The implementation supports both linear and exponential annealing of the learning rate. Gradient clipping is also applied to further stabilize the backpropagation process. This update process is repeated multiple times per training epoch or until the difference between the old and new policies (measured by the Kullback–Leibler (KL) divergence) exceeds a predefined threshold. KL divergence quantifies the difference between two probability distributions—in this case, the old and new policy distributions. If this divergence exceeds the threshold, the training episode is terminated early. The entire PPO implementation, excluding specific formulas, is summarized in Pseudocode 1. Lastly, it is important to note that for both network we used a structure comprised of two hidden layers of 256 neurons each followed by a ReLU activation function in between the respective input and output layers. The implementation of PPO used here follows a decentralized training and decentralized execution framework, where each agent is trained independently. As such, we refer to this implementation as Independent PPO (**IPPO**).

Algorithm 1 Baseline PPO implementation

```
Ensure: Initialize RL environment, Policy Network $\theta$ and Value Function network $\pi$ parameters
  for episode=1:MAX_Episodes do
    for step=1:Max_Steps do
      for all $agents \in AgentsSet$ do
        Get observation $s_t$
        Choose action $a_t$ based on current policy $\pi_k = \pi(\theta_k)$ and observation $s_t$
        Compute state value $V_t$ from Critic Network
        Save $D_k = (a_t, s_t, d_t, V_t)$, where $d_t$ is the termination flag of the episode
      end for
    end for
    if episode % Rollout then
      for idx=1:epochs do
        Retrieve episode data $D$
        Compute GAE $A_t$ based on Equation 2 and Reward-to-go $R_t$
        Compute actor loss $a_l$ based on Equation 1
        Compute critic loss $c_l$ based on Equation 3
        Use $a_l$ and $c_l$ for network updates
        Compute KL Divergence $kl$
        if $kl \geq target\_kl$ then
          End training
        end if
      end for
      Anneal Learning Rate
      Reset episode memory $D$
    end if
  end for
```

2.3.2 MAPPO

As previously mentioned, IPPO follows a decentralized training and decentralized execution approach. While it can be adapted to multi-agent environments, it was primarily designed for single-agent systems. For those use cases, a multi-agent version of PPO called Multi-Agent PPO (MAPPO) can be used instead. In MAPPO, execution remains decentralized, but the training process is centralized. This is achieved by using a centralized Critic network, rather than having each agent maintain its own Critic. The centralized Critic takes in the global state observations from all agents and outputs a shared global value function, enabling more coordinated decision-making among agents, as they are not solely reliant on their individual observations and returns.

Aside from the change in state representation, the other implementation details remain the same as in the Independent PPO version, so we will not repeat those here.

2.3.3 QMIX

QMIX is a value-based reinforcement learning method introduced by Rashid et al. [70] as an improvement over the Value Decomposition Networks (VDN) approach [90]. Similar to MAPPO, QMIX operates under the principle of Centralized Training with Decentralized Execution. The main goal of the method is to enforce monotonicity between the global and individual action-value functions of agents, meaning that an increase in an agent's action-value will also result in an increase in the global action-value function. This guarantees that improvements in an individual agent's local policy will lead to an overall improvement in the global policy. To achieve this, QMIX employs a **mixing network** that takes the individual Q-values (action-values) of each agent and outputs a global Q-value. The monotonicity constraint is enforced by using a separate *hypernetwork*, which generates positive weights for the mixing network based on the global state inputs. In our implementation, the mixing network consists of two sequential layers each including a linear transformation of size 256 followed by a ReLU activation function and Dropout regularization with a probability of 0.2. The bias terms of the network are computed via two separate linear layers. Each agent also has its own network that processes the agent's observations and actions to output its Q-value at each timestep. The agent network consists of one hidden layer of 256 neurons in between the input and output layers, all followed by a ReLU activation function. QMIX is an off-policy algorithm, meaning it uses a replay buffer to store past data, which is then sampled during training. This enhances sample efficiency since episodes can be used for multiple training rounds. However, this also raises the risk of outdated data in the replay buffer. Regarding exploration strategies, QMIX typically employs the ϵ-greedy strategy. In this approach, agents have a probability of ϵ to choose a random action (exploration) and a probability of $1-\epsilon$ to select the action with the highest estimated return (exploitation) in the action set, as shown in Equation 4. The pseudocode for QMIX is illustrated in Algorithm 2.

$$action = max_{a \in A} Q_t(a) \tag{4}$$

The loss function for the method is shown in Equation 2.3.3, where γ is the discount factor, b is the batch size, and θ^- are the target network parameters. τ and u represent the joint observation and action history fed into the mixing network and its target counterpart.

$$\begin{aligned} y_{tot} &= r + \gamma max_{u'}(\tau', u', s'; \theta^-) \\ L(\theta) &= \sum_{i=1}^{b} [(y_i^{tot} - Q_{tot}(\tau, u, s; \theta))^2] \end{aligned} \tag{5}$$

2.3.4 MADDPG

The Multi-agent Deep Deterministic Policy Gradient (MADDPG) is a reinforcement learning method introduced by Lowe et al. [55], designed as a multi-agent extension of the Deep Deterministic Policy Gradient (DDPG) method. Like PPO, MADDPG uses an actor-critic framework, but its key distinction lies in its off-policy approach, where it aims to improve a target policy while utilizing a replay buffer to store past state transitions. Since it is multi-agent, MADDPG follows a centralized training and decentralized execution strategy, similar to MAPPO. However, unlike MAPPO, in MADDPG each agent has its own critic that receives the global observations and actions of all agents as input. In our implementation, both actor and critic networks have one hidden layer of size 256 in between the respective input and output layers, each followed by ReLU activation functions. One of the challenges with MADDPG is that, as an extension of DDPG, it was originally designed for continuous action spaces. CybORG, on the other hand, operates in a discrete action space, which may seem incompatible. However, Lowe et al. [55] propose a solution for discrete environments by approximating discrete actions using the Gumbel-Softmax (GS) estimator. In this context, the work by Tilbury et al. [96] is particularly relevant, as they explore various softmax operators for MADDPG. They find that for many test environments, the Gapped Straight-Through (GST) estimator performs better than the GS estimator and other tested softmax methods. The GST estimator, introduced by Fan et al. [33], reduces variance without adding significant computational overhead, making it a preferable choice for our

Algorithm 2 Baseline QMIX implementation

```
Ensure: Initialize RL environment, Replay Buffer D, $Q_i$ and $Q_tot$ with parameters $\theta$ as well as target
    networks with parameters $\theta^- = \theta$
  for episode=1:MAX_Episodes do
    for step=1:Max_Steps do
      for all $agents \in AgentsSet$ do
        Get observation $s_t$
        Choose action $a_t$ based on Equations 4 or 9
        Save in $D$ $(a_t, s_t, d_t, V_t)$, where $d_t$ is the termination flag of the episode
      end for
    end for
    if len(D) $\geq$ batch size then
      Select batch $b$ from $D$
      Compute loss $l$ based on Equation 2.3.3
      Update network parameters $\theta$ by minimizing $l$
      Anneal Learning Rate
      Anneal $\epsilon$ or $\tau$ based on the chosen exploration strategy
    end if
  end for
```

project. The MADDPG algorithm is presented in Pseudocode 3, with the Actor and Critic loss functions described in Equations 6 and 7, respectively. For the Critic loss, the algorithm minimizes the error between the Q-values predicted by the target critic network, which is used to compute the target actions and their returns, and the Q-values based on the current policy's action choices and state transitions.

Algorithm 3 Baseline MADDPG implementation

```
Ensure: Initialize RL environment, Replay Buffer D, Policy Network $\theta$ and Value Function network $\pi$
    parameters, as well as target networks with parameters $\theta^- = \theta$ and $\pi = -\pi$
  for episode=1:MAX_Episodes do
    for step=1:MAX_Steps do
      for all $agents \in AgentsSet$ do
        Get observation $s_t$
        Choose action $a_t$ based on current policy and GST estimator
        Save in $D$ $(a_t, s_t, d_t)$, where $d_t$ is the termination flag of the episode
      end for
    end for
    if len(D) $\geq$ batch size then
      Select batch $b$ from $D$
      Compute Critic Loss $l_c$ based on Equation 6
      Compute Actor Loss $l_a$ based on Equation 7
      Perform network update for each agent based on Equation 8
      Anneal Learning Rate
    end if
  end for
```

$$L(\theta_i) = \frac{1}{S} \sum_j ((r_i^j + \gamma Q_i^{\mu'}(x'_j, a'_1 ... a'_N)) - Q_i^{\mu}(x_j, a_1 .... a_N))^2 \tag{6}$$

For the Actor loss, a gradient estimator ∇ is used to compare the actions selected under the old policy to those chosen by the current policy, with the critic serving as an evaluator of these actions. This process is performed for each episode in the sampled batch j in the batch set S, with μ representing the agent's policy. Thus, each action in the equation results from querying the policy μ with a specific state s^i based on the policy parameter θ. The term $\nabla_{a_i} Q_i^{\mu}(x_j, a_1....a_N)$ indicates the critic network evaluation of the joint action set, as it is performed for the critic loss.

$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum_j \nabla_{\theta_i} \mu_i(s_i^j) \nabla_{a_i} Q_i^{\mu}(x_j, a_1....a_N) \tag{7}$$

Finally, the policy update is carried out as shown in Equation 8, with τ representing a hyperparameter that smooths the policy updates over time.

$$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_{i'} \tag{8}$$

2.4 Related Works

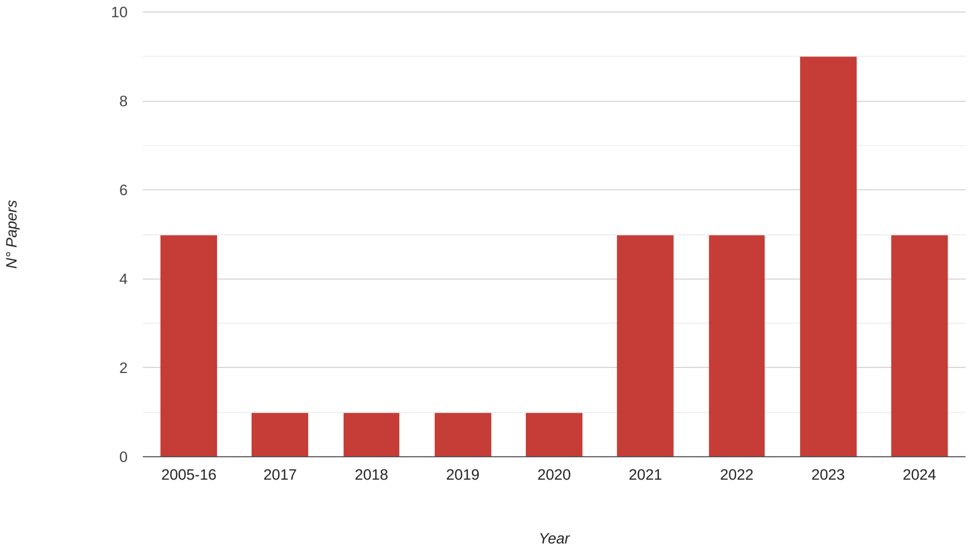


Figure 4: Number of papers published per year on collaborative reinforcement learning in the field of cybersecurity

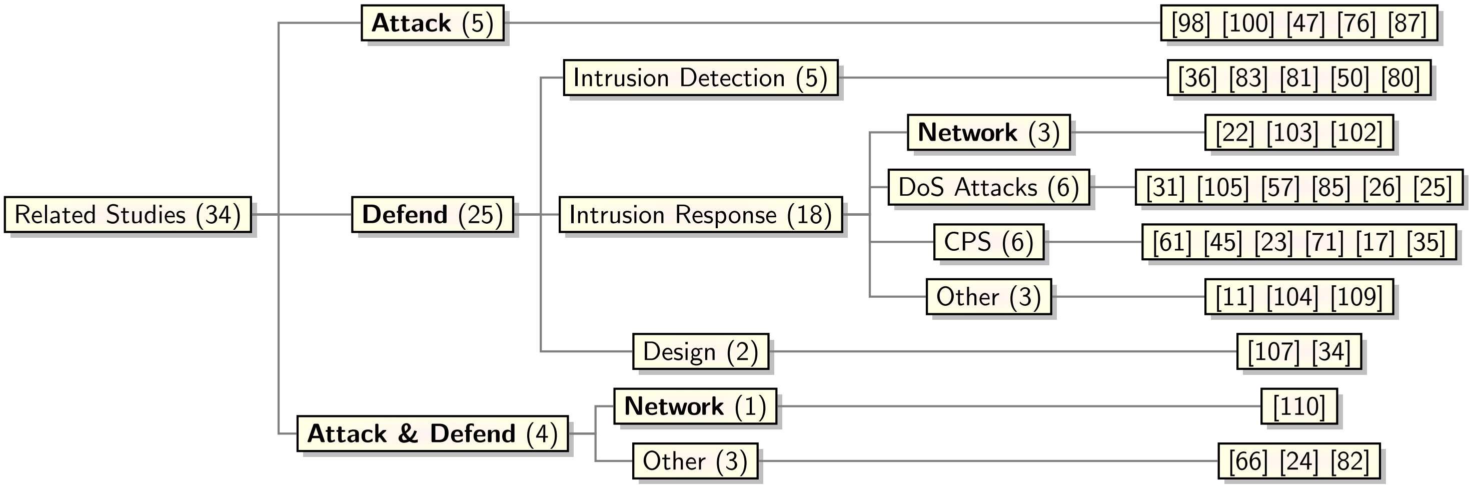
In this section, we will explore the current literature on collaborative multi-agent reinforcement learning in the field of cybersecurity, as well as review studies that compare various cybersecurity environments for reinforcement learning. It's worth noting that our review includes works up to March 2024. Initially, we observed a notable surge in research about collaborative multi-agent reinforcement learning, particularly evident from 2023 onwards. Notably, by March 2024, there were already five papers published within just three months, indicating a positive trend in the research landscape of collaborative reinforcement learning. This positive trend can be seen in Figure 4.

Furthermore, our investigation resulted in the categorization of the identified papers, as depicted in Figure 5. These studies were initially categorized based on the introduction of collaborative reinforcement learning within the context of *Attack*, *Defense*, or both domains. Additionally, we further partitioned papers focusing on *Defense* and *Attack and Defense*, given their alignment with the conceptual framework of our project, particularly concerning the creation of collaborative defenders.

Within this classification, a significant portion of studies revolved around the Detection or Mitigation of Denial of Service (DoS) attacks. Typically, in such scenarios, agents are deployed across various gateways and critical links connected to diverse devices. Some studies outline their architecture with a simplistic network setup, positioning defenders at specific edges, while others leverage existing datasets such as the NSL-KDD dataset, as utilized in the study by Shi et al. [83].

Moreover, Cyber-Physical System (CPS) defense emerges as another prominent research area. Many studies utilize a 2D grid environment, as exemplified in the work by Wan et al. [100], or represent physical circuits to simulate scenarios involving electricity flow, as demonstrated by Mukherjee et al. [61].

Figure 5: Papers using collaborative reinforcement learning within cybersecurity, categorized by whether agents collaborate in offense, defense, or both, and then further divided within the field of cybersecurity.



Certain papers, particularly those focusing on *Attack & Defense*, fall under the category of design, where the objective is not solely to train agents for defense but also to identify optimal security configurations. This category aims to employ reinforcement learning to optimize environmental design or resource allocation to improve defense mechanisms.

Additionally, some papers, like the one by Chen et al. [24], delve into network security for other applications such as anti-jamming, where the channel bandwidth emerges as a critical specification akin to DoS scenarios.

Finally, papers relevant to our study are those addressing network defense in a collaborative manner. Notably, the following papers loosely relate to our work:

- The paper by Cardelline et al. [22] presents a simulated environment with pre-existing agent implementations, focusing on defending a web application. Although the primary focus is on a web application, the inclusion in related works is somewhat relevant, given the networking capabilities among its components.

- Wilson et al. [103] deploy agents in a network representing different control systems of military ships or submarines. While the simulation encompasses varying topologies, it appears simpler compared to our chosen environment, CybORG. Nonetheless, it stands as the most similar paper.

- Zhu et al. [110] implement collaborative attacker and defender agents within a network environment. However, the simulation primarily addresses a 'design' problem, specifically targeting optimal resource allocation. It employs a Colonel Blotto Game model, where attackers and defenders allocate resources within the network, with the victor determined by the highest resource allocation level.

- Wiebe et al. [102] implemented a collaborative defender agent in the CybORG environment, specifically from the Cage Challenge 2 [89]. In their study, the authors introduced an additional defender agent to the baseline environment to defend against a deterministic attacker. Although the network topology is simpler, with fewer total agents (attackers, defenders, and user agents) and a more limited action set, this previous iteration of CybORG closely resembles our work. The primary difference is the additional complexity introduced in newer iterations of the environment.

In conclusion, to the best of our knowledge, no paper encompasses the same combination of use case and environment as ours. Finally, Table 1 offers a concise summary of the aforementioned related works, primarily highlighting key differences from our project.

Instead, regarding the first research question, as previously mentioned, we aim to categorize and evaluate different reinforcement learning environments for cybersecurity. Unlike the more expansive research on the use of multi-agent reinforcement learning in cybersecurity, there is comparatively little literature focused

Table 1: List of related works that use collaborative reinforcement learning for the defense of a network structure.

Paper	Focus	Difference with our study
[22]	Collaborative Blue team agents defending a web application and its components from an external attacker.	- Loose similarity with our study, as the environment is not a network although the application components have some network capabilities between them.
[103]	Collaborative Blue team agents defending a network of a military ship	- The network itself is focus on representing entities of a military ship, not ones in a network. - Less network complexity compared to the environment chosen for our study.
[110]	Collaborative Blue and Red agents which compete for the optimal resources allocation in a network.	- Simpler network representation - Different goal for the agents, as their objective is only to allocate resources in advantageous nodes.
[102]	Collaborative Blue team agents defending a network architecture against a deterministic Red agent.	- Less network complexity compared to the environment chosen for our study. - Less agent complexity

on evaluating and comparing these environments. Most papers that utilize a cybersecurity environment for reinforcement learning tend to include brief sections describing the chosen environment, occasionally comparing it to a few alternatives. These comparisons typically highlight conceptual differences between environments without conducting a thorough analysis. It is important to note that the primary focus of these studies is not environment comparison, but rather the broader topic of applying reinforcement learning in cybersecurity [68] [64] [52].

However, some papers in the existing literature do provide more detailed comparisons of cybersecurity environments. Studies that incorporate such comparisons as an integral part of their research include:

- The study by Palmer et al. [65] reviews various reinforcement learning environments, categorizing them based on properties like observability, action sets, and other reinforcement learning-related characteristics. While the primary focus is on autonomous cyber operations, the authors also review environments outside of cybersecurity, such as SMAC [73], discussing only three cybersecurity environments.

- The paper by Drasar et al. [29] compares available simulation environments based on MITRE-defined metrics, such as adversarial behavior, operational architecture, and defender actions. The study also offers a detailed comparison between two specific environments.

- The paper by Standen et al. [89] introduces CybORG as an open-source environment for autonomous agents and compares it to other existing environments, offering a similar meter of comparison as the one presented in this study.

- The study by Byas et al. [99] provides a comprehensive review of automated cyber defense, encompassing various fields of artificial intelligence, including reinforcement learning and game theory. It also includes a review of both open and closed-source environments classifying them based on different categories, such as the presence of pre-trained agents, emulation, as well as the different action and observation sets available.

In summary, while there are papers that present similar reviews of cybersecurity environments, none offer the exhaustive, comprehensive analysis of open-source environments that this study provides. Table 2 summarizes the key differences between our project and the related works discussed above.

3 Environments

In this section, we will explore open-source environments designed for reinforcement learning agents, focusing particularly on environments tailored to the field of cybersecurity. Furthermore, we will assess the cybersecurity environments identified based on metrics outlined in Section 3.3. This evaluation process will yield

Table 2: List of related works that focus on a comparison between the different reinforcement learning environments in the field of cybersecurity.

Paper	Focus	Difference with our study
[65]	Literature review on deep reinforcement learning in the context of cyber security, with a section comparing reinforcement learning environments based on their properties.	- The review includes only three cybersecurity environments
[29]	Comparison of cybersecurity simulation environments using MITRE metrics, with an in-depth analysis of two selected environments.	- The study includes only four cybersecurity environments - The review excludes emulation environments
[89]	The paper introduces the CybORG environment and performs a brief comparison with other environments, highlighting differences and improvements.	- Lack of in-depth analysis we provide for certain environments - Many of the environments discussed are not open-source
[99]	Review of automated cyber defense techniques, including an evaluation of different open and closed-source environments.	- Exclusion of many environments covered in our study - Lack of in-depth analysis we provide for certain environments

a shortlist, which will undergo a comprehensive examination to determine the primary environment for the project.

3.1 Reinforcement Learning Environments

The success of a project in reinforcement learning often depends on the definition and handling of its environment by the agent. Therefore, for a project to thrive, it's essential to have not only a well-defined specification for the reinforcement learning agent but also a robust environment. Before delving into the discussion and analysis of environments tailored for the project, we aim to introduce reinforcement learning environments designed for general tasks. These baseline environments serve as foundational elements for many open-source cybersecurity environments. They are frequently employed as the foundational components for constructing more intricate environments or utilized as wrappers to facilitate the training and evaluation of various agents.

The initial environment we will discuss is **OpenAI Gym**. Introduced by OpenAI in 2016, Gym serves as an open-source platform dedicated to facilitating the creation of environments tailored for training reinforcement learning agents [18]. Apart from offering a collection of predefined environments for researchers to test, Gym enables researchers to define and share their environments by the standards established by its codebase. Gym aims to establish a standard for environment specification and utilization in the training of AI-based agents. Owing to its simplicity and early release in the realm of reinforcement learning, Gym swiftly emerged as one of the most popular foundational environments for training and evaluating reinforcement learning agents, effectively fulfilling its intended objectives. However, Gym's development gradually slowed over time and effectively ceased in 2021, prompting the *Farama Foundation* to take over and evolve the Gym environment into a new framework named Gymnasium [97]. With OpenAI discontinuing further Gym production, Gymnasium now stands as the sole maintained version slated to receive ongoing updates. It's worth noting that the new developers made minimal adjustments to Gym's API to facilitate a seamless transition between the two environments, prioritizing bug fixes and enhancements in functionality.

Another prominent environment developed by the Farama Foundation is **PettingZoo** [94], which was released in 2020. The primary distinction between PettingZoo and Gym/Gymnasium lies in their respective focuses: PettingZoo is tailored explicitly for research in multi-agent reinforcement learning, while Gym primarily caters to environments implementing single agents. This positioning makes PettingZoo the most notable environment directly supporting multi-agent scenarios, and its maintenance by the same organization ensures a similar level of utility as Gymnasium. It's worth noting that, at the time of writing, PettingZoo and Gymnasium are not integrated, creating a divide between the two environments. This division is particularly relevant for the scope of this project, as PettingZoo is specifically designed for training multiple agents, while most cybersecurity-based environments are built on Gym, as we will explore in Section 3.2.

The last notable environment we wish to highlight is **RLlib** [54]. RLlib is another open-source library offering a diverse range of options, capable of training both single and multi-agent setups while also supporting distributed reinforcement learning workloads. Since 2023, RLlib has been able to integrate with Gymnasium, further expanding its capabilities. The primary strength of RLlib lies in its distributed learning approach, achieved through parallelizing the learning process. Additionally, RLlib excels in supporting multi-agent training and can interface seamlessly with Gymnasium.

Lastly, it's important to acknowledge the existence of several other reinforcement learning environments not discussed here, as they are not relevant to our project. Examples include the **DeepMind Control Suite**, designed primarily for solving continuous tasks [92], and the **Unity toolkit** for agents development [49], which leverages the Unity engine to train agents in various 2D and 3D environments.

3.2 Reinforcement Learning and Cyber Security Environments

After providing an initial overview of reinforcement learning environments in the preceding section, we now turn our attention to presenting the various reinforcement learning environments considered for our project. We aim to offer a concise explanation of each environment, enabling this paper to serve as a valuable resource for future developers seeking environments that best suit their needs. To compile our initial list of environments, we referenced the curated list available on the GitHub repository of Hammar et al. [41].

From this list, we initially selected `open-source` environments, with their use cases and environment specifications briefly outlined in Table 3. It's worth noting that the descriptions provided in the table are general and will be expanded upon for the environments that advance beyond the initial evaluation stage. The use case described in the table follows a similar pattern, offering a broad overview. However, the specific use case tends to vary depending on the agent(s) that are implemented in most scenarios.

It's worth mentioning that the listed use cases represent general scenarios outlined or implemented within the respective projects. However, it's important to consider that these descriptions can evolve. For instance, if a defender agent is introduced in an environment initially intended for Penetration Testing, the focus might shift to aspects of intrusion prevention. Additionally, we'd like to highlight that the environments listed in Table 3 are related to each other:

- From NASim: NASimEmu and CLAP
- From Yawning-Titan: PrimAITE
- From Gym-malware: malware_rl
- From Gym-optimal-intrusion-response: gym-idsgame

Table 3: List of open-source environments along with their description and respective use case(s)

Name	**Description**	**Use Case**
PrimAITE [58]	Network Environment: flexible network, in which a firewall simulator is implemented. The system can be configured through YAML files.	Network Related use cases - Intrusion Prevention
CSLE [43]	Network Environment: implements both Emulation and Simulation using Docker images.	Network Related use cases - Intrusion Prevention
AutoPentest DRL [46]	Agent Implementation: provides a DRL agent to perform penetration testing on a logical or real network	Penetration Testing
NASimEmu [48]	Network Environment: provides both simulation and emulation of network with different topologies and features.	Penetration Testing
Gym-idsgame [42]	Network Environment: provides a simulated network environment which supports the implementation of blue and/or red team agents	Network Related use cases - Intrusion Prevention

Table 3 continued from previous page

CyberBattleSim [93]	Network Environment: simulation of a network environment with variyng topologies and vulnerabilities	Network Related use cases - Lateral Movement
Gym-malware [8]	Sample Generator: an environment which generates malware samples that the agents can modify to avoid detection	Malware Detection
malware_rl [38]	Sample Generator: an environment which generates malware samples that the agents can modify to avoid detection	Malware Detection
Gym-flipit [63]	Resource Environment: provides and implementation of blue and red agent to take control of a common set of resources (crypto keys).	Stealthy Takeover
Gym Threat Defense [44]	Network Environment: representation of a network as an attack graph, where the edges between nodes represent the exploits an attacker can use	Network Related use cases - Intrusion Prevention
NASim [79]	Network Environment: provides the simulation of a network with different vulnerabilities, exploits and topologies is available.	Network Related use cases - Priviledge Escalation
Gym-optimal-intrusion-response [40]	Network Environment: simulation of a network infrastructure, including different clients, servers and gateways.	Network Related use cases - Intrusion Prevention
sql-env [28]	Sample Generator: Provides list of SQL statements in which the agents can perform the injection to	SQL Injection
ATMoS [6]	Network Environment: framework for the application of agents to SDN (Software Defined Networks), using Docker container for the different network host.	Network Related use cases - Mitigation of APT.
MAB-Malware [1]	Sample Generator: framework that generates adversarial examples (for malware classifiers) given a sample.	Malware Detection
asap [10]	Network Environment: simulated environment which from gets an input network and models it as a graph adding different vulnerabilities to the nodes.	Penetration Testing
Yawning-Titan [9]	Network Environment: graph based simulation environment in which each node has a vulnerability score associated to it	Network Related use cases - Intrusion Prevention
CybORG [3]	Network Environment: provides a set simulated infrastructure for agents to interact with. Different scenarios (with their own infrastructure and use case) are published each year for researchers to test with.	Network Related use cases - Intrusion Prevention
SecureAI [84]	Network Environment: simulation environment providing a different set of network topologies with a simple nodes/edges specification.	Network Related use cases - Intrusion Prevention
CYST [30]	Network Environment: network simulator with an in-depth infrastructure that can be easily changed. The nodes message exchange mimicks the packet transport in real applications.	Network Related use cases - Priviledge Escalation
CLAP [106]	Network Environment: provides a network simulation environment to train red agents to perform penetration testing on the network.	Penetration Testing
BRAWL [59]	Network Environment: simulated an enterprise network inside of a cloud environment and provides already for the implementation of static bots as red/blue agents.	Network Related use cases - Intrusion Prevention

3.3 Environment Classification Metrics

Our primary objective is to identify an environment suitable for our intrusion prevention use case. Table 3 reveals a total of 22 open-source environments. Consequently, we aim to streamline this list to facilitate a thorough examination of the shortlisted environments, enabling us to select the most fitting solution for our project. This necessitates the adoption of classification metrics to trim down the initial list and prioritize those environments exhibiting high scores in these metrics. It's important to emphasize that these metrics will serve only as a means to initially shortlist the environments. Subsequent evaluations will rely solely on a comprehensive analysis of the shortlisted options and our firsthand experience with them.

For the **first evaluation phase**, we have established the following classification metrics, assigning scores ranging from 1 to 5, with 5 indicating the highest and 1 indicating the lowest:

- **Last Update**: This metric assesses the recency of updates to the repository of each environment (as of March 2024). Recent updates suggest active maintenance and compatibility with evolving dependencies, ensuring that the environment remains relevant. Scores are allocated as follows:
  - 4 for environments updated within the last 6 months
  - 3 for environments updated within the last year
  - 2 for environments updated within the last three years
  - 1 for environments with no updates in over three years
- **Studies**: This metric quantifies the number of studies conducted on each environment, providing insights into its utilization and significance within the scientific community. Scores are assigned as follows:
  - 1 for environments with fewer than three studies
  - 2 for environments with fewer than ten studies
  - 3 for environments with fewer than 30 studies
  - 4 for environments with fewer than 50 studies
  - 5 for environments with more than 50 studies
- **Flexibility**: This metric gauges the adaptability of the environment, considering both its inherent flexibility (e.g., support for various network topologies and vulnerabilities) and its capacity to accommodate different agents.
- **Documentation**: This metric evaluates the comprehensiveness of documentation accompanying each environment. A robust documentation should include user guides, installation instructions, API specifications (if applicable), and illustrative examples to facilitate users in getting started with the project

Furthermore, the **final assessment of environments** will rely on different metrics that won't be measured on a numerical scale like those used in the initial evaluation. Instead, the objective of these metrics is to offer researchers a broad understanding of an environment, facilitating informed decisions regarding its suitability for their research. Therefore, for the final evaluation, we will consider the following categorical variables:

- **Environment Description**: Provides a comprehensive overview of the environment, covering all relevant aspects while omitting details such as agent implementation and abstraction level, which will be addressed separately.
- **Usability**: Assesses the ease of use and deployment of the system, offering developers insights into what to expect when utilizing the environment.
- **Direct Multi-Agent Support**: Indicates whether the environment directly supports multi-agent reinforcement learning in a cooperative way, catering to both blue and red team agents.

- **Abstraction Level**: Evaluates the level of abstraction of the environment, offering insights into how closely it mimics real-life networks and operation centers. This assessment will be based on the information provided in the Environment Description, facilitating a discussion on the environment's complexity.
- **Agents Implementation**: Provides a brief overview of the agent implementation within the environment, including red, blue, and potentially green agents if applicable.
- **Studies Discussion**: Mention any existing studies on the environment that focus on multi-agent reinforcement learning.
- **General Impressions**: Offers general impressions and observations on the environment after utilizing and testing it.

3.4 Environments Evaluation

In this section, we will conduct and analyze the evaluation of the environments identified earlier. The initial evaluation will be guided by the metrics outlined in the preceding section. From this assessment, a subset of environments will be shortlisted for further scrutiny in the second evaluation phase. Subsequently, a final selection of environments will undergo meticulous examination and testing, accompanied by an assessment of their respective strengths and weaknesses.

This comprehensive evaluation process will furnish us with a systematic approach to selecting the most suitable environment for our requirements. Moreover, it will serve as a valuable resource for future developers seeking guidance in selecting environments for similar projects.

3.4.1 First Environments Evaluation

As shown in Table 4, none of the environments attained a perfect score of 19/19. Instead, the highest scores were achieved by CyberBattleSim, Yawning-Titan, and CybORG, each garnering a score of 16/19. Furthermore, for the second phase of evaluation, we opted to include the top 30% of environments based on their ratings. With a total of 22 environments, approximately 7 environments were selected for further evaluation, encompassing all those that attained a score of 14 or higher. Notably, we wish to highlight that all of the chosen environments are network-centric, whereas environments serving as sample generators or catering to different use cases fared poorly, largely attributed to their infrequent updates on their repository. This trend highlights a growing emphasis, particularly in recent times, on network-oriented environments for the training of artificial intelligence-based agents.

Moreover, it's important to emphasize that this initial evaluation serves as a preliminary screening of the environments. Subsequent evaluations will adopt a broader scope in the second phase, leading to an in-depth analysis in the third phase.

3.4.2 Second Environments Evaluation

It's worth noting that in this section, we will provide brief discussions of the environments we have excluded from the third evaluation, accompanied by concise descriptions and reasons for their exclusion, in line with the metrics outlined in Section 3.3, which will guide the final evaluation process.

We have chosen not to delve deeply into the environments that have passed this initial screening, as they will be thoroughly described and analyzed in the subsequent section. However, it's crucial to mention that this screening process is directly relevant to the theme of our project and its requirements, particularly concerning Multi-Agent implementations. With that said, from the initial screening, we have excluded the following two environments:

- **NASim**:
  - *Environment Description*: Simulation environment without emulation capability:
    * Offers diverse network scenarios, varying from small (with an action space of 18) to large (with an action space of 3000).
    * Networks are segmented into subnetworks, comprising various nodes, services (such as FTP, SSH, HTTP), and firewalls, contingent upon the network topology.

Table 4: First evaluation of the environments based on the metrics described in Section 3.3

Name	Last Update	Studies	Flexibility	Documentation	Total
PrimAITE [58]	4	1	5	4	**14**
CSLE [43]	4	1	4	5	**14**
AutoPentest DRL [46]	2	1	5	3	11
NASimEmu [48]	4	1	4	4	13
Gym-idsgame [42]	4	1	4	3	12
CyberBattleSim [93]	4	5	4	3	**16**
Gym-malware [8]	1	4	2	2	9
malware_rl [38]	4	1	3	2	10
Gym-flipit [63]	1	1	2	1	5
Gym Threat Defense [44]	1	1	1	2	5
NASim [79]	3	2	4	5	**14**
Gym-optimal-intrusion-response [40]	2	2	2	2	8
sql-env [28]	2	1	1	1	5
ATMoS [6]	1	3	3	4	11
MAB-Malware [1]	2	5	1	2	10
asap [10]	1	3	1	1	6
Yawning-Titan [9]	4	2	5	5	**16**
CybORG [3]	4	4	4	4	**16**
SecureAI [84]	1	1	3	1	6
CYST [30]	4	1	5	4	**14**
CLAP [106]	3	1	5	3	12
BRAWL [59]	1	1	2	3	7

    - ∗ Typically, attacks are executed by exploiting the services hosted on each machine within the network.
  - – *Direct Multi-Agent Support*: No
  - – *Agent Implementation* Only Red Agent implemented which engage in attacks resembling complete scans (like Nmap), followed by necessary offensive maneuvers to compromise distinct nodes, with each action incurring a cost.
  - – *Reason for exclusion* The environment is primarily geared to penetration testing, lacking even a deterministic defensive agent in its implementation.

- **CYST**:
  - – *Environment Description*: Simulation environment without emulation capability:
    - ∗ The network comprises elements such as Nodes, Firewalls, Connections, Routers, and Sessions.
    - ∗ During the simulation, these components (network nodes) transmit messages, replicating the packet transport observed in real-world applications
    - ∗ The environment allows the definition of various network topologies.
  - – *Direct Multi-Agent Support*: No
  - – *Agent Implementation* Only a Red Agent is implemented in both Reinforcement Learning (RL) and offers support for non-RL-based agents. Its actions include Reconnaissance, Brute-Force attacks, privilege escalation, and more.
  - – *Reason for exclusion* Very interesting framework; however, it appears that some of the discussed features have yet to be implemented. This suggests that the framework may still be in development and not yet finalized.

3.4.3 Final Environments Evaluation

In this concluding segment of our analysis, we will delve into the five shortlisted environments based on the metrics outlined in Section 3.3. Subsequently, we will arrive at a decision guided by these categories, aiming to select an environment that aligns most closely with the requirements of our project. With that in mind, let's proceed with the evaluation of the environments.

- **Yawning-Titan**
  - *Environment Description*: Simulation environment without emulation capability:
    - Network topologies are customizable, offering standard options while emphasizing the ease of creating diverse configurations.
    - Networks consist of nodes (computers) interconnected by edges (connections).
    - Each node is assigned a preset value (High or Low) and a vulnerability score indicating its susceptibility to attacks.
    - Various entry nodes can be designated for attackers to initiate assaults.
    - Game modes are adaptable, allowing for adjustments to train or configure agents according to preferences, including altering the action space, observation space, and final objectives.
  - *Usability*: Highly user-friendly with comprehensive documentation and strong support.
  - *Direct Multi-Agent Support*: No
  - *Agents Implementation*: A probabilistic red agent and blue agent are implemented. ‘
    - The red agent can infect nodes with a probability determined by both the node vulnerability score and the agent's skills.
    - The blue agent defends the network through actions like reducing node vulnerability, scanning the network for intrusions, and isolating nodes.
  - *Abstraction Level*: The environment is highly abstract, with each node represented in a very simplistic manner, characterized solely by a vulnerability score and associated value. Furthermore, the edges lack any attributes.
  - *Studies Discussion*: No studies implementing multi-agent reinforcement learning for this framework were found.
  - *General Impressions*: The environment is highly adaptable, allowing for the testing of agents across a wide range of network topologies and game modes. However, its negatives lie in its simplicity and abstraction, as it lacks the intricacies typically found in real-life networks.
- **PrimAITE**
  - *Environment Description*: Simulation environment without emulation capability:
    - Offers a baseline environment similar to Yawning-Titan, providing comparable flexibility in creating game modes and network topologies.
    - Node definitions are slightly more intricate compared to Yawning-Titan; each node includes a Priority value, Hardware State, Software State, File System State, and Service State, allowing for a more detailed representation.
    - Nodes are linked by edges, primarily defined by maximum bandwidth (bits/s) and a list of protocols running in the link.
    - Includes an access control list (ACL) to regulate traffic flow in the system, which blue agents can interact with.
  - *Usability*: Highly user-friendly with comprehensive documentation, although slightly more challenging to navigate compared to Yawning-Titan due to its less accessible documentation.
  - *Direct Multi-Agent Support*: No
  - *Agents Implementation*: No agents are directly implemented, although wrapper implementations are provided for any RLlib agent. The actions of the Blue and Red agents resemble those in Yawning-Titan, with consideration given to network traffic, modeled by Green Agents.

    - Red Agents not implemented, but similar action space to Yawning-Titan.
    - Blue Agents not implemented, but similar action space to Yawning-Titan.
    - Green Agents operate on different nodes and exchange information, simulating network traffic.
  - *Abstraction Level*: This environment exhibits a high level of abstraction regarding network infrastructure, albeit slightly more detailed than Yawning-Titan. This is evident in both the definition of edges and nodes, as well as in its ability to approximate real network traffic through the implementation of green agents.
  - *Studies Discussion*: No research found on it
  - *General Impressions*: A highly flexible environment that offers greater complexity compared to Yawning-Titan while retaining its beneficial functionalities. It is relatively new and lacks research, and does not include direct red/blue agent implementation.

- **CyberBattleSim**
  - *Environment Description*: Simulation environment without emulation capability:
    - Provides some standard networks with varying topologies and the ability to customize them according to our requirements.
    - Nodes within the graph symbolize network entities, such as computers, while edges represent their communication channels.
    - Edges can denote different connection types, such as HTTP between two computers, or SSH or SQL with the database.
    - Nodes possess specific reward values; for instance, compromising a SQL server yields greater value than compromising a test machine. Additionally, nodes have predefined vulnerabilities, each with a predetermined outcome if exploited, such as leaking credentials or information to another node.
  - *Usability*: Highly user-friendly and conducive to testing, thanks to its comprehensive documentation and the provision of various notebooks for testing different implementations.
  - *Direct Multi-Agent Support*: No
  - *Agents Implementation*: A reinforcement learning-based red agent is implemented, alongside a deterministic blue agent.
    - The Red Agent is modeled on the Mitre Attack framework for lateral movement, incorporating a range of local and remote attacks outlined in the framework.
    - The deterministic Blue Agent executes simple actions such as node isolation and attacker eviction upon detection.
  - *Abstraction Level*: This environment is highly abstract, featuring a simplistic representation of nodes and edges. It lacks representation of network traffic, as indicated by the absence of green agents, yet offers more complexity than environments like Yawning-Titan.
  - *Studies Discussion*: Research focuses on implementing multi-agent reinforcement learning on the defensive side, incorporating a blue agent to counter the already deployed red agent.
  - *General Impressions*: Regarded as the most prominent reinforcement learning cybersecurity environment, it maintains a suitable abstraction level compared to others. With existing implementation of the Red Agent and various topologies, it's ideal for testing different agents in diverse simple networks.

- **CSLE**
  - *Environment Description*: Emulation environment with simulation capabilities:
    - Emulation mimics the functionalities of a target infrastructure.
    - Physical hosts are emulated using Docker containers, some equipped with predefined vulnerabilities, while others host SSH, FTP, and Telnet Servers.
    - Network links are emulated via Linux Bridges.

    - Switches are emulated through Docker containers running Open vSwitch.
    - A monitoring system gathers information on the emulated devices, which the RL agents utilize to determine the optimal course of action.
    - The monitoring system also constructs the simulation by representing its underlying infrastructure as a network graph.
    - The agents run on the simulation and their actions are mapped to the emulation.
    - Provides a selection of predefined network topologies.
    - Presents the option of using different game modes, with different goals for the agents. This includes use cases ranging from optimal stopping problems to local intrusion response games.
  - *Usability*: Challenging to use and install, requiring significant memory (minimum 50+ GB) and RAM (minimum 16+ GB) to operate. Additionally, it entails a lengthy installation process and is generally heavier (for the agents' training) compared to other simulation environments.
  - *Direct Multi-Agent Support*: Yes (competitive)
  - *Agents Implementation*: The system provides the implementation of both blue and red team agents.
    - Red Agents execute various moves within the emulation, such as Reconnaissance (TCP/UDP port scan, etc.), Brute-force attacks (Telnet, SSH, attacks on databases like Cassandra, MongoDB, etc.), and different types of exploits like SQL Injections.
    - Blue Agents undertake actions to stop the red agents, including Revoking user certificates, Blacklisting IPs, Blocking Gateways, Isolating/Shutting down servers, Redirecting traffic, etc.
    - Green agents client population, which performs general operations, such as performing database queries or connecting to different servers.
  - *Abstraction Level*: Given its emulation, this environment exhibits the least abstraction among those considered, as agent movements directly correspond to actions feasible on real systems. However, it lacks traffic representation since no green agent has been implemented yet.
  - *Studies Discussion*: No studies have been performed on this environment.
  - *General Impressions*: Ideal for researchers seeking an environment akin to real-life systems. It is less adaptable than others and presents greater challenges in setup and usage.

- **CybORG**
  - *Environment Description*: Simulation environment without emulation capability:
    - Offers various annual challenges featuring distinct objectives, environments, and agent modifications.
    - All challenges primarily focus on cybersecurity. Both challenges 3 and 4 target multi-agent reinforcement learning.
    - Challenge 3 introduces a scenario involving a multi-agent drone swarm, where defenders must protect a formation of drones.
    - Challenge 4 is notably intriguing, in the context of this project, presenting a mixed framework for multi-agent reinforcement learning.
    - In this framework (challenge 4), a single network is divided into 4 subnetworks.
    - Each subnetwork is further subdivided into different zones, with each zone containing varying numbers of servers (1-6), user hosts (3-10), and services per host (1-5, randomized).
    - In this challenge, the blue team is tasked with defending the network and ensuring its availability for the green agents, which represent the users.
  - *Usability*: User-friendly interface facilitated by comprehensive documentation and informative examples for each of the challenges.
  - *Direct Multi-Agent Support*: Yes, available for challenges 3 and 4.
  - *Agents Implementation*: Each challenge provides agent implementations. In challenge 4, red team and green agents are implemented: '

    - ∗ Finite State Red agents can execute stealth/aggressive discovery, exploit services, perform privilege escalation, withdraw, discover deception, etc.
    - ∗ No blue agent is implemented, but the action set is defined. This includes actions such as monitoring, analyzing, and deploying decoys in the network, as well as removing, restoring, and blocking/allowing traffic.
    - ∗ Green agents can expose themselves to risk, thereby allowing new red agents to access the network (e.g., expose IP and port addresses, fall victim to phishing emails, etc.).
  - – *Abstraction Level*: The environment (challenge 4) offers a good level of abstraction, encompassing both network structures, node configurations, and traffic dynamics, including the client population represented by the green team. Agent actions closely resemble real-world cybersecurity scenarios.
  - – *Studies Discussion*: Numerous studies have been conducted on this environment, with some available for each challenge (except Challenge 4). These studies primarily delve into their implementation within the respective challenges and discuss the obtained results.
  - – *General Impressions*: This environment stands out as the optimal choice for accessing a variety of environments to test agents annually. It features both single-agent and multi-agent challenges. Challenge 4 exhibits a higher abstraction level compared to emulation environments, but at the same time offers greater complexity than the other selected simulation environments.

In conclusion, we have chosen CybORG as the environment for our project. This decision was based on CybORG's superior complexity compared to other simulation environments, without the limitations associated with emulation environments, such as low flexibility and lengthy setup and training times for agents. Additionally, with the introduction of the new challenge (published at the time of writing this paper), CybORG provides a prime opportunity to explore multi-agent reinforcement learning within a dedicated framework, complete with an existing implementation of a red agent team. For these reasons, we opted for **CybORG** as our environment of choice.

3.5 CybORG

In this section, we will delve deeper into the CybORG environment, which has been selected as the primary environment for our project. It's important to specify that our analysis of CybORG focuses solely on the environment and agent specifications provided by CAGE Challenge 4 [39]. This section aims to condense the essential information available in the GitHub repository of the challenge, cited above. The emphasis is on providing readers with a comprehensive overview to facilitate their understanding of the subsequent sections of this study. For those seeking a more detailed examination, the GitHub repository of the challenge offers additional insights.

3.5.1 Environment

As detailed in the preceding section, CybORG CAGE Challenge 4 introduces an environment structured around a network subdivided into four distinct subnetworks interconnected via internet connections. Shown in Figure 6, these subnetworks comprise two deployed networks, each further segmented into restricted and operational zones, a Headquarters network featuring three security zones, and a contractor network serving as the launching point for red team attacks. As previously mentioned, the distribution of servers, user hosts, and services across security zones is randomized. Within each subnetwork, hosts contain varying degrees of information that red agents can extract, ranging from basic system details to session access credentials, effectively granting user or root privileges to the attacker.

It's worth noting that each episode within the environment is non-uniform and progresses through different phases, each assigning varying degrees of priority to missions associated with specific Deployed Networks. This prioritization effectively dictates the actions of agents operating within those networks during each phase.

Finally, it is important to note that, as mentioned in previous sections, most multi-agent environments are modeled as Dec-POMDPs, which is also true for CybORG. The following subsections will provide a more detailed discussion of the various components that make up CybORG's Dec-POMDP as it is described in Section 2.

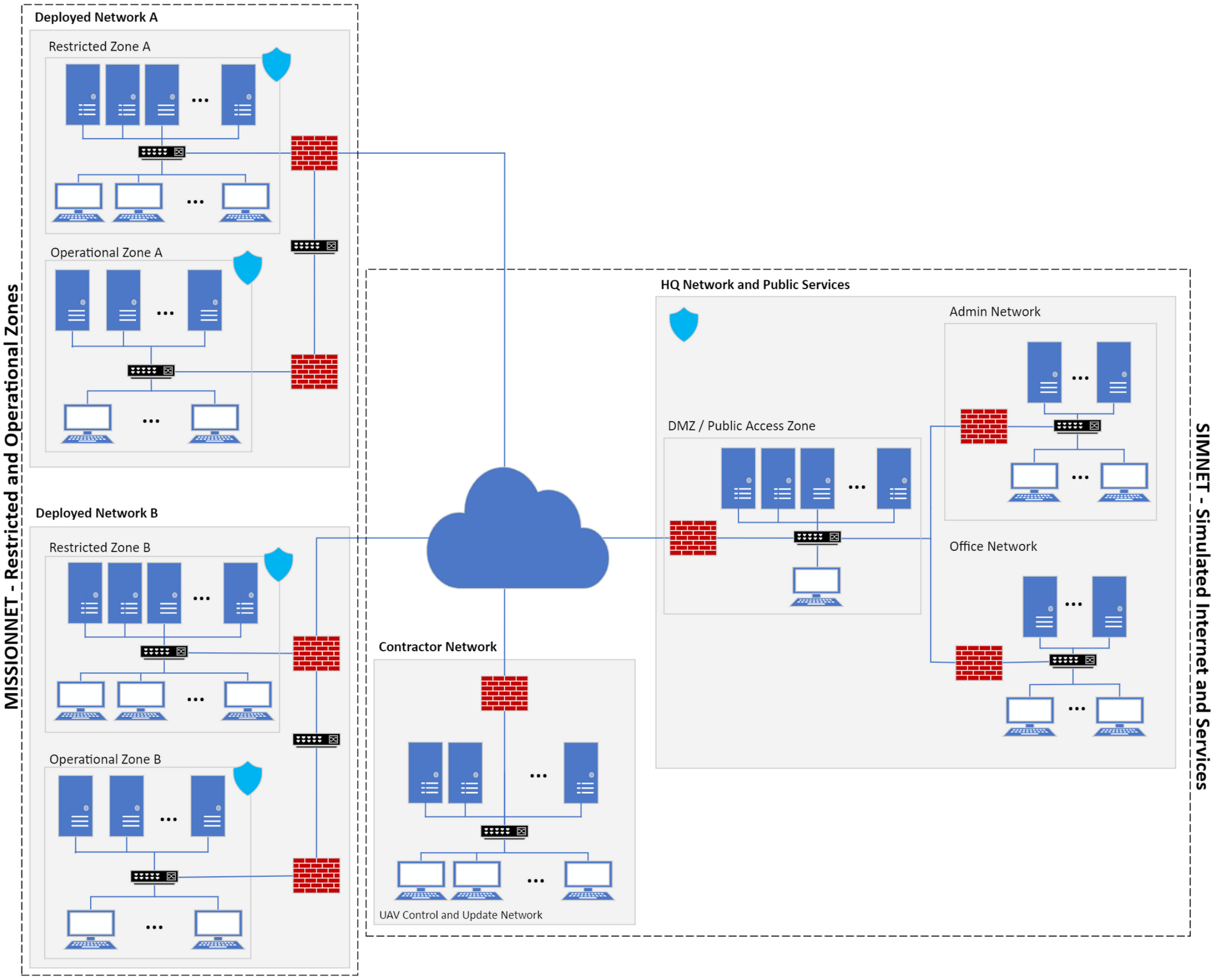


Figure 6: Network layout implemented in CybORG CAGE Challenge 4 [39]

3.5.2 Agents

As previously outlined, a central aspect of CAGE Challenge 4 is the integration of Green Agents, which play a crucial role within the network. These agents, as we'll delve into later, represent standard users within the network, making them vulnerable to attacks such as phishing attempts orchestrated by red agents. Notably, the availability of the network for green agents is of vital importance, with penalties imposed on blue agents in the event of green agent downtime.

Furthermore, in line with expectations for a Multi-Agent environment, red and blue team agents also inhabit the network. The primary objective of the Red Agent is to disrupt network availability by infecting as many hosts as possible, whereas the Blue Agent attempts to counteract these efforts. Each agent has a predefined set of actions to use. It is noteworthy that each action requires a certain amount of time, measured in 'ticks,' to execute. While simple actions are immediate, more intricate actions necessitate additional time.

For the Blue Agent:

- *Monitor* (time - 1): Observes a specific portion of the subnetwork.
- *Analyze* (time - 2): Identifies malware occurrences on a designated host.
- *Decoy* (time - 2): Deploys a decoy on a specified host, akin to a honeypot.
- *Remove* (time - 3): Eliminates a red agent from a targeted host.
- *Restore* (time - 5): Returns a host to a 'clean' state, devoid of malicious processes or attackers.
- *Control Traffic* (time - 1): Blocks incoming traffic from a particular zone.
- *Messages*: Agents also can exchange messages for coordination purposes. These messages consist of 8 bits and are transmitted to a designated agent exclusively.

The Red Agent's actions correspond to various activities outlined in the MITRE ATT&CK Framework [2] and include:

- Action 0 - *Remote discovery* (time - 1): Discover the IP address of a host in the subnet.
- Action 1 - *Aggressive Discovery* (time - 1): Conducts an intensive scan to uncover different hosts, with a high risk of detection.
- Action 2 - *Stealth Discovery* (time - 3): Executes a stealth scan to identify different hosts, with a lower risk of detection.
- Action 3 - *Deception* (time - 2): Checks for the presence of honeypots deployed by blue agents.
- Action 4 - *Exploit Services* (time - 4): Exploits services on a designated host remotely.
- Action 5 - *Privilege Escalation* (time - 2): Performs privilege escalation on a host.
- Action 6 - *Impact* (time - 2): Disrupts network availability between hosts.
- Action 7 - *Degrade Service* (time - 2): Worsen the service of a green host if the attacker has root access.
- Action 8 - *Withdraw* (time - 1): Retracts from a host.

Additionally, it's important to note that all agents possess a *sleep* action, during which they remain inactive for a single time unit.

This results in each agent having over 80 available actions for the subnet they manage, with agent 4 having an action set of more than 200 actions.

Figure 7 depicts the Finite State Machine (FSM) representation of the red agent, where state K represents the initial state and F is the final state. The states of each host are categorized based on whether they were encountered through the discovery action (action 0 for the red agent) or through other actions.

- States K and KD: Indicate a host whose IP address is known to the red agent.

- States S and SD: Represent a host whose services are known to the red agent.
- States U and UD: Indicate a host where the red agent has an active user shell.
- States R and RD: Represent a host where the red agent has an active root shell session.

The transitions between these states are labeled with the action numbers described in the list above containing the actions of the red agents.

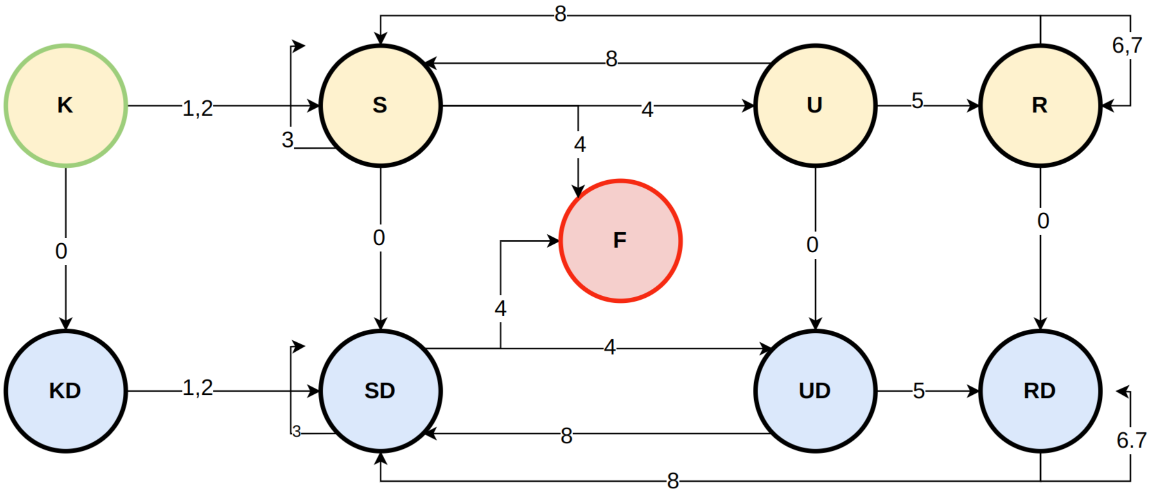


Figure 7: Finite State Machine representation of the red agents as presented in CybORG CAGE Challenge 4 [39]

3.5.3 Rewards

As mentioned earlier, the rewards for the blue agent are based upon the network's availability and the quality of work performed by the green agents at any given time. To minimize these disruptions, the blue agent begins with a reward of 0, which gradually decreases each time the green agents are unable to perform their tasks or if there are any disruptions caused by the red agents. Furthermore, Phase 2a focuses on maintaining operational activities for Zone A, while Phase 2b pertains to maintaining operational activities for Zone B. Consequently, failure to ensure service availability for these operational zones during their respective phases will result in significant penalties for the blue agents. In summary, the blue agents can achieve a **maximum reward** of 0 if they successfully prevent all red agent attacks proactively.

3.5.4 Observations

The observation space for the agents is an important aspect to consider. It's essential to clarify that the agents in this simulation do not learn in a centralized manner, meaning there isn't a central agent coordinating the actions of all others based on their observations. Instead, each agent operates autonomously and can communicate with neighboring agents by sending and receiving messages.

At each time step, agents receive the following observations:

- *Mission Phase*: Indicates the current stage of the mission.
- *Subnet Information*: Provides various details about each observable subnet, including:

  - State of the subnet (blocked or unblocked)
  - Communication policy
  - Detection of malicious activity or connections

- *Messages*: Consists of messages sent by other agents to facilitate coordination. It's important to note that agents' messages must be defined by the user and default to zero if not specified.

Additionally, it's worth noting that not all agents have the same observation space size. Some agents are responsible for defending only one subnet, while others have to defend three. Consequently, agents overseeing multiple subnets will have a larger observation space. A general representation of the agents observation space is given by Figure 8.

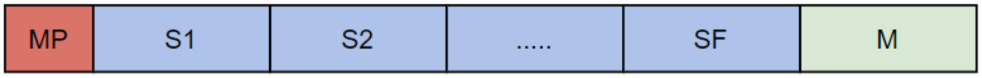


Figure 8: State representation of the CybORG environment for Agent with Mission Phase (MP), Subnetworks 1,2...etc.(S1, S2,...SF) and list of messages (M)

This results in each agent having a flat observation space of size 92, with agent 4 having an observation space of size 210.

4 Methodology

In this section, we will outline the methodology used in this project. It begins with the selection of reinforcement learning methods for the CybORG environment, detailing their specifications and our implementation. Additionally, we will discuss the changes made to the environment, rewards, and observations to address the research questions posed in this paper. Lastly, we will briefly outline the various technologies used in this project.

4.1 Reinforcement Learning Methods

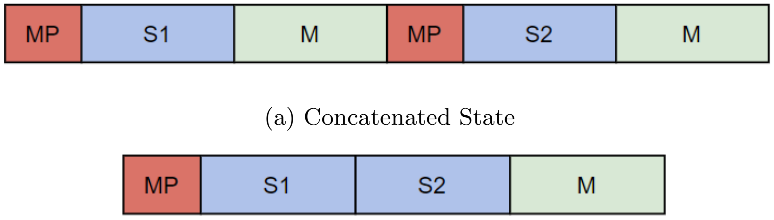
In this section, we will explore various enhancements to the reinforcement learning methods outlined in Section 2, specifically in the context of CybORG. These enhancements will be discussed both in relation to the CybORG environment and to the reinforcement learning methods that utilize them.

4.1.1 Global State Representation

In their paper, Yu et al. [108] explore the challenges of representing the global state for the Critic in MAPPO. The standard approach involves concatenating the observation vectors of all agents into a single global state representation, as shown in Figure 9. However, Yu et al. suggest that this approach might be inefficient, as it can include redundant or unnecessary information across different states. To address this, they propose a more compact state representation, which we also adopted in this study. This refined state representation, illustrated in Figure 9, combines the message states of all agents and the initial values representing the agents' mission states, as described in Section 3.5. This global state representation is 20% smaller than the concatenated version, and throughout this paper, we refer to this representation as C_MAPPO.

4.1.2 Exploration Strategy

An alternative approach we considered is the Boltzmann softmax action selection operator [13]. Unlike ϵ-greedy, which might result in suboptimal random actions during exploration, Boltzmann softmax assigns a probability distribution to actions based on their Q-values, with the balance between exploration and exploitation controlled by a temperature parameter τ. Higher values of τ increase exploration by making action probabilities more similar, while lower values emphasize exploitation by favoring actions with higher



(a) Concatenated State

(b) Compact State

Figure 9: Global state representation of the observations of different CybORG agents as initially described in Figure 8

Q-values as can be seen from Equation 9. In the result of discussion of this study, we will refer to the QMIX implementation using the Boltzmann softmax as B_QMIX.

$$action = \frac{e^{Q_t(a)/\tau}}{\sum_{a'}^{n} e^{Q_t(a')/\tau}} \tag{9}$$

4.1.3 Prioritized Experience Replay

In the previous sections, we introduced QMIX and MADDPG as two off-policy methods to use in the CybORG use case. As discussed, both methods, being off-policy, rely on a replay buffer to store past transitions for sampling during training. While most implementations use a basic buffer that samples episodes uniformly, we instead adopt the **Prioritized Experience Replay (PeR) buffer** introduced by Schaul et al. [75]. Using PeR allows the agent to sample more important experiences, potentially improving performance early in the training process. A common issue with standard replay buffers is that newer experiences may never be utilized, leading to situations where the agent trains on the same batch of data repeatedly, rather than incorporating new observations from updated policy versions. In PeR, the priority of each episode is based on its Temporal-Difference (TD) error, which is already calculated in both MADDPG and QMIX. However, as Schaul et al. suggest, relying solely on TD error for prioritization has drawbacks, such as favoring transitions with the best TD error too frequently while ignoring those with lower TD errors. To address this, the authors propose a formula, shown in Equation 10, to compute episode prioritization, where α controls the influence of the prioritization value p_i at runtime.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{10}$$

Additionally, to ensure all transitions are utilized at least once during training, new transitions are initially given the highest possible prioritization upon entering the buffer, increasing their likelihood of being selected for the next training step. In the following sections, we will refer to the QMIX and MADDPG implementations using the Prioritized Experience Replay buffer with the PeR prefix.

4.2 Recurrency in CybORG

One significant change we can introduce to the previously mentioned methods is the incorporation of recurrency into the network structure. Recurrency can be a powerful tool in reinforcement learning, particularly in environments with partial observability, like CybORG. Additionally, the fact that actions in CybORG take varying timesteps to become effective highlights the temporal dependencies between past actions and subsequent observations.

Figure 10 illustrates this concept with an example network consisting of three subnetworks. In this scenario, the agent has full observability of two subnetworks but only partial observability of subnetwork 3. Due to this limited visibility, the agent lacks complete information when deciding on actions. Here, the recurrent hidden state acts as a type of auxiliary memory, helping the agent track network patterns over different training runs. This memory proves especially valuable when chosen actions have delayed effects

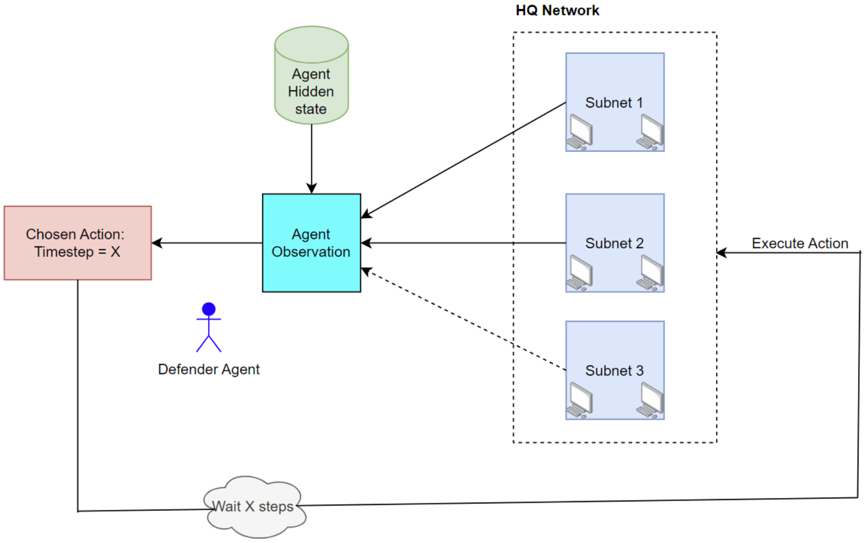


Figure 10: Example of use of recurrency in a network simulation like CybORG.

on the environment, providing the agent with *extra memory* that helps in handling the challenges of partial observability and action delays.

In the CybORG use case, we complete entire episodic sequences before training the defensive agents. As a result, our recurrent implementation does not use truncated episode sequences, meaning we do not need to store hidden states at truncation points. This ensures that our buffer implementation remains unaffected by the addition of recurrence. The initial hidden states for a new sequence are initialized to zero, making the integration of recurrence straightforward and minimally altering the pseudocode for the previously discussed algorithms.

For this project, we implemented recurrent versions of various PPO algorithms as well as QMIX, providing both policy-based and value-based reinforcement learning representatives. These versions will be denoted with an R prefix, such as R_QMIX for the recurrent QMIX implementation.

Finally, the choice of the recurrent neural network (RNN) architecture is a critical aspect of the recurrent implementation, specifically the decision between Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) [69]. LSTM is more complex, requiring an additional gate, which enables it to learn more intricate dependencies, albeit at the cost of increased training time and a higher risk of overfitting. This could be a drawback in CybORG, where the network's entities are always variable. GRU, by contrast, is simpler and more resource-efficient but may not capture complex relationships as effectively as LSTM. For our implementation, we decided to use LSTM as the recurrent cell for all PPO-based implementations and GRU for the recurrent version of QMIX.

4.3 Messages

As discussed in Section 3.5, one component of CybORG's observation space is the ability for blue agents to send messages to one another. This gives developers the option to use the 8 bits available to each agent to communicate important information about the environment, with the goal of improving coordination. Based on this, we designed three different types of messages, drawing on the coordination theory presented by Buşoniu et al. [21], and applying it to the Cage Challenge 4 use case. These messages can be divided into two main categories based on the type of data being exchanged: **action messages** and **state messages**.

The first category, action messages, is intended to reduce the non-stationarity of the environment by helping agents with action selection. These messages would be unnecessary if actions had an immediate effect, as they are sent at the time of action choice. However, because many actions take several timesteps to become effective, the messages provide valuable information about future actions. Each action is encoded in the 8 available bits as a message code. Since each agent has at most 242 available actions at runtime, we represent each action as an 8-bit number, as shown in the example below:

$$\begin{aligned} &action\ 64 = [0,1,0,0,0,0,0,0] \\ &action\ 132 = [1,0,0,0,0,1,0,0] \end{aligned}$$

The second category, state messages, aims to enhance the agents' observation of different subnetworks. These messages are structured using either **2 bits** or **8 bits**. In the 2-bit version, only the first two bits are used to indicate whether malicious processes or network events have been detected in a subnetwork. As the example below shows, there is no distinction in the messages between agents encountering one or multiple malicious activities in their subnetwork.

$$\begin{aligned} &Malicious\ Processes : 1, Malicious\ Events : 0 = [1,0,0,0,0,0,0,0] \\ &Malicious\ Processes : 4, Malicious\ Events : 2 = [1,1,0,0,0,0,0,0] \end{aligned}$$

These messages still provide useful information about other subnetworks' current state without significantly increasing the observation space. In contrast, the 8-bit messages convey the exact number of infected processes. The first 4 bits represent the number of infected processes (from 0 to 16), while the last 4 bits denote the number of infected networks. An example of this message strategy is shown below:

$$\begin{aligned} &Malicious\ Processes : 1, Malicious\ Events : 2 = [1,0,0,0,0,0,1,0] \\ &Malicious\ Processes : 4, Malicious\ Events : 3 = [0,1,0,0,0,0,1,1] \end{aligned}$$

In conclusion, with these message strategies, each agent can gain a general understanding of what is happening in other subnetworks, even though we use a decentralized training approach and a global reward system. Additionally, the messages do not transmit highly sensitive information, which is important if we envision a scenario where the defending agents operate under strict segregation and cannot share sensitive data. Thus, the network subdivision remains valid and logical.

4.4 Rewards

One possible challenge in multi-agent reinforcement learning is the issue of reward attribution when agents share the same reward. In such cases, it can be difficult for individual agents to determine whether their actions directly contributed to the change in reward or if it was due to the actions of other agents. Atrazhev et al. [14] specifically discuss how different reward structures can significantly impact the performance of reinforcement learning agents depending on the method used, particularly contrasting individual rewards with joint rewards in cooperative multi-agent settings. As mentioned in Section 3.5, CybORG uses a global reward shared among all agents, which is calculated based on the costs of certain actions and the rewards provided by green agents within the network. In response, we propose an alternative reward structure to contrast with CybORG's joint global reward system. This new reward system is individual, where each agent receives a reward based on the cost of their own actions and the activity of green agents within their own subnetwork. The total reward value remains the same, but it is distributed differently among the agents. Figure 11 illustrates a case for this new reward calculation: for the joint global reward, the rewards from each subnetwork are combined for all agents therefore obtaining the same reward of -21. In the second case, they are partitioned among individual agents, with Agent 1 obtaining a reward of -1 and Agent 5 of -10 in this case. Additionally, we introduce a hybrid reward system that combines both individual and joint rewards for each agent. The goal of this approach is to provide agents with information on both the overall network state and the impact of their own actions. Since the values of the two rewards differ, a simple summation would not be ideal, as some agents might become too reliant on the global reward, ignoring their individual contributions. To address this, the rewards are combined in a way that balances their influence. For example, if agent 1 controls 1/7 of the subnetworks, we expect their individual reward to account for 1/6 of the global joint reward on average. Therefore, the joint reward will be scaled by 1/7 and added to the individual reward. In contrast, Agent 5, which controls 3 subnetworks, will have the global reward scaled by almost half during the reward calculation. Following the example of the obtained rewards in Figure 11, Agent 5 would obtain the following hybrid reward:

$$R_5 = (-2-8) + \frac{3}{7} \cdot (-4-1-6-2-8) = -10 + \frac{3}{7}(-21) = -19$$

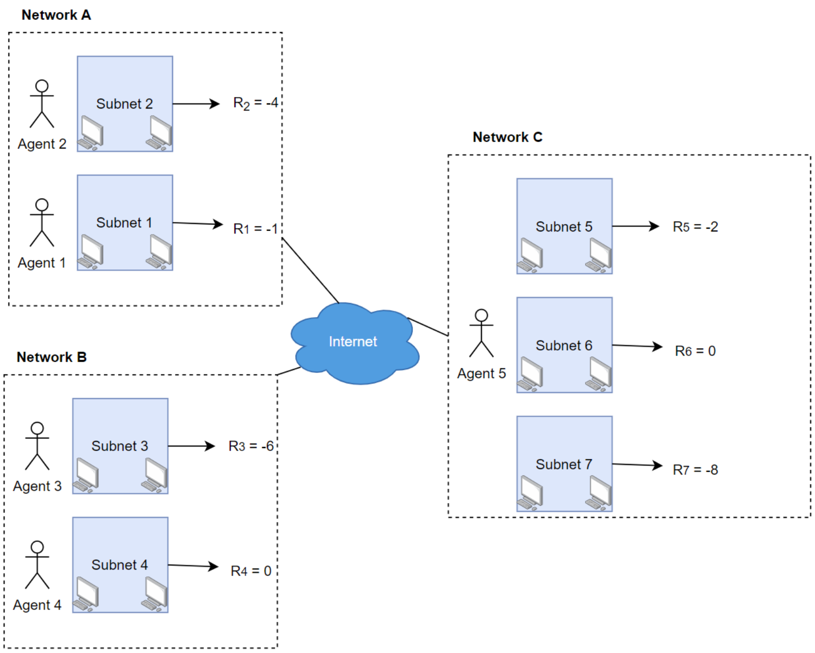


Figure 11: Example of reward partitioning between the agents in the CybORG network.

4.4.1 Intrinsic Rewards

In the previous section, we have discussed extrinsic rewards provided to the agents by the environment, specifically in the form of feedback from the green agents. However, in reinforcement learning, we can also incorporate `intrinsic rewards`, which are designed to encourage exploration rather than merely focusing on task completion. These intrinsic rewards can be particularly useful during the training phase, as they motivate agents to explore new states that might otherwise go unnoticed in typical scenarios. This is especially valuable in environments with sparse reward structures, where intrinsic rewards can guide agents in the absence of frequent extrinsic feedback. That said, they can also be applied in environments with more frequent rewards as an additional incentive for exploration.

Puigdomènech Badia et al. [16] discuss this concept in their work, highlighting the role of intrinsic rewards as a bonus for promoting exploration. The authors introduce the **Never Give Up (NGU)** module, depicted in Figure 12, which aims to incentivize agents to explore new states. The NGU module consists of two main components: the episodic novelty module and the life-long novelty module [16].

The episodic novelty module compares each new embedded state to previously encountered states; the greater the difference between the new state and its closest neighbors in memory, the larger the exploration reward. The life-long novelty module, on the other hand, is based on Random Network Distillation (RND) [19]. It provides an exploration bonus by measuring the error between a neural network's predictions of new observation features and the outputs of a fixed, randomly initialized network. The larger the prediction error, the greater the reward for exploring that state.

The total intrinsic reward from the NGU module is the combination of these two components and is added during training to encourage exploration. Importantly, reinforcement learning algorithms discussed in this paper that incorporate this module will include the NGU suffix in their name. Therefore, for instance, IPPO would be referred to as IPPO_NGU in this context.

4.5 Environment Changes

In this section, we outline the changes made to the baseline CybORG environment. Our focus was on implementing fundamental changes to the environment and the agents within it to observe the impact of the resulting policy. Beyond the previously discussed differences in reinforcement learning methods and

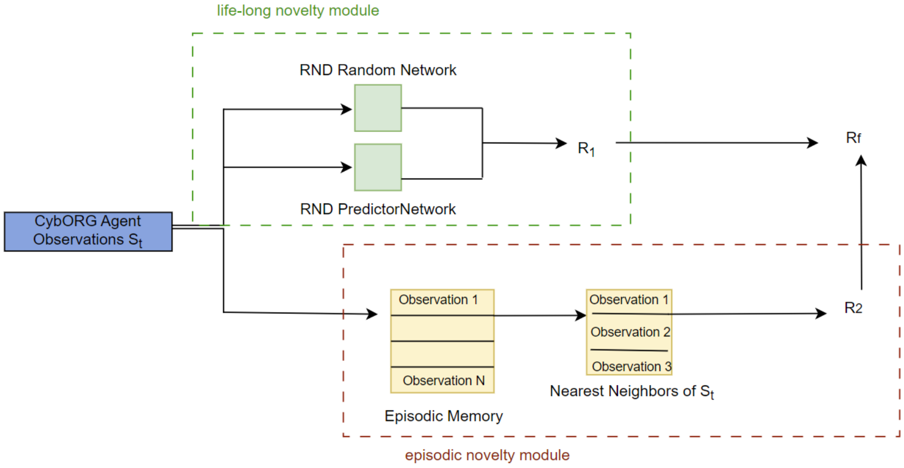


Figure 12: Never Give Up (NGU) module as described in the study of Puigdomènech Badia et al. [16]

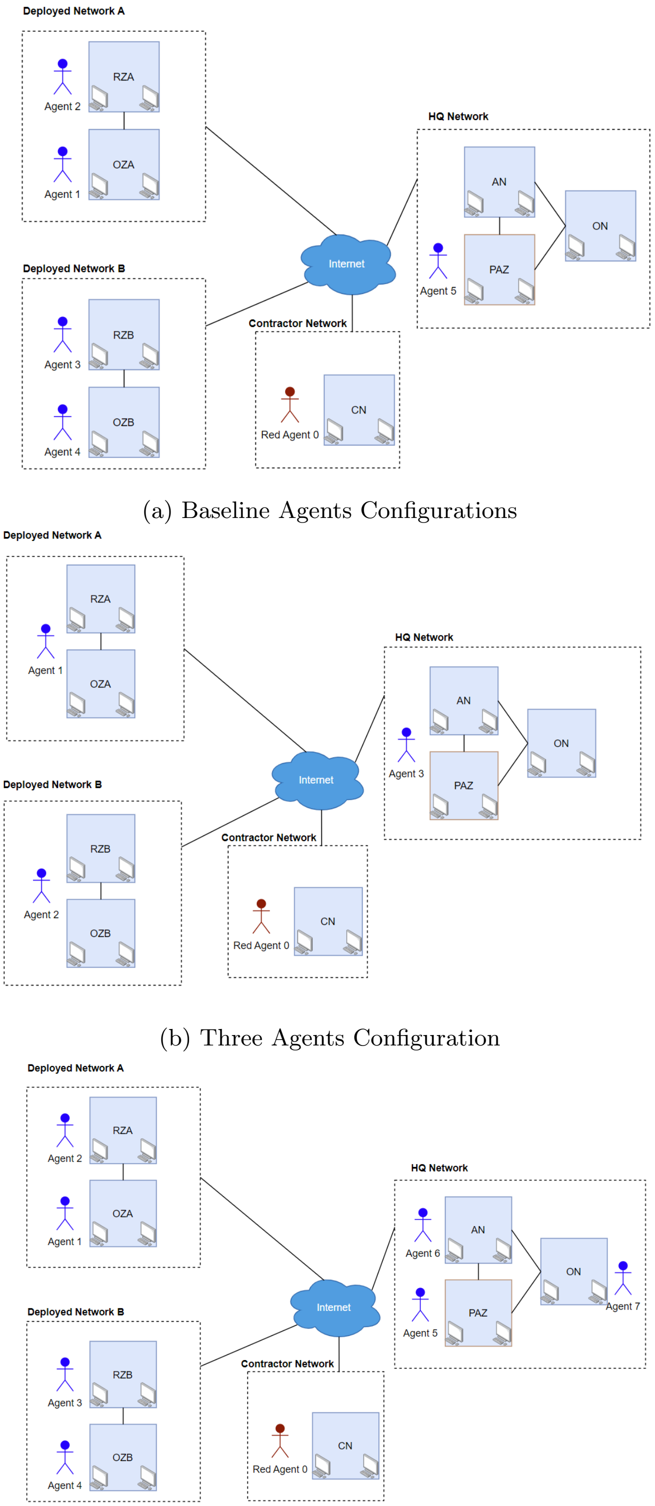
reward structures, here we introduce changes centered on agent configurations, available actions, and network topology.

4.5.1 Agents Configurations

One of the initial modifications to the CybORG environment aimed to examine how the agents' general policy changes when the number of defenders in the network varies. As discussed in Section 3.5, the baseline CybORG environment includes five agents, each responsible for defending specific sections of the network, as illustrated in Figure ??. Notably, Agent 5 manages the defense of three subnetworks, namely the Admin Network (AN), Office Network (ON), and the Public Access Zone (PAZ) which also serves as a point of contact with the rest of the networks. On the other hand, the remaining agents each defend a single subnetwork, consisting of different Restricted Zones (RZA and RZB) and Operational Zones (OZA and OZB). Since there are a total of seven defensible subnetworks in the network topology shown in Figure 13, we propose two new configurations for agents in the CybORG environment. The first configuration reduces the number of agents from five to three, redistributing the network defense as depicted in Figure ??. Here, the first two subnetworks have reduced defense, each protected by only one agent instead of two; henceforth both Restricted and Operational zones in such subnetworks will each be defended by one agent. The Headquarters network remains unchanged. This configuration simplifies the environment's non-stationarity in the first two subnetworks but increases the action and observation space for the two responsible agents. From a cybersecurity perspective, the loss of defending agents could lead to decreased performance against red agents. In contrast, the second configuration involves adding more defender agents, increasing the total from five to seven. The partitioning of network defense in this setup is shown in Figure ??, where Agents 1 through 4 retain their original roles, while Agent 5 now defends only one subnetwork (PAZ), with the remaining two subnetworks (AN and ON) assigned to new agents. Compared to the three-agent setup, this configuration reduces the action and observation space for the Headquarters defender agent but increases the environment's non-stationarity due to the presence of additional agents. In this scenario, we anticipate improved performance relative to the baseline CybORG setup, given the increased number of defenders.

4.5.2 Action Timestep

The actions in CybORG, as described in Section 3.5, each come with specific costs and execution times. As a result, the agent must weigh which actions will yield the best outcomes within the shortest timeframe, potentially leading to policies that prioritize execution time over the optimality of the action itself. But how would the policy shift if agents could select actions with immediate effects instead? We implemented this modification to observe how the policy adapts to the change in action timing and to compare the resulting



(a) Baseline Agents Configurations

(b) Three Agents Configuration

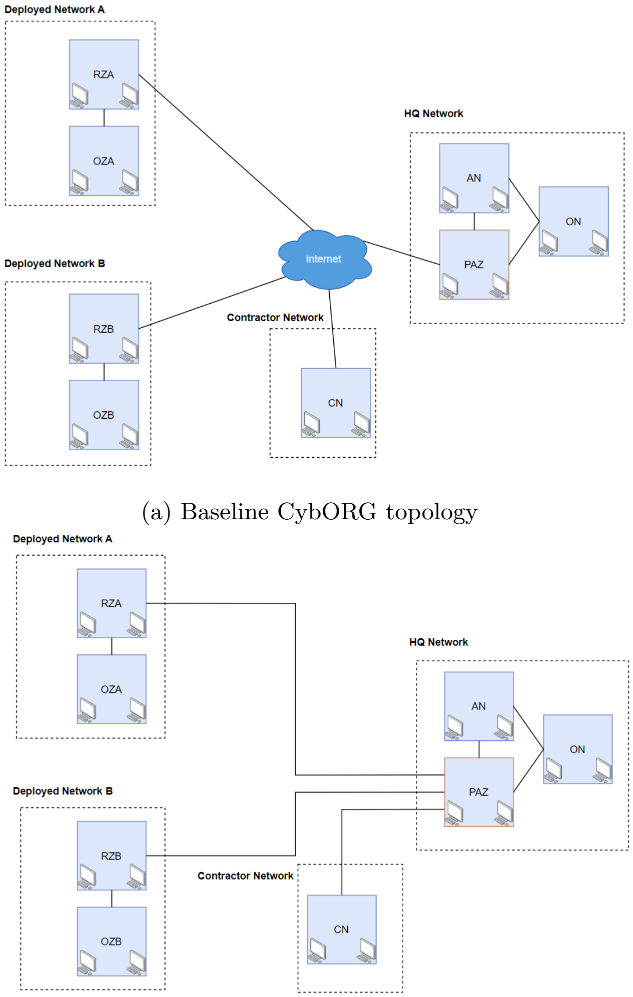
(c) Seven Agents Configuration

Figure 13: Different agents configurations used in the standard CybORG network environment.

strategies. Importantly, to maintain realism in the simulation, we also adjusted the red agents' actions to have immediate effects on the environment.

4.5.3 Network Topology

The final modification to the environment involves altering the underlying environment model used to train the agents. For this change, we focused primarily on the network topology, a crucial factor in network defense. The baseline CybORG topology, depicted in Figure **??**, closely resembles a star topology, with a central internet node connecting various parts of the network. To maintain a solid basis for comparing policy changes in defender agents, we chose not to introduce entirely new topologies, such as bus or ring structures, which would fundamentally alter the network's connections from the CybORG baseline. Instead, we modified the existing star topology by introducing a hub node that can fail. This design emphasizes the central hub as a potential single point of failure, which is an inherent vulnerability in star topologies that often demands heightened defense and maintenance. Currently, the CybORG hub node acts solely as a server node through which all communications pass, and it is unaffected by attacks from red agents. In our modified topology, shown in Figure **??**, we replace this central node with the entire headquarters network. This shift highlights the importance of HQ defense as both a critical communication point and a single point of failure for the entire network, allowing us to observe how defender policies adjust in response.



(a) Baseline CybORG topology

(b) Star Topology CybORG

Figure 14: Comparison of the two different network topologies of CybORG.

5 Results

In the following section, we will discuss the results obtained for *Research Question 2* and *Research Question 3*. In the former, we will analyze the results obtained from the use of different reinforcement learning methods and practices in the CybORG environment. In the latter, the analysis shifts to the learned policy by the agents using the different methods and in the different configurations of the environment. All the results will be thoroughly discussed in Section 6.

5.1 Research Question 2

In this section, we are going to discuss and present the results obtained from employing the methods previously discussed. Such results are obtained on the baseline settings of CybORG as described in Section 3.5.

The hyperparameters used for each of the reinforcement learning methods are shown in Appendix A. Moreover, it is important to note the training of the agents was held in the same CybORG environment with the seed being set to 1 as well as each episode consisting of 500 steps.

Figure 15 shows the results obtained from employing all baseline methods compared to one another. The graph evidences the superior performance of all PPO-related methods over the rest of the implemented methods. Of those policy-gradient methods, we notice how the centralized version of the methods achieves higher initial results to then converge around the same values as the non-centralized ones. Additionally, in Figure 16 we can see how the recurrent version of the methods performs slightly worse compared to the non-recurrent counterpart for all the tested methods. Moreover, Figure 17 illustrates the implementation of various message types and presents the results of applying the methods without any messages. These results demonstrate that the performance of the methods does not improve when messages are added to the agent's observation space. Notably, the 2 Bits message type shows worse results than any other message implementation that, in turn, achieves similar results to the message-less implementations. Those results raise the question of whether the messages, as defined in Section 4.3, are a positive implementation in the baseline CybORG setting.

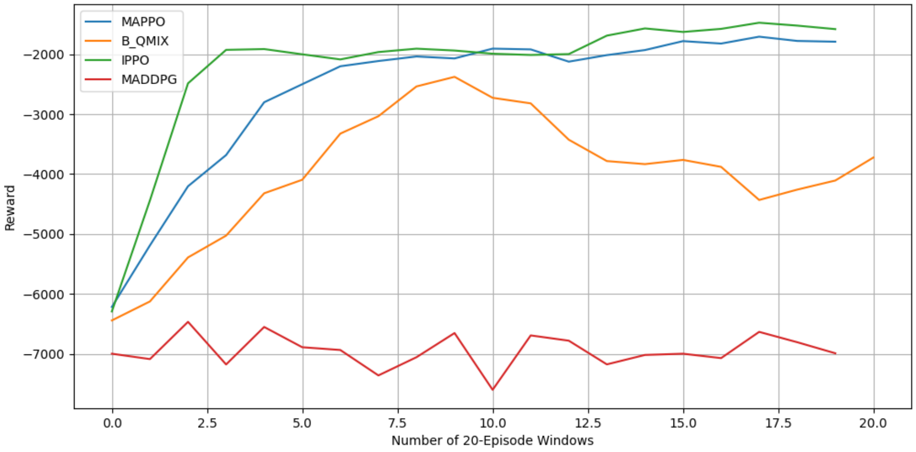
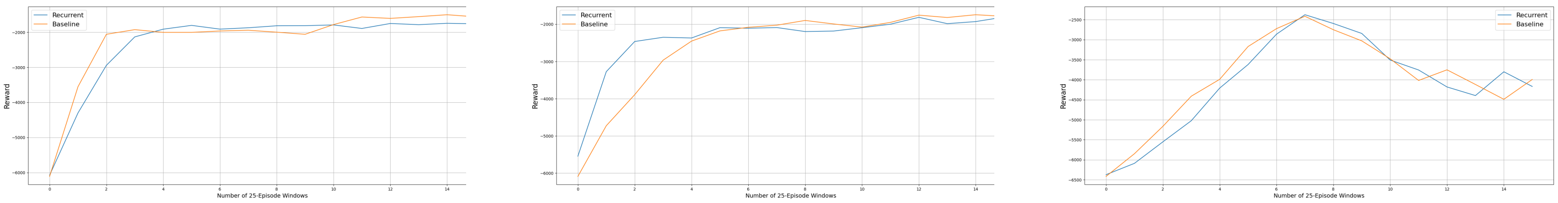


Figure 15: Result obtained from using different reinforcement learning methods in the baseline CybORG environment.

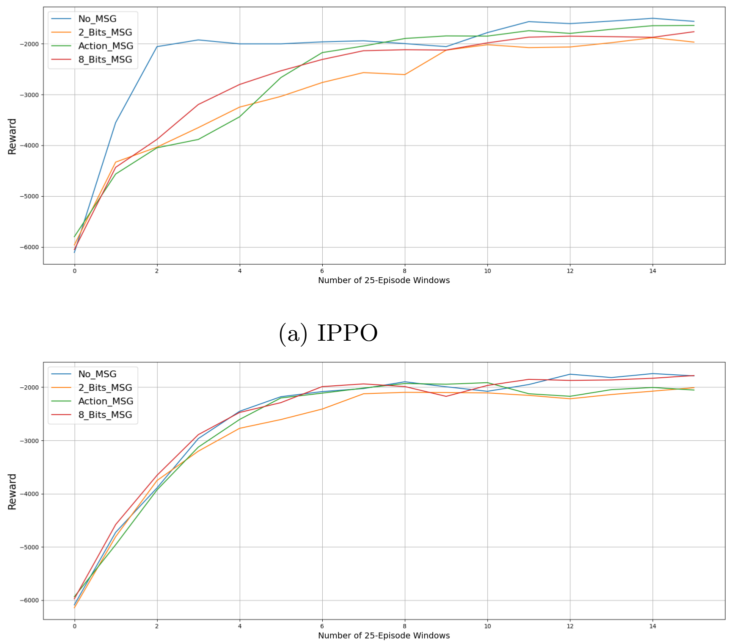
Figures 18 show the results obtained by MADDPG, and QMIX, with the implementation of different replay buffers and exploration strategies respectively.

The results show how neither of those methods of baseline implementation learns in the environment at hand, with them only displaying very marginal improvement in certain training runs. Nonetheless, it can be seen how the use of a different exploration strategy greatly improves the results in the case of QMIX. Similarly, Figure 19 shows the results obtained from MAPPO two different global state representations, both discussed in Section 2.3.1.



(a) IPPO (b) MAPPO (c) QMIX

Figure 16: Result obtained from using different reinforcement learning methods and their recurrent version in the baseline CybORG environment.



(a) IPPO

(b) MAPPO

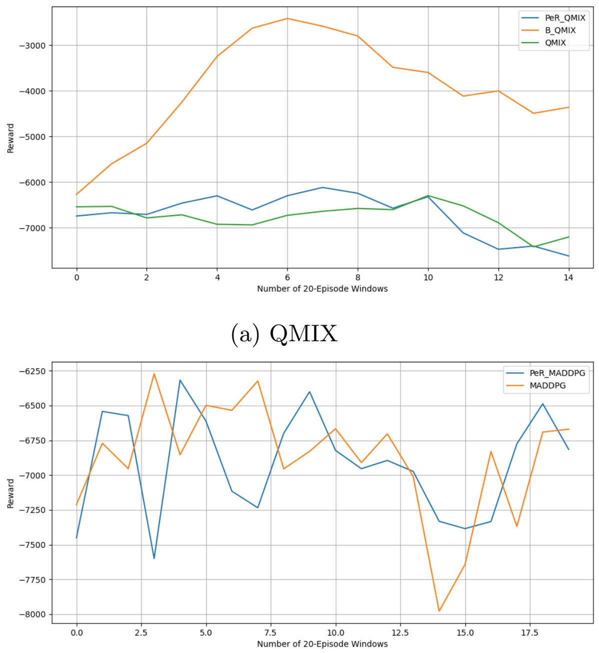
Figure 17: Result obtained from using different reinforcement learning methods with different message versions in the baseline CybORG environment. (**QMIX still need data with Msg to test**)

Lastly, Figure 20 shows the different results obtained from using the IPPO baseline method together with different reward schemas described in Section 4.4. Those results show that the original reward structure produces the best outcomes, and modifications to it do not lead to any improvements. Additionally, the inclusion of the NGU module has a negative effect on the agent's training performance.

5.2 Research Question 3

In this section, we are going to discuss and present the results obtained from analyzing the policy the agent learned in the baseline CybORG environment and its modifications described in Section 4.

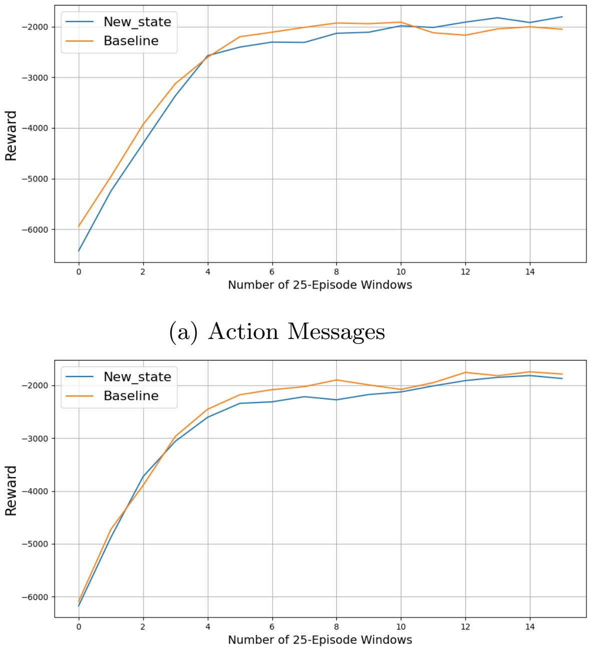
Table 6 shows the most used actions obtained by employing the policy learned by the trained agents in the case with and without messages. Those actions are obtained from the resulting policy of different agents trained through different methods with different message implementations. Moreover, Table 8 shows the actions selected by the blue agent when the red agents execute actions within the defense area of each blue agent. It is important to note that these blue agent actions are not always direct responses to the red team's actions, as certain red team actions may go undetected by the blue agents initially. In this presented scenario, an action-mask was applied to remove the different instances of the Sleep action.



(a) QMIX

(b) MADDPG

Figure 18: Result obtained from using QMIX and MADDPG with different buffer and exploration configurations in the baseline CybORG environment



(a) Action Messages

(b) No Messages

Figure 19: Result obtained from using MAPPO with a different global state representation in the baseline CybORG environment with the inclusion of Action Messages

The aforementioned results concern policy evaluation in the baseline CybORG environment without any

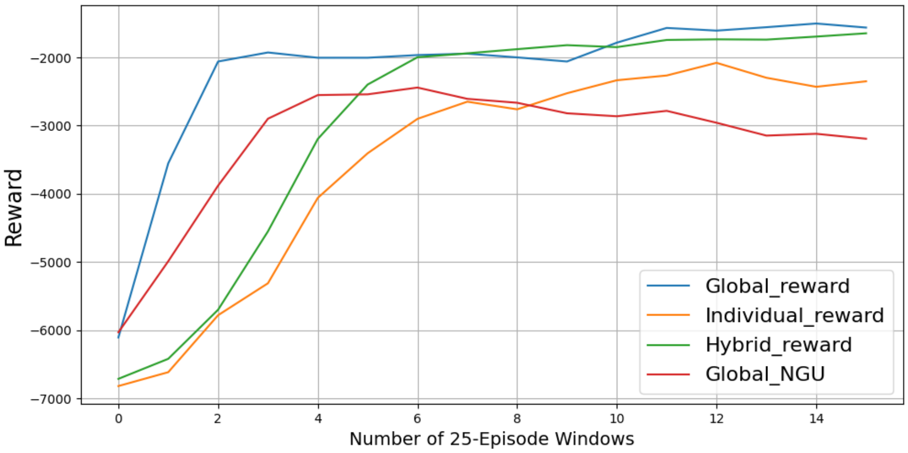


Figure 20: Result obtained from using IPPO with different rewards configurations described in Section 4.4

modifications to its models. In contrast, Table 5 presents the most frequently used actions by agents trained with MAPPO across various environment specifications outlined in Section 4. Additionally, alongside these environment specifications, we include results for agents trained using different reward schemes, as detailed in Section 4.4. Similarly, Table 8 illustrates the correlation between red and blue agent actions under different model configurations. The correlations are expressed as intervals, capturing the maximum and minimum values observed across the various environments. Finally, Figure 21 provides a Finite State Machine representation of the red agents, as introduced earlier in Section 3.5. This figure highlights the most frequently selected defensive actions per state for each environment modification, including the baseline configuration. For each state, the most commonly chosen actions are tallied across model changes and incorporated into the graph. It is important to note that the figure does not depict the totality of actions chosen but rather reflects how consistently agents select the same actions relative to the host state.

Table 5: Table showing the most chosen defensive actions by the blue agents trained with MAPPO in different modifications of the CybORG environment.

	Baseline	**Individual Rewards**	**New Topology**	**7 Agents**	**3 Agents**	**Action Time**
Allow Traffic	25%	**26%**	24%	22%	23%	25%
Analyze	18%	**22%**	17%	17%	17%	**22%**
Remove	18%	15%	**20%**	19%	16%	15%
Deploy Decoy	18%	15%	**23%**	19%	22%	15%
Restore	14%	19%	15%	16%	16%	**20%**
Block Traffic	**4%**	0.1%	0.5%	2%	1%	0.1%
Monitor	**2%**	1%	0.5%	**2%**	1%	1%

Table 6: Table showing the most chosen defensive actions by the blue agents in CybORG trained with different reinforcement learning methods.

	IPPO	**IPPO - Action MSG**	**R_IPPO**	**MAPPO**	**QMIX**
Sleep	32.0%	32.1%	33.3%	31.6%	**40.2%**
Allow Traffic	12.5%	14.2%	**15.6%**	13.6%	13.03%
Analyze	**15.1%**	13.2%	14.0%	12.7%	14.4%
Remove	13.5%	13.0%	**13.6%**	12.5%	11.6%
Deploy Decoy	14.9%	**16.6%**	13.5%	13.6%	10.9%
Restore	**9.7%**	9.2%	8.0%	9.6%	9.5%
Block Traffic	0.7%	0.5%	0.4%	**4.8%**	0.1%
Monitor	**1.7%**	1.2%	1.5%	1.5%	0.2%

Table 7: Table showing the most chosen defensive actions by the blue agents in CybORG, trained with IPPO, in correlation to the actions chosen by the red agents active in their area when the 'Sleep' actions are removed.

	Aggressive Discovery	**Degrade Service**	**Discover Deception**	**Stealth Discovery**	**Exploit Service**	**Impact**	**Privilege Escalation**
Allow Traffic	25.0%	26.9%	24.5%	24.3%	24.5%	26.7%	**29.7%**
Analyze	**18.6%**	17.9%	**18.6%**	18.1%	**18.6%**	17.9%	16.9%
Remove	18.3%	17.9%	18.5%	**18.7%**	18.6%	18.2%	17.0%
Deploy Decoy	18.2%	18.0%	**18.7%**	18.1%	18.6%	17.7%	17.1%
Restore	13.9%	13.3%	13.9%	**14.4%**	14.0%	13.5%	12.9%
Block Traffic	2.9%	2.5%	2.8%	**3.4%**	2.7%	2.5%	2.2%
Monitor	1.2%	1.2%	1.1%	1.2%	1.1%	1.3%	**1.5%**

Table 8: Table showing the most chosen defensive actions by the blue agents in CybORG, trained with MAPPO in different environments, in correlation to the actions chosen by the red agents active in their area when the 'Sleep' actions are removed.

	Aggressive Discovery	**Degrade Service**	**Discover Deception**	**Stealth Discovery**	**Exploit Service**	**Impact**	**Privilege Escalation**
Allow Traffic	21.2 - 25.8%	21.7 - 26.9%	21.2 - 24.5%	20.9-26.3%	20.9 - 24.5%	21.6 - 26.7%	**21.5 - 29.7%**
Analyze	18.6 - 22.5%	17.1 - 24.9%	18.1 - 21.9%	18.1-22.1%	18.2 - 21.6%	**17.9 - 25.2%**	16.9 - 23.2%
Remove	**14.5 - 19.4%**	14.2 - 19.2%	15.1 - 19.3%	**14.2 - 18.7%**	15.3 - 18.6%	15.1 - 18.2%	14.5 - 19.4%
Deploy Decoy	15.1 - 22.7%	**15.3 - 23.0%**	16.5 - 22.5%	16.5 - 22.3%	18.6 - 22.6%	15.5 - 22.1%	15.4 - 22.2%
Restore	13.9 - 19.5%	13.3 - 18.8%	**13.9 - 20.1%**	14.4 - 19.4%	14.0 - 19.9%	13.5 - 18.9%	12.9 - 19.3%
Block Traffic	0.1 - 2.9%	0.1 - 2.5%	0.1 - 2.8%	**0.2 - 3.4%**	0.1 - 2.7%	0.1 - 2.5%	0.1 - 2.2%
Monitor	0.5 - 1.8%	0.5- 2.1%	1.1 - 1.9%	0.5- 1.9%	**1.1 - 2.2%**	0.5 - 1.3%	**0.6 - 2%**

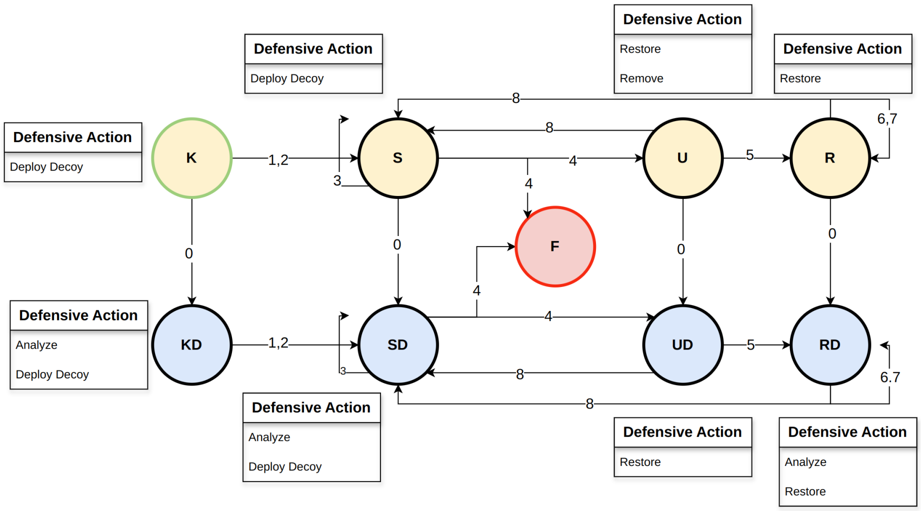


Figure 21: Most chosen action of Blue Agent per each state of the network host as presented in the red agent FSM

6 Discussion

In this section, the results obtained from the first part of the study will be discussed. It is important to note that, especially in the field of Multi-Agent Reinforcement Learning, the specific implementation of a method and the choice of its hyperparameters can significantly change its performance. Therefore, we consider the possibility that some of the implemented methods might lack specific implementation techniques that could enhance their performance in this particular environment. Although the objective of this paper was to apply the current state-of-the-art methods to solve the CybORG environment, we acknowledge the possibility that some methods could achieve better results with alternative configurations or implementations that were not tested. Nevertheless, the results shown in Section 5 can lead us to make some interesting conclusions about the different methods and the types of messages used.

Firstly, the main difference in performance is between the two main classes of algorithms tested. Although there are many different reasons why there can be a discrepancy in the obtained results, we want to highlight some possible factors that led our PPO-based implementations to learn in the environment while the others did not.

6.1 Reinforcement Learning Methods

The first main difference is that PPO and MAPPO are on-policy methods, using data for training that comes directly from the current policy. Conversely, methods like MADDPG, and QMIX are all off-policy methods, employing a replay buffer to store data from past policies.

This leads those methods to be more sample-efficient in theory but can lead to some drawbacks in environments such as CybORG. One can consider that the presence of a large replay buffer might lead to methods relying on old data to perform their policy updates which can lead to sub-optimal policy updates. Furthermore, in highly dynamic environments such as CybORG, the replay buffer may directly hinder the performances of the method, as they might not be able to capture those changes by using data collected on past policies. Although possible, this reasoning is not in line with the findings of our study, as the result presented in the previous section shows clearly how the replay buffer is not the reason for the off-policy methods' inferior performance. The usage of the PeR as described in section 4.1.3 is shown to not bring a performance boost to those methods compared to the use of a standard replay buffer, as previously shown in Figure 18.

Additionally, another important difference between these methods is their exploration strategy. All the PPO-based methods are distinguished by using a stochastic policy which intrinsically includes exploration within them. On the contrary, methods like MADDPG, and QMIX use a deterministic policy, with elements of stochasticity during the exploration phase. This policy can be stochastic during exploration using methods like the Gumbel-Softmax for MADDPG and epsilon-greedy exploration strategies for QMIX, both described in Section 4. While valid, those exploration methods can be less efficient than the inherently stochastic nature of PPO in an environment like CybORG, characterized by partial observability and a degree of unpredictability. Concerning this, the partial observability of the environment can lead to state aliasing which occurs when different states in the environment appear identical to an agent due to partial observability. Using a stochastic policy in that situation can still promote a better exploration of the different actions even in the late stages of the training. Therefore, the flexibility and inherent exploration strategy of PPO can lead it to achieve better results in this specific environment. This hypothesis is further corroborated by the results obtained by employing a different exploration strategy than epsilon-greedy for QMIX. The use of Boltzmann Exploration, described in Section 2.3.3, is shown to bring considerable advantage to the learning of the blue agents when employing QMIX as a reinforcement learning method. The difference, in this case, is significant, as in the CybORG environment, using QMIX with a standard epsilon-greedy strategy resulted in the agents failing to learn anything. However, when Boltzmann softmax was used for exploration, the agents were able to achieve results closer to those of on-policy methods. Similar findings were reported in the studies by Bradley et al. [88] and Azizzadenesheli et al. [15], where both concluded that exploration strategies like Boltzmann softmax and Thompson Sampling outperformed epsilon-greedy strategies in single-agent Atari games. Additionally, Mahajan et al. [56] demonstrated that QMIX with the epsilon-greedy exploration strategy also performed poorly on certain SMAC [72] maps.

In conclusion, we believe that the specific characteristics of PPO contribute to its superior performance compared to the other methods evaluated. Specifically, we have discussed how the exploration strategy is one of the main factors in explaining why PPO excels while other methods perform poorly in the CybORG environment. The study from Yu et al. [108] corroborates our findings, as the authors found that PPO-based solutions often outperform other state-of-the-art methods, including QMIX, in most of their tested multi-agent reinforcement learning environments.

6.2 Centralized and independent learning comparison

A separate analysis should focus on comparing the performance of various PPO variants implemented in this study. Initially, we discuss the performance comparison between independent PPO (IPPO) and its multi-agent counterpart, MAPPO. In this scenario, IPPO slightly outperforms MAPPO.

However, it is important to note that both methods eventually converge similarly. This finding parallels previous studies by Schroeder de Witt et al. [27] and Yu et al. [108], which underscore IPPO's effectiveness in multi-agent environments, particularly highlighting its superior performance over MAPPO in challenges like SMAC challenge [72]. We particularly emphasize again the study by Yu et al., as it serves as a benchmark for comparing the performance of various reinforcement learning methods across different open-source multi-agent environments like SMAC [72] and MPE [60]. The results from their study demonstrate that IPPO and MAPPO yield very similar outcomes in most environments.

The authors discuss also how these results can be partly attributed to the increased complexity introduced by MAPPO's centralized critic, which integrates all individual agent state spaces. While this complexity initially boosts coordination and performance, it may challenge the critic network's ability to learn the diverse features within the combined state space effectively. However, our results indicate that this approach does not affect MAPPO's learning, as demonstrated in Figure 19, at least when using the new global state proposed in Section 2.3.1. While this result is likely due to the new state representation primarily affecting message representation, we also observe no improvement when incorporating messages alongside the new state representation. As discussed in later sections, it is possible that the specific message implementation described in Section 4.3 does not enhance the agents' overall learning, and therefore, the new state representation does not either.

Alternatively, both methods might converge to similar rewards by reaching local optima and encountering challenges in further improvement. Nevertheless, our PPO implementation incorporates methods aimed at addressing such issues, as discussed in the previous section.

6.3 Use of recurrency

Another topic of discussion regarding the results of the various methods implementations concerns the use of a recurrent model to enhance the baseline methods' performance. As discussed in the previous section, while including a recurrent model can be computationally more demanding, it can aid agent learning in partially observable environments.

However, the results obtained in this environment indicate that this benefit is not as straightforward as it might seem. Ni et al. [62] highlight that many studies have reported subpar results from using recurrent policies, attributing these outcomes to the design and implementation of such networks. They also discuss implementation details that can enhance the performance of recurrent methods, most of which have already been covered in Section 4.2. One notable improvement not considered in our recurrent network implementation is the addition of data such as rewards to the agent's input, rather than only providing the state. Nevertheless, Ni et al. emphasize that the effectiveness of different implementation techniques may vary across environments.

In conclusion, the addition of recurrent layers to reinforcement learning methods can be highly dependent on both the model and the method. In this study, we found that the architecture proposed and implemented in Section 4.2 did not yield better results than the baseline.

6.4 Messages Implementation

The final discussion regarding the first part of the obtained results focuses on the use of different messaging systems as described in Section 4.3. The main observation is that agents using 8 Bits and Action messages achieve better results compared to those using 2 Bits messages. Additionally, agents utilizing these message types perform better in the initial stages of training than those using 2 Bits messages. This suggests that the utility of the data exchanged between agents outweighs the increased complexity in their state space. The results indicate that more comprehensive data from other agents leads to better performance than less comprehensive data about the current agent's state. However, it is interesting to note that agents do not see performance improvement by adding these messages, regardless of whether the data pertains to the current observation space (8 Bits messages) or the agent's actions (Action-related messages). Overall, these results can be explained by two main factors. First, the usefulness of the messages: if the data sent is not useful, it will amount to noise and worsen the agents' performance. However, as argued in Section 4.3, we believe the current message implementation is useful, so this might not be the reason. Another possible explanation is that the agents learn sufficiently well on their own, as found by Schroeder de Witt et al. [27] and discussed in the previous section. Finally, the methods themselves might be a factor. For instance, PPO-based methods might reach a local optimum, and the messages may not be sufficient to encourage new exploration or a significant policy change.

In conclusion, our message implementation indicates that expanding the agent's state space can be beneficial if the additional information is valuable. However, implementing messages with the currently used methods in the CybORG environment may not lead to significant improvements in the results.

6.5 Individual Rewards

Another improvement to the CybORG environment was discussed in Section 4.4, focusing on a restructuring of the reward scheme. However, as shown in Figure 20, the new reward structures did not yield improved results compared to the baseline rewards when tested using the Independent version of PPO. The first observation is that agents using individual rewards struggled to learn effectively. Upon closer examination, we found that this was primarily due to the sparsity of rewards given to the agents, particularly those responsible for only one subnetwork, who received zero rewards in up to 70% of cases. Given PPO's reliance on consistent feedback, this reward sparsity hindered the agents from learning the policy well, which explains their poor performance.

On the other hand, when rewards are normalized, as done for all other implementations, we observed a significant improvement, with the performance of individual rewards approaching that of the original global reward scheme.

Additionally, combining the two reward types did not result in better outcomes compared to the original reward. This was unexpected, as we had anticipated that the addition of individual rewards, to provide more feedback on the agents' specific actions, would lead to a general reward improvement. One possible explanation for the lack of improvement is that the change was insufficient to shift the agents away from their current local maxima when both reward types were combined. We had hypothesized that the inclusion of individual rewards would help the agents better understand the consequences of their actions, but the results show that using these rewards, whether entirely or in a hybrid format, did not enhance the agents' overall learning.

6.6 CAGE 4 Results

The final topic of discussion focuses on the results of the CAGE 4 Challenge. At the time of writing, we are aware that the top-performing participants in the challenge have achieved final evaluation scores close to '-100', which are significantly better than the results presented in this study. However, we lack insight into the methods and techniques employed by the participants to reach those scores. It's possible that these results were obtained using approaches other than reinforcement learning or through the use of a different messaging schema than the ones proposed in this study.

As mentioned earlier, our best-performing method (PPO) may have reached a local maximum, preventing further improvement in its reward. Although the challenge results suggest better performance is possible, we have taken steps to mitigate this issue. For example, we incorporated an entropy bonus and learning rate annealing to encourage exploration, as discussed in Section 2.3.1. Additionally, to address the possibility of a local maximum, we introduced the NGU module (described in Section **??**), which adds extrinsic rewards to incentivize exploration. Unfortunately, this addition did not lead to better results and, in fact, slightly worsened the performance. Figure 19 illustrates the changes in IPPO training before and after the NGU module was added. As seen, both iterations of the module resulted in worse results, particularly when the scaling factor for extrinsic rewards was annealed. Consequently, we believe it is unlikely that our PPO implementation is stuck in a local maximum. In fact, the suboptimal performance of the methods in this study may stem from the choice of hyperparameters. The paper by Wang et al. [101] also employs PPO for both individual and multi-agent applications, achieving significantly better results than those presented here. However, a key difference lies in the selection of hyperparameters. This difference in results emphasizes that while reinforcement learning approaches can indeed be effective, hyperparameters play a critical role in determining the ultimate performance of the method, as discussed in Section **??**. Additionally, the study by Singh et al. [86] utilizes PPO as its primary reinforcement learning method, but with a notable distinction: the agents are organized hierarchically based on their decision-making importance within the network. This hierarchical arrangement reportedly yields excellent results, though it represents a slightly different use case due to the hierarchical nature of the solution.

Another potential reason for the superior results achieved by other participants may be the use of a different reward function. This idea was explored further in Section **??**, where we discussed our adjustments to the baseline reward function during the agent's training.

6.7 Agent's Policy

The final point we want to address regarding the agent's results is the policy derived from the agents that performed well in the environment. Specifically, we focus on the policy learned by the agents trained using the different versions of PPO, as well as how messaging impacts that policy.

First, we discuss the agent's action choices in the environment. Looking at Tables 6 and **??** it can be assumed that agents are selecting actions primarily based on how frequently those actions appear in the action set. For instance, the "Sleep" action, which is the most frequently selected action by a large margin (around 30% for most agents), might seem like it's being chosen simply because it appears often. One reason for this is that "Sleep" replaces invalid actions early in training, which depends on the environment's initialization. However, it's worth noting that the proportion of "Sleep" actions present in the agent's action set varies between 8% and 40%, depending on the seed, but all agents stabilize around a 25-35% usage rate of the ”Sleep” action over time.

Furthermore, even though Block and Allow traffic actions are equally represented in the action set, the ”Allow Traffic” action is chosen far more frequently than ”Block Traffic.” This trend holds true across all settings except in the centralized training of PPO, where the ”Block Traffic” action is selected more frequently than in other methods. This may be because ”Block Traffic” has a significant impact on the activity of Green agents and therefore on the reward, meaning agents need better coordination to execute these actions without negatively affecting the reward. Aside from this, the overall policy learned by the various methods presented does not differ significantly, as shown in Table 8.

Additionally, the correlation between Red and Blue actions largely follows the pattern of general action choices. Even when excluding ”Sleep,” the order in which actions are chosen tends to remain consistent. As discussed earlier, a potential issue is that agents do not always become immediately aware of Red team actions within their subnet. We also observed that agent performance slightly improves when ”Sleep” actions are removed, though the average reward increase is typically less than 100 points.

7 Threat to validity

In this section, we address the potential threats to the validity of our study.

The first concern involves the classification of open-source reinforcement learning environments, as discussed in research question 1, particularly in evaluating metrics like flexibility and documentation quality. Objectively assessing these metrics is challenging, as extensive documentation may lack coherence or clarity, affecting its rating in our analysis. Similarly, the second round of classification, which includes more subjective metrics like usability and abstraction level, is also vulnerable to subjective interpretation. As noted in Section 3.3, our review balances objectivity and subjectivity, given that its purpose was not only evaluative but also to select an environment for further research. While we do not consider this a limitation, we highlight it as a validity threat, particularly for researchers looking strictly for objective reviews.

The remaining threats to validity relate to research questions 2 and 3, with implementation details posing the primary risks. As detailed in Section 2, the performance of multi-agent reinforcement learning methods can depend heavily on implementation specifics and selected hyperparameters. To mitigate this, we followed the official pseudocode implementations of each algorithm and used recommended hyperparameters appropriate for our experiments. While we standardized hyperparameters across methods to facilitate a controlled comparison, we recognize that alternative configurations could yield better results. Additionally, minor implementation differences could influence agent performance in specific environments.

The final threat involves the policy analysis conducted following agent training. As discussed in Section 6, deriving a complete policy from a complex multi-agent system is inherently challenging. The extensive observation and action spaces make it impractical to capture all relevant details, especially in a partially observable environment with variable action timesteps. These factors complicate our ability to draw clear correlations between attacker and defender actions. To address this, we included an analysis of network host states, providing a more detailed view of agent policies. Nevertheless, partial observability still presents challenges in making definitive conclusions. We mitigate this by sampling numerous episodes to capture emergent patterns in the agent's policy, and we further supported our analysis by manually reviewing individual defense episodes and noting significant observations. While this manual review is less extensive, it provides additional insight into the agent's policy behavior.

8 Conclusion

References

[1] GitHub - weisong-ucr/MAB-malware: MAB-Malware an open-source reinforcement learning framework to generate AEs for PE malware. We model this problem as a classic multi-armed bandit (MAB) problem, by treating each action-content pair as an independent slot machine. — github.com. `https://github.com/weisong-ucr/MAB-malware`. [Accessed 11-03-2024].

[2] MITRE ATT&amp;CK&xAE; — attack.mitre.org. `https://attack.mitre.org/`. [Accessed 13-03-2024].

[3] Cyber operations research gym. `https://github.com/cage-challenge/CybORG`, 2022. Created by Maxwell Standen, David Bowman, Son Hoang, Toby Richer, Martin Lucas, Richard Van Tassel, Phillip Vu, Mitchell Kiely, KC C., Natalie Konschnik, Joshua Collyer.

[4] Amrin Maria Khan Adawadkar and Nilima Kulkarni. Cyber-security and reinforcement learning — A brief survey. *Engineering Applications of Artificial Intelligence*, 114:105116, September 2022.

[5] Mostofa Ahsan, Kendall E. Nygard, Rahul Gomes, Md Minhaz Chowdhury, Nafiz Rifat, and Jayden F Connolly. Cybersecurity threats and their mitigation approaches using machine learning—a review. *Journal of Cybersecurity and Privacy*, 2(3):527–555, July 2022.

[6] Iman Akbari, Ezzeldin Tahoun, Mohammad A. Salahuddin, Noura Limam, and Raouf Boutaba. Atmos: Autonomous threat mitigation in sdn using reinforcement learning. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2020.

[7] Stefano V Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-agent reinforcement learning: Foundations and modern approaches*. MIT Press, 2024.

[8] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, January 2018.

[9] Alex Andrew, Sam Spillard, Joshua Collyer, and Neil Dhir. Developing optimal causal cyber-defence agents via cyber security simulation. In *Workshop on Machine Learning for Cybersecurity (ML4Cyber)*, 07 2022.

[10] Jayasurya Sevalur Mahendran Ankur Chowdhary. GitHub - ankur8931/asap: Autonomous Security Analysis and Penetration Testing — github.com. `https://github.com/ankur8931/asap`. [Accessed 11-03-2024].

[11] Mohamed A Aref, Sudharman K Jayaweera, and Stephen Machuzak. Multi-agent reinforcement learning based cognitive anti-jamming. In *2017 IEEE wireless communications and networking conference (WCNC)*, pages 1–6. IEEE, 2017.

[12] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. 2017.

[13] Kavosh Asadi and Michael L. Littman. An alternative softmax operator for reinforcement learning, 2017.

[14] Peter Atrazhev and Petr Musilek. It's all about reward: Contrasting joint rewards and individual reward in centralized learning decentralized execution algorithms. *Systems*, 11(4), 2023.

[15] Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9, 2018.

[16] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, and Charles Blundell. Never give up: Learning directed exploration strategies, 2020.

[17] Manoj Basnet and Mohd Hasan Ali. Multi-agent deep reinforcement learning-driven mitigation of adverse effects of cyber-attacks on electric vehicle charging station. *arXiv preprint arXiv:2207.07041*, 2022.

[18] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[19] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation, 2018.

[20] L. Buşoniu, R. Babuška, and B. De Schutter. Multi-agent reinforcement learning: An overview. 310:183–221, 2010.

[21] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. *Multi-agent Reinforcement Learning: An Overview*, pages 183–221. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[22] Valeria Cardellini, Emiliano Casalicchio, Stefano Iannucci, Matteo Lucantonio, Sudip Mittal, Damodar Panigrahi, and Andrea Silvi. irs-partition: An intrusion response system utilizing deep q-networks and system partitions. *SoftwareX*, 19:101120, 2022.

[23] Madeline Cheah, Jack Stone, Peter Haubrick, Samuel Bailey, David Rimmer, Demian Till, Matt Lacey, Jo Kruczynska, and Mark Dorn. Co-decyber: Co-operative decision making for cybersecurity using deep multi-agent reinforcement learning. In Sokratis Katsikas, Habtamu Abie, Silvio Ranise, Luca Verderame, Enrico Cambiaso, Rita Ugarelli, Isabel Praça, Wenjuan Li, Weizhi Meng, Steven Furnell, Basel Katt, Sandeep Pirbhulal, Ankur Shukla, Michele Ianni, Mila Dalla Preda, Kim-Kwang Raymond Choo, Miguel Pupo Correia, Abhishta Abhishta, Giovanni Sileno, Mina Alishahi, Harsha Kalutarage, and Naoto Yanai, editors, *Computer Security. ESORICS 2023 International Workshops*, pages 628–643, Cham, 2024. Springer Nature Switzerland.

[24] Miaojiang Chen, Wei Liu, Ning Zhang, Junling Li, Yingying Ren, Meng Yi, and Anfeng Liu. Gpds: A multi-agent deep reinforcement learning game for anti-jamming secure computing in mec network. *Expert Systems with Applications*, 210:118394, 2022.

[25] Pengcheng Chen, Shichao Liu, Bo Chen, and Li Yu. Multi-agent reinforcement learning for decentralized resilient secondary control of energy storage systems against dos attacks. *IEEE Transactions on Smart Grid*, 13(3):1739–1750, 2022.

[26] Delali Kwasi Dake, James Dzisi Gadze, Griffith Selorm Klogo, and Henry Nunoo-Mensah. Multi-agent reinforcement learning framework in sdn-iot for transient load detection and prevention. *Technologies*, 9(3):44, 2021.

[27] Christian Schröder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *CoRR*, abs/2011.09533, 2020.

[28] Manuel del Verme. GitHub - manuel-delverme/sql_env — github.com. `https://github.com/manuel-delverme/sql_env`. [Accessed 11-03-2024].

[29] Martin Drašar, Ádám Ruman, Pavel Čeleda, and Shanchieh Jay Yang. The road towards autonomous cybersecurity agents: Remedies for simulation environments. In Sokratis Katsikas, Habtamu Abie, Silvio Ranise, Luca Verderame, Enrico Cambiaso, Rita Ugarelli, Isabel Praça, Wenjuan Li, Weizhi Meng, Steven Furnell, Basel Katt, Sandeep Pirbhulal, Ankur Shukla, Michele Ianni, Mila Dalla Preda, Kim-Kwang Raymond Choo, Miguel Pupo Correia, Abhishta Abhishta, Giovanni Sileno, Mina Alishahi, Harsha Kalutarage, and Naoto Yanai, editors, *Computer Security. ESORICS 2023 International Workshops*, pages 738–749, Cham, 2024. Springer Nature Switzerland.

[30] Martin Drašar. CYST-Public / cyst-core · GitLab — gitlab.ics.muni.cz. `https://gitlab.ics.muni.cz/cyst-public/cyst-core`. [Accessed 11-03-2024].

[31] Ashutosh Dutta, Ehab Al-Shaer, Samrat Chatterjee, and Qi Duan. Autonomous cyber defense against dynamic multi-strategy infrastructural ddos attacks. In *2023 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, 2023.

[32] Jianqing Fan, Cong Ma, and Yiqiao Zhong. A selective overview of deep learning. *Statistical Science*, 36(2), May 2021.

[33] Ting-Han Fan, Ta-Chung Chi, Alexander I. Rudnicky, and Peter J Ramadge. Training discrete deep generative models via gapped straight-through estimator. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 6059–6073. PMLR, 17–23 Jul 2022.

[34] Xuecai Feng, Jikai Han, Rui Zhang, Shuo Xu, and Hui Xia. Security defense strategy algorithm for internet of things based on deep reinforcement learning. *High-Confidence Computing*, 4(1):100167, 2024.

[35] Xuecai Feng, Hui Xia, Shuo Xu, Lijuan Xu, and Rui Zhang. Tsgs: Two-stage security game solution based on deep reinforcement learning for internet of things. *Expert Systems with Applications*, 234:120965, 2023.

[36] Yuming Feng, Weizhe Zhang, Shujun Yin, Hao Tang, Yang Xiang, and Yu Zhang. A collaborative stealthy ddos detection method based on reinforcement learning at the edge of internet of things. *IEEE Internet of Things Journal*, 10(20):17934–17948, 2023.

[37] Youssef Fenjiro and Houda Benbrahim. Deep reinforcement learning overview of the state of the art. *Journal of Automation, Mobile Robotics and Intelligent Systems*, 12(3):20–39, December 2018.

[38] Bobby Filar. GitHub - bfilar/malware_rl: Malware Bypass Research using Reinforcement Learning — github.com. `https://github.com/bfilar/malware_rl?tab=readme-ov-file`. [Accessed 11-03-2024].

[39] TTCP CAGE Working Group. Ttcp cage challenge 4. `https://github.com/cage-challenge/cage-challenge-4`, 2023.

[40] Kim Hammar. GitHub - Limmen/gym-optimal-intrusion-response: A Simulated Optimal Intrusion Response Game — github.com. `https://github.com/Limmen/gym-optimal-intrusion-response`. [Accessed 11-03-2024].

[41] Kim Hammar. GitHub - Limmen/awesome-rl-for-cybersecurity: A curated list of resources dedicated to reinforcement learning applied to cyber security. — github.com. `https://github.com/Limmen/awesome-rl-for-cybersecurity?tab=readme-ov-file`, 2022. [Accessed 07-03-2024].

[42] Kim Hammar and Rolf Stadler. Finding Effective Security Strategies through Reinforcement Learning and Self-Play. `https://github.com/Limmen/gym-idsgame`.

[43] Kim Hammar and Rolf Stadler. Intrusion Prevention through Optimal Stopping. `https://github.com/Limmen/csle`.

[44] Johan Backman Hampus Ramström. GitHub - hampusramstrom/gym-threat-defense — github.com. `https://github.com/hampusramstrom/gym-threat-defense`. [Accessed 11-03-2024].

[45] Chris Hicks, Vasilios Mavroudis, Myles Foley, Thomas Davies, Kate Highnam, and Tim Watson. Canaries and whistles: Resilient drone communication networks with (or without) deep reinforcement learning. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, AISec '23, page 91–101, New York, NY, USA, 2023. Association for Computing Machinery.

[46] Zhenguo Hu. GitHub - crond-jaist/AutoPentest-DRL: AutoPentest-DRL: Automated Penetration Testing Using Deep Reinforcement Learning — github.com. `https://github.com/crond-jaist/AutoPentest-DRL`. [Accessed 11-03-2024].

[47] Nemanja Ilić, Dejan Dašić, Miljan Vučetić, Aleksej Makarov, and Ranko Petrović. Distributed web hacking by adaptive consensus-based reinforcement learning. *Artificial Intelligence*, page 104032, 2023.

[48] Jaromír Janisch, Tomáš Pevný, and Viliam Lisý. Nasimemu: Network attack simulator emulator for training agents generalizing to novel scenarios, 2023.

[49] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

[50] Kiarash Kazari, Ezzeldin Shereen, and György Dán. Decentralized anomaly detection in cooperative multi-agent reinforcement learning. In *32nd International Joint Conference on Artificial Intelligence, IJCAI 2023, Macao, China, Aug 19 2023-Aug 25 2023*, pages 162–170. International Joint Conferences on Artificial Intelligence, 2023.

[51] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[52] Li Li, Raed Fayad, and Adrian Taylor. Cygil: A cyber gym for training autonomous agents over emulated network systems. *arXiv preprint arXiv:2109.03331*, 2021.

[53] Yuxi Li. Deep reinforcement learning: An overview. 2017.

[54] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

[55] Ryan Lowe, YI WU, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[56] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[57] Kleanthis Malialis and Daniel Kudenko. Multiagent router throttling: Decentralized coordinated response against ddos attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pages 1551–1556, 2013.

[58] Chris McCarthy. GitHub - Autonomous-Resilient-Cyber-Defence/PrimAITE: ARCD Primary-Level AI Training Environment (PrimAITE) — github.com. `https://github.com/Autonomous-Resilient-Cyber-Defence/PrimAITE`. [Accessed 11-03-2024].

[59] Craig Wampler Mike Kemmer. GitHub - mitre/brawl-public-game-001: Data from a BRAWL Automated Adversary Emulation Exercise — github.com. `https://github.com/mitre/brawl-public-game-001`. [Accessed 11-03-2024].

[60] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[61] Sayak Mukherjee and Veronica Adetola. A secure learning control strategy via dynamic camouflaging for unknown dynamical systems under attacks. In *2021 IEEE Conference on Control Technology and Applications (CCTA)*, pages 905–910, 2021.

[62] Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps, 2022.

[63] Lisa Oakley and Alina Oprea. Qflip: An adaptive reinforcement learning strategy for the flipit security game. In *Proceedings of the 10th Conference on Decision and Game Theory for Security*, pages 364–384. Springer International Publishing, 2019.

[64] Merve Ozkan-Okay, Erdal Akin, ÖMER Aslan, Selahattin Kosunalp, Teodor Iliev, Ivaylo Stoyanov, and Ivan Beloev. A comprehensive survey: Evaluating the efficiency of artificial intelligence and machine learning techniques on cyber security solutions. *IEEE Access*, 12:12229–12256, 2024.

[65] Gregory Palmer, Chris Parry, Daniel J. B. Harrold, and Chris Willis. Deep reinforcement learning for autonomous cyber defence: A survey, 2024.

[66] Martina Panfili, Alessandro Giuseppi, Andrea Fiaschetti, Homoud B. Al-Jibreen, Antonio Pietrabissa, and Franchisco Delli Priscoli. A game-theoretical approach to cyber-security of critical infrastructures based on multi-agent reinforcement learning. In *2018 26th Mediterranean Conference on Control and Automation (MED)*, pages 460–465, 2018.

[67] Paolo Passeri. 2024 cyber attacks statistics. `https://www.hackmageddon.com/2024/03/26/2024-cyber-attacks-statistics/`, Mar 2024.

[68] Aritran Piplai, Mike Anoruo, Kayode Fasaye, Anupam Joshi, Tim Finin, and Ahmad Ridley. Knowledge guided two-player reinforcement learning for cyber attacks and defenses. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1342–1349, 2022.

[69] Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Memory gym: Towards endless tasks to benchmark memory capabilities of agents, 2024.

[70] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning, 2018.

[71] Ciaran Roberts, Sy-Toan Ngo, Alexandre Milesi, Anna Scaglione, Sean Peisert, and Daniel Arnold. Deep reinforcement learning for mitigating cyber-physical der voltage unbalance attacks. In *2021 American Control Conference (ACC)*, pages 2861–2867, 2021.

[72] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge, 2019.

[73] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

[74] Iqbal H. Sarker, Md Hasan Furhad, and Raza Nowrozy. AI-driven cybersecurity: An overview, security intelligence modeling and research directions. *SN Computer Science*, 2(3), March 2021.

[75] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

[76] Thomas Schiller, Bruce Caulkins, Annie S Wu, and Sean Mondesire. Security awareness in smart homes and internet of things networks through swarm-based cybersecurity penetration testing. *Information*, 14(10):536, 2023.

[77] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[78] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[79] Jonathon Schwartz. GitHub - Jjschwartz/NetworkAttackSimulator: An environment for testing AI pentesting agents against a simulated network. — github.com. `https://github.com/Jjschwartz/NetworkAttackSimulator`. [Accessed 11-03-2024].

[80] Arturo Servin and Daniel Kudenko. Multi-agent reinforcement learning for intrusion detection. In *European Symposium on Adaptive Agents and Multi-Agent Systems*, pages 211–223. Springer, 2005.

[81] Shahaboddin Shamshirband, Ahmed Patel, Nor Badrul Anuar, Miss Laiha Mat Kiah, and Ajith Abraham. Cooperative game theoretic approach using fuzzy q-learning for detecting and preventing intrusions in wireless sensor networks. *Engineering Applications of Artificial Intelligence*, 32:228–241, 2014.

[82] Hassam Ullah Sheikh, Mina Razghandi, and Ladislau Boloni. Learning distributed cooperative policies for security games via deep reinforcement learning. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 489–494. IEEE, 2019.

[83] Guochen Shi and Gang He. Collaborative multi-agent reinforcement learning for intrusion detection. In *2021 7th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC)*, pages 245–249, 2021.

[84] Andrea Silvi. GitHub - ansi-code/secureai-java: SecureAI: Deep Reinforcement Learning for Self-Protection in Non-Stationary Cloud Architectures — github.com. `https://github.com/ansi-code/secureai-java`. [Accessed 11-03-2024].

[85] Kyle A Simpson, Simon Rogers, and Dimitrios P Pezaros. Per-host ddos mitigation by direct-control reinforcement learning. *IEEE Transactions on Network and Service Management*, 17(1):103–117, 2019.

[86] Aditya Vikram Singh, Ethan Rathbun, Emma Graham, Lisa Oakley, Simona Boboila, Alina Oprea, and Peter Chin. Hierarchical multi-agent reinforcement learning for cyber network defense, 2024.

[87] Åvald Åslaugson Sommervoll, László Erdődi, and Fabio Massimo Zennaro. Simulating all archetypes of sql injection vulnerability exploitation using reinforcement learning agents. *International Journal of Information Security*, 23(1):225–246, 2024.

[88] Bradly C. Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *CoRR*, abs/1507.00814, 2015.

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

[90] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017.

[91] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018.

[92] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018.

[93] 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.

[94] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

[95] Manikant Thakur. Cyber security threats and countermeasures in digital age. *Journal of Applied Science and Education (JASE)*, 4(1):1–20, Apr. 2024.

[96] Callum Rhys Tilbury, Filippos Christianos, and Stefano V. Albrecht. Revisiting the gumbel-softmax in maddpg, 2023.

[97] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[98] Khuong Tran, Maxwell Standen, Junae Kim, David Bowman, Toby Richer, Ashlesha Akella, and Chin-Teng Lin. Cascaded reinforcement learning agents for large action spaces in autonomous penetration testing. *Applied Sciences*, 12(21), 2022.

[99] Sanyam Vyas, John Hannay, Andrew Bolton, and Professor Pete Burnap. Automated cyber defence: A review, 2023.

[100] Yihao Wan and Tomislav Dragičević. Data-driven cyber-attack detection of intelligent attacks in islanded dc microgrids. *IEEE Transactions on Industrial Electronics*, 70(4):4293–4299, 2023.

[101] Mingjun Wang and Remington Dechene. Multi-agent actor-critics in autonomous cyber defense, 2024.

[102] Jacob Wiebe, Ranwa Al Mallah, and Li Li. Learning cyber defence tactics from scratch with multi-agent reinforcement learning, 2023.

[103] Alec Wilson, Ryan Menzies, Neela Morarji, David Foster, Marco Casassa Mont, Esin Turkbeyler, and Lisa Gralewski. Multi-agent reinforcement learning for maritime operational technology cyber security. *arXiv preprint arXiv:2401.10149*, 2024.

[104] Liang Xiao, Yan Li, Jinliang Liu, and Yifeng Zhao. Power control with reinforcement learning in cooperative cognitive radio networks against jamming. *The Journal of Supercomputing*, 71:3237–3257, 2015.

[105] Xin Xu, Yongqiang Sun, and Zunguo Huang. Defending ddos attacks using hidden markov models and cooperative reinforcement learning. In *Pacific-Asia Workshop on Intelligence and Security Informatics*, pages 196–207. Springer, 2007.

[106] Yizhou Yang and Xin Liu. Behaviour-diverse automatic penetration testing: A curiosity-driven multi-objective deep reinforcement learning approach. *arXiv preprint arXiv:2202.10630*, 2022.

[107] Seunghyun Yoon, Jin-Hee Cho, Dong Seong Kim, Terrence J. Moore, Frederica Free-Nelson, and Hyuk Lim. Desolater: Deep reinforcement learning-based resource allocation and moving target defense deployment framework. *IEEE Access*, 9:70700–70714, 2021.

[108] 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. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24611–24624. Curran Associates, Inc., 2022.

[109] Huifeng Zhang, Dong Yue, Chunxia Dou, and Gerhard P Hancke. Resilient optimal defensive strategy of micro-grids system via distributed deep reinforcement learning approach against fdi attack. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[110] Tianqing Zhu, Dayong Ye, Zishuo Cheng, Wanlei Zhou, and S Yu Philip. Learning games for defending advanced persistent threats in cyber systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(4):2410–2422, 2022.

Appendix A Appendix