LEARNING CYBER DEFENCE TACTICS FROM SCRATCH WITH COOPERATIVE MULTI-AGENT REINFORCEMENT LEARNING

FORMATION AUTONOME DE TACTIQUES DE CYBERDÉFENSE À L'AIDE DE L'APPRENTISSAGE PAR RENFORCEMENT MULTI-AGENT COOPÉRATIF

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

Jacob Wiebe
Captain

In Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science

March, 2023

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Ranwa Al Mallah, for her mentorship. I would also like to thank our DRDC colleagues, Dr. Li Li and Dr. Adrian Taylor, for their guidance and expertise.

Abstract

Recent advancements in deep learning techniques have opened new possibilities for designing solutions for autonomous cyber operations. Teams of intelligent agents in computer network defence roles may reveal promising avenues to safeguard cyber and kinetic assets. The aim of this thesis is to provide evidence to support or refute the applicability of cooperative MARL to a range of tactical cyber defence tasks. In a simulated game environment, agents are evaluated on their ability to jointly mitigate attacker activity in a host-based defence scenario. The complex and interrelated effects of game design elements on the performance of learning systems are explored. The results demonstrate the adaptability of MARL systems to learn in the context of various game objectives and network sizes while being sufficiently robust to perform in large, dynamic problem spaces.

Résumé

Les récents progrès des techniques d'apprentissage en profondeur ont ouvert de nombreuses nouvelles possibilités dans le domaine des cyber opérations. Des groupes d'agents autonomes et intelligents chargés de défendre un réseau informatique pourraient ouvrir de nouvelles voies pour protéger les actifs cyber et cinétiques. L'objectif de cette thèse est de démontrer l'applicabilité de l'Apprentissage par Renforcement Multi-Agent (ARMA) coopératif à une gamme de tâches tactiques en cyberdéfense. Dans un environnement de jeu simulé, les agents sont évalués selon leur capacité d'atténuer conjointement l'activité des menances provoquées par un attaquant dans un scénario visant la défense de l'hôte. Les effets des différents paramètres du jeu sur la performance des systèmes d'apprentissage sont complexes et interdépendants et seront explorés. Les résultats démontrent l'adaptabilité des systèmes ARMA à apprendre dans un contexte de divers tailles de réseau et d'objectifs de jeu tout en étant suffisamment robustes pour les espaces de problèmes qui soient grands et dynamiques.

Contents

List of Tables

List of Figures

1 Introduction

Within the field of cybersecurity, there is an ever-growing number of complex problems requiring proportionally complex solutions. Demand for highly skilled and experienced cybersecurity professionals in roles such as penetration testing, digital forensics, cyber threat hunting, and incident response is increasing. Meanwhile, machine learning and deep learning have risen as dominant technologies in the space of complex problem-solving. The emerging field of autonomous cyber operations has sought to harness the potential of Reinforcement Learning (RL), deep learning, and multi-agent systems to model the complexity of the cyber battlespace.

Autonomous Cyber Operations (ACO) utilizes a computer network environment in which two opposing agents or teams of agents compete to realize their goals of either attack or defence using only information from within the network environment. For attacker agents, the goal is to penetrate the network to access or deny access to some high-value element. For defender agents, the goal is to prevent attacker activity on the network. By modelling this environment as a game with specified actions, states, and rewards, autonomous RL agents can learn to optimize the decision-making process without a prior understanding of the game mechanics. Moreover, multiple learning agents can share individual information to cooperate toward common goals.

This research presents the design of cooperative Multi-Agent Reinforcement Learning (MARL) systems to learn tactical decision-making for cyber defence. It identifies key advancements in the rapidly developing field of MARL game design and leverages these approaches to evaluate their applicability to cyber defence scenarios.

1.1 Motivation

A cyber defence analyst will typically employ a chain of decisions leading them from the first discovery of a threat to incident response or actions to

mitigate it. Human experts are the best tool available for performing this chain of decision-making. However, human ability in cyber defence tasks faces the challenges of, among others, attention allocation, cognitive load, lack of measurable impact, and reaction time [1]. RL has the potential to overcome these limitations if systems can be trained to perform decision-making tasks to a sufficiently high level. Perhaps a more near-term objective is machine-assisted decision-making for cyber defence, in which algorithms can provide tactical suggestions that may not have been obvious to human analysts.

It is unrealistic for current RL algorithms to learn tactical decision-making from scratch in real-world scenarios. Enterprise networks play an essential role in business operations and are not conducive to training iterative learning programs or experimenting with cyber threats. Moreover, modern networks contain massive complexity, and running RL experiments using real hardware is prohibitively expensive. Instead, a simulated environment can be used for RL systems to learn tactics from abstracted network information. Such an environment does not require learning agents to parse directly from raw data but instead to access host-based information akin to what a human analyst may have, such as running sessions, processes, and open ports. The agent could then decide to perform further analysis and take defensive actions. Given a suspected intrusion, the RL agent could reboot hosts, reset services, or create decoys. An RL agent can only learn to perform actions if it has an indication of its goals. In the case of a cyber attack on a network, minimizing attacker activity, such as attacker observation time, hosts infected, or files tampered with, is the objective.

A significant challenge RL systems face in cyber defence is learning to take actions using a large, dynamic input space. One promising advancement in the field of RL is the use of multiple autonomous agents working collaboratively to solve complex problems, referred to as cooperative MARL. The cooperative MARL approach allows for the division of large action and observation spaces, thus reducing the dimensionality of the problem space. Moreover, it allows for the division of complex tasks into decentralized, independent agents, thus enabling more robust learned policies. This research utilizes cooperative MARL to perform tactical-level decision-making in a host-based cyber defence simulation.

Exploration of various design strategies in the learning system and the game environment can provide novel insights into how cooperative MARL may be employed in a tactical cyber defence setting. This research is motivated by the potential for state-of-the-art methods to learn collaborative cyber defence tactics in progressively more realistic scenarios.

1.2 Statement of Deficiency

Recently, many novel algorithms emerging within the field of deep cooperative MARL have demonstrated increasingly performant skill adoption and competence at a variety of tasks. Work that has sought to leverage cooperative MARL in a cyber context has predominantly focused on a specific implementation of the offensive or defensive arsenal, such as autonomous penetration testing or anomaly detection. Studies into the more general problem of autonomous tactical decision-making for cyber defence have seen success using single-agent RL games. No empirical study currently exists investigating the merits of cooperative MARL as a solution to the tactical enterprise network defence problem.

1.3 Aim

The aim of this research is to demonstrate the applicability of cooperative MARL for tactical-level decision-making in cyber defence. The success of this aim is contingent on the provision of empirical evidence to support or refute the applicability of this approach to a range of tactical cyber defence tasks. Applicability in this context is the ability of a MARL system to learn to perform a variety of tasks that inform its potential utility in cyber defence.

This experimentation evaluates the ability of cooperative MARL to develop a set of decentralized control policies that can collectively defend a simulated network. To assess applicability, game design parameters will be varied based on two factors: adaptability and complexity. Applicable MARL systems must show the ability to learn many types of tasks from scratch. Likewise, complex tasks are necessary to show the potential transferability of these results to more realistic settings. A MARL game implementation is presented that enables agents to learn from simulated host-based monitoring data and to take tactical-level actions to prevent and mitigate malicious activity.

This research is concerned with the evaluation of current state-of-the-art MARL techniques in a cyber defence simulation by investigating game design elements. It does not introduce novel algorithm design or cyber defence tactics, techniques, and procedures (TTP). Instead, it provides an analysis of game design applied to a cyber defence context for training MARL systems. MARL systems do not have any *a priori* knowledge of the environment. They must learn to make decisions from only the actions, observations, and rewards provided by the game environment. Additionally, all learning is conducted online in simulation and in the presence of a heuristic attacker agent. The

product of this research is a MARL system capable of learning various challenging cyber defence scenarios to inform the future design of training games for autonomous cyber operations.

1.4 Summary of Results

The cooperative MARL systems used in this evaluation outperformed a heuristic defender at multi-agent tactical cyber defence in a range of network sizes and against multiple attacker types. A centralized learning method showed improved performance and greater robustness against complexity than a system of independent agents. The behaviour of MARL systems had a low sensitivity to changing observations, suggesting that a less sparse input space could benefit learning.

1.5 Organization

The remainder of this thesis is organized as follows. Chapter 2 discusses the background of RL, cooperative MARL, and autonomous cyber defence. Chapter 3 reviews related work in autonomous and multi-agent applications for cyber defence and advancements in cooperative MARL. Chapter 4 describes the methodology for evaluating the selected MARL systems. Chapter 5 presents the results and discussion of the experimentation. Chapter 6 outlines this work's contributions and suggests future work, and Chapter 7 concludes.

2 Background

This chapter presents the foundational concepts of RL, deep learning, MARL, and autonomous cyber defence as they relate to the research objective. For a learning system to provide utility in tactical cyber defence, it must be able to interpret its environment and derive a logical chain of decisions. RL selectively reinforces patterns of behaviour that maximize its expected reward each episode. This research intersects two main fields of study: reinforcement learning and computer network security. Specifically, it investigates cooperative multi-agent deep reinforcement learning methods as a solution to tactical-level decision-making for autonomous cyber defence games. RL is a branch of machine learning occupied with the problem of sequential decision-making. The learner, an autonomous agent, takes sequential actions given inputs from its environment. The environment provides an observation and possible actions to the agent, and after the agent takes an action, it provides a reward and a new observation. The reward may take any scalar value and allows the agent to evaluate its behaviour. The agent's goal, therefore, is to choose actions that maximize its cumulative reward, referred to as *return*. RL games can be used to model cybersecurity problems for autonomous learning. This research applies the RL framework to a set of multi-agent tactical cyber defence tasks.

This chapter is broken down into the following sections: Section 2.1 presents the foundational concepts for RL used in this research. Section 2.2 explores the use of deep learning to augment RL capability. Section 2.3 describes partial observability in the environment. Section 2.4 presents the challenge of stochasticity. Section 2.5 describes progressive advancements in value-based cooperative MARL. Section 2.6 discusses the challenge of autonomous cyber defence, and Section 2.7 provides a summary.

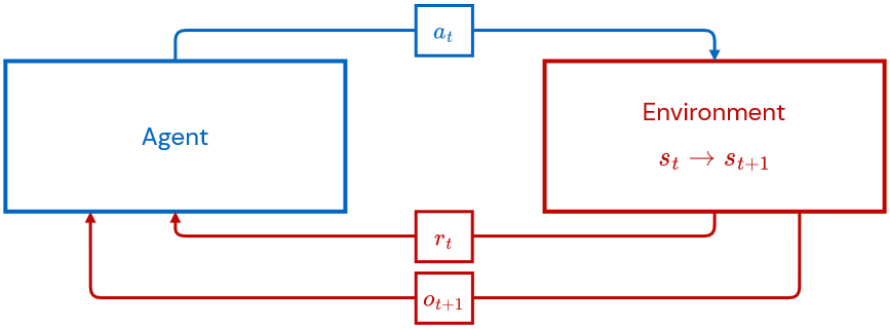


Figure 2.1: RL agent-environment framework.

2.1 The RL Framework

Figure 2.1 illustrates the agent taking action a_t causing the environment to step to the next state s_{t+1} based on a transition probability p, where $p(s_{t+1}|s_t, a_t)$ is the transition function. The environment generates two outputs from the state transition: the reward $r(s, a)$ as a function of the state and action at time $t-1$, and the observation $o(s)$ as a function of the state at time t. The observation may be the entire state or specific information determined by the observation function. In either case, the agent will interpret the observation o_t as the state s_t. This framework is formally a Markov Decision Process (MDP) and serves as a foundation for designing and evaluating learning[1]. A *game* generalizes the MDP and refers to an environment with a consistent set of rules in which one or more agents interact.

An agent's *policy* $\pi(a|o)$ is a mapping of actions to the observation space. The policy, therefore, determines a series of actions for an agent to take. A deterministic policy maps a single action to each state. Hence, an agent following a deterministic policy will always take a given action a for a corresponding observation o. A stochastic policy maps an action probability over the observation space. An agent following a stochastic policy will have a set chance of taking action a for a given observation o.

RL tasks have one or more optimal policies π^* that maximize return when acted upon by an agent. An agent can learn to solve a problem by adjusting its policy based on its received rewards until its policy converges to the optimal policy. An optimal policy is determined when no change in policy will cause an improvement to the agent's return [2]. In complex practical applications, discovering an optimal policy is often difficult, and it suffices to converge on a

[1]In the case where the observation does not include the entire state information, the framework is typically referred to as a *Partially-Observable* MDP (POMDP).

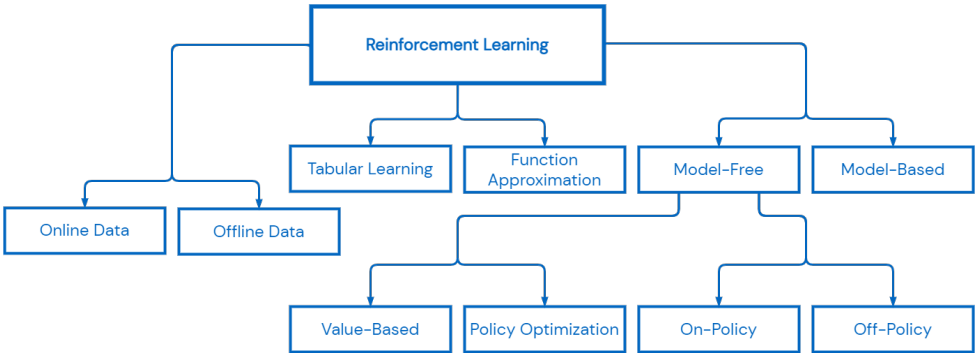


Figure 2.2: Overview of RL algorithm methods.

functional policy that may not be optimal. When evaluating learning system performance, agents act in accordance with a static learned policy.

Agents will select actions at each timestep for a fixed period, referred to as an *episode* of play. A game can be played over multiple episodes, each consisting of a series of turns. Upon completion of each episode, the game is reset. There are many possible approaches to optimizing RL tasks using this framework, this research examines methods that are model-free, value-based, off-policy, and use online learning [2]. Each of these terms is discussed and contrasted to alternative approaches in this section; see Figure 2.2 for an overview of RL algorithm methods.

2.1.1 Model-Free Learning

To learn what series of actions will produce favourable results, an agent must explore the environment, receive rewards, and update its understanding. This process occurs iteratively, allowing the agent to refine its knowledge over many steps and episodes of play. An RL algorithm is a process in which an agent or a set of agents develop a policy representation and evaluate the policy's utility, referred to as *control* and *prediction*, respectively. In model-free learning, agents learn directly from experience with the environment. Agents learn to optimize their behaviour to complete a task by taking actions and observing how they impact rewards.

Conversely, in model-based learning, the agent's task is to approximate a model of the game mechanics with which to derive a policy. Model-based learning typically requires less interaction with the environment and is often more sample-efficient than model-free methods. The end-state of both types of RL is to approximate the optimal policy. Model-based learning approaches this

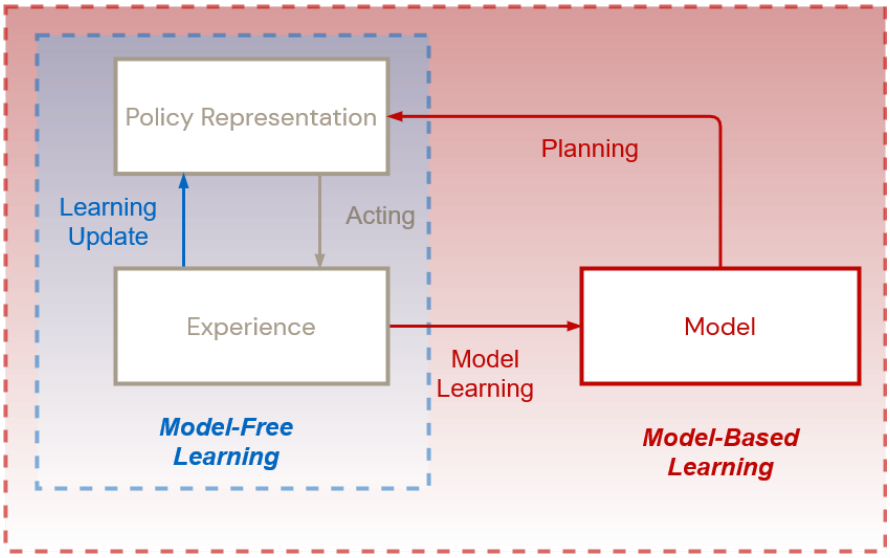


Figure 2.3: Comparison of Model-Free and Model-Based learning steps.

objective indirectly through model learning, depicted in Figure 2.3. Model-based methods are generally more complex in implementation than model-free approaches, owing to an indirect method of policy optimization [2]. A popular example of model-based learning is the Monte-Carlo Tree Search used to great success in the game of Go [3]. This method performs look-ahead searches to plan its series of future actions. Model-based learning may have benefits in cyber defence applications where training data is limited [4]. However, the gains in sample efficiency are less relevant for a simulated environment that can generate millions of samples at a relatively low computational cost. Research into model-based methods is left for future work.

Model-free algorithms have seen widespread adoption across a range of cybersecurity applications and in cooperative MARL, and are discussed in Chapter 3. These methods do not specifically address enterprise network defence using MARL. Due to promising results in model-free learning and the availability of training data from simulation, it is best suited for this experimentation of cooperative MARL for tactical cyber defence.

2.1.2 Value-Based RL

Value-based RL methods derive their policy from a value function. An agent will predict the value of states or state-action pairs using its value function and decide on an action that will maximize its expected future return. The value

of state s under policy π is the expected sum of discounted future rewards r, formally:

$$V_\pi(s) = \mathbb{E}_\pi[\sum_{k=1}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi] \tag{2.1}$$

Where the discount rate ($\gamma \in (0, 1)$) weights rewards less as the agent looks to further k states. This value function, adapted from the Bellman Equation [5], can be extended to a definition of the value of taking an action given an observation, the *Q-value*:

$$Q_\pi(o, a) = \mathbb{E}_\pi[\sum_{k=1}^{\infty} \gamma^k r_{t+k} | o_t = o, a_t = a, \pi] \tag{2.2}$$

A value function based on the observed state ($V(o)$) or based on state-action pairs ($Q(o, a)$) can be used for value-based RL prediction. The action-value function, also referred to as the *Q-function*, is necessary to compute a policy in model-free learning [2]. The algorithms discussed in this work use Q-functions to represent value. A deterministic policy π is derived from the Q-function by selecting the action with the highest Q-value at each observed state:

$$\pi(o) = \arg\max_{a \in A} Q(o, a) \tag{2.3}$$

There exist one or more value functions that result in an optimal policy. Therefore, the value-based learner's objective is to discover an optimal action-value function $Q^*(o, a)$ from which to derive its policy. These concepts form the central mechanism of Q-learning, an RL algorithm that provides much of the foundation for modern methods [6]. A basic Q-learning agent will maintain a table of Q-values for each state-action pair in the environment. In an MDP framework, a Q-learning agent can approach the optimal set of Q-values Q^* by iteratively improving its estimated Q-values using the equation:

$$Q(o, a) = \alpha[r + \gamma \max_{a_{t+1}} Q(o_{t+1}, a_{t+1}) + (1 - \alpha)Q(o, a)] \tag{2.4}$$

where $\alpha \in (0, 1)$ is the *learning rate*. To compute the new Q-value, the agent takes action a_t given its observation o_t, and the environment returns the next observation o_{t+1} and reward r. The agent then updates its Q-value for the observation and action at time t by using the reward and the maximum Q-value for its new observed state as inputs.

In contrast to value-based methods, policy optimization methods iteratively update a parameterized policy directly without intermediate value functions. It is commonly updated using gradient ascent. Adapted from the REINFORCE algorithm definition [7], using the derivation from OpenAI *Spinning Up* [8], the update to the policy function $J(\pi)$ can be written as:

$$\nabla_\theta J(\pi) = \nabla_\theta \log \pi(a_t|o_t;\theta) R(\tau) \tag{2.5}$$

Where θ represents the set of parameters for policy π and $R(\tau)$ is the cumulative discounted reward over the historical trajectory τ from following policy π. Policy optimization methods offer advantages in their ability to approximate optimal stochastic policies and to intrinsically learn exploration, which are not possible using a standard value-based method as described here [2]. Policy optimization methods can suffer from high variance [9], although this has been mitigated by clipping the output of the policy probability ratio, thus restricting the gradient update to within a certain proximity [10]. Moreover, policy-optimization methods can be more sample-efficient than value-based methods. In the multi-agent setting, the class of policy-based, actor-critic algorithms commonly demonstrate high performance, however, value-based MARL algorithms such as QMIX [11] have shown comparable results [12]. Only value-based MARL methods are examined in this work. This allows for direct comparison of Q-learning-based algorithms using fully-decentralized and centralized learning architectures, discussed in greater detail in Section 2.5.

2.1.3 Off-Policy Learning

An agent that always chooses the highest Q-value is said to act greedily. While an agent is learning a policy, greedy behaviour tends to reinforce a bias towards previously seen state-action pairs with known good outcomes rather than exploring unseen state-action pairs in search of greater rewards. A common strategy to encourage exploration is to set a probability ϵ that an agent will randomly select an action rather than taking the action with the highest expected reward, known as the ϵ-greedy strategy. An ϵ-greedy agent is an example of *off-policy* learning since the learned *target policy* approximates the optimal policy but is different from the *behaviour policy* that the agent follows. More generally, off-policy learning characterizes a method in which the data used by an agent to make decisions is not identical to the data used to evaluate a policy or value function [2].

Off-policy algorithms commonly learn from sampling experience replay [13] in randomly sampled batches. Experience replay often consists of the

action, observation, and Q-value. Using replay can improve sample efficiency by exposing the agent to more learning samples without generating them through actions. However, random sampling from historical data without accounting for the changing policy induces bias [14]. The target value function is typically averaged over the behaviour policy state distribution to avoid bias toward any particular state-action pairs. Prioritizing specific replay samples based on the magnitude of the error it produced when encountered (i.e., the *Temporal Difference (TD) error*) has also been used to improve learning for off-policy methods [15].

In complex environments, on-policy learning solely from direct interaction is problematic due to a lack of generalizability. Learning off-policy from past experience is an effective strategy to generalize a value function to a large or highly stochastic state space. Scalability of the problem space is critical in the design of realistic tactical cyber defence games. Off-policy learning has clear advantages over on-policy learning in more complex environments.

2.1.4 Online Learning

Online learning is the general case in which an agent interacts with a dynamic environment that generates state transitions in an episode of play. Conversely, offline learning refers to an RL agent interacting with a limited dataset of state information. In the offline environment, an agent risks overfitting its policy representation to the dataset. Moreover, finite samples also induce bias since the learner approximates its policy representation based on available data, which does not provide a complete model of the MDP [16]. The constraints of offline learning can be avoided by using a simulated environment that can generate infinite samples. Many real cyber applications cannot be accurately modelled using simulation. Instead, offline datasets pulled from real settings can be used to train or evaluate RL systems. The representational capacity of current RL methods is a limiting factor in learning from complex environments. The intent of this research is to train RL systems to take high-level tactical actions. Therefore, an abstract simulation is better suited to the objective than training from offline data.

2.1.5 Function Approximation

Function approximation iteratively improves a policy representation using a parameterized function, rather than by storing individual data points, such as Q-values for each visited state. As the size of the environment's state space increases to accommodate more complex problems, tabular methods such as

Q-learning become ineffective due to a lack of generalization and time required for learning [2]. Tabular methods are limited by the number of states they can explore. Since it is unreasonable to rigorously explore large state spaces, agents must rely on information derived from a subset of the state space since predictions are made from individual state encounters. As the size of the state space increases, it becomes less likely that predictions from tabular methods will generalize. In contrast to predicting discrete Q-values, value functions can be parameterized so that a single approximation is relevant to the entire space of Q-values. The parameters θ of the value function can be optimized by iteratively minimizing the loss function $\mathcal{L}(\theta)$. Building on the iterative Q-update from Equation 2.4, the loss function can, for example, use the mean squared error to optimize θ [17]:

$$\mathcal{L}(\theta_t) = [r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{t-1}) - Q(s_t, a_t; \theta_t)]^2 \tag{2.6}$$

Using a parameterized Q-function $Q(s, a; \theta)$ allows for each sample to influence the entire function, improving the efficiency and generalizability of the learning algorithm. In practice, using a single function approximator for learning a value function tends to be unstable due to correlations between Q-values and the optimization target $r + \gamma Q(s_{t+1}, a_{t+1})$ [17]. Mitigating techniques are discussed in Section 2.2.

2.2 Deep Learning

Deep Learning (DL) enables the representation of complex non-linear functions through a composition of layered interconnected nodes, an artificial Neural Network (NN). DL uses NNs with at least one hidden layer, a layer of nodes that is neither the input nor output. In a basic feed-forward fully-connected NN, a set of scalar inputs passes into the first layer of nodes. Each node computes a weighted sum of its inputs that is then outputted through an activation function producing a non-linear representation at each layer [18]. A forward pass of a data sample through the NN produces an abstraction of the raw input's features. To train a NN, the weights of the nodes are commonly updated by a backpropagation method using gradient descent. Backpropagation is commonly a recursive update function that uses the gradient of the previous layer of nodes to modify the weights at each node. The network can learn features of the input data through iterative forward and backward passes, *prediction* and *adjustment*, respectively. Intuitively, DL is well suited for function approximation in RL problems due to its representational capac-

ity. There are, however, some challenges faced by deep RL methods that must be considered.

Many machine learning and DL methods assume that data is Independent and Identically Distributed (I.I.D.) and stationary. I.I.D. characterizes independent training samples and training and testing data with identical distributions. Independent and identically distributed data allows the training error to accurately represent the expected error in testing [19]. However, the data for training and testing RL systems are generated during play due to the agent's actions. The data are not independent since the rewards are commonly delayed until some game objective is met. As a result, samples are temporally correlated as it is typically the *sequence* of observed states that corresponds to a particular reward. The data in RL environments tend not to be identically distributed since the agent explores the state space unevenly. Moreover, data in RL environments are typically *non-stationary* due to a stochastic transition function, discussed further in Section 2.4.

There are many methods to overcome the instability fundamental to non-linear function approximation in deep RL, characterized in part by non-I.I.D. and non-stationary data. One prominent work that mitigated these challenges is Deep Q-Networks (DQN), which proposed using an experience replay buffer and a secondary NN to improve stability through off-policy learning [17]. The experience replay buffer allows for data to be randomly sampled from history while the secondary network stores a copy of the NN parameters for training, breaking some of the correlation between samples. Experience replay has the effect of alleviating some of the challenges of non-stationary and non-I.I.D. data.

2.3 Partial Observability

Many real-world problems cannot be realistically modelled by providing the agent with complete state information. A central challenge in cyber defence is the classification of threats based on incomplete information. Entire state visibility would give the defender a complete picture of the attacker's status in the environment, eliminating the threat identification aspect of the task. This aspect of the game is essential to consider in RL applicability to tactical cyber defence. Problems can be modelled as *partially observable* in which the observation provided by the environment is a function of the state. Partial observability adds more variability in the observation space, adding more noise than complete observability. In this case, agents must condition on their action-observation history rather than just the last viewed observation [20],

[21]. In a cyber defence setting, the learner could be required to identify samples of data over time that correspond to patterns of malicious activity. Understanding how actions influence these patterns and the task outcome is a key capability that RL provides to autonomous cyber defence.

One method of conditioning a deep RL system on its history is the use of a recurrent neural network (RNN) to capture historical trajectories [22]. RNNs use a looping structure to pass outputs back through hidden layers. An RNN receives input data as a temporal sequence, passing a copy of the weights at the current time t back into the hidden layer, influencing the next input at $t+1$. This allows the model to condition on its history, represented in the recurrent activations in the network. RNNs allow RL models to learn long-term dependencies between data samples which is useful for accurate predictions in partially observable environments [23].

2.4 Stochasticity

Stochasticity refers to the characteristic of an environment to produce non-deterministic outcomes. In a network setting, there is normally some level of uncertainty that a particular action will have a particular effect. Sometimes packets are dropped, or code does not execute correctly or at the precise time. As the vast majority of real-world tasks involve some element of stochasticity, a realistic RL environment will often model this using a stochastic transition function. Stochastic policies, in which action probabilities are mapped to states, can be used in this case to approximate an optimal policy. Optimizing a policy in a stochastic environment is generally more difficult, as predictions are not guaranteed.

2.5 Multi-Agent Reinforcement Learning

The cooperative multi-agent game builds upon the MDP defined previously by allowing multiple agents to take actions and receive rewards simultaneously within a single timestep [24], [25]. Although each agent, denoted by i, receives individual observations o^i from the environment, a joint reward r is determined based on the state and actions of all agents. Agents collectively seek to optimize the joint reward. Figure 2.4 provides a cooperative MARL revision of Figure 2.1. The game's global state depends on the vector of actions $\mathbf{a}_t = \{a_t^{i=1}, ..., a_t^{i=n}\}$ containing n agents. At each timestep, the environment will output a joint reward as a scalar and an observation vector $\mathbf{o}_{t+1} = \{o_{t+1}^{i=1}, ..., o_{t+1}^{i=n}\}$ that includes individual observations for each agent.

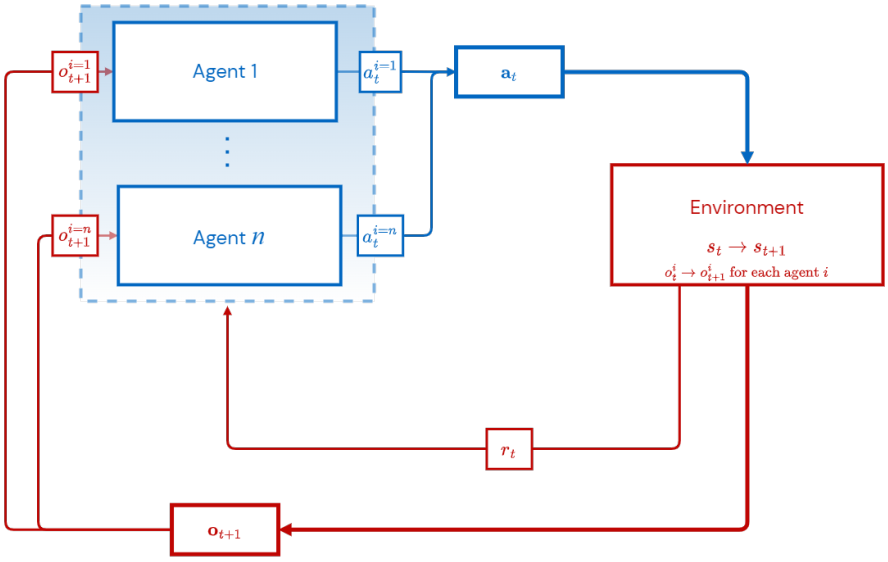


Figure 2.4: Multi-Agent RL Agent-Environment Framework.

Simultaneous actions in a cooperative MARL setting allow for greater exploration. Cooperative MARL also has the advantage of dividing large tasks into manageable goals, thus handling greater task complexity. However, challenges exist in cooperative MARL that are critical to the design of performant solutions. The game's current state depends on the joint actions of all agents since the action vector is an input to the transition function. The joint reward and individual observations agents receive depend on the new state. Therefore, an agent's actions can affect other agents' observations, depending on the observation function. If two agents interact the same game element, such as a host machine, the actions of one agent on the element influence the observations of the other. Since both agents are learning and updating their policies, this interaction creates *multi-agent*-non-stationarity in the environment; the optimal policy is a moving target. In a single-agent game, the fixed probability of a stochastic transition function can be incorporated into the learned policy, for example, with the use of a replay buffer in DQN (see Section 2.2). However, in the multi-agent case, actions of "external agents" affect observations in a way that cannot be explained by changes in the observing agent's policy [26]. It can therefore be challenging for an agent to learn a stable policy when the actions of its peers are silently affecting its reward and observations. As a result, agents tend to exhibit high variance due to inaccuracy in their estimates. The remainder of this section discusses value-based algorithm de-

sign for decentralizing control while highlighting the challenges of cooperative MARL.

2.5.1 Independent Q-Learning

A naive approach to multi-agent systems in terms of cooperation is to decentralize prediction and control, referred to as *independent learning*. This approach employs self-contained RL agents that act out separate policies based on independent observations while seeking to maximize a joint reward. Independent learning does not force any cooperation or account for communication between agents. Independent Q-Learning (IQL) is a fully-decentralized value-based MARL algorithm that uses independent DQN [17] agents. Each agent is responsible for learning a policy from their individual observations. Agents perform learning updates using a batch of experience history $\mathbf{o}^i = \{o_1^i, ..., o_m^i\}$ for a batch of m samples, minimizing the loss of their Q-functions. IQL ignores the non-stationarity problem completely but has performed well at simple tasks [27]. A decentralized approach allows for greater scalability, with the advantage of smaller input-output spaces for each agent, at the cost of high variance due to the non-stationary game. The input-output space refers to the number of inputs to an agent, the size of the observation space, added to the number of outputs, the size of the action space.

2.5.2 Centralized Training with Decentralized Execution

A hybrid structural approach is Centralized Training with Decentralized Execution (CTDE) [26]. A CTDE architecture is compared to independent learning in Figure 2.5. With CTDE, semi-independent agents follow separate policies and receive updates periodically from a central learner. The central learner trains on information from all agents and provides a learning update, allowing each agent's policy to condition on the policies of their peers, mitigating the non-stationarity problem. Using a central learner encourages greater cooperation as agents can learn complementary policies directly.

The addition of a central learner introduces the *credit assignment problem*. Credit assignment refers to agents learning to reinforce certain behaviours that result directly from their individual effect on the joint task. Without credit assignment, the joint reward can reinforce good behaviour in one agent while incentivizing poor performance in another. The poorer performer, sometimes referred to as a "lazy agent," will be discouraged from exploring for better rewards. A naive CTDE approach that provides a single policy update for all agents based on joint training information will suffer from lazy agents since

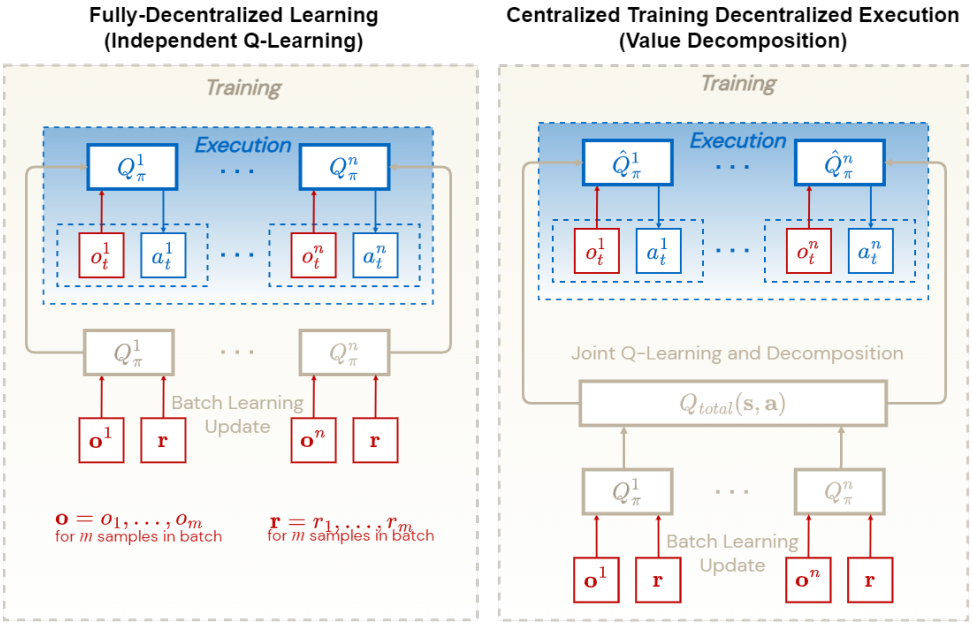


Figure 2.5: Fully-Decentralized Learning and CTDE.

it does not differentiate between agents. A solution to the credit assignment problem is *value-decomposition* which allows a joint reward to be factored into multiple policy representations [28].

2.5.3 Value-Decomposition

Value-Decomposition Networks (VDN) is a CTDE approach that performs Q-function learning centrally before providing separate policy updates to the agents. As shown in Figure 2.5, individual Q-functions are learned from separate observations and are summed together into a joint Q-function (Q_{total}). VDN assumes that Q_{total} can be additively decomposed into individual Q-functions. As a result, distinct functions $\hat{Q}^i(o^i, a^i)$ for each agent i are simultaneously produced at each training iteration. From Equation 2.1, the joint Q-function of the set of observations and actions is:

$$
\begin{aligned}
Q^{\pi}_{total}(\mathbf{o}, \mathbf{a}) &= \mathbb{E}_{\pi}[\sum_{k=1}^{\infty} \gamma^k r_{t+k} | \mathbf{o}_t = \mathbf{o}, \mathbf{a}_t = \mathbf{a}, \pi] \\
&:= \sum_{k=1}^{\infty} \hat{Q}^i_{\pi}(\mathbf{o}, \mathbf{a})
\end{aligned}
\tag{2.7}
$$

Where $\mathbf{s}$ and $\mathbf{a}$ are a vectors such that $\mathbf{o} = \{o^{i=1}, ..., o^{i=n}\}$ and $\mathbf{a} = \{a^{i=1}, ..., a^{i=n}\}$ for n agents. As with IQL, $Q^i(o^i, a^i)$ is calculated first from each agent's independent inputs. Then the Q-functions are combined into Q_{total} from which a learning update is produced. The Q_{total} update is the same as the individual Q-function update but uses the batched history of all agents. Q_{total} is then decomposed into individual $\hat{Q}$-functions through backpropagation to be used as the new decentralized action-value functions. Through this process, each individual function is conditioned on the complete observable state of the team of agents, addressing the non-stationarity problem. Credit assignment is also addressed via the decomposition of Q_{total} as the joint reward is an input to this function. Therefore the reward is represented implicitly in the $\hat{Q}$-function of each agent.

The primary limitation of VDN is that it relies on simple additive decomposition to perform complete factorization. Therefore, its joint Q-function can only represent linear functions. Rashid *et. al* proposed QMIX to improve the representational complexity of the joint value function using a mixing architecture [11].

2.5.4 QMIX

QMIX leverages the advantages of VDN in efficiency, multi-agent conditioning, and credit assignment while improving upon the representational complexity of its Q-functions by introducing a mixing architecture. QMIX, like VDN, utilizes independent DQN-style agents that act according to their respective policies and are trained centrally. As opposed to VDN's summation of Q-functions to generate Q_{total}, QMIX employs a NN, referred to as a *mixing network*, to construct Q_{total} from individual action-value functions. The weights of the mixing network are controlled by the output of hypernetworks: single-layer NNs that take the entire observation space as inputs. QMIX enforces a monotonicity constraint such that $\frac{\partial Q_{total}}{\partial Q^i} \geq 0$ for each i agent. This constraint allows for QMIX to decompose the central Q-function into non-linear $\hat{Q}$-functions. QMIX has greater representational capacity than VDN

which can only represent linear decomposed functions, though the monotonicity constraint limits its range of possible $\hat{Q}$-function outputs. Despite this, Hu *et al.* demonstrate that QMIX often outperforms algorithms with more relaxed monotonicity criteria, if the MARL system can interpret the game objective as purely cooperative [29].

2.6 Autonomous Cyber Defence

This chapter has discussed foundational concepts in RL, deep learning, and cooperative MARL. Autonomous Cyber Operations (ACO) is discussed to justify framing this problem using MARL. ACO is concerned with the decision-making models of attacker and defender agents acting autonomously within a computer network environment [30]. The goal with the design of both agents and environment is two-fold: first, agents must be trainable on a variety of network topologies, and second, the task, and therefore the environment, should represent as much realism as possible.

RL is a natural approach for ACO due to the ability of agents to adapt via interaction with the environment. In contrast to supervised learning methods, which typically require static, pre-established training data, online RL is dynamic. It allows agents to learn not only to make predictions but to take actions and to understand the effect of those actions on the environment over time. The RL approach, therefore, is more relevant to the problem space. A drawback to RL is that it requires large amounts of data to sample. Many thousands or millions of iterations are often required to learn reasonable policies. As a result, complex environments are often hindered by extensive computational resources and clock-time. By simulating ACO, RL systems can be trained over many iterations in an abstraction of the underlying computer network environment. Simulation requires less compute than the alternative emulated approach while still offering enough complexity to challenge current state-of-the-art designs.

Tactical decisions in cyber defence are typically not derived from raw data. Instead, many tools exist to assist an analyst with finding and mitigating threats. ACO does not seek to replace tools that are successful in the field, such as anomaly-based intrusion detection. It instead aims to build a tactical-level decision-making framework to integrate with existing technologies. In this research, the decision space is modelled as a simulation on top of a tool-based abstraction of the network state. An RL simulation for ACO must model the elements of the environment to the level of detail that will allow the agent to learn tactical-level action chains when presented with a series of alerts

about the underlying network status. A multi-agent framework further allows for the decentralization of the decision-making processes while maintaining an input-output space that is constrained to a level that can be learned effectively by current RL methods.

2.7 Summary

The field of RL stretches back into game theory fundamentals while reaching the cutting edge in deep learning techniques. This chapter framed the cyber defence problem within the RL setting and focused on model-free, value-based, cooperative multi-agent learning solutions. Function approximation using deep learning has emerged as the dominant technique for learning policies and value functions. In cooperative MARL, significant advancements have been made in the design of CTDE systems. How these approaches may generalize to a range of specific decision-making applications remains to be seen. A systematic evaluation of game design elements in this work demonstrates their applicability for use in tactical cyber defence.

3 Related Work

This chapter investigates how research into autonomous cybersecurity, multi-agent systems, and deep MARL could inform the design of cooperative MARL solutions to learn cyber defence tactics. To this end, experimental environments are compared qualitatively based on game complexity and task relevance. Game complexity can be assessed in terms of the number and variety of actions, observations, and rewards an agent may encounter via its interaction with the environment. Complexity in the computer network environment refers to the scale and variability of network segments, devices, protocols, applications, and services. Task relevance is explained by the similarity of learnable behaviour to cyber defence operations. For example, agents that must learn to interpret host process connections and can misinform their attacker are more relevant to the target task than agents that only receive a compromised or uncompromised signal for a host and can only perform a reimage action. Complexity and task relevance provide greater realism to the game, allowing for a more robust assessment of MARL applicability to cyber defence tasks.

RL for specific cybersecurity tasks has been studied in the context of DDoS protection [31], anomaly-based intrusion detection [32], and penetration testing [33], [34], among other specific use cases [4]. These works demonstrate the utility of RL system research for cybersecurity, however, they have limited application to the more general problem of tactical-level decision-making for cyber defence. A tactical defender should be able to detect and disrupt attacker activity at multiple possible phases of the attacker kill chain. Specific use cases are generally too narrow in scope to demonstrate utility at managing defensive sensors and actuators. Furthermore, tactical decision making for ACO should involve an element of action selection, in which acting on some object to the exclusion of another. For example, taking an action to gather additional information about a host comes at the cost of the lost opportunity to affect the state of a different host. Designing a game with tactical action selection tradeoffs requires actions with parameters for different targets.

Although penetration testing methods provide targeted actions, their application to cyber defence is limited due to the differences in objectives from cyber defence games.

A game that provides defenders complete observability of attacker actions has limited utility in cyber defence applications. Complete observability removes a core element of cyber defence: detecting and attributing malicious activity. In a realistic scenario, threats can often be mitigated if an attacker is detected before they can complete their objective. The challenge for the tactical defender, then, is to detect malicious activity by using information about its environment that would be reasonably available in a real cyber defence scenario. As such, this chapter primarily discusses partially-observable environments.

The development of more capacitive and powerful algorithms is a central driver for solving increasingly complex RL problems. Progress has rapidly improved due to the integration of deep learning methods into RL algorithm design, one primary force being the work of Mnih *et al.* in DQN [17]. DQN dramatically improves upon the classic Q-learning algorithm, using deep neural networks and experience replay with random sampling. DQN unlocked the ability for autonomous agents to learn expert-level strategy in complex environments, demonstrated through near-optimal performance at a range of Atari video games. Significant advancements have since been made in the capabilities of RL systems, notably in the ability to outperform professional players at challenging games such as Go [3], the real-time strategy game, StarCraft II [35], and the racing simulation game, Gran Turismo Sport [36]. These milestones in machine learning research motivate the use of RL methods for complex cybersecurity tasks. It remains to be discovered how MARL could influence the future of autonomous cyber defence using techniques fostered by state-of-the-art learning algorithms.

This chapter is broken down into the following sections: Section 3.1 discusses the uses of RL for autonomous cyber defence. Section 3.2 presents the state-of-the-art in cyber defence environments for training RL agents. Section 3.3 explores how MAS can be used in a cyber defence setting. Section 3.4 discusses the use of MARL for intrusion detection and response. Section 3.5 contrasts pertinent approaches to value-based cooperative MARL, and Section 3.6 summarizes.

3.1 RL for Autonomous Cyber Defence

Autonomous cyber defence is concerned with protecting computer networks by selecting actions at a tactical level. Decision-making for cyber defence tactics can be carried out by rule-based expert systems using heuristics to react to state changes by taking actions (e.g., [37]). However, RL provides a richer capability and adaptability due to self-learned behaviours, particularly in high-dimensional state spaces [14].

The complexity of game environments is a limitation to current research in RL game design for autonomous cyber defence. Much of the work that uses RL algorithms to perform tactical cyber defence tasks relies on highly abstract representations of the underlying computer network. As a result, tasks are necessarily simplified, restricting the possible policies available to an agent. Computer network attack and defence game environments commonly use a graph representation. In the graph-network framework, each node represents a host machine, and links represent the possible connections an attacker may use to move through a network. The host is either in a healthy state or the attacker has compromised it (e.g., [38]).

Hu *et al.* present a partially-observable graph-network environment and a model-based RL algorithm that approximates the underlying game model via an estimate of the transition probabilities [39]. It applies a Q-learning-based algorithm to approximate an optimal policy from the model. The defender agent's action space consists of a "detect" and a "re-image" action, each requiring a specified network host as the target. Using network links, a simulated attacker penetrates the network of ten hosts by performing an exploit to compromise each host it encounters. The reward received by the defender includes a cost based on the NIST Common Vulnerability Scoring System [40] score of the exploits used on each host. This work is limited in its representation of the network, each host having a binary "compromised" or "uncompromised" state. Moreover, this approach uses a tabular learning algorithm, which limits the scalability of action and observation spaces compared to function approximation methods. For a game design to have demonstrable task relevance, it requires a greater level of detail in terms of the action and observation spaces, likely necessitating more complex algorithms using deep learning.

Dutta *et al.* present a model-free RL defender that learns to modify the threshold of detection (i.e., false-positive and false-negative rate) of simulated anomaly-based intrusion detection sensors in a graph-network environment of 200 hosts [41]. Assigning resources to improve the accuracy of sensors has an associated cost that the defender must balance against the potential for detecting malicious activity on a given network host. This work uses a

hybrid solver in which an RL agent selects actions while a hand-specified heuristic model adapts the action space based on received rewards. If the defender has sufficient confidence of a host compromise, it will take an action to re-image the host. A heuristic attacker begins with a single compromised host and exploits up to one host per turn, propagating across neighbouring nodes. No simulated vulnerabilities exist, and all compromises have the same penalty for the defender at each turn. The RL defender agent optimizes its policy using the Proximal Policy Optimization (PPO) algorithm [10]. This task is significantly more complex than the previous example, and the system leverages a more advanced algorithm design. Moreover, the task involves the management of attention in the form of a limited detection resource that must be shared among multiple hosts. Prioritization of the purview of defensive resources across many network hosts depending on the perceived attacker behaviour is central to the task relevance of tactical decision-making. However, the system also relies on a heuristic to restrict the action space. It is not evident how much of a performance improvement the heuristic model provides over a purely RL approach.

Cyber defence games that exclusively use deep RL have also shown to be successful at protecting a graph-network [42], [43]. Han *et al.* demonstrate DDQN [44] and A3C [45] defender agents capable of optimally minimizing the propagation of an attacker on a network of up to 100 hosts [43]. This work has imperfect detection. In ten percent of cases, the environment will return a false-negative rather than correctly alerting the agent of attacker presence. Imperfect detection adds a layer of uncertainty to the task, providing greater realism. The defender can migrate hosts between subnets and isolate/reconnect hosts suspected of compromise. The defender receives penalties for each turn that hosts are disconnected from the network and for host compromise. By including migration actions, defenders can proactively preserve network assets if the source of the attack is not obvious.

The graph-network games discussed demonstrate the scalability of a single RL agent to defend a relatively large set of hosts. Prioritizing certain assets over others is central to the utility of a tactical defender. However, the tactical policies are limited if the defender can only react to a binary (compromised or healthy) observation signal with a minimal set of actions (e.g., re-image, migrate). This game design does not allow RL agents to learn detailed, relevant policies for defender action selection and attention. As such, it is difficult to infer the potential applicability of these approaches to real cyber operations.

Charpentier *et al.* frame the game of cyber defence between an attacker and defender as a competition for control of a single host [46]. A heuristic attacker agent performs scans and exploits with varying costs and chances of

success. Upon successfully finding a vulnerability with a scan, the attacker can use a corresponding exploit before finally attempting to escalate its privileges. For example, some running services can be exploited using code injection. An attacker would need to scan for this specific vulnerability and perform the exploit. If successful, it will seek out options to escalate privileges on the host. The reward provided at each turn is proportional to the severity of the attack. Meanwhile, a DQN-based defender attempts to deceive the attacker while accessing incomplete information about the attacker's actions. The defender will choose from eight possible strategies, from generating a honeypot process to changing IP addresses or port numbers. Additionally, costs are associated with each agent's actions, and there is a risk that they could fail. This design improves upon the level of task relevance from the previously discussed work by including real host characteristics such as vulnerable processes and more tactical options for the defender. However, since the action space includes only a single host, it is difficult to extrapolate the results of this simulation to a tactical network defence setting.

In this section, the selected works have suggested a variety of design elements that can be used to develop a tactical cyber defence game that is complex and relevant to the setting. These works aim not to produce a learning system that can be deployed on a real network but to aid in developing an understanding of how RL game design can influence an automated defender's role and capability. Many of these works tend to favour network scale at the expense of detailed observations and actions. The exception being Charpentier *et al.*, which minimizes network scale but utilizes more detailed host simulation. An environment enabling aspects from both would have greater applicability to real network operations.

3.2 Cyber Defence Environments

A challenge in the development of autonomous cyber defence research is the diversity of game environments. Environments are often specifically built for the evaluation of a particular learning system. This diversity presents a challenge for the development of adaptable, intelligent tactical defender agents. Moreover, many RL research environments for cyber defence use limited graph-networks and a simple set of agent actions. Some research groups have sought to overcome these challenges by creating general, configurable, and detailed environments for training RL-based attacker and defender agents in repeatable experiments.

Molina-Markham *et al.* present a cyber defence environment (FARLAND)

that can be evaluated on emulated hardware [47]. FARLAND includes a network topology in which hosts have static and dynamic characteristics, such as OS version and running processes, respectively. Additionally, these hosts contain services and users that generate network traffic, providing a quality of service metric to the defender and likewise an opportunity for the attacker to obfuscate its connections to hosts. The defender agent receives health monitoring information from the hosts relating to network traffic volume and new processes and files, among other elements. From these observations, the defender can choose to isolate a compromised host, migrate or replicate services, or deceive the attacker by creating decoy services. Meanwhile, a heuristic attacker will attempt to gain access to network hosts by scanning for open services and applying specific exploits to vulnerable services or network configurations. On a network of ten hosts, APEX-DQN [48] and PPO [10] defender agents learned to defend a high-value server for 100 in-game steps. A drawback to the FARLAND environment is that it is currently closed-source.

The primary alternative to FARLAND is the open-source CybORG environment [30]. It provides many of the same features, including a configurable network topology, detailed simulated host-based elements, and a large action space for attacker and defender agents. Although initially released as a single-agent, tactical defender training environment, CybORG is in active development and version 3 includes a MARL drone-based game with networking elements. To the best of our knowledge, this version is the first published project for cooperative MARL for tactical cyber defence. However, the scenario deviates significantly from previous releases and is not concerned with the enterprise defence of hosts with multiple users, processes, and vulnerabilities. Rather, CybORG 3 contains mobile hosts with a single vulnerable pathway that can be exploited by two teams of agents, red and blue. CybORG 3 improves upon its previous releases by incorporating multi-agent cooperation, network traffic simulation, and dynamic network configuration. However, it lacks the simulated host elements that support the applicability of this environment to enterprise cyber defence settings. CybORG could be adapted into a MARL environment for enterprise cyber defence, retaining the host simulation from previous versions.

The potentially massive observation and action spaces are a pertinent limitation to advanced environments such as FARLAND and CybORG. One means for handling complexity is the introduction of expert systems to simplify the defending agent's observation space (e.g., [41]). The CybORG team released an open challenge [49] accepting RL defender models trained on the CybORG 2 environment (the enterprise defence version) with the help of a hand-specified observation module. This module simplifies the defender's ob-

servations, saving the state (healthy, scanned, or exploited) of each host in memory. Without this module, the CybORG defender receives over 11 000 unique observation elements on a network of thirteen hosts. The results of this challenge found that PPO methods could effectively mitigate attacker activity, limiting attacker movement onto higher-valued hosts. The observation module in the CybORG 2 challenge abstracts the complex environment so that the actions the agent selects are based on a simplified understanding of the underlying state. A decentralized defender system would not realistically have this input and instead be required to evaluate the state of its host data and coordinate with other security devices to make decisions. CybORG could be extended to allow for the recognition of raw sensor data rather than through a game-specific filter, supporting a more task-relevant observation space.

Emulated cyber game frameworks can provide autonomous agents with actual virtual machines and command line tools (e.g., [47], [50]), providing a greater level of realism to the learning scenario. A drawback of emulated environments is their computational cost relative to simulated environments. In an emulated environment, experimentation requires more clock-time and the game design elements are more rigid. However, emulated environments provide an important link to real network TTPs. This research focuses solely on training MARL systems in simulation, leaving emulation for future work.

3.3 Multi-Agent Systems for Cyber Defence

The works discussed in the previous sections primarily employ a single learning agent to take actions in a network environment. Central to the design of any RL-based cyber defence game is the agent-environment interaction, characterized by observations, possible actions, and rewards. In a cooperative multi-agent system, agent-agent interaction must also be considered. The following works provide relevant insight into multi-agent considerations for cyber defence games but are not specifically MARL implementations[1].

Communication between defender agents has been shown to improve their ability to protect a simulated network from DDoS attacks [51]. The simulation places defender teams on different subnets monitoring network traffic. The attacker generates traffic in an attempt to overwhelm network hosts, and the defending teams must adaptively apply filters to drop offending network packets at their respective firewalls. Defender agents are programmed

[1]To the best of our knowledge, the only cooperative MARL implementation for tactical cyber defence (aside from the one presented in this work) is CybORG version 3, available at https://github.com/cage-challenge/cage-challenge-3.

to communicate filter configurations or packet data between teams to improve reaction time and effectiveness. The experimentation found that teams that used communication performed better than those that did not, particularly in the case of shareable filter configurations. This work is limited to heuristic decision agents and a specific DDoS protection case. However, it demonstrates how cooperative agents in a cyber defence setting can benefit from sharing tactical information.

Liu *et al.* propose a multi-agent online learning system using an evolutionary game framework to adopt better strategies based on outcomes [52]. It considers fully-decentralized agents that can elect to share information among themselves. Although actions are selected based on the learned expected value of future states, the value function is hand-specified, unlike in RL systems in which value functions are derived from experience. This work provides a framework for independent agents to learn to share features that will assist their teammates, thus extending teammate observation spaces. Learning communication is a developing area in cooperative MARL [53] that could be advantageous in large decentralized network defence applications. Although agents learning from shared experience is an aspect of this research, explicitly learning communication between agents is left for future work.

3.4 MARL for Intrusion Detection and Response

Multi-agent systems provide important details regarding the design of autonomous cyber defence systems where applications of MARL systems are limited. However, there are MARL-specific considerations that are unique to the problem space, such as non-stationarity and credit assignment, discussed in Section 2.5. This section examines how cooperative MARL has been applied to intrusion detection and response scenarios with considerations of cross-applicability into tactical cyber defence.

MARL systems have been shown to perform anomaly-based intrusion detection when framed as a classification task [54], [55]. The work of [54] proposes a design for multiple agents to collectively alert of intrusions in a simulated network environment based on separate observation spaces containing unique sets of network features. Features may include, for example, the connection destination of packets or the number of ACK packets received. A hierarchical distribution of agents is used in which two agents positioned at network traffic sensors send signals to a decision agent, which performs the classification. A joint reward corresponding to the correctness of the classification is returned to the learning system at each timestep. A hierarchical

structuring of RL agents reduces the policy space of each agent, potentially allowing for more complex behaviours without exceeding the representational capacity of the system. Although the detection of threats plays a central role in cyber defence, the tactical challenge must also include an element of management or threat response.

MARL systems have been compared to traditional machine learning and deep learning classifiers at an anomaly-based intrusion detection task using an offline dataset [55]. It showed to have improved accuracy and precision over the comparative approaches. This approach employs a set of DDQN [44] "minor" agents that each receive a subset of the feature space, which they use to classify samples as anomalous or normal. Likewise, a "major" agent with identical architecture performs the same task while using the entire feature space. With sufficient consensus among the minor agents, they may overturn the default action selected by the single major agent. Although offline training is commonly used to train machine learning-powered intrusion detection systems, it presents significant constraints to exploration due to a limited set of samples [14]. The possibilities for agent-learned behaviour are constrained by the information contained within the dataset.

MARL agents have been shown to learn to defend a network against a DDoS attack by selectively throttling network traffic [56]. In a simulated network game, a group of 1000 agents was trained to restrict traffic flows on their assigned network routers. The agents were shown to keep the total load on a central server within its operating range while maximizing the flow of legitimate traffic. This work demonstrated a significant improvement when using *difference rewards* to mitigate the credit assignment problem by comparing the observed reward to a counterfactual [57]. Credit assignment may significantly impact the performance of cooperative MARL for tactical cyber defence, particularly in large-scale settings. Similar to previous MARL works discussed, the observation spaces are restricted to a single aspect of the network, and the action space involves increasing or decreasing a certain parameter. MARL can decentralize control of systems for cyber defence, handling very large input spaces. This suggests that cooperative MARL could be an effective tool for managing complexity in tactical-decision making for cyber defence tasks.

3.5 Value-Based Cooperative MARL

Despite the small amount of published literature in cooperative multi-agent cyber defence, the field of general algorithm design for cooperative MARL is rapidly expanding. This research seeks to bridge the gap between the capa-

bility of MARL designed for general games and tactical cyber defence simulation. This section provides an overview of value-based cooperative MARL algorithms commonly applied to research environments.

Experimentation on RL systems typically utilizes a baseline environment to provide a performance benchmark between algorithm designs. One of the most commonly used cooperative MARL environments is the StarCraft Multi-Agent Challenge (SMAC) [58]. SMAC provides a variety of scenarios ranging from easy to very hard in which virtual units learn to defeat each other in a battle video game. A comparative study of cooperative MARL algorithms across a variety of environments, including SMAC, showed that CTDE methods are among the highest performing [12]. However, it remains an open question how these results may translate to a tactical cyber defence game. Moreover, centralized or decentralized models may have a performance edge when faced with different opponent attack strategies, network topologies or observation spaces.

The underlying principle of value-based RL is the use of a value function to represent the expected return from a series of game states or state-action pairs. Many value-based methods are based on the principles of Q-learning [6], which is discussed in greater detail in Section 2.1. The RL methods explored in this work perform value function approximation using deep neural networks.

A simple decentralized approach to learning in a multi-agent setting is IQL, in which multiple Q-learning agents take actions simultaneously in a network to achieve a joint goal [59]. IQL has been demonstrated to learn to play simple games despite the non-stationarity of multiple agents interacting with the environment simultaneously [60], [28]. However, IQL does not perform as strongly as many state-of-the-art cooperative MARL algorithms across a variety of tasks, often due in part to the non-stationarity problem [12]. Based on its performance in other environments, IQL methods may struggle to learn in complex cyber defence scenarios.

QMIX is a CTDE value-decomposition algorithm that employs a mixing architecture of NNs to generate a joint Q-function [11]. Each agent uses a decentralized Q-function that is trained centrally. QMIX extends the potential for value-based methods by improving the representational capacity of the central Q-function using a trainable system of networks. See Section 2.5.4 for additional discussion of the QMIX architecture. In practice, QMIX has outperformed state-of-the-art value-based and policy-based methods at a range of tasks [12], [58]. Most notably, QMIX has been shown, with the help of specific implementation tricks, to discover optimal or near-optimal policies on all SMAC [58] scenarios [29].

SMAC tasks are assumed to have transferability for general tactical-level

decision-making as a result of agents needing to direct attention and action in a coordinated way toward individual enemies in order to achieve success. The cyber defence problem is modelled similarly, using actions to selectively monitor or affect host characteristics. Due to the high potential for performance on cooperative tasks, QMIX may be well suited for autonomous cyber defence.

3.6 Summary

This chapter examined two main critiques of existing game design approaches: (1) Many cyber defence game environments are greatly abstracted from actual networks or involve optimizing a specific task. These approaches offer limited task relevance to tactical cyber defence. (2) learning systems in tactical cyber defence research rarely employ cutting-edge techniques, including multi-agent methods, to expand the scope and complexity of the solvable problem space. A sophisticated, partially-observable environment such as CybORG or FARLAND allows for the design of complex, relevant tasks.

4 Methodology

This chapter describes the research methodology for the evaluation of state-of-the-art MARL algorithms in diverse scenarios. It provides a framework from which to uncover promising game designs that demonstrate cooperative MARL system adaptability and ability to handle complexity. Evaluation is divided into two phases: the first phase exposes MARL systems to three attacker strategies and three network sizes. The second phase explores the effect of environmental complexity on the learning ability of MARL systems through the addition of noise sources, the use of an additional action, and an increased area of responsibility. Each phase relates to one of the applicability criteria in this work, adaptability and complexity, respectively. CyMARL (Cyber-MARL), the simulated cyber defence game environment for training and evaluating cooperative MARL presented in this work, allows agents to learn from host-based monitoring data to take tactical-level actions to prevent and mitigate malicious activity.

A learning system is a cooperative deep MARL design consisting of an architecture of neural networks and algorithmic structures. A learning model is a specific instance of the learning system that can be trained to perform a specific task; its objective in a game. This use of *model* is separate from its use to discuss model-based learning in Section 2.1.1. A game is an entire system, including the learning model and simulated environment. The game elements include, for example, the attacker's behaviour, the simulated network architecture, and the actions available to the defender. A scenario is a set of specific game elements. The terms *defender* and *attacker* are used to represent each of these forces' roles within the game. The defender refers to the behaviour exhibited by the learning system, composed of multiple agents.

The outcome of the evaluation is ultimately to provide evidence to support or refute the applicability of the selected cooperative MARL approaches for the task of tactical cyber defence decision-making. Applicability refers to the capacity for a particular learning system to scale to more realistic problems. For example, a system with a high level of applicability generates a policy

that identifies anomalous behaviour and carries out a logical chain of decision-making to mitigate the threat across a variety of game designs. MARL systems are exposed to a variety of game design elements and sources of complexity within the environment to depict the ability of particular architectures to learn cyber defence tasks. A complete guarantee of applicability to real cyber defence is not possible within the scope of the simulated environment or in the number of trials. Instead, arguments for the applicability and limitations of MARL approaches are presented based on the experimental evidence.

To validate the objective of applicability, learner and environment parameters are evaluated based on the learning ability of MARL systems. Learning ability is quantified by the mean return of a policy relative to a benchmark. The benchmark in Phase 1 is a heuristic multi-agent defender. In Phase 2, the parameters of one scenario type are varied, so performance is benchmarked against the original scores of each MARL architecture.

This work assumes that there is transferability of performance gains between the CyMARL environment, other game environments, and more realistic applications. This assumption is based on the evidence of performance transfer between different learning tasks for cooperative MARL (e.g., [12]) and between simulated and emulated cyber tasks (e.g., [47], [50]). The transfer of learning ability from an abstracted simulation to emulation and real network environments is assumed to be possible given an appropriately sized input-output space. That is, the representations of tactical elements in the simulation are sufficiently precise for performance gains to transfer to more realistic environments. The applicability of MARL systems is therefore correlated with their learning ability in complex environments.

This chapter is broken down into the following sections: Section 4.1 introduces CyMARL, the simulated cyber defence environment. Section 4.2 discusses the evaluation criteria of cooperative MARL methods. Section 4.3 explores the game design elements to be varied as part of the evaluation. Section 4.4 presents the two phases of the evaluation using the framework and game design elements discussed in this chapter, and Section 4.5 summarizes.

4.1 CyMARL: Cyber Defence Environment

Game design, including elements such as observations, actions, rewards, network topology, and game mechanics, define the cyber defence task the learning system will attempt to optimize. To demonstrate applicability, the game environment must support a variety of game design elements to evaluate tactical cyber defence decision-making by cooperative MARL learners. The simulated

environment presented in this section allows for extensible, repeatable experimentation of MARL for host-based cyber defence tasks.

Cyber Operations Research Gym (CybORG) is a framework that provides a simulated cyber operations environment for training and evaluating RL agents [30]. CybORG simulates information technology systems related to network security that a cyber professional may use in network defence or penetration testing. CyMARL adapts the CybORG simulation project to create a multi-agent host-based monitoring game for training tactical cyber defence. CyMARL includes a PyMARL[1] environment similar to SMAC [58], allowing for integration with the tens of open-source cooperative MARL algorithms built on the PyMARL framework.

Host-based monitoring systems are a common and effective tool for cyber defence, particularly in situations where an attacker has made an initial breach and is attempting to gain Command and Control (C2) channels within a network. An agent that can effectively learn to perform tactical cyber defence decision-making has the potential to provide host-based alerting and suggested decision chains to a human cyber analyst. By excluding network traffic from the simulation, the feature space of the MARL system can be constrained to manageable levels. The results of a host-based game could inform the design of more complex games, including network traffic elements. A host-based monitoring system may use many attributes of computer systems that are not simulated in this game environment. However, the intent of this game design is to train an agent to react appropriately to some abstraction of the underlying state of the network hosts. There are few limitations to the possibilities of RL in this field. With sufficiently advanced design, the pattern recognition potential of deep learning could enable extremely precise, efficient targeting and mitigation of threats that may not be obvious to humans or traditional tools.

4.1.1 Simulated Network

The core function of the simulated environment is to maintain and update the true state of the game: a dictionary of all hosts, processes, and artifacts within the network. Hosts, representing machines on the network, typically contain system information, running processes, relevant files, running sessions, and network interfaces. Each of these elements has various parameters stored

[1] PyMARL is an open-source MARL research project that allows algorithms to be built from existing deep RL components such as NN agents and trainers. It is included as part of the SMAC environment and paper [58]. The repository for PyMARL can be found at https://github.com/oxwhirl/pymarl.

in a hierarchical dictionary object. The true state is modified by the actions of the defender learning agents, the scripted attacker agent, and, optionally, heuristic grey users. Grey users are neutral actors in the scenario that take actions such as connecting between hosts or scanning for IP addresses. Such actions may be interpreted by the observing defender agent as malicious and therefore create false positives.

Each scenario has an operational subnet that contains an operational server and a defender server, shown in Figure 4.1. Each scenario also contains a user subnet in which the attacker has compromised an initial host (User0) at the start of the game. The attacker's session on User0 cannot be removed, and the defender server cannot be compromised. This allows for the infinite play to be separated into episodes with a fixed number of turns. In this experimentation, thirty-turn episodes are used based on the parameter from CAGE Challenge 2 [49].

At the start of an episode of play, the IP space for the network is randomly generated. Each host is assigned to a pre-defined subnet and given an IP address as well as appropriate networking information to other hosts on its subnet. Specific hosts are provided with routing information of other subnets; the attacker must exploit these hosts to penetrate into other subnets. Connections between hosts are not explicitly simulated. Instead, agents use a directory of remote sessions to execute actions on. Services are run on hosts, and they can be exploited by the attacker from a remote host, so long as the attacker has appropriate routing information (i.e., IP address and port number) and an applicable exploit.

Hosts can be Linux or Windows, which affects which processes, file system, and vulnerabilities they have. Hosts are divided into up to three subnets: User, Enterprise, and Operational. Each host has an importance score based on the subnet. Hosts on the "user" subnet have a value of 0.1, on the "enterprise" subnet, a value of 1, and on the "operational" subnet, a value of 10.

Hosts also store information about users and active sessions. When the attacker exploits a host, it generates a reverse TCP shell as a C2 channel. The appearance of new sessions is one of the key indicators that a defender agent can use to attribute host compromise. Some types of exploits or data tampering create files stored on hosts. Identifying file creation is another method a defender agent can use to detect possible attacker actions.

Simulated data structures are encoded into observation vectors for training and evaluation with RL agents. The data-structure-based host simulation allows host-monitoring observations and actions to be computed without the overhead of simulated data flows. This method allows for an efficient training process in terms of clock-time for the simulation and the learning system. A

description of the network environment elements can be found in Appendix A.

4.1.2 Heuristic Attacker

The heuristic attacker follows a static policy. It begins each episode with an initial foothold on a user host and moves between hosts by scanning for and exploiting vulnerable services. Figure 4.1 presents a simple network diagram showing attacker actions. The attacker is rewarded at each timestep equal to the sum of the importance scores of all currently compromised hosts, though reward does not modify its behaviour. The attacker can move laterally to a target host if one of its sessions' hosts has visibility of the target host through its network interface. The attacker will choose hosts to exploit as it follows the steps in Algorithm 1. The selected exploit is determined based on the port number and a fixed probability, discussed in Appendix A.

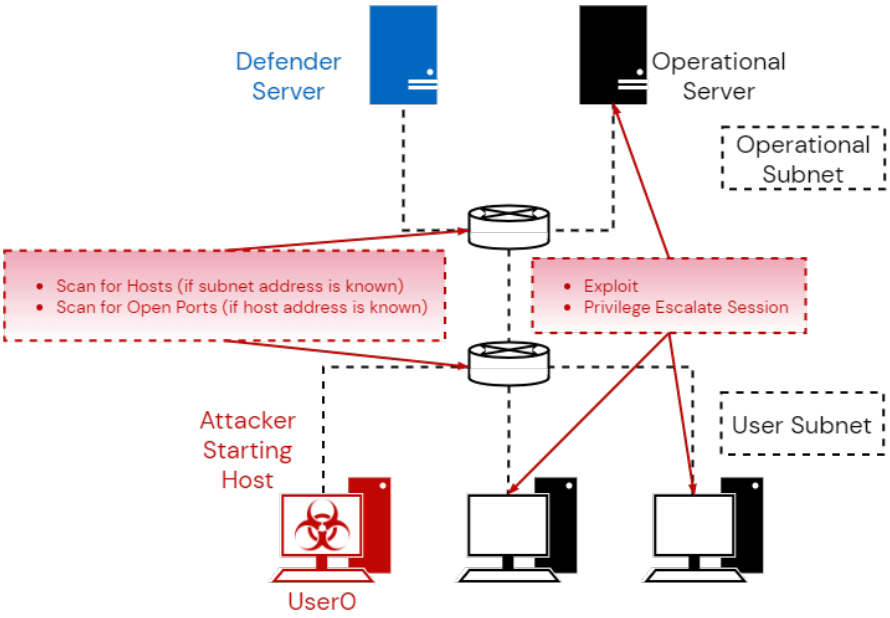


Figure 4.1: Simple network diagram with attacker actions and targets.

A real-world attacker is not always solely motivated to establish communication channels on a network, although this is often critical in the success of advanced persistent threats. Attackers may be motivated to modify data within a computer network or deny users the ability to use IT infrastructure. These principles are captured within the definitions of *integrity* and *availabil-*

ity, respectively. A generic attack scenario in which the attacker attempts to control as many hosts as possible fits the principle of *confidentiality*. These three principles of cybersecurity provide variation to the attacker's behaviour and thus the task of the defender, allowing for an evaluation of learning adaptability.

Algorithm 1 Attacker heuristic behaviour.

for `subnet` in `network` **do**
  `known_ip_addresses` ← *DiscoverRemoteHosts*
end for
for `ip_address` in `known_ip_addresses` **do**
  `scanned_ip_addresses`, `known_ports` ← *DiscoverNetworkServices*
end for
for `ip_address` in `scanned_ip_addresses` **do**
  `exploited_ip_addresses` ← *Exploit* ▷ Exploit type is based on port number and probability, see Appendix A.
end for
for `ip_address` in `exploited_ip_addresses` **do**
  `ip_address` ← *ScoringAction* ▷ Take scoring action if applicable.
end for
for `ip_address` in `exploited_ip_addresses` **do**
  `ip_address` ← *PrivilegeEscalate*
end for

Attacker Goals The attacker's behaviour is differentiated by its goal, which can be *confidentiality*, *integrity*, or *availability*. In each case, the attacker takes different actions upon establishing a session on a host, though the heuristic for lateral movement is the same across all scenarios. In the confidentiality scenario, the attacker's intent is to spy on privileged information within the network but not take actions to modify it directly. The confidentiality attacker receives rewards for each host to which it is connected at each timestep. The integrity attacker seeks to *tamper* with files stored on a host. At each timestep that this modified data exists, rewards are provided to the attacker based on the importance score of the host. The availability attacker will spawn a Denial of Service (DoS) malware process that represents a consumption of CPU cycles, affecting user availability. Similarly, at each timestep in which the malware process is running, the attacker receives rewards equivalent to the importance score of the host. The attacker's reward is used to train the

defending system on whether the defender's series of actions is effective at mitigating this activity.

The attacker goals are designed to assess the adaptability of the defender system to learn to defend against variations in threat type on a network. Although the attacker's behaviour is a simplification of these attack patterns, the difference in objective affects how the threat appears to the defender and which actions it should take. The cumulative reward for each strategy serves to model the risk to the network while malicious artifacts exist on network hosts. As the number of timesteps increases, there is a greater probability that the attacker will have collected privileged information or disrupted normal processes on the network. Likewise, the severity of the risk is proportional to the importance of the host, which is based on the value of its subnet.

Attacker Actions The attacker will choose from a set of parameterized, discrete actions at each timestep. The parameters allow the attacker to choose targeting information such as an IP address, session, process, and port, as shown in Table 4.1. Beginning with its initial foothold on the user subnet, the attacker's only valid action is to perform a scan for IP addresses of connected hosts. It will save the list of discovered hosts and select one at random to scan for open ports. With a target IP address and port number, the attacker can choose from a set of exploits with a probability of successfully spawning a shell, gaining the attacker greater information on a compromised host, including connections to other hosts that can be hijacked. Attacker exploits are based on the vulnerable service, they are listed and decribed in Appendix A. The attacker's actions follow a hierarchy: it must first discover a host, then a port, before attempting to move laterally on the network using an exploit. Escalating session privilege requires the existence of a session created via an exploit. The attacker's possible actions at each step are based on the saved information it has gathered.

Similar to performing exploits, the action to escalate privileges is successful based on a fixed probability for all hosts. The scenario assumes that privilege escalation vulnerabilities are related to the image of the host system and are common among all hosts of a given OS. Escalating privileges does not provide points, but it immediately obfuscates the attacker's session, removing it from the defender agents' observations. A file is left on the host as a result of the privilege escalation action providing the defender with a clue that malicious activity has occurred on a host. The heuristic attacker will attempt to gain a session on each host within the network before escalating its privilege.

In the integrity and availability scenarios, an additional *scoring action* is

required to realize the attacker's goal. Scoring actions can be taken once a session is established on a victim host using the `exploit` action. The `tamper` action is taken by the integrity attacker to simulate the modification of important files. The files will appear to the defender agent monitoring the subnet. However, the defender can only distinguish between legitimate and tampered files using a targeted `analyze` action (see Section 4.1.3). The `deny` action is taken by the availability attacker to simulate the creation of a DoS process. This does not affect the defender's ability to monitor the victim host. However, the attacker receives rewards over time for each active DoS process.

Table 4.1: Attacker actions and their parameters.

Action	Parameters			
DiscoverNetworkSystems	Session	Subnet		
DiscoverHostServices	Session	IP Address		
Exploit	Session	IP Address	Port	Exploit Name
Privilege Escalate	Session	IP Address	Exploit Name	
Tamper[1]	Session	IP Address		
Deny[1]	Session	IP Address		

[1] `tamper` and `deny` scoring actions are exclusive to their respective scenario types, integrity and availability.

4.1.3 Multi-Agent Defender

Within the simulation, a defender server runs a controller process for each RL agent executing commands using its active sessions on connected hosts. Each agent has exclusive access to its designated simulated controller. Figure 4.2 illustrates the agent-environment interaction from the point of view of the defender. Agents are external to the network as they are the representations of the MARL decision-making policies.

Defender Goals The defender's goal is to minimize the harm caused by the attacker. Harm is quantified by the attacker's rewards. The defender cannot receive positive rewards. Rewards are either a result of the attacker's actions or a defender's invalid actions. The MARL agents seek to minimize the success of the attacker by jointly maximizing their cumulative reward. Rewards, and by extension, the importance of protecting a host, are scaled based on the value of the subnet. This reward signal incentivizes proactive protection of lower-valued subnets to avoid harsh penalties incurred if the attacker penetrates into higher-valued subnets.

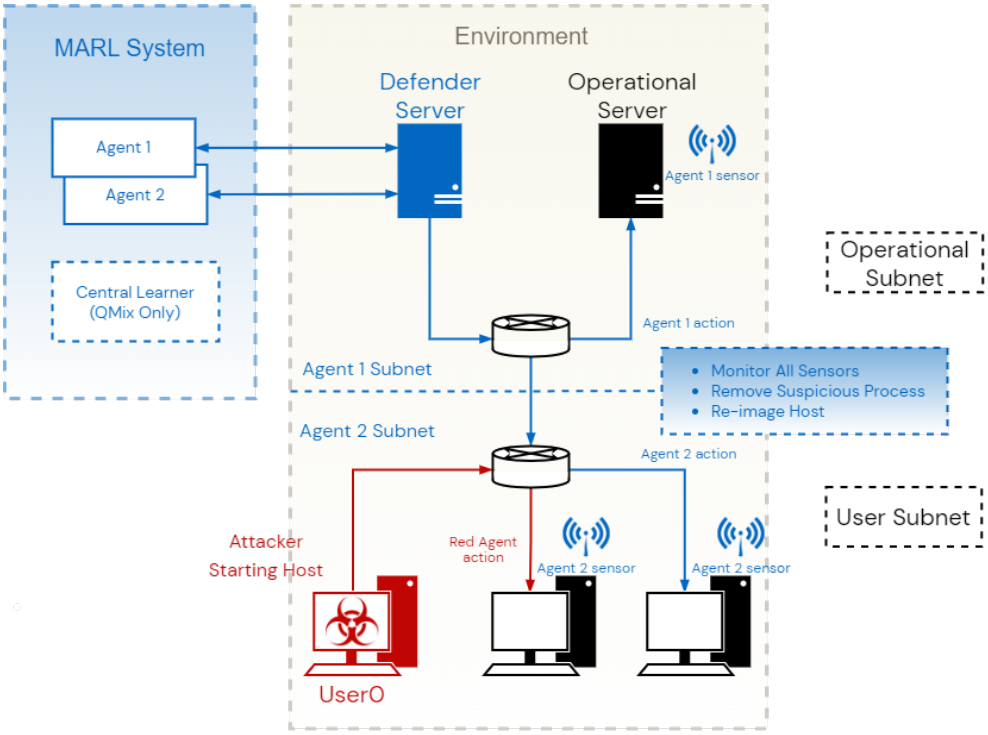


Figure 4.2: Agent-environment interaction with multiple defender agents.

Defender Actions To mitigate attacker activity, the defender selects actions to monitor, remove suspicious processes or files, restore hosts, or gather file information. RL defender agents must choose to take the `monitor` action to receive observation information from all hosts within its area of responsibility. As with the attacker, each action has parameters that allow the agent to choose targeting information such as an IP address, session, process, and port, as presented in Table 4.2.

`monitor` checks all hosts within an agent's vision for new processes added to a suspicious process list. The action returns an observation containing an encoded listing of network information, running processes, sessions, and system information. An agent can also choose to `remove` suspected malicious processes from a specific host. All new processes are pre-identified using the `monitor` action. As a more aggressive action, an agent can re-image a host using `restore`. This action restores a host to its original state at the start of the game. It is the only way to remove an attacker that has escalated its privilege on a host. It will also remove any malicious processes or files. However, the `restore` action has an associated penalty that scales with the

importance score of the target host (e.g., -0.1 for a user host or -10 for an operational host).

The three actions, `monitor`, `remove`, and `restore`, are available to the defender for all scenarios. In the integrity scenario, agents have two additional actions to manage the specific threat type. Agents can take an `analyze` action to determine security characteristics of files on a host, allowing it to identify any malicious files. To remove the discovered files, the `data_repair`, is required. In the availability scenario, the `remove` action can be used to target DoS processes and simulated TCP sessions. In more advanced trials, the `misinform` action is introduced, and the defender's area of responsibility is modified. `misinform` generates a service on a host that can be misinterpreted by an attacker as vulnerable.

Table 4.2: Defender actions and their parameters.

Action	Parameters		
Monitor	None		
Remove	Session (to execute from)	IP Address	PID
Restore	Session	IP Address	
Analyze[1]	Session	IP Address	
Data Repair[1]	Session	IP Address	
Misinform[2]	Session	IP Address	Service to create

[1]Exclusive to the integrity scenario. [2]Action added as part of advanced trials (see Section 4.4).

A defender agent may choose to perform any action on any of its connected hosts. However, some actions are invalid. For example, a `remove` action on a host without an attacker process will fail, resulting in a *invalid action*. Taking an invalid action has a penalty equal to the importance score of the host's subnet that serves to reinforce the avoidance of these actions. The game is designed to allow the defender to receive information about certain activities that may be malicious, such as the establishment of new sessions on a host, by selecting the `monitor` action whenever a specific response is not required. The intuition behind this mechanism is that a human cyber operator will typically focus their attention on monitoring to maintain a general picture of events on the network from their detection tools. Given details about the environement, an analyst may select a specific tool and target for either analysis or mitigation of a suspected threat. The high-level actions of the defender in this game represent the choices in attention that a defender would need to take to understand or act against the threat. The tactical decision-making

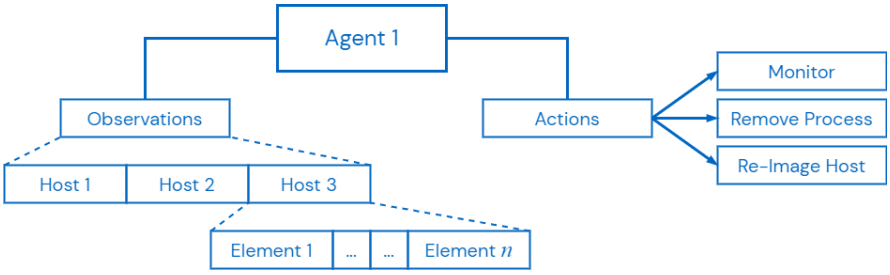


Figure 4.3: Observation and action space of a defender agent.

competency of agents can then be inferred from their ability to minimize the impact of the attack.

Defender Observations The defender's observations consist of a set of host attributes that could be available to a host-based monitoring system, listed in Table 4.3. The observation space is a fixed vector containing an encoding of host attributes for all hosts to which an agent is connected. Within each observation, the status of each host is encoded into a feature vector as illustrated in Figure 4.3. Feature vectors contain a floating point value for each simulated element of a host. All host vectors are concatenated before being sent to the simulated controller for each agent. The information contained within the vector includes static attributes such as IP address and hostname and dynamic attributes such as running processes, connections, and running sessions. For example, a trained defender agent may suspect attacker activity by noting that a new session has been created by a previously zeroed segment of its observation space containing encoded information. The fixed observation space integrates with existing MARL training libraries such as PyMARL [58]. A detailed description of action and observation in CyMARL can be found in Appendix A. A limitation of fixed observation spaces is poor generalizability of trained models to new tasks. However, this does not limit an evaluation of the adaptability of MARL architectures that are trained from scratch for each task.

Each agent is responsible for a subnet within the network. Network defence in real applications is commonly divided by subnet to segregate security access levels. Consequently, the compromise of an RL agent does not expose or compromise agents in other subnets, enabling greater security of the learning system within the game.

Table 4.3: Defender observation features of a network host.

Category	Feature
Networking Interface	IP address, Subnet
Running Processes	PID, User
Sessions	PID, User
System Information	OS Version, Hostname
Users	Users, Groups, Passwords if known

4.1.4 Optimal Score

A score of zero infers that the attacker did not take any successful scoring actions over thirty turns of play. The attacker has the first-move advantage, as is common in real operations. In the confidentiality scenarios, the first attacker action that the defender can observe is an exploit. It will therefore reliably accept some loss in the first turn that the attacker moves onto one of its neighbouring hosts. Although the defender could get lucky with the removal of the right host, this risks an invalid action with the same penalty. A non-zero score is expected in most confidentiality games, even with optimal defender play. In the other two scenarios, integrity and availabilty, the attacker performs two actions to begin accumulating rewards. It is possible, therefore, for an intelligent defender to break the attacker's connection to a host before it can tamper with files or spawn malicious processes. If this is done every time and no invalid actions are selected, the defender could hypothetically score zero in the integrity and availability scenario types. Moreover, the deception scenario, which allows the defender to use the `misinform` action, provides opportunities for the defender to guess at the possible hosts that an attacker may randomly select. Thus the defenders can improve the probability of detecting the attacker without exposing hosts.

4.2 Evaluation Design

This section describes a framework to evaluate the learning ability of MARL systems across a range of tactical cyber defence tasks. Models begin with randomized policies for each agent. Through iterative episodes of interaction, the system learns to optimize its policy in the game by seeking to minimize the rewards that the attacker accrues.

For each experiment, two algorithms are evaluated: IQL and QMIX, implementing fully-decentralized and CTDE architectures, respectively. Agents

do not directly share information with each other, but in the case of QMIX, a joint policy representation is learned centrally before being decomposed into separate behaviour policies. The architecture of cooperative MARL systems has been shown to significantly affect the learning ability of models at a range of tasks [11], [12]. As such, a comparison of centralized and decentralized training allows for direct observation of the effect of centralized learning for MARL in a cyber defence setting.

Learning ability is the capacity of an RL model to generate a policy from its initial randomized state. A trained model that achieves a high return at a task has a higher learning ability than a lower-performing model given the same training opportunity. The design of this experimentation relies on the assumption that the applicability of a learning system is correlated to its learning ability across a range of cyber defence tasks. Learning ability is evaluated using the relative mean performance of a model compared to a benchmark at a given task. Tasks are differentiated by the inclusion or modification of various design elements. Each design element is tested independently as a trial for each architecture type.

4.2.1 Learning System Training

Agents update their Q-function based on the joint reward, observations, and actions. Since the defender cannot gain positive rewards, its objective is to minimize the negative rewards it accrues in an episode by preventing the attacker from realizing its goals.

IQL is equivalent to training independent DQN agents to learn to take simultaneous actions to maximize their joint reward. Each IQL agent performs a prediction of its expected reward from local experience using a Q-function. By updating its Q-function based on new experience and acting on its policy representation, each IQL agent is a self-contained learner-actor. The foundational difference between QMIX and IQL is that QMIX performs learning updates centrally. Each agent will generate a separate Q-function based on its local observations and joint reward, but the weights of the Q-function NNs are sent to a centralized learner, which performs a joint policy update.

4.2.2 Evaluation Metrics

Since the task of an RL system is to maximize its return, the mean return of a system is a direct indication of how well it has learned to perform its intended task. The mean return is the average of the defender system's return over multiple episodes of play. Return is provided by the environment based

on the reward function of the game. Each experiment has a unique set of parameters affecting the optimal policy, hence the optimal score. To measure performance, a model's mean return is compared against a benchmark for the specific task using percentage difference. This benchmark is a heuristic defender model for each scenario in Phase 1, and the original performance of each architecture in Phase 2. Since the optimal return varies between tasks, a heuristic model provides a base value for learning system performance.

Mean return is a common metric for evaluating trained model performance and convergence in cooperative MARL (e.g., [11], [12], [28], [58]). In many cases, novel algorithm designs are benchmarked against established environments to compare their performance to current state-of-the-art approaches [11], [29]. Likewise, the difficulty of MARL tasks can be quantified using the mean return of models trained using existing algorithms, such as MADDPG [26], IQL, and QMIX. Works such as SMAC [58] provide a benchmark environment for these comparisons. This research is concerned with evaluating the performance of state-of-the-art cooperative MARL architectures when faced with a novel environment.

An alternative evaluation metric is an agent's Q-function which outputs an agent's estimated return for following its policy from a given state. The Q-function provides an agent-defined value of its current policy. There are three primary drawbacks to using the Q-function as an evaluation metric: First the Q-function is an internal estimate of expected performance rather than observed performance. Second, in independent multi-agent learning, each agent utilizes an independent Q-function. The average of these functions does not necessarily indicate the system's overall performance. And third, the heuristic defender benchmark does not produce an equivalent value metric with which to compare. Q-value distribution is sometimes used as a metric for learning stability. However, it suffers from the same drawbacks as the Q-function discussed previously. Instead, the standard deviation of the mean return can provide a metric for the stability and convergence of the learning model over time.

Two evaluation outputs are used to analyze learning ability: *evaluation score* and *training performance*. The evaluation score for a model is its average performance, measured in mean return, over 1000 steps of play using a trained, static policy. The training performance of a model refers to the mean return of the model taken at 10 000 timestep intervals over its training run[2]. All evaluation outputs are specific to the experiment and must therefore be

[2]Averaging over 1000 timesteps for evaluation and 10 000 timesteps for training are standard in the literature (e.g., [17], [58]).

compared to relevant benchmarks. Training performance curves provide an overview of model learning speed and performance relative to other architectures and benchmarks.

All models were trained to one million timesteps. Learning speed is an implicit evaluation criterion to mitigate the impracticality of long training times. Although there may be greater performance gains outside of the bounds of the training time for the model, this evaluation will discuss the behaviour of the model from its initial state to one million steps, inferring expected future behaviour.

4.2.3 Heuristic Defender Benchmark

In Phase 1, a heuristic defender is used as the benchmark. This rule-based multi-agent defender model can detect suspected malicious activity and restore hosts. Its agent-environment interface mirrors the MARL defender system for each scenario, controlling agents with the same action spaces and limited observations. The heuristic issues one action to each agent at each timestep.

The heuristic defender's design assumes it has access to a tailored expert system that provides a precise signal of malicious process creation. The defender's action at each step is to monitor until it receives this signal, at which point it will always restore the target host. This design is based upon a hypothetical script that reacts to the signal of a signature-based detection tool that reveals targets based on a ruleset. The MARL systems do not assume that this signal is available and instead rely on the features produced directly from the host simulation.

Fewer constraints on data preprocessing allow for a potentially richer set of behaviours that cannot be otherwise achieved due to preconceptions about the meaning of inputs. In the heuristic defender model, or in the CybORG 2 observation module discussed in Section 3.2, elements of the simulation are aggregated and labelled before being sent to the decision-making model. These observations provide specific context, derived by hand, that allow models to infer which actions to take. By providing raw data without context, learning agents have the opportunity to develop more nuanced relationships between observations and actions that may not be obvious to the human designer. Bypassing the feature discovery required by the MARL systems gives the heuristic defender an advantage, but this is mediated by its unsophisticated behaviour. In testing, the heuristic defender model performed well relative to MARL models, particularly on larger networks where MARL systems have a greater observation space to manage.

4.3 Game Design Elements

Game design elements define the learning task and thus enable the evaluation of MARL systems at a range of tasks. Elements are selected based on three outcomes: (1) to expose MARL systems to a range of scenarios that may be encountered in a real setting, enabling an assessment of the adaptability of model performance across tasks, (2) to increase the complexity of the tasks by adding realism, and (3) to increase its scope of possible behaviours, by increasing the action space of the defender system. MARL models are also tuned to establish hyperparameter values to improve the potential for success at their tasks.

4.3.1 Scenario

Variation of the scenario is designed to challenge MARL systems in terms of their scalability by introducing larger network topologies and in terms of adaptability by varying the attacker's objective. In a cyber setting, post-breach attack vectors and goals are many and varied. Therefore, a MARL approach to tactical cyber defence should demonstrate competence when dealing with a range of threat types. CyMARL includes three network sizes and three attacker behaviours (scenario types). Each combination allows for nine scenarios to train and evaluate each MARL defender system.

Network Size Three network sizes vary the task complexity, as shown in Table 4.4. Adding more hosts and subnets exponentially increases the size of the observation and action space of the agents, thus adding more dimensionality to optimizing over policy space. The small scenario is designed to have the minimum feature space while requiring the attacker to move between hosts and subnets to achieve higher rewards. The network has a single server on the operational network (disregarding the defender server, which cannot be compromised or monitored) and three user hosts that the attacker can infect. The medium-sized scenarios increase the number of hosts on the operational and user subnets. The large scenarios introduce the enterprise subnet. All network sizes are depicted in Figure 4.4. The large scenarios do not have direct routing between the user subnet and the operational subnet. Instead, the attacker must pivot through a specific enterprise network host with the operational subnet's routing information. With three subnets in this scenario, the defender system is also charged with managing three agents. The network sizes are based on the sizes evaluated in similar single-agent RL work into autonomous cyber defence (e.g., [30], [47]).

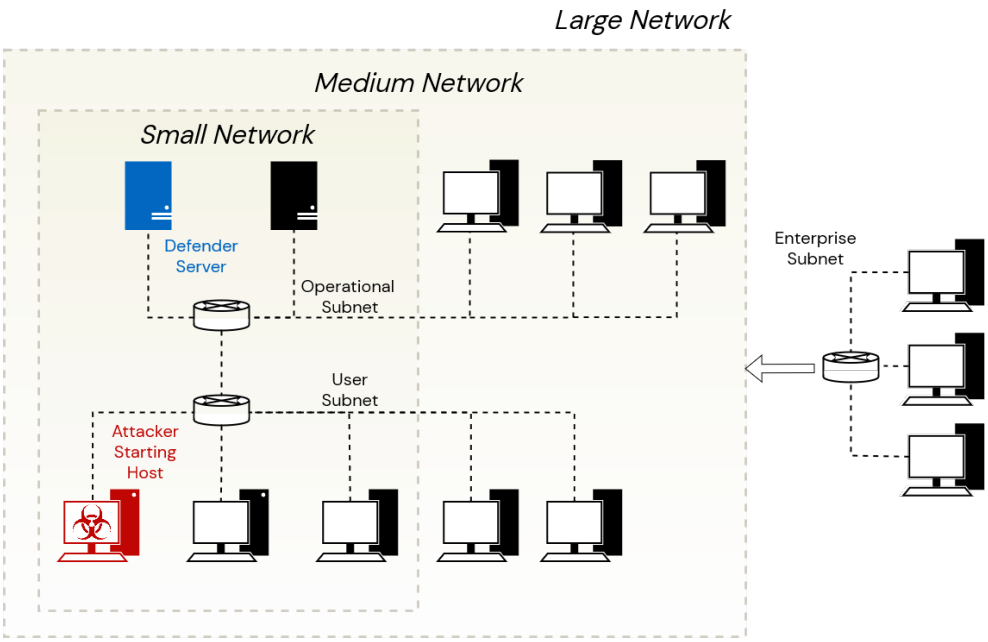


Figure 4.4: Network diagram of small, medium, and large sized networks in CyMARL. In the large scenario, the enterprise subnet divides the user and operational subnets; the attacker must move between each subnet in order.

Table 4.4: Network sizes by defender agents.

	Small	Medium	Large
Number of hosts[1]	5	10	13
Number of subnets	2	2	3
Number of agents	2	2	3

[1]Each network contains an initial foothold host (User0) and a defender server which cannot be affected by attacker and defender actions.

Attacker Behaviour The attacker must perform its scoring action to begin collecting points on a given host. In the confidentiality scenario, rewards are provided to the attacker for maintaining active sessions on hosts, generated as part of the `exploit` action. In the integrity and availability scenarios, the attacker must establish a session before it can perform its scoring action, `tamper` or `deny`, respectively. As outlined in Table 4.5, in response to the attacker's `tamper` action, defender agents are given additional actions to combat the specific threat, `analyze` and `data_repair`. The `remove` action will remove DoS processes and attacker sessions if they exist on the target host.

Table 4.5: Scoring versus defensive action for each scenario.

	Confidentiality	Integrity	Availability
Red Scoring Action	Exploit	Tamper	Deny
Blue Defensive Action	Remove	Analyze, Data Repair	Remove

Given the limitations of the simulation, and the possible policies of attackers and defenders, these three scenarios neglect many TTPs that an attacker may use in a real-life scenario to realize its goals. However, this work is a first step towards understanding how a MARL architecture may learn distinct tactical cyber defence policies in the face of different threat types.

4.3.2 Hyperparameters

The intent of hyperparameter tuning in this experimentation is to maximize the potential performance of MARL models at specific tasks. The hyperparameter values chosen were based on initial trials with the environment and the findings in Hu *et al.* and Hessel *et al.* that evaluated the effect of RL implementation designs in QMIX and DQN, respectively [29], [61]. These works specifically aim to derive the best-performing architectures using implementation tricks. Moreover, Hu *et. al* demonstrates a near-perfect win rate using QMIX on the most challenging tasks in SMAC, and the implementation is open source [29]. Based on the assumption that learning ability will transfer between tasks, this experimentation assumes a degree of transferability between the "hard" SMAC [58] tasks and the scenarios presented in CyMARL.

The use of recurrent neural networks is assumed to outperform the alternative fully-connected NN. This assumption is supported by initial testing in the CyMARL environment. Since the environment is partially-observable, there is a strong intuition that the use of RNNs will benefit learning, expanded on in Section 2.3.

A one-factor-at-a-time technique was used for each trial to vary a single hyperparameter from the baseline in [29]. This technique is limited by its robustness, as relationships can exist between hyperparameter values that lead to better outcomes. However, it has a significant speed advantage over other tuning approaches, such as grid search, allowing for a fast search over a larger range of values. This approach was taken because a grid search is later carried out before the values are finalized for Phase 1 and because it allowed for a greater breadth of search that would otherwise not be possible within the scope of this experimentation.

It is important to identify the differences in optimization strategy between SMAC and CyMARL, as the two environments have significant differences. Eight hyperparameters were evaluated in initial testing:

1) **Batch size**: The number of samples used from experience replay for one learning update.
2) **Buffer size**: The size of the sliding window buffer that stores experience replay samples.
3) **Learning rate**: Weighs the magnitude of learning updates.
4) **Number of rollouts**: The number of parallel environments that a learner samples from.
5) **Orthogonal initialization gain**: Establishes the initial weights of the agents' linear NN layers with a semi-orthogonal matrix by a factor of the gain value.
6) **RNN hidden layer dimension**: The number of nodes (the width) of the hidden layer in each agent's RNN.
7) **Target update interval**: The frequency at which the target network is updated from the primary network. Two-network architecture is adapted from DQN [17].
8) **TD(λ)**: Value of λ, the trace-decay parameter, weights the effect of eligibility traces by applying less discounting to expected future rewards.

Once the baseline is established, three hyperparameters are used to tune the two MARL architectures for each of the nine scenarios using a grid search: batch size, buffer size, and learning rate. These parameters were selected based on their ability to affect performance in initial testing and their effect across tasks presented in [29]. Parameters that, when varied, caused a greater evaluation performance without severe negative changes in performance were preferred.

4.3.3 Noise

Noise refers to stochastic variance in the input to the RL system. In the case of a cyber defence game, noise can be any observation that is not relevant to achieving the defender's goal. Real settings are assumed to have significantly more noise than in simulation. Therefore, the addition of realistic noise sources allows for a closer approximation of real applications. The two sources used in this experimentation are modifying the reliability of the detection sensor and the addition of grey users.

Imperfect Detection A major challenge confronting intrusion detection and host-based security tools is detecting false positives. The enormous volume of data these systems receive compared to the few actual malicious events

makes correctly determining the malicious activity difficult and a major driver in the development of these expert systems. Cyber defence sensors are typically designed to accept higher levels of false-positives if they can reduce the probability of harmful false-negatives. A realistic cyber defence scenario should contain sensors that are not entirely reliable, allowing some false alerts. Inaccurate alerting, in the mislabeling of positive and negative samples, introduces noise into the system that can affect the accuracy of a learning system. In simulation, defender agents can be withheld observation information at a fixed probability. A probability of correct detections allows for tuning of this parameter to determine how noise from imperfect detection affects the MARL system.

Grey Users Grey users are heurisitic models that randomly perform actions in the network. With the addition of grey user actions such as port scanning and process spawning, there is a risk that grey activity may be interpreted as attacker activity, thus leading to false positives. In a realistic network setting, the vast majority of activity is conducted by benign actors. Although the scale of grey users in this experimentation is significantly less than what could be considered average, the actions of just one grey user may significantly affect the MARL system's ability to discern malicious from benign activity.

4.3.4 Defender Capabilities

Defender capability can be extended with the addition of actions, representing tactical options for the defender in the second phase of evaluation. The addition of the deception action allows the defender system to take more proactive tactics when presented with a threat. Furthermore, by providing defender agents with visibility over the entire network environment, they may leverage greater collaboration.

Deception Deception actions allow a simulated defender to stand up a decoy process that appears vulnerable from the attacker's perspective. The attacker may choose to exploit this process which will always fail. The exploit action can alert the defender of the threat without compromising the target host. By deceiving the attacker, the defender system has learned about the attacker's position as a result of the exploit. Moreover, this wastes one of the attacker's thirty turns in the episode. If the defender is frequently successful at deceiving the attacker through an episode of play, it could be an effective tactic to slow the attacker's movement toward higher-value hosts. However, the reward is typically delayed when experimenting with this action using

RL. A defender may set up a decoy that the attacker will not see for a few timesteps or even at all. Learning from delayed reward presents a significant challenge, but through doing so, agents can be significantly more effective at network defence.

Defender Area of Responsibility The final element of the experimentation will examine how changing the observation and action space affects learning in the two approaches. Because the game sets each agent on a segregated subnet, their options for cooperative action are constrained. A re-framing of the defence game, giving all agents visibility and the ability to act over the entire network, may lead to improved learning of decentralized policies. It may, however, exceed the capacity of the MARL systems to parse relevant features due to the curse of dimensionality. Comparing a full visibility approach to the original subnet-separated agents provides insight into MARL systems' ability to scale.

4.4 Evaluation Process

This experimentation comprises two phases: the first phase of evaluation assesses the viability of two cooperative MARL architectures at performing nine host-based tactical cyber defence tasks. The second phase builds upon these results by introducing new game design elements: noise, a deception action, and an increased area of responsibility, to improve the game's applicability to real applications. The intent of this experimentation is not to assess the generalizability of trained models across different scenarios. Rather, specific models will be trained at a variety of tasks to determine how game design elements affect performance. If learning systems that exceed the benchmark performance are limited to simple scenarios, the applicability of the approach would appear to be low. Conversely, learning systems capable of high performance in a large set of game scenarios suggest higher applicability.

Training and evaluation of learning models are performed over a series of trials, each evaluating performance with a particular game design element. As shown in Figure 4.5, each architecture is compared for each experimental trial. Each trial compares the effect of a single design element to the benchmark in terms of mean return. The evaluation scores and training performance curves are used to compare model performance for each trial. Results of training and evaluation runs are measured from the mean of five randomized training seeds to provide more representative results.

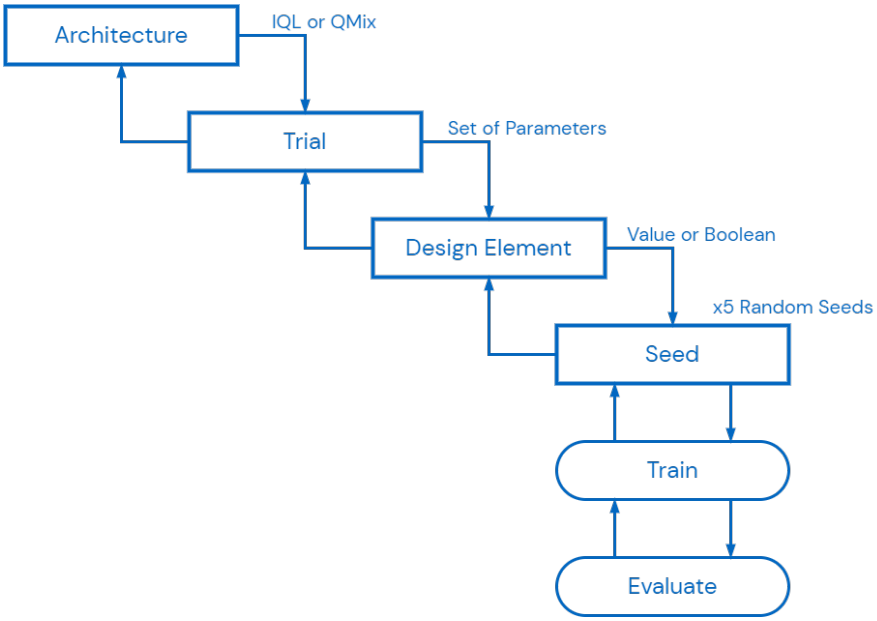


Figure 4.5: The evaluation loop.

4.4.1 Initial Conditions

It is impractical to perform extensive hyperparameter tuning for each scenario and MARL architecture included in this experimentation. Training is constrained by scope and computational resources available. The intent of this experimentation is to be reasonably repeatable, for example, for validation of new scenarios or simulations. The confidentiality-medium scenario was used to establish the baseline for the experimental trials. The medium-sized network provides the greatest probability of representing the median values, minimizing the skew that hyperparameters may have toward a particular scenario. The confidentiality scenarios consist of the core gameplay upon which the other two scenarios build.

4.4.2 Phase 1 Trials

The first phase of trials evaluates the learning ability of each MARL architecture across nine scenarios. The scenarios consist of the three attacker behaviours: confidentiality, integrity, and availability, for each network size: small, medium, and large. Three hyperparameters are tuned using a grid search. The evaluation scores and the training curves of IQL and QMIX are compared. An assessment of MARL system adaptability and its ability

to scale to more complex environments can then be supported based on the performance of learning models relative to the benchmark.

4.4.3 Phase 2 Trials

This phase adds complexity to the most promising game configurations from the previous phase to provide a more robust assessment of MARL system adaptability. For each trial, models are trained in the presence of one game design element and compared to a benchmark policy generated in Phase 1. The objective is to demonstrate how these elements affect learning ability. Each trial relates to one of the design elements discussed in Section 4.3:

1) Grey users
2) Detection noise
3) Deception action
4) Defender area of responsibility

The introduction of complexity from these design elements revealed specific advantages for RL decision-making. By providing a greater range of states and actions to the learning system, there are more opportunities to expand upon the range of applicability of MARL approaches. Moreover, models have greater potential to develop more sophisticated behaviours tailored to their objective, thus showing greater promise for specific, realistic applications.

4.5 Summary

A tactical cyber defender must be capable of recognizing specific patterns within their purview and be capable of protecting, mitigating, and recovering cyber assets through a series of actions. Cooperative MARL may show evidence for such decision-making. The learning ability of two value-based methods is compared across a range of host-based defence tasks to provide an assessment of the applicability of cooperative MARL to cyber defence operations. By varying game design elements, this research seeks to establish a range of applicable tasks in which the evaluated MARL approaches show the potential to learn effective tactics. The framework presented in this chapter provides a set of tactical-level scenarios for multi-agent defender systems. The provision of trained policies at the range of scenarios demonstrates their adaptability and ability to handle complexity in a cyber defence context.

5 Results

This chapter presents the findings of the two phases of evaluation. First, two cooperative MARL architectures, IQL and QMIX, were trained on nine host-based network defence scenarios. The second phase evaluated MARL systems in scenarios with sources of increased complexity. The *learning ability* of a MARL system is quantified by the actual performance of a trained model, the *evaluation score*. From the results of both phases of experimentation, the capabilities and limitations of MARL systems in this game are discussed in terms of learning adaptability and game complexity. These results are limited by a small sample size of trials investigating a particular game design element. In a limited number of trials, this experimentation sought primarily to evaluate individual parameters that may point to utility in real network settings.

This chapter is broken into the following sections: Section 5.1 discusses the experimental setup, Section 5.2 presents the initial conditions, Section 5.3 presents the results of the first phase of trials, Section 5.4 presents the results of the second phase of trials, Section 5.5 discusses the findings and points to trends in network size, scenario type, noise, and MARL architecture. Section 5.6 summarizes.

5.1 Experimental Setup

MARL systems are trained for one million timesteps. A timestep is represented in the game as a turn in which each agent takes one action. The outputs of training are the training results and a saved model. The training results include a variety of metrics collected at 10 000 timestep intervals. The saved model contains all of the parameters of the learning model, effectively a saved policy. Each trained model is evaluated over 1000 timesteps of play without learning. The evaluation score is the average of the mean return of five evaluation runs using separately trained static policies for each trial. Evaluation scores are compared using percent difference, calculated by sub-

tracting the smaller score from the larger score and then dividing the result by the smaller score. The variance of models is expressed in the standard deviation of the mean score. Standard deviation is calculated as the square root of the variance of the mean return of the evaluation runs. When variance is compared between results, the Coefficient of Variation (CV) is used. CV is calculated from the standard deviation divided by the mean return, expressed as a percentage.

Action masking is not used on the action space by default, so agents must learn which actions are invalid based on the context of their observations. Actions are parameterized with information that allows agents to specify the type and target of their actions. The PyMARL framework does not natively support parameterized actions, so the action space of each agent is parsed as a vector with a single element for each possible combination of action parameters.

Trial data were collected using the PyMARL [58] command line interface. This work used the algorithms from Hu *et al.* 's implementation, PyMARL2, which builds additional capability into some of the MARL algorithms in the original project [29]. PyMARL saves training and evaluation data for each run within a trial to JSON-formatted output, from which the relevant information is read into CSV format for analysis and plotting. Trials were run on Windows 10 using a batch script to run sequential experiments.

5.1.1 Limitations

Where robust evidence was required, evaluation scores were calculated from five separately trained policies. The objective of this technique is to reduce the impact of variance between runs skewing the final values, which can be severe in some RL settings. Due to computational constraints, this technique was used to validate final results and not for every experiment run. For example, it was not feasible within the scope of this experimentation to train five randomized seeds for each grid search combination. Rather the grid search was run for a single iteration for each combination of hyperparameter values. The best-performing models were then validated using five randomized training seeds. This approach does not account for "unlucky" runs in which hyperparameter values may have performed better if taken from an average, however, this is an acceptable tradeoff in limiting computation time.

In some cases, one million training steps was insufficient to observe model convergence to a policy. However, one million timesteps of training over five seeds can show how quickly and how well models learn in this setting given changing conditions. The findings of this experimentation provide evidence to

suggest performance potential within the training period rather than to find the performance limit of these methods.

Three hyperparameters, batch size, buffer size, and learning rate, were varied as part of the grid search in Phase 1 of this evaluation, a total of eight values. This is less tuning than what is commonly performed for MARL grid searches in the literature. The grid search is not intended to be a robust assessment of hyperparameter values. Instead, it is meant to provide models with an improved representation of their actual potential. To mitigate the relatively small tuning set, a larger set of hyperparameters were varied using one-factor-at-a-time to establish the initial conditions.

5.1.2 Benchmarks

Phase 1 uses the evaluation score of a heuristic defender model as a benchmark of performance for each scenario. The heuristic defenders are designed to simulate the performance of a basic defence policy. In the confidentiality scenarios, this policy monitors all hosts within each agent's subnet, and restores hosts that provide an alert. The `restore` action was chosen because it scored higher in testing than using combinations of other actions. The heuristic defender will always take the monitor action until it observes the presence of one or many new sessions on its hosts, a which point it will respond. In the availability scenario, detection occurs due to a DoS process being created, not an attacker session. The availability defender will restore on detection. In the integrity scenarios, the heuristic defender will attempt to remove malware on a host that it has discovered to have a tampered file with a 50% probability and restore otherwise. To achieve the resulting behaviour, the heuristic defender models uses specific information from the true state of the game rather than through an observation signal, as is the case with the MARL systems. Since using `monitor` to detect and `restore` to re-image affected hosts takes two turns, the attacker has the opportunity to stay one step ahead if it continues to exploit new hosts. Although the heuristic defender has perfect game information, its policy is restrictive enough to allow the attacker opportunities to overcome its defences.

5.2 Initial Conditions

Eight hyperparameters for each architecture were trained against the CyMARL confidentially-medium scenario using a one-factor-at-a-time method. Table 5.1 presents the best-performing hyperparameter values in terms of

mean return. The values that produced the highest mean return for each hyperparameter for each architecture were used as the initial configurations for the experimentation.

Table 5.1: Selected hyperparameter values for each architecture from initial tuning at the confidentiality-medium scenario.

Trial	Values	Initial Setup	
		IQL	QMIX
Batch size	64, 128, 256	256	128
Buffer size	5000, 10000, 20000	10000	10000
Learning rate	0.001, 0.002	0.0005	0.001
Number of rollouts	4, 8, 16	8	8
Orthogonal initialization gain	0.01, 0.02, 0.04	0.02	0.01
RNN hidden layer dimension	64, 128, 256	64	64
Target update interval	100, 200, 400	200	200
TD(λ)	0.0, 0.2, 0.6, 0.9	0	0.9

5.3 Phase 1

The first phase of trials evaluated the learning ability of each MARL architecture in nine scenarios. After training for one million timesteps, the policies of each model were evaluated in terms of mean return for each scenario. There is no clear preference of an architecture for a certain set of hyperparameters across the scenarios, nor is there a strong correlation between the size of the network and the magnitude of the best hyperparameter values. IQL and QMIX observed a respective 3.24% and 12.35% increase in score as a result of hyperparameter tuning, averaged over all scenarios. The selected hyperparameter values for each scenario are provided in Appendix B.

Figure 5.1 presents the training performance curve of each model for the nine scenarios[1]. The curves for IQL in red and QMIX in blue are the mean return over the five randomized seeds, calculated from the previous 10 000 training steps. The learning curve shows the change in performance via iteratively policy updates for one million timesteps beginning from a randomized initial state at timestep zero. The shaded area of each curve is the standard

[1]Re-scaled learning curves separated by network size can be found in Appendix B.

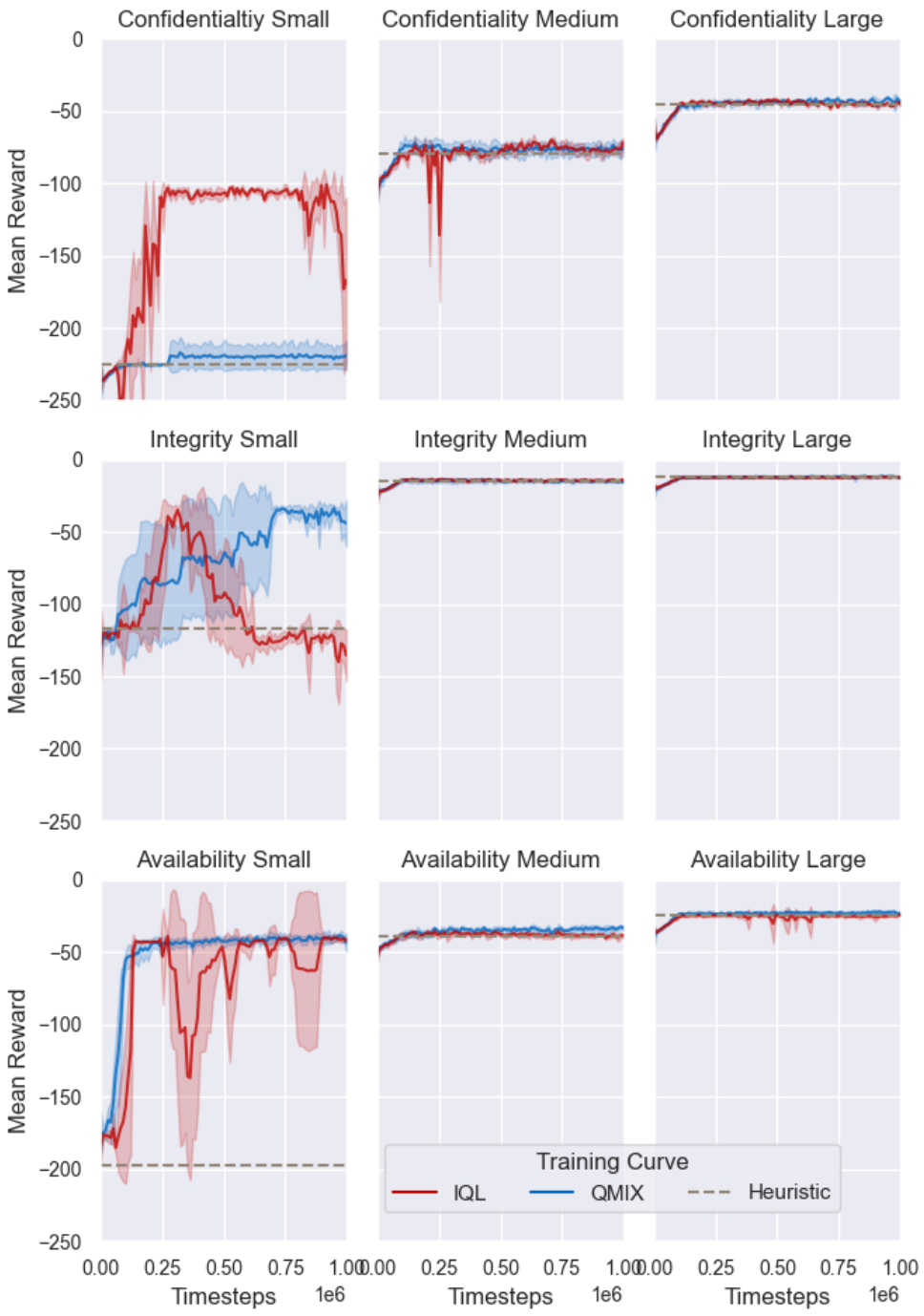


Figure 5.1: Learning curve of IQL and QMIX in each Phase 1 scenario. The shaded area represents the standard deviation.

deviation of the average training return. The dashed line is the mean heuristic defender score for each scenario.

In most scenarios, the learning curves closely fit (within one standard deviation of) the heuristic score after a period of improvement. The small scenarios exhibited the greatest difference in score between the learning systems and the heuristic. The small scenarios also exhibit the greatest variance in both architectures. In the confidentiality-small and integrity-small scenarios, IQL is observed to lose performance after a period of improvement. Figure 5.2 shows each MARL model's evaluation score and standard deviation in the nine scenarios. Figure 5.3 groups the evaluation scores together by average over network size and scenario type.

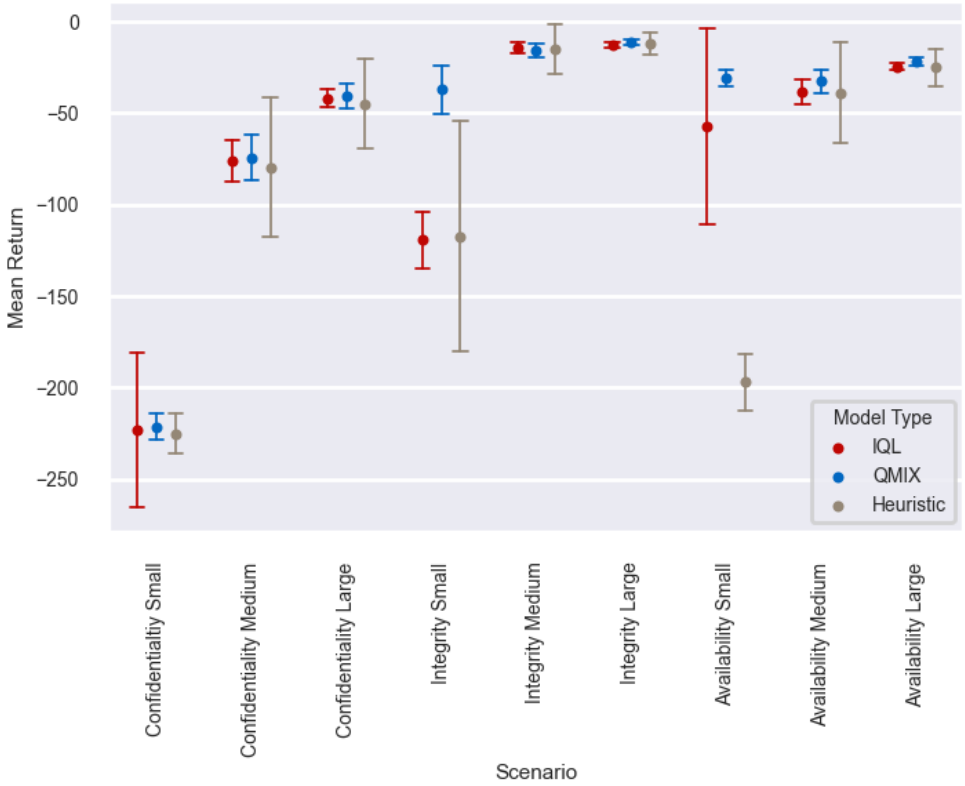
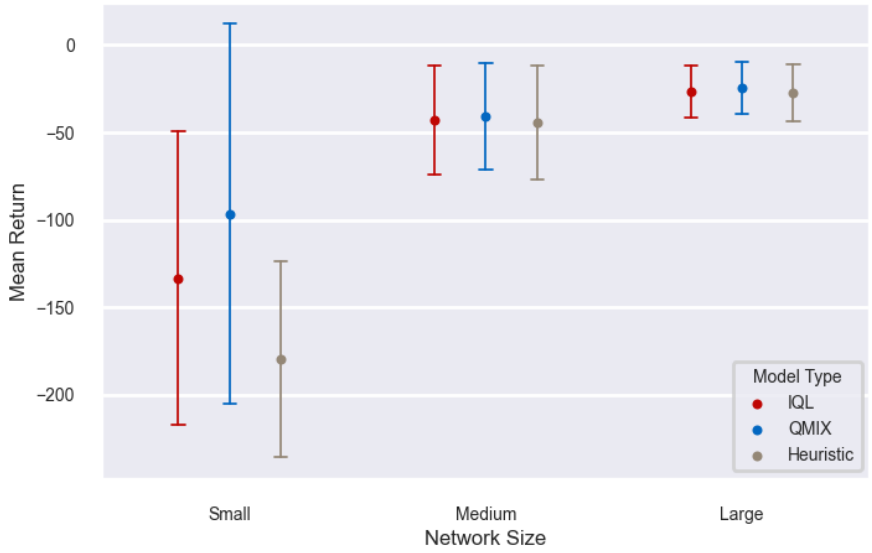
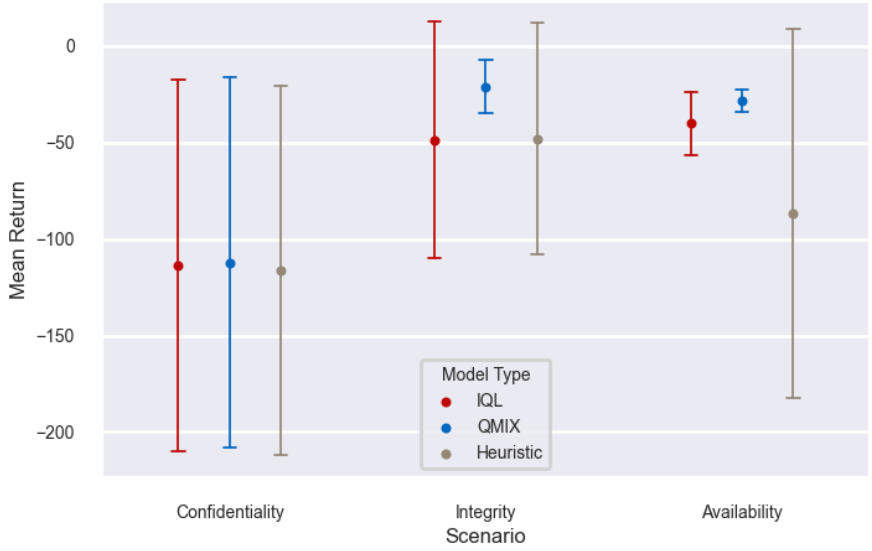


Figure 5.2: Evaluation score of IQL, QMIX, and heuristic defender in each Phase 1 scenario. Bars represent the standard deviation.

In Table 5.2, the evaluation score of each learning model is listed against the score of the benchmark heuristic for each scenario. IQL and QMIX each



(a) Mean evaluation score of each network size.



(b) Mean evaluation score of each scenario type.

Figure 5.3: Evaluation score of IQL, QMIX, and heuristic defender in each Phase 1 scenario grouped by network size in Subfigure 5.3a and by scenario type in Subfigure 5.3b. Bars represent the standard deviation.

generated policies that outperformed the heuristic at seven and eight scenarios, respectively. QMIX outperformed IQL in every scenario except for integrity-medium. QMIX had a 16.86% greater evaluation score than IQL on average. IQL exhibited high variance in the small scenarios and significantly lower scores than QMIX in the case of integrity-small and availability-small. Averaging over only the medium and large scenarios, QMIX outperformed IQL by 5.83%. The average CV of QMIX in the medium and large scenarios was 16.95% compared to the IQL CV of 14.23%. Despite the high variance of IQL in the small scenarios, both architectures observed the greatest performance improvement over the heuristic defender in the small scenarios.

Table 5.2: Evaluation Score by scenario '+/-' the standard deviation of five randomized training seeds. The highest score for each scenario is given in bold font.

Scenario		IQL	QMIX	Heuristic
Confidentiality	Small	-223.12 ±42.46	**-221.47 ±7.25**	-225.13 ±11.02
	Medium	-75.78 ±11.4	**-74.27 ±12.38**	-79.5 ±37.85
	Large	-42.07 ±4.86	**-40.64 ±6.56**	-44.92 ±24.35
Integrity	Small	-119.34 ±15.54	**-37.01 ±13.09**	-117.18 ±62.99
	Medium	**-14.35 ±3.23**	-15.58 ±3.9	-14.91 ±13.3
	Large	-12.56 ±1.58	**-11.16 ±1.62**	-12.21 ±6.06
Availability	Small	-57.16 ±53.54	**-30.75 ±4.62**	-196.96 ±15.32
	Medium	-38.52 ±6.62	**-32.53 ±6.21**	-38.8 ±27.39
	Large	-24.65 ±1.9	**-21.77 ±2.53**	-24.96 ±10.19

The integrity scenarios posed the greatest challenge to IQL, scoring slightly below the heuristic on average (-1.33%), whereas QMIX performed well (+55.82%). Both architectures scored similarly in the confidentiality and availability scenarios. The learning systems scored on average 3.11% and 64.93% greater than the heuristic defender at the confidentiality and availability scenarios, respectively.

5.4 Phase 2

The confidentiality-large scenario was selected as the benchmark score for the investigation of the Phase 2 game elements on the MARL systems. The confidentiality scenario type uses a simpler attacker behaviour and defender action space, and the larger network size is more desirable for evaluating the scalability of the MARL systems. The intent of the second phase of

trials is to evaluate how the MARL systems respond to different sources of complexity relating to realistic aspects of cyber defence. This phase does not perform a robust assessment by taking all models to their failure point but rather points to trends in learning ability under different conditions. The ability of MARL systems to maintain levels of performance under adverse conditions, or to show improvement under beneficial conditions, is key to supporting their applicability to the network defence setting. This phase of trials consists of three parts. First, MARL models are evaluated in settings with increased noise using two design elements: inaccurate sensors and grey agents. Second, an additional action, `misinform`, is added to the defender's action space. Finally, agent action and observation spaces are modified to increase their scope of responsibility within the network. Evaluation scores in these trials are compared to scores from the original confidentiality-large scenario.

Noise Noise challenges an agent's ability to learn from the effect of environmental changes, such as the attacker's actions and the effect of defender actions in the game. This set of trials used two sources of noise: inaccurate sensors and the presence of grey users taking actions on the network. The inaccurate sensor scenario decreased each defender agent's probability of being alerted of attacker activities. In the default scenario, `monitor` action always reports process creation on a host accurately. The noise scenarios cut the probability of accurate alerting from the `monitor` action to 50% and to 25%. The defender agents still have visibility of their hosts' processes; only the alerting via the `monitor` action is disrupted. Figure 5.4 shows the mean return of each architecture when its `monitor` action is 50% and 25% effective, compared to 100% effectiveness in the original scenario.

Reducing sensor accuracy had only a small effect on the ability of both MARL architectures to learn a policy similar to the benchmark scenario. Notably, in the 50% scenario, the effect on the QMIX evaluation score was negligible (-0.14%), whereas IQL lost 3.27% from the benchmark. Reducing the alerting accuracy of `monitor` to 25% followed a similar trend. QMIX scored 0.94% lower than the benchmark, and IQL scored 6.93% lower.

The second design element relating to game noise was the use of grey users to confuse the defender. These agents take actions similar to an attacker, such as performing scans and establishing connections with other hosts. A scenario with a single grey agent and with five grey agents were trialled to evaluate their effect on learning ability. The mean returns of these scenarios are shown in Figure 5.5.

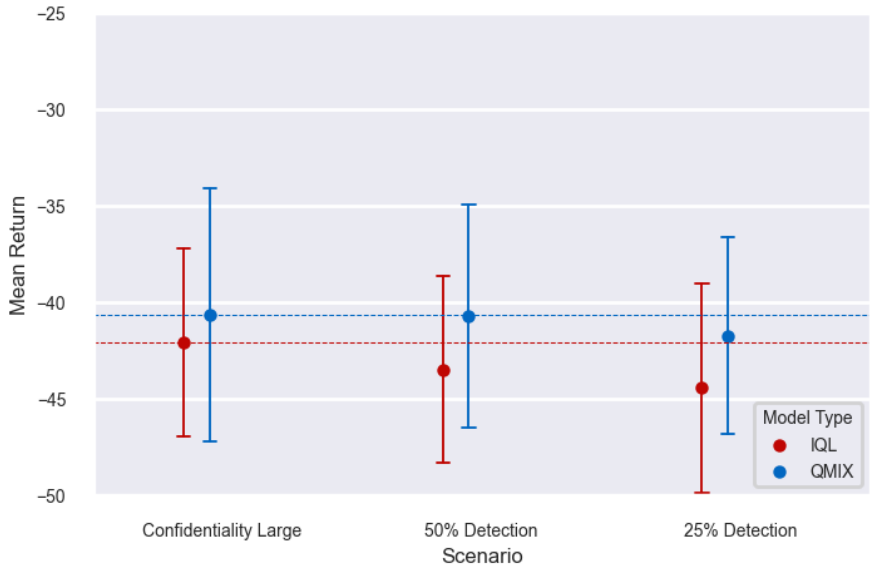


Figure 5.4: Evaluation score and standard deviation of IQL and QMIX at 50% and 25% alert accuracy. Dashed lines indicate the original confidentiality-large scores for each architecture.

Similarly to unreliable sensors, the addition of grey agents does not significantly impact the final policies of the MARL models. Surprisingly, the models performed better in the presence of five agents than with one. As with the other noise source, IQL was more negatively affected by this trial, though only slightly, with a 4.30% lower than benchmark evaluation score compared with the QMIX loss of 3.86% in the single grey agent scenario. Table 5.3 shows the IQL and QMIX evaluation scores and standard deviation at each scenario averaged over five randomized seeds.

The greatest effect of the noise scenarios was an average reduction of 4.02% of both MARL systems in the 25%-accuracy scenario. The noise scenarios did not significantly affect the variance of the models compared with the original scenario.

Defender Capability The next experiment evaluated the ability of MARL systems to learn to use a deception action in addition to standard actions. The `misinform` action allows an agent to create a decoy process to mislead an attacker to attempt to exploit the perceived vulnerable process. Exploiting a decoy does not provide the attacker with visibility of the host system or

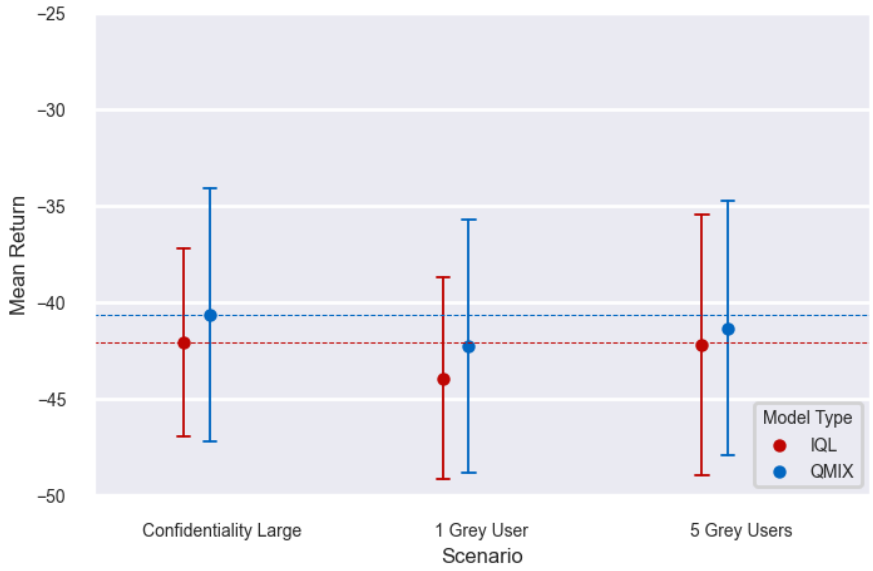


Figure 5.5: Evaluation score and standard deviation of IQL and QMIX in the presence of one and five grey agents.

Table 5.3: Evaluation Score by noise trial '+/-' the standard deviation of five randomized training seeds.

Scenario	IQL	QMIX
Confidentiality-Large (benchmark)	-42.07 ±4.86	-40.64 ±6.56
One grey user	-43.96 ±5.22	-42.27 ±6.58
Five grey users	-42.24 ±6.76	-41.34 ±6.61
50% sensor accuracy	-43.49 ±4.82	-40.70 ±5.78
25% sensor accuracy	-44.43 ±5.42	-41.74 ±5.09

rewards. The evaluation score and standard deviation of models using this action are compared with the benchmark in Figure 5.6 and Table 5.4

The use of the `misinform` action had a negligible effect on IQL performance relative to the benchmark (-0.54%). Conversely, the new action allowed QMIX to make a significant improvement of 18.39% over the benchmark score. The effect on the standard deviation of IQL and QMIX was an increase of 10.12%, and a decrease of 0.63%, respectively.

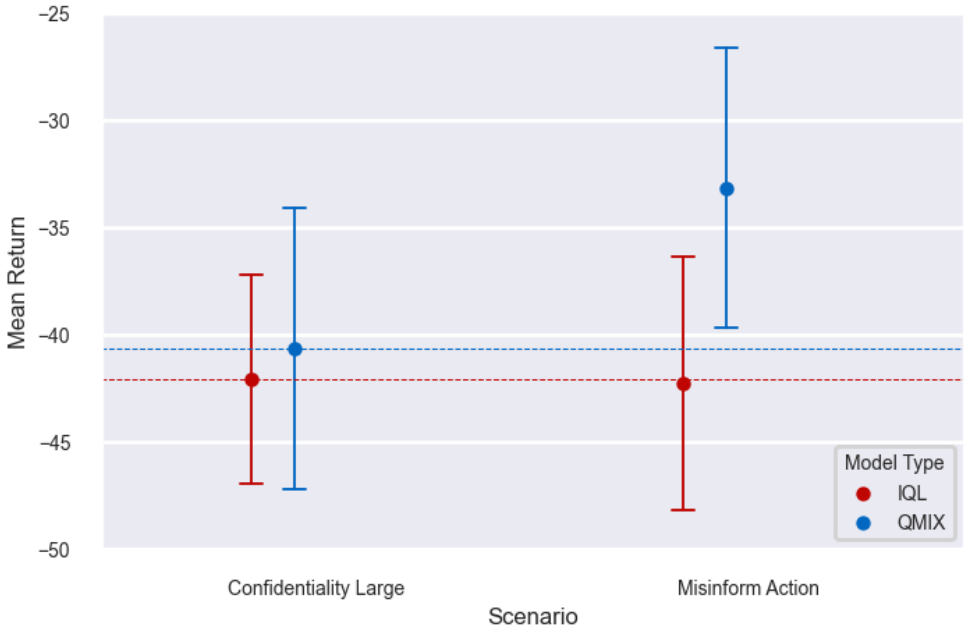


Figure 5.6: Evaluation score and standard deviation of IQL and QMIX when provided with the `misinform` action.

Table 5.4: Evaluation Score of `misinform` trial '+/-' the standard deviation of five randomized training seeds.

Scenario	IQL	QMIX
Confidentiality-Large (benchmark)	-42.07 ±4.86	-40.64 ±6.56
`misinform` action	-42.29 ±5.92	-33.16 ±6.52

Agent-Environment Interaction The final trials in this phase remove the subnet constraints to assess the learning ability of MARL agents with a larger state and action space. The first scenario changes the original framework by allowing all three agents access to observations and actions across all subnets. The second scenario uses the same complete visibility task but employs two agents instead of three. The final scenario utilizes two heterogeneous agents with observability of the entire network. Heterogeneous agents in this game are differentiated by their action spaces. Rather than identical agents with the same policy space as seen up to this point, heterogeneous agents must learn to use different actions. Both agents can use the `monitor` action as it is

fundamental to collecting information about the network, but each agent has access to only one other action, either `remove` or `restore`. The evaluation scores of each scenario are compared to the benchmark scenario in Figure 5.7 and Table 5.5. Each scenario except for the benchmark allows MARL models to observe the entire network and target any host with its actions. The complete visibility scenario adapts the three agents from confidentiality large whereas the two-agent scenario and the heterogeneous scenario each use two agents.

IQL and QMIX exhibited improved performance in the "complete visibility" scenario, an average increase of 20.81% on average. The two-agent scenario scored above the benchmark by 26.63%, and the heterogeneous agents scored above the benchmark by 30.71%.

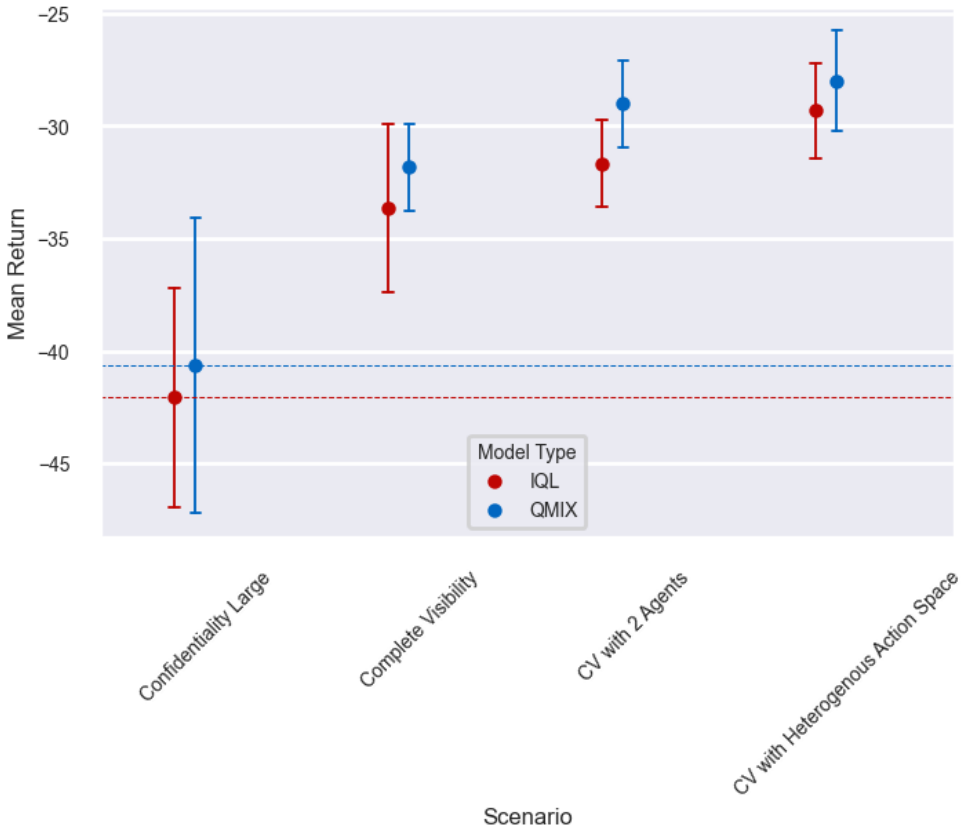


Figure 5.7: Evaluation score and standard deviation of IQL and QMIX in a complete network visibility scenario with three agents, two agents, and two agents with a heterogeneous action space.

Table 5.5: Evaluation Score by agent responsibility trial '+/-' the standard deviation of five randomized training seeds.

Scenario	IQL	QMIX
Confidentiality-Large (benchmark)	-42.07 ±4.86	-40.64 ±6.56
Complete visibility of network (3 Agents)	-33.65 ±3.76	-31.84 ±1.93
Complete visibility of network (2 Agents)	-31.67 ±1.92	-29.01 ±1.94
CV with heterogeneous action space	-29.31 ±2.12	-28.00 ±2.25

5.5 Discussion

The design of this experimentation opted for an exploration of game design elements. The advantage of this approach is that, with a limited number of trials, many game design effects can be observed thus providing a more rounded assessment of MARL applicability. However, it is limited by a small sample size of trials from which to draw relationships. For example, the effect of an increased network size, without increasing the number of agents, is observed in three samples for each algorithm; between the small and medium scenarios of each type. The effect of noisy sensors and grey agents are observed in two samples each for each algorithm. The sample size brings the difficulty of attributing results to specific game elements or learning patterns. The intent of this work is not to present correlations that generalize the effects of specific game design parameters on learning ability. Instead, this experimentation demonstrated how game design, particularly in how the reward and observation signals are shaped, significantly affects the learning ability and, therefore, the applicability of learners to autonomous cyber defence.

This section discusses the effects of network size and scenario type on MARL learning ability before discussing the observed differences between the two architectures. This discussion provides a depiction of MARL system behaviour in this game by linking observed performance to game design elements and to related work in cooperative MARL.

5.5.1 Network Size

Network size had the greatest effect on model outcomes. In the small scenarios, each agent was responsible for three and two hosts on the user and operational subnets, respectively. In the medium scenarios, the input-output size of the observation space and action space was increased to five hosts occupying each

subnet. In the large scenarios, the observation space and action space were not increased because the addition of hosts comes with a new subnet, enterprise, with an added defender agent.

When the input-output size was increased, MARL model performance decreased relative to the heuristic benchmark, depicted in Figure 5.3a. On average, MARL systems maintained their performance advantage over the heuristic between medium and large scenarios. Increasing input-output space negatively affected learning, whereas the addition of an agent did not have a significant effect on relative performance.

Game Design There is a distinct difference between the learning curves in the small scenarios and those of the medium and large scenarios. MARL models trained at medium and large scenarios followed a common period of policy improvement, typically under 200 000 timesteps, followed by convergence to a suboptimal policy, as seen in Figure 5.1. In the small scenarios, MARL systems exhibited significantly more variance, particularly in the case of IQL.

Larger scenarios offer more choices for an attacker. The attacker will randomly choose which host to exploit, so more hosts leads to greater stochasticity in the game environment. This is a defensive benefit since it decreases the probability that the attacker will choose the optimal path. With an attacker taking more actions in the user subnet, the defender has more opportunities to intercept or mitigate the attacker's activities. In a larger network scenario, over many episodes of play, MARL systems will have more consistent training stimulus for the early removal of an attacker. The large scenarios have the additional benefit of an intermediary enterprise subnet. Since the penalty for attacker activity in the enterprise network is not as severe as in the operational subnet, the defender scores higher in a large scenario than in a medium scenario for an attacker following the same heuristic.

Given that medium and large scenarios have the same model input-output space and the learning curves follow similar patterns across all scenario types, the higher average score observed in the large scenarios can be accounted for by the lower subnet importance score of hosts in the enterprise subnet. An experiment was run to validate this claim changing the importance score of hosts in the enterprise subnet to equal to those of the operational subnet in the confidentiality-large scenario. The trained policies of IQL and QMIX performed 83.99% worse on average in with the increased importance score of the enterprise subnet than in the benchmark scenario.

Input-Output Space and Score As the size of the network increases, the ability of the attacker to achieve its objective is negatively affected. Correspondingly, the heuristic defender model and both MARL architectures were observed to have an improvement in score between each level of network size, as shown in Table 5.6. The change in score between the medium and large-sized scenarios was similar for IQL and QMIX. When comparing the average MARL system change to the heuristic defender, the small to medium change differed by 85.49% between the average learning system and the heuristic defender. Conversely, the medium to large change differed by 4.13%. This suggests that the additional subnet and agent in the large scenario do not significantly affect how the MARL models learned.

Table 5.6: Average change in Evaluation Score between network sizes, in absolute value and percentage difference.

Network size change	IQL	QMIX	Heuristic
Small to Medium	90.33 (+67.81%)	55.62 (+57.69%)	135.35 (+75.3%)
Medium to Large	16.45 (+38.37%)	16.28 (+39.9%)	17.04 (+40.33%)

Increasing the input-output space can reduce performance as a result of the curse of dimensionality as models may derive less effective patterns from their observations. However, this reasoning is not supported by later experimentation. MARL systems were observed to improve performance in the presence of complete network visibility; a significant increase in input-output space size. Moreover, they were observed to be resistant to noisy observations. These findings suggest that the MARL systems trained in this game had a low sensitivity to their observations. MARL models were likely relying primarily on the reward signal to learn policies. This game reveals opportunities for MARL scalability since the increased network size, hence input-output space, was not observed to negatively affect learning ability at this scale.

5.5.2 Scenario Type

The availability and integrity scenario types employ an attacker that must perform an additional action upon establishing a session to score points. The attacker in these scenario types is therefore slower and will tend to score less in a fixed-timestep game. A validation experiment was run in which no defenders were played against the attackers in each Phase 1 scenario. In these trials, the score is higher in the integrity and availability scenarios than in the confidentiality scenarios, 55.74% and 24.93% higher, respectively. This

score difference is likewise represented in the average scores of the MARL systems: 69.00%[2] and 67.19% greater in the integrity and availability scenarios, respectively, than in the confidentiality scenarios (see Table 5.2).

The integrity scenario type requires the use of incident response actions to remove attacker presence, `analyze` and `data_repair`. Since the standard `remove` action is only effective for attacker sessions and DoS processes, these additional actions allow the defender to detect and remove malicious files without needing to use the costly `restore` action. The optimal tactical policy in the integrity scenarios is to detect and remove all attacker sessions before the option to score becomes available. Assuming that the defender does not act optimally and the attacker is occasionally able to perform the scenario scoring action, the best policy involves a combination of proactive and reactive responses to the threat. The heuristic defender only acts reactively to attacker scoring using the `data_repair` or `restore` actions to remove the attacker's scoring artifacts. To understand how MARL-derived policies deviated from those of the heuristic defender models, action selections were collected from a sample of trained policies over 1000 timesteps. This experiment revealed that both IQL and QMIX strongly preferred acting proactively with the `remove` action to kill attacker sessions in all scenarios. Further analysis of action selection is tangential to the methodology set out in this work. An investigation of policy selection for cyber defence decision-making is left for future work. The `remove` action prevents the attacker from proliferating, but it will not remove tampered files, which continue to generate negative rewards in the integrity scenarios. Although MARL-generated policies outperformed the heuristic model, it is expected that greater gains are possible if a defender learns to balance the use of proactive and reactive tactics depending on its circumstances.

5.5.3 Architecture

The evaluation score of QMIX is within one standard deviation of IQL in eight of nine Phase 1 scenarios and all Phase 2 scenarios. Moreover, the training curves followed a similar pattern in the medium and large scenarios. These findings suggest that QMIX was learning similarly to IQL. Despite this, QMIX consistently outperformed IQL by a small margin, only conceding the top score in one scenario, integrity-medium.

[2]The average MARL system score difference between the integrity and confidentiality scenarios is skewed by the poor performance of IQL at the integrity-small scenario. If this outlier is excluded from the mean, the percent difference is 83.93%.

The multi-agent host-monitoring game presented does not require explicit coordination for a defender team to be successful. Moreover, the segregation of agents into subnets limits their ability to influence the state of other agents. However, the findings of the second phase of evaluation suggest that it is not just the design of the scenario, but also the observations that may limit the effectiveness of CTDE as compared to independent learning. In Phase 2, agents exhibited low sensitivity to changes in their observation spaces. The CTDE advantages of multi-agent coordination and mitigation of the non-stationarity problem are severely limited if agents struggle to derive utility from their observations. A sparse observation space may hinder agents' ability to derive useful information from their inputs. In turn, QMIX's central learning updates may have done little to condition agents on the policies of their peers. The effect of non-stationarity should be greater in a scenario like complete visibility where agents' actions influence the same elements in the environment. Due to the relatively similar performance between the two architectures, it is likely that QMIX did not learn to leverage multi-agent coordination.

QMIX was more resistant than IQL to noisy observations, and it was able to learn to leverage the `misinform` action, which IQL failed to do. This discrepancy can be explained as a result of QMIX learning from the batched experience of all agents rather than solely from independent experience. Although the observations in this game are expected to have a small impact on learning, QMIX also trains on each agent's actions over the sampled history. The performance advantage of QMIX is dampened by its limitation of scalability as compared to IQL. QMIX requires that Q-functions are learned centrally, thus limiting the number of possible agents since the central learner will eventually create a bottleneck. IQL does not require any central learning, therefore, the number of agents in a game does not pose a constraint.

5.6 Summary

Increasing network size had a negative effect on learning ability due to the consequential increase in action and observation space. However, it also insulated the higher-valued hosts from a randomly exploring attacker. The defender scored higher at the integrity and availability scenarios in large part due to the attacker needing to take two turns to perform its scoring action on a host. Both MARL architectures performed similarly at the majority of tasks. They exhibited this similar behaviour in the complete network visibility scenarios and were resistant to noise. Both approaches likely had low sensitivity to changing observations, which may be due to a sparse observation signal.

MARL models did not fail to learn even in extreme cases, still converging to a reasonable policy. Although these systems outperformed the heuristic, the information provided in the agents' observations appeared to limit the learning ability of both architecture types. More performant solutions to tactical cyber defence problems are possible through further study and experimentation with game design.

As a first exploration of cooperative MARL for the task of host-based enterprise network defence, this experimentation evaluated diverse game designs to challenge the MARL systems in the context of a broad range of design elements. This work applied realism and technical challenge to demonstrate the applicability of cooperative MARL approaches to the tactical cyber defence setting. These results support the adaptability and complexity of cooperative MARL systems in tactical-level decision-making for cyber defence.

6 Contributions and Future Work

This chapter discusses the contributions and limitations of this work as it relates to the applicability of cooperative MARL to learn tactical-level decision-making for cyber defence. The primary contribution of this work is a demonstration of how cooperative MARL can be used in a host-based monitoring setting to learn diverse tasks. The simulation was designed to have many states to more closely model realistic network defence settings. The results of this work show that independent and CTDE value-based cooperative MARL architectures can learn policies that outperform a basic heuristic model at this game. Future gains in performance are expected to be possible both in terms of the learning ability of MARL systems and in the realism of training games. Future work in this area could provide greater confidence in the applicability of this approach to learning cyber defence tactics. This chapter is broken into three sections: Section 6.1 lists the main contributions of this work, Section 6.2 suggests future work in four promising areas, and Section 6.3 summarizes.

6.1 Contributions

This work makes the following contributions:

1. To the best of our knowledge, CyMARL is the first cooperative MARL training environment for enterprise network defence tasks. It extends the CybORG simulator with additional game types, actions, and network topologies and it provides a PyMARL environment interface with which it is possible to train tens of open-source algorithms.
2. This work compared the utility of cooperative MARL algorithms in a tactical cyber defence context. It provided a comparison of independent and centrally-learned policies for multi-agent enterprise network defence.

3. This work demonstrated the adaptability of cooperative MARL methods to learn to defend a network in a variety of scenarios without any *a priori* knowledge of the game environment.
4. This work identified the need for more robust feature selection in complex cyber defence RL games such as CyMARL and CybORG to enable more effective learning of observations. Due to the large feature space necessary to model the complexity of computer networks for RL, meaningful information is sparse. This work suggests a variety of possible solutions to improve learning opportunities in future experimentation.

6.2 Future Work

This work indicated the potential of MARL to learn decentralized policies in varied host-based cyber defence settings. In particular, the performance of IQL at these tasks suggests that this approach may scale well to larger networks with more agents. The performance of MARL systems was limited by their insensitivity to observations. It is likely that this behaviour is intrinsic to the scenario design and therefore we may expect that IQL and QMIX can achieve higher levels of performance given modifications to the game. To overcome the limitations of the environment brought forward by this work, and to advance the use of cooperative MARL for tactical cyber defence tasks, the following topics for future work are discussed:

1. Techniques to approximate greater environmental realism, including the use of generative programs to increase the sample size of experimentation,
2. Games offering more sophisticated and varied threat types,
3. Different MARL agent-environment interaction techniques to improve learning ability, and
4. Leveraging promising techniques in machine learning, evolutionary learning, and game theory to improve the representational capacity of the learning system.

Environment The simulated cyber defence game presented here provides a base level of information that may be available from a host-monitoring service. The use of simulated processes, files, and sessions allows for significantly more detailed observations than graph-network abstractions commonly used to frame the cyber defence task for RL systems. More complex simulations are possible, including abstractions of network traffic (e.g., [47]). This may, however, be at the expense of RL skill adoption.

A suspected limitation of the performance of MARL models observed in this work is the large, highly variable observation space. Dynamic allocation of input features could be adopted to reduce the overall size of the observation vector while mitigating the loss of useful information. Observation embedding has been shown to reduce the dimensionality characteristic of multiple multi-agent tasks [62].

Cyber defence operations have a large amount of variability, and the number and type of scenarios in this work provide only a small sampling of data to support the use of MARL in this setting. Generative programs could be used to train MARL models on a larger range of tasks. Generative tasks have been adopted by the latest version of SMAC [63] and in the FARLAND cyber defence environment [47]. The resulting data could provide stronger support for correlations between scenario elements and training techniques.

A critical link to real cyber defence applications is the use of emulated network artifacts and typologies. Emulation allows for improved task realism for training RL systems. Specifically, the use of emulation can parameterize the problem space with realistic actions and observations for simulated tasks, and it can be used as a means to validate the performance of RL trained in simulation.

Adversaries The heuristic attacker behaviour used in this work set a basic policy that was sufficiently challenging to measure the skill adoption of MARL systems in the presented scenarios. Just as there is potential for MARL systems to improve learning and for environments to adopt more complexity, attackers can be modelled with more realism by behaving with a greater diversity of behaviours, more closely approximating real threats. For example, attacker behaviour could be modelled using the CALDERA framework [64], using real TTPs to penetrate a network to service a specific goal. Likewise, the integration of real tooling into the defensive force could allow MARL defender agents to develop realistic, useful policies.

Furthermore, competitive RL self-play may be investigated to provide even more realism via an attacker that adapts to the defender agents' strategies. Self-play has been demonstrated to train attacker and defender agents in a one-on-one network defence game [65], and teams of agents have learned to compete in non-security games [66]. These techniques could be applied to improve the capabilities of one or multiple attacker agents, thus allowing for the training of more robust defender policies.

An advanced threat facing RL defender agents is adversarial attacks on RL models. Adversarial RL attacks are particularly relevant to real network

applicability since the tool providing the defensive policy must itself be secure and reliable. In applications of RL for tactical computer network defence, poisoning and evasion attacks have been leveraged by attackers to reduce the ability of RL defender agents [43], [47]. Adversarial techniques have seen use in MARL settings also [67]. These techniques could be implemented as part of the threat model in a cyber defence game for robust MARL defenders.

Agent-Environment Interaction Complexity is necessary for developing training environments that can provide better approximations of real cyber defence settings and, consequently, more realistic learned behaviours. Beyond the addition of the `misinform` action, agents may be given control of software-defined networks enabling them to isolate, honeypot, or migrate hosts as seen in [43], bolstering their capability.

Based on the findings in this work, a heterogeneous approach in which agents take on different roles may benefit the handling of complexity and leveraging cooperation through the division of responsibility. Future environments including network traffic simulation could employ agents responsible for host data and network data to collaborate to identify and mitigate threats. Beyond centralized learning used in QMIX, agents could learn to communicate with each other about their local observations, asking for help or notifying each other of important features and events. Notably, message-passing methods for MARL can leverage the scalability of independent, decentralized learning while also benefiting from coordination between agents [53].

Different architectures and algorithms should be evaluated to provide a more robust assessment of the applicability of MARL systems. Methods such as MAPPO [68] leverage the state-of-the-art performance seen in the PPO algorithm in a CTDE architecture. The differences between value-based methods and policy-optimization methods could be explored in the context of multi-agent cyber defence. Other approaches to cooperative MARL include using hierarchical role assignment [69] and model-based learning [70].

Advancements in MARL In this work, and in the game environments that inspired it, MARL architectures are relied upon to perform start-to-end feature processing and decision-making. This approach could be improved by outsourcing feature processing to a separate data mining module. For example, by using supervised learning in an anomaly detection role (e.g., [71]), notable features or samples from the environment could be extracted before being passed to RL agents for decision-making.

Implementation techniques are critical in the design of performant MARL architectures. Building on the success of QMIX, researchers have modified the mixing network with an attention mechanism [72], a duplex duelling network architecture [73], and a joint action weight [74], among others. Furthermore, transformers have been used to perform value function decomposition for cooperative MARL [75]. More advanced methods, such as transfer learning, can iteratively train RL models at progressively more difficult scenarios to reach higher levels of skill adoption [76]. League training has been used to select for specific characteristics of trained models, thus fine-tuning their behaviour [35]. Each of these approaches has theoretical and empirical qualities that could be evaluated in MARL cyber defence environments to optimize for learning ability.

6.3 Summary

CyMARL provides a training environment and experimental results of the use of cooperative MARL systems at host-based enterprise defence. Significant advancements to model performance and game realism could be observed by continuing to apply state-of-the-art advancements in MARL to this game setting. Moreover, greater environmental realism is possible through emulation, adversarial attack and advanced TTPs.

7 Conclusion

Tactical decision-making in cyber defence contends with an expansive problem space necessitating expert knowledge and, in many cases, decentralization of effort to provide essential coverage. The knowledge required is context-dependent, thus the correct identification of patterns is indispensable. Reinforcement learning has been demonstrated to perform pattern recognition and decision-making to an expert level. Applicable learning methods must be adaptable to many scenarios and capable of extracting policies from large, dynamic inputs. This work demonstrates how cooperative MARL can be used to learn decentralized behaviour policies for a variety of security games for host-based network defence from scratch. To evaluate the adaptability and ability to handle complexity of MARL methods, games were varied by attacker behaviour, network size, amount of sensor noise, grey agent presence, and agent area of responsibility. Fully-decentralized and CTDE approaches were compared across the range of tasks.

From this experimentation, meaningful insights were derived with regard to game design for multi-agent cyber defence tasks. Particularly, CTDE consistently outperformed the fully-decentralized IQL method, but by a smaller margin than is typical for collaborative and cooperative tasks. For this use case, the possibility for greater training scalability with IQL may outweigh its lower performance relative to QMIX. This work presented the initial evidence for cooperative MARL as a tool applicable to the generation of decentralized, tactical-level control policies in cyber defence. In the rapidly advancing field of RL game design, there are many directions of future work to continue to build this capability to handle greater realism, larger action spaces, and more sophisticated tactics.

References

[1] R. S. Gutzwiller, S. Fugate, B. D. Sawyer, and P. Hancock, "The human factors of cyber network defense," in *Proceedings of the human factors and ergonomics society annual meeting*, vol. 59, 2015, pp. 322–326.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[5] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[6] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[7] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[8] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018. [Online]. Available: `https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html`.

[9] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[11] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *International conference on machine learning*, PMLR, 2018, pp. 4295–4304.

[12] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks," *arXiv preprint arXiv:2006.07869*, 2020.

[13] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3, pp. 293–321, 1992.

[14] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[16] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International conference on machine learning*, PMLR, 2019, pp. 2052–2062.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016, pp. 108–118.

[20] T. Jaakkola, M. I. Jordan, and S. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," *Neural Computation*, vol. 6, pp. 1185–1201, 1994.

[21] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," *Advances in neural information processing systems*, vol. 29, 2016.

[22] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 aaai fall symposium series*, 2015.

[23] B. Bakker, "Reinforcement learning with long short-term memory," *Advances in neural information processing systems*, vol. 14, 2001.

[24] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*, Elsevier, 1994, pp. 157–163.

[25] L. Buşoniu, R. Babuška, and B. D. Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in multi-agent systems and applications-1*, pp. 183–221, 2010.

[26] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

[27] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative markov games: A survey regarding coordination problems," *The Knowledge Engineering Review*, vol. 27, no. 1, pp. 1–31, 2012.

[28] P. Sunehag, G. Lever, A. Gruslys, *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017.

[29] J. Hu, S. Jiang, S. A. Harding, H. Wu, and S.-w. Liao, "Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning," *arXiv e-prints*, arXiv–2102, 2021.

[30] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, and D. Marriott, "Cyborg: A gym for the development of autonomous cyber agents," *arXiv preprint arXiv:2108.09118*, 2021.

[31] X. Xu, Y. Sun, and Z. Huang, "Defending ddos attacks using hidden markov models and cooperative reinforcement learning," in *Pacific-Asia Workshop on Intelligence and Security Informatics*, Springer, 2007, pp. 196–207.

[32] Z. Utic and K. Ramachandran, "A survey of reinforcement learning in intrusion detection," in *2022 1st International Conference on AI in Cybersecurity (ICAIC)*, IEEE, 2022, pp. 1–8.

[33] A. Chowdhary, D. Huang, J. S. Mahendran, D. Romo, Y. Deng, and A. Sabur, "Autonomous security analysis and penetration testing," in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, IEEE, 2020, pp. 508–515.

[34] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2020, pp. 2–10.

[35] O. Vinyals, I. Babuschkin, W. M. Czarnecki, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[36] P. R. Wurman, S. Barrett, K. Kawamoto, *et al.*, "Outracing champion gran turismo drivers with deep reinforcement learning," *Nature*, vol. 602, no. 7896, pp. 223–228, 2022.

[37] L. Huang and Q. Zhu, "Adaptive strategic cyber defense for advanced persistent threats in critical infrastructure networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 2, pp. 52–56, 2019.

[38] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings 2002 IEEE Symposium on Security and Privacy*, IEEE, 2002, pp. 273–284.

[39] Z. Hu, M. Zhu, and P. Liu, "Adaptive cyber defense against multi-stage attacks using learning-based pomdp," *ACM Transactions on Privacy and Security (TOPS)*, vol. 24, no. 1, pp. 1–25, 2020.

[40] *NIST CVSS V3 Calculator*. [Online]. Available: `https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator`.

[41] A. K. Dutta, R. Negi, and S. K. Shukla, "Robust multivariate anomaly-based intrusion detection system for cyber-physical systems," in *International Symposium on Cyber Security Cryptography and Machine Learning*, Springer, 2021, pp. 86–93.

[42] Y. Han, B. I. Rubinstein, T. Abraham, *et al.*, "Reinforcement learning for autonomous defence in software-defined networking," in *International Conference on Decision and Game Theory for Security*, Springer, 2018, pp. 145–165.

[43] Y. Han, D. Hubczenko, P. Montague, *et al.*, "Adversarial reinforcement learning under partial observability in autonomous computer network defence," in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–8.

[44] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[45] V. Mnih, A. P. Badia, M. Mirza, *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1928–1937.

[46] A. Charpentier, N. Boulahia Cuppens, F. Cuppens, and R. Yaich, "Deep reinforcement learning-based defense strategy selection," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, 2022, pp. 1–11.

[47] A. Molina-Markham, C. Miniter, B. Powell, and A. Ridley, "Network environment design for autonomous cyberdefense," *arXiv preprint arXiv:2103.07583*, 2021.

[48] D. Horgan, J. Quan, D. Budden, *et al.*, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.

[49] *TTCP CAGE Challenge 2*, 2022. [Online]. Available: `https://github.com/cage-challenge/cage-challenge-2`.

[50] L. Li, R. Fayad, and A. Taylor, "Cygil: A cyber gym for training autonomous agents over emulated network systems," *arXiv preprint arXiv:2109.03331*, 2021.

[51] I. Kotenko, "Multi-agent modelling and simulation of cyber-attacks and cyber-defense for homeland security," in *2007 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, IEEE, 2007, pp. 614–619.

[52] M. Liu, L. Ma, C. Li, *et al.*, "Design and analysis of decentralized interactive cyber defense approach based on multi-agent coordination," in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, IEEE, 2020, pp. 659–664.

[53] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019.

[54] A. Servin and D. Kudenko, "Multi-agent reinforcement learning for intrusion detection: A case study and evaluation," in *German Conference on Multiagent System Technologies*, Springer, 2008, pp. 159–170.

[55] G. Shi and G. He, "Collaborative multi-agent reinforcement learning for intrusion detection," in *2021 7th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC)*, IEEE, 2021, pp. 245–249.

[56] K. Malialis and D. Kudenko, "Distributed response to network intrusions using multiagent reinforcement learning," *Engineering Applications of Artificial Intelligence*, vol. 41, pp. 270–284, 2015.

[57] C. HolmesParker, A. Agogino, and K. Tumer, "Clean rewards for improving multiagent coordination in the presence of exploration," in *Proceedings of the 2013 international conference on autonomous agents and multi-agent systems*, 2013, pp. 1113–1114.

[58] M. Samvelyan, T. Rashid, C. S. de Witt, *et al.*, "The StarCraft Multi-Agent Challenge," *CoRR*, vol. abs/1902.04043, 2019.

[59] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," *AAAI/IAAI*, vol. 1998, no. 746-752, p. 2, 1998.

[60] A. Tampuu, T. Matiisen, D. Kodelja, *et al.*, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, e0172395, 2017.

[61] M. Hessel, J. Modayil, H. Van Hasselt, *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[62] J. Zhang, Y. Pan, H. Yang, and Y. Fang, "Scalable deep multi-agent reinforcement learning via observation embedding and parameter noise," *IEEE Access*, vol. 7, pp. 54 615–54 622, 2019.

[63] B. Ellis, S. Moalla, M. Samvelyan, *et al.*, "Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning," *arXiv preprint arXiv:2212.07489*, 2022.

[64] R. Alford, D. Lawrence, and M. Kouremetis, "Caldera: A red-blue cyber operations automation platform," 2022.

[65] K. Hammar and R. Stadler, "Finding effective security strategies through reinforcement learning and self-play," in *2020 16th International Conference on Network and Service Management (CNSM)*, IEEE, 2020, pp. 1–9.

[66] B. Baker, I. Kanitscheider, T. Markov, *et al.*, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019.

[67] M. Standen, J. Kim, and C. Szabo, "Sok: Adversarial machine learning attacks and defences in multi-agent reinforcement learning," *arXiv preprint arXiv:2301.04299*, 2023.

[68] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," *arXiv preprint arXiv:2103.01955*, 2021.

[69] T. Wang, T. Gupta, A. Mahajan, B. Peng, S. Whiteson, and C. Zhang, "Rode: Learning roles to decompose multi-agent tasks," *arXiv preprint arXiv:2010.01523*, 2020.

[70] V. Egorov and A. Shpilman, "Scalable multi-agent model-based reinforcement learning," *arXiv preprint arXiv:2205.15023*, 2022.

[71] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[72] Y. Yang, J. Hao, B. Liao, *et al.*, "Qatten: A general framework for cooperative multiagent reinforcement learning," *arXiv preprint arXiv:2002.03939*, 2020.

[73] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "Qplex: Duplex dueling multi-agent q-learning," *arXiv preprint arXiv:2008.01062*, 2020.

[74] T. Rashid, G. Farquhar, B. Peng, and S. Whiteson, "Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning," *Advances in neural information processing systems*, vol. 33, pp. 10 199–10 210, 2020.

[75] M. J. Khan, S. H. Ahmed, and G. Sukthankar, "Transformer-based value function decomposition for cooperative multi-agent reinforcement learning in starcraft," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 18, 2022, pp. 113–119.

[76] W. Wang, T. Yang, Y. Liu, *et al.*, "From few to more: Large-scale dynamic multiagent curriculum learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 7293–7300.

Appendices

A CyMARL Simulated Environment Details

A.1 Scenario

The scenario in CybORG defines the network topology, which agents are playing the game, including the heuristic attacker and defender models, and which actions the agents can take. The variables in the scenario and their parameters are:

1) Agent
   a) Team (red, blue, or grey)
   b) Internal. *Whether the agent is controlled with RL or by a heuristic.*
   c) Heuristic model (as applicable)
   d) Accessible subnets
   e) Host vision. *What information it has about each host at the start of the game.*
   f) Starting sessions
      (i) Artifacts (privileges, events, etc.)
      (ii) Host connected to
      (iii) User
   g) Actions
   h) Reward calculator. *Six reward calculators are used in CyMARL, one for each scenario type for each team (red and blue).*
2) Host
   a) Availability value. *Indicates negative reward for attacker gaining root access.*
   b) Confidentiality value. *Indicates negative reward for attacker disrupting operations.*

c) Integrity value. *Indicates negative reward for attacker tampering with files.*

d) Connected interfaces. *Defines which subnets the host has access to.*

e) Image. *OS and system information. And configures the following simulation elements:*

(i) Files

(ii) Local sessions

(iii) Running processes

(iv) Services

(v) Users and groups

3) Subnet

a) Access Control List

b) Listing of hosts by subnet

The attacker objectives are varied by scenario type, listed in Table A.1. The number of hosts in each subnet is varied by network size, listed in Table A.2.

Table A.1: Attacker objectives in each scenario type.

Scenario Type	Goal	Scoring (per turn)
Confidentialtiy	Gain access to hosts	for each host with an active attacker session
Integrity	Tamper with filesystem	for each host with tampered files
Availability	Establish DoS processes	for each host with an active DoS process

Table A.2: Number of hosts per subnet in each network size scenario.

Subnet	Small	Medium	Large
User	3	5	5
Enterprise	0	0	3
Operational	2	5	5

A.2 Action Space

The actions that an agent can take in CyMARL are categorized into two types: concrete actions and abstract actions. Concrete actions typically simulate a single parameterized command that an agent may use to learn about or affect the network. A pingsweep, portscan, specific exploit, or privilege escalation

are examples of concrete actions. These actions assume that basic vulnerabilities can be found and exploited with a similar level of knowledge as an attacker using basic automated tools. Likewise, a defender's concrete actions may require the level of detail necessary for a user of cyber defence software to find and isolate threats. A host may be vulnerable to a variety of exploits, some of which may be tied to a running process or service. To use a concrete exploit action the attacker identifies the target service or feature of the host and its corresponding vulnerability.

Abstract actions are not necessarily linked directly to command line parameters but instead constitute a logical task. For example, an attacker's `ExploitRemoteService` action simply automates the process of trying an exploit to and establishing a reverse TCP session if it is successful. A defender's `remove` action will kill a target process using the concrete action `stop_process`. If the process was benign, it is restarted.

The abstract actions for the baseline scenario are:

1) Attacker actions:
   a) DiscoverRemoteSystems: run a pingsweep on target subnet to gather IP addresses.
   b) DiscoverNetworkServices: run a portscan on target host.
   c) ExploitRemoteService: select and attempt an exploit based on open ports on target host, and a set preference for using certain exploits, quantified by the *exploit weight* value. If a host has multiple open ports with corresponding exploits, the attacker will select one based on its weighted probability. Available exploits are chose:
      i. BlueKeep: if port 3389 is open. Weight: 1.
      ii. EternalBlue: if port 139 is open. Weight: 2.
      iii. FTPDirectoryTraversal: if port 21 is open. Weight: 7.
      iv. HarakaRCE: if port 25 is open. Weight: 6.
      v. HTTPRFI: if port 80 is open. Weight: 3.
      vi. HTTPSRFI: if port 443 is open. Weight: 4.
      vii. SQLInjection: if port 3390 and one of port 80 or port 443 are open. Weight: 5.
      viii. SSHBruteForce: if port 22 is open. Weight: 0.1.
   d) PrivilegeEscalate: Create an administrator session for target OS on target host. The session provides visibility of all host information and it cannot be killed with the defender `Remove` action.
2) Defender actions:

a) Analyse: Get file info on target host. Concrete actions `DensityScout` will save a probability of malware of each file, `SigCheck` saves a flag if a signature is present for each file.
b) Data Repair: Removes suspected malicious files on a host.
c) Misinform: Deploy a decoy process on target host.
d) Monitor: Detect new events and sessions on each host.
e) Remove: Kill processes detected with `monitor` on target host.
f) Restore: Restart target host, removes files and processes.

A.3 Observation Space

The defender's observation space constitutes a list of agent observations since each agent observes its area of responsibility separately. Observations are generated from simulated host information and are only provided to agents performing the `monitor` action. Observations are encoded into a floating point vector before being outputted by the simulated environment observations contain the following information for each host:

1) System information
2) Network Interface information
3) Processes
4) Files
5) Users
6) New network connection flag
7) New process creation flag

A.4 CyMARL implementation tasks

CyMARL extends the CybORG simulated environment to allow for multi-agent training and evaluation using PyMARL. The following tasks were conducted to achieve this implementation:

1) Extend CybORG, scenario, environment controller, and wrapper classes to accept multiple RL agents.
2) Make a CybORG multi-agent gym environment based on SMAC for interface with PyMARL.
3) Add seventeen scenarios and five actions to support experiments.
4) Extend the heuristic attacker and defender models to three behaviour types with additional actions and unique reward calculators.

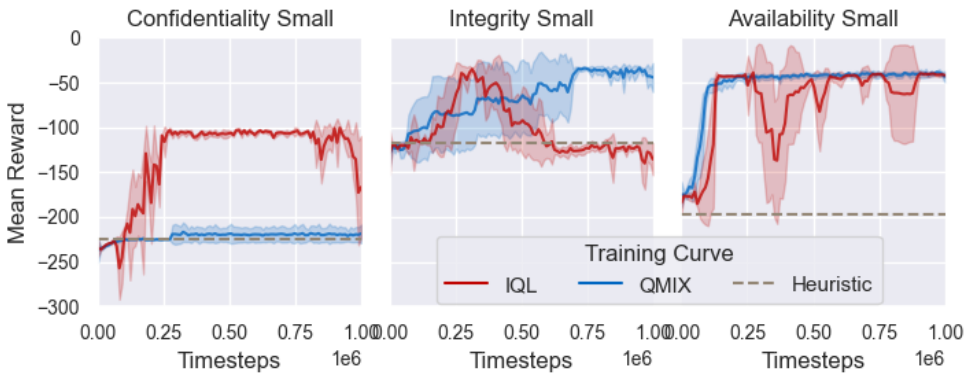
B Supplementary Results

The best-performing hyperparameter values in each phase 1 scenario are presented in Table B.1.

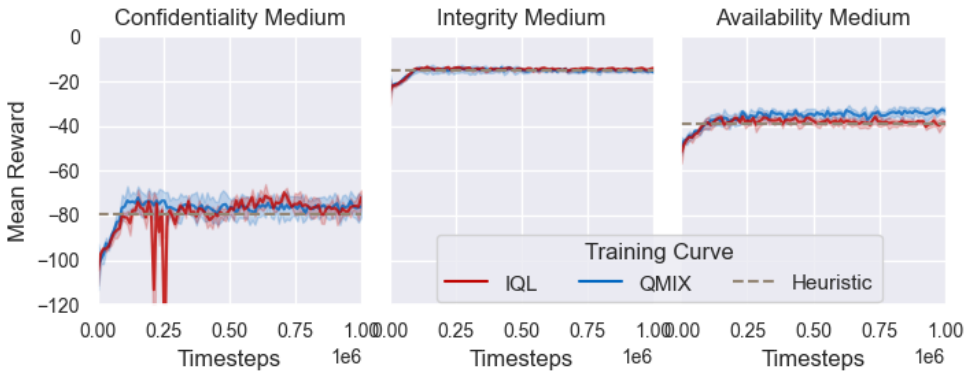
Table B.1: Selected hyperparameter values from Phase 1 grid search by scenario.

Scenario		IQL			QMIX		
		Batch size	Buffer size	Learning Rate	Batch Size	Buffer Size	Learning Rate
Confidentiality	Small	256	20000	0.001	64	20000	0.002
	Medium	64	5000	0.001	128	10000	0.002
	Large	128	5000	0.002	128	10000	0.001
Integrity	Small	128	20000	0.002	128	10000	0.001
	Medium	64	10000	0.002	256	20000	0.002
	Large	256	10000	0.001	256	5000	0.001
Availability	Small	64	10000/20000	0.001/0.002	64	20000	0.002
	Medium	64	5000	0.002	64	5000	0.001
	Large	128	10000	0.001	64	5000	0.001

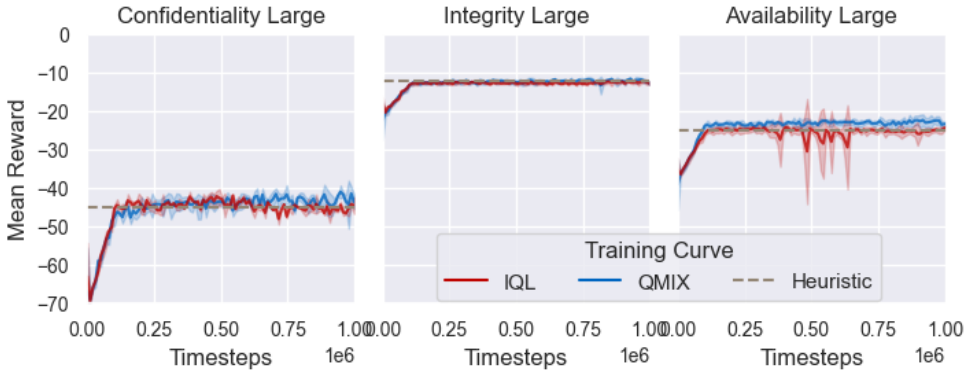
The training curves presented in Figure 5.1 are shown in Figure B.1 separated by network size and rescaled along the y-axis.



(a) Training curves in small scenarios.



(b) Training curves in medium scenarios.



(c) Training curves in large scenarios.

Figure B.1: Learning curve of IQL and QMIX in each Phase 1 scenario separated by network size. The shaded area represents the standard deviation.