Autonomous Network Defence using Reinforcement Learning

An Investigation of Robustness using Enhanced Adversarial Strategies

Candidate Number: QTHG1[1]

MSc Information Security

Supervisors: Vasilios Mavroudis, Chris Hicks, and Steven Murdoch

Submission Date: 12/09/2022

[1]**Disclaimer:** This report is submitted as part requirement for the MSc Degree in Information Security at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Network defence becomes more challenging because of the dramatic increase in network size and the growing shortage of cyber security professionals. In contrast, current adversaries (e.g., Advanced Persistent Threat attackers) keep evolving their capabilities through information gathering and the development of defence evasion tools. A promising research direction to solve such problems is Artificial Intelligence for Security. Because of the success of Reinforcement Learning in games, it is possible to extend its applications to network security. The current works mainly focus on automated penetrating testing and intelligent intrusion detection systems. One of the limitations of these works is that their environments lack competitive interactions between attackers and defenders. Therefore, we investigate the capabilities of an autonomous network defender who can actively select actions to mitigate the impact of adversaries in an interactive cyber environment named CybORG. The main contributions of this work are extending the performance of prior adversarial strategies, training novel autonomous agents, and evaluating performance. Our results indicate that the hierarchical Reinforcement Learning method can successfully defend against multiple types of adversaries over varying lengths of time with high performance and robustness.

Keywords: Network Security; Advanced Persistent Threat; Reinforcement Learning; Robustness

Acknowledgements

I would like to sincerely thank my great supervisors, Dr. Vasilios Mavroudis, Dr. Chris Hicks, and Prof. Steven Murdoch, for their full range of guidance and support in researching, coding, and writing during my dissertation project. Every time I had a new idea or confusion, they always encouraged me and provided helpful advice. Besides, the Department of Computer Science at UCL provides me with systematical learning resources for Information Security, which has become the knowledge basis of my dissertation. I am also grateful for all staff in this department who provide high-quality teaching and technical support in high performance computing for this dissertation. Finally, I express my gratitude to my family and friends, who are always there for me.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Informatisation techniques have altered how enterprises, organisations, and governments operate and improved the efficiency of communication and data processing. However, more opportunities are also provided to adversaries who aim to collect sensitive information from victims for financial gains or political missions. Advanced Persistent Threats (APT) are taken seriously by many organisations and nations due to their complexity and insidious features. This kind of adversary is usually state-sponsored and has advanced techniques and enough patience to hide in the target network. Their common goals are to steal valuable data and impact essential services when necessary. Since APT adversaries prefer to perform slow and covert movements over a relatively long period, it is difficult for victims to find evidence even after the attackers have accomplished their goals. Therefore, continuous and active network defence should become an indispensable part in operations and managements to mitigate the impact caused by the APT attackers.

Network defence against APT has become a significant area in cyber security that attracts global interest from industry and academia [10]. Defence is a more complex task than attack since defence requires correct combinations of various defending tools and comprehensive knowledge in cyber security. It must also remove the threat, find tracks or evidence, and fix related vulnerabilities. This fact indicates that network defence usually needs enough professional employees. However, the reality is the opposite since many countries have an extreme shortage of cyber security experts. Moreover, this human operation has high operational costs and a long response time compared with an autonomous system. The former allows only large companies and essential government departments to adopt a comprehensive defence strategy, while the latter makes the applied defence strategies less reliable than expected. The situation worsens when information technology becomes more popular as more devices are added to networks, which will increase network traffic exponentially. This phenomenon leads to even experts struggling to pre-empt attack strategies in large-scale networks. Therefore, autonomous network defence should be developed to gather cyber threat intelligence and defend against attackers actively.

One cutting-edge approach to implementing autonomous network defence is to apply Artificial Intelligence (AI). Supervised Learning has already been used in malware detection [3], and Natural Language Processing (NLP) can extract valuable information from extensive daily logs and alerts, which can greatly reduce response times. As a significant direction in the AI area, Reinforcement Learning (RL) has been shown to outperform humans in games. Therefore, it is a promising direction to investigate the capabilities of RL algorithms in network attack and defence. Recent research mainly focuses on penetrating testing [39] [46] and intrusion detection systems [50] [11] [24]. One limitation of these works is that the tested environments lack competitive interactions between adversaries and attackers. Although their efficiency and accuracy are optimised compared with previous works, there is still a gap with the real world. In contrast, the CybORG environment [2] where attackers and defenders can interact with each other is used in this research. In the CybORG environment, if one action that APT attackers perform has harmful impacts, the network defender will receive a punishment (i.e., negative performance). Thus, if the trained defender wants higher rewards (i.e., positive performance), it must select the correct combination of actions to mitigate the attack.

APT attackers are usually sponsored by different organisations or governments, and their targets also vary. Their capabilities might also differ since they could be based on knowledge about the victim's network environment and the chosen advanced tools that can bypass the traditional detection methods. This fact indicates that APT adversaries prefer to develop their bespoke strategies to compromise hosts in a target network. These different strategies can make it difficult to model a particular attacker for a network defender that needs to be trained by the RL methods. Furthermore, APT attackers will keep trying to attack more devices on the network as long as they exist. This behaviour helps the attackers gather sensitive data for further attacks and to achieve more advanced goals. A recent report shows that the average time before an APT attacker can be detected in 2020 is about 24 days [18]. Since the number of infected machines is usually proportional to the attack time, adversaries may have already obtained most sensitive data they need before detection. The autonomous network defender must therefore adapt to APT adversaries with different strategies over varying lengths of time.

1.2 Achievements and Thesis Structure

This research aims to investigate the robustness of RL-enhanced agents when defending against multiple APT adversaries over different lengths of time. One achievement of this research is three extended adversarial strategies that APT attackers can apply in the CAGE challenge [25]:

- The first strategy is based on a newly designed action named *DefenceEvade*. This action is inspired by the ATT&CK Matrix [43] and aims to help adversaries hide their tracks during their attacking processes.
- The second strategy is adding randomness when discovering the possible attacking path, which tries to mislead the network defender.
- The third strategy explores the potentiality that adversaries can use the RL algorithm to find a proper approach to execute attacks and recover from the defender's interrupts.

This research also evaluates the performance and robustness of the network defender using the algorithm developed in the state-of-the-art [13]. We train the updated network defender which aims to defend against attackers using different adversarial strategies. We also investigate what will happen in its learning process and why it holds advantages compared with the unmodified network defender. According to our experiment results, the newly trained defender can successfully mitigate the influences of different APT adversaries at a satisfactory level. Furthermore, there are some extra technical contributions to the CAGE challenge code repository.

The rest of this dissertation is organised as follows. Chapter 2 introduces several essential concepts about RL and APT. It also discusses related works in automated penetrating testing, intelligent intrusion systems, and applications of hierarchical RL in network defence. A brief outline of the challenge environment is introduced in Chapter 3 for the experimental deployment. Chapter 4 is about the state-of-the-art autonomous defensive algorithm, which includes the Proximal Policy Optimisation algorithm [38] for deep RL, the extended curiosity module [30], and the hierarchical architecture. Chapter 5 focuses on the performance and robustness of the algorithm in Chapter 4. In this chapter, three kinds of novel extended adversarial strategies are designed, the corresponding autonomous adversaries and defenders enhanced by RL are trained, and evaluations of their performances are also presented and compared. Finally, this dissertation is concluded in Chapter 6 with contributions and limitations that could be improved for future works.

Chapter 2

Literature Review

2.1 Reinforcement Learning

Reinforcement Learning, Supervised Learning, and Unsupervised Learning are three main categories of Machine Learning. Different from two other paradigms, RL has no supervisor but is based on the delayed feedback called *Reward*. Its main idea combines concepts from computer science and behavioural psychology, and tries to learn which behaviour to do next would lead to a maximal cumulative reward. According to this definition, *trial-and-error search* and *delayed reward* are two significant characteristics of the RL approach [44].

As shown in Figure 2.1, the RL process can be described as interactions between *Agent* and *Environment*. At each time step t, the agent will receive an observation O_t and a scalar reward R_t from the environment, and then execute the action A_t. After the environment receives the action A_t, it will emit the observation O_{t+1} and reward R_{t+1} for the next time step.

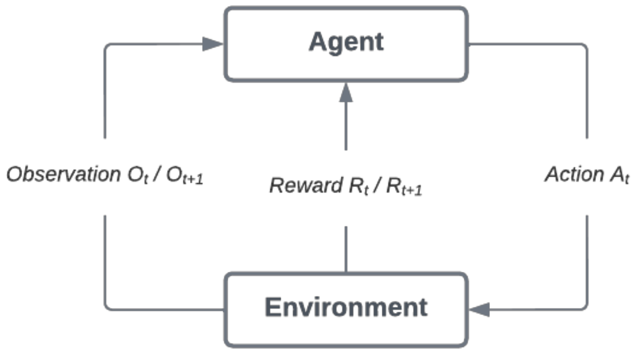

Figure 2.1: Agent and Environment

To represent the RL problem, the environment is formally described as a Markov Decision Process (MDP). MDP is modified from a Markov Process with rewards and decisions, and can be defined as a five-element tuple:

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle \tag{2.1}$$

where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is a state transition probability matrix with $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$, $\mathcal{R}$ is the reward with $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$, and $\gamma \in [0, 1]$ is the discount factor which shows uncertainty about the future [40].

Another important concept of RL is *Policy* π which fully defines the behaviour of the agent. To estimate the how good some state s it is under the policy π, the state-value function $v_\pi(s)$ and action-value function $q_\pi(s, a)$ are defined as follow:

$$v_\pi(s) = \mathbb{E}_\pi[\, G_t \mid S_t = s \,] = \mathbb{E}_\pi[\, R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \,] \tag{2.2}$$

$$q_\pi(s, a) = \mathbb{E}_\pi[\, G_t \mid S_t = s, A_t = a \,] = \mathbb{E}_\pi[\, R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \,] \tag{2.3}$$

Therefore, RL methods aim to find the *Optimal Policy* π_* which can achieve the optimal (maximal) state-value function $v_*(s)$ and action-value function $q_*(s, a)$:

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.4}$$

$$q_*(s, a) = \max_\pi q_\pi(s, a) \tag{2.5}$$

Then, the agent following the optimal policy π_* will maximise the long-term reward, and finally solve the RL problem.

The success of RL methods has been established in games where it has achieved performance far beyond human capabilities. Its applications range from perfect information games (e.g., Atari arcade games [27] and Go [42]) to imperfect information games (e.g., StarCraft II [48] and Stratego [31]). To explore more adaptability of RL solutions, network attack and defence, which can be regarded as a two-player game with imperfect information, is a research direction with high potentiality.

2.1.1 Hierarchical Reinforcement Learning

Because the network environments are usually large-scale and hacker approaches have become increasingly sophisticated, traditional RL approaches might not be sufficiently effective to explore the action space. This problem is sometimes called "the curse of dimensionality". Therefore, hierarchical RL is currently a new research direction for a realistic defence approach to handle the high dimensionality problem in network security.

Hierarchical RL aims to divide and conquer a large-scale problem into a hierarchy of subtasks [29]. Each subtask can be solved by a single-agent (subagent) RL approach, and the controller of this hierarchical architecture will select actions from the optimal subagent. One interesting algorithm in this area is named H-DQN [22]. It uses the temporal abstraction and the intrinsic motivation to solve the insufficient exploration problem. Its hierarchical architecture contains two layers. The upper level controller selects a goal based on the upper level policy over a longer time span, while the lower level controller selects an action over a shorter time span. Both the upper and lower level controller use the Deep Q-Network (DQN) algorithm, where the upper level policy is updated based on external rewards from the environment and the lower level policy is updated based on internal rewards from the upper level controller. This newly designed hierarchical RL algorithm has been proven to have better performance than standard algorithms in long-horizon problems [29], which might be a breakthrough for autonomous network defence against APT adversaries.

There are many other promising algorithms in hierarchical RL for network defence. For example, Tran et al. [46] introduced HA-DRL, a hierarchical architecture with multiple subagents, to establish a real-like penetration testing process. HA-DRL uses an action decomposition scheme to solve the unstable problem of DQN with a large action space. This scheme is similar to a tree with multi-layers such that the number of subagents is also acceptable where $logN$ subagents are sufficient for N actions. Moreover, each subagent with a neural network does not need all state features to find the optimal policy, and experiments show that HA-DRL also has less training time and better convergence than a single DQN agent.

2.2 Advanced Persistent Threats

The concept of APT was first introduced by a patent in 2008 [35]. As the name describes, APT attacks are much stronger than traditional attacks. The APT actor will not be a single person but groups who are highly organised and usually have sufficient resources to execute attacks [10]. For example, they work for government agents or military institutions, and could attack education, energy, transport, health, and other critical infrastructures in other countries. Moreover, behaviours of APT attacks aim to run in the long term, which means that they keep trying to attack the specific targets and find ways to hide their track in the target network. The global median dwell-time of APT attacks is 99 days in 2016 and still remains 24 days in 2020 [18]. Additionally, the threat aspect of APT attacks is usually based on 0-day or N-day exploits. For example, DeputyDog uses CVE-2014-0322 [5] to attack US military intelligence, and WannaCry ransomware uses CVE-2017-0144 (EternalBlue) [6] which was found by Equation Group. Although activities of APT groups might be paused due to public disclosures and patch fixes, they can update their tools to keep ahead of network defenders, and continue to execute their long-term strategies [19].

2.2.1 Processes of APT Attacks

The process of a typical APT attack can be described by continuous steps [10], which is similar to Intrusion Kill Chain (IKC) [15] but has more phases in its lifecycle:

(a) **Reconnaissance.** This is one of two preparation steps for information gathering. In this step, attackers will investigate situations of the target and collect as much information as possible. The information usually includes the network topology, software versions, and personal details of staff or employees. The mainstream technique here is *Social Engineering* (SE) [21]. SE does not target systems but humans who can access sensitive information. Therefore, technical mitigation approaches are usually ineffective against this kind of attacks. Furthermore, humans are usually regarded as the weakest point in information security due to their lack of awareness and poor performance on detecting deception [21].

(b) **Weaponisation.** After reconnaissance, attackers analyse the gathered information (e.g., attacking path, software vulnerabilities, and weak password) to design the attacking plan and tools. To make sure that the APT attack will be successful, weaponisation usually requires multiple attack vectors and should find ways to hide tracks for future actions.

(c) **Delivery.** In this stage, attackers transfer the attack vectors to the target environment directly or indirectly. The direct method also uses social engineering tools like phishing. Phishing, especially spear phishing, uses the gathered personal information during reconnaissance to increase the probability of success. After inducing people to click a link or download an attachment in the phishing email, further exploitation will start through the potential malware. The indirect method is to use a watering hole attack. This kind of attacks targets websites or services that victims frequently visit or use. Because these websites and services are usually trusted, watering hole attacks are more deceptive and more efficient.

(d) **Initial Access.** The simplest way is to use credentials collected during reconnaissance, which is challenging to detect since this behaviour is regarded as legitimate access. The standard method for access usually includes *Remote Exploitation* and *Privilege Escalation*. The attack vectors might be constructed by unauthorised credentials, software vulnerabilities (including 0-day exploits and N-day exploits without patches), misconfigurations, and malware. After privilege escalation, attackers can execute more severe malicious activities. For example, they can deepen their access to other connected systems and install additional backdoor programs. Therefore, this step is the key for the following attack phases.

(e) **Command and Control.** Because most network perimeter defences are effective, it is hard for attackers to contact the target host directly from the external network. However, internal connections often do not have this strict restriction. In this phase, Command and Control (C2) channels are built between APT actors and exploited hosts, which are aimed to enable adversaries to control the following attacking steps in the target environment.

(f) **Lateral Movement.** Because significant targets are heavily guarded and difficult to attack directly, it is reasonable that the first compromised host might not be a valuable target. Adversaries prefer to use the first host as a springboard to gain access to other hosts in the target network. They can perform internal reconnaissance, send the gathered information for weaponisation via the C2 channel, deliver attack payloads, and then have access to the next host. APT attacks execute this process repeatedly until adversaries reach the highly valuable target.

(g) **Actions.** This is the final step of APT attacks where adversaries take action to complete their tasks. One typical goal is data exfiltration using the hidden C2 channel, which will be a high risk to sensitive information (e.g., intellectual property and military intelligence). Another typical aim is to impact the services of the infiltrated target. For financial institutions, attackers can forge or modify transaction data to steal funds. Considering infrastructures like energy and transportation, APT attacks can take control of critical devices. This kind of impact can also reduce the prestige of organisations or governments. Moreover, APT attackers can use exploited hosts as zombies for Distributed Denial-of-Service (DDoS) attacks. This leads to huge numbers of simultaneous requests to the target server such that legitimate users will be prevented from using services.

2.2.2 MITRE ATT&CK Matrix

The ATT&CK Matrix [43] is established by MITRE as a globally-accessible knowledge framework that records the known strategies and techniques of observed adversaries. This matrix defines a general taxonomy for network offences and defences, and is a useful theoretical tool for situational awareness and penetration testing. Because of the threat types and the range of sources for its research basis, ATT&CK is more suitable for network attack and defence scenarios with network boundaries. It can also provide a mapping relationship between the knowledge base and APT attacks in the real world.

The current ATT&CK Matrix for Enterprise [26] covers 14 different tactics, 222 techniques, and several sub-techniques. The tactics include Reconnaissance, Resource Development, Initial Access, Execution, Persistence, Privilege Escalation, Defence Evasion, Credential Access, Discovery, Lateral Movement, Collection, Command and Control, Exfiltration, and Impact [26]. Inside each tactic, there are a series of techniques and sub-techniques that lead to different outcomes. The framework includes technique description, platform, minimal permission required, and mitigation approaches.

There are three necessary tactics for a successful APT attack: Initial Access, Persistence, and Defence Evasion. Currently, most APT research [10] and reports [20] [16] [17] focus on analysis of the delivered payloads and malicious tools. A possible reason is that these are easier to collect and capture. However, they might not be enough to restore the entire lifecycle, and these attack tools are often updated and changed frequently. Therefore, the ATT&CK Matrix needs to be combined with other threat models. One is the Diamond Model of intrusion analysis [9]. This model establishes four basic subjects in each intrusion: adversary, infrastructure, capability, and victim. The main reason to use this model is that APT attacks in the real world usually have specific targets, resources, and capabilities. Identifying this information will help defenders know the current attack phase and intention and then complete the attribution analysis.

2.3 Autonomous Network Defence

In recent years, network defence has been an essential mechanism for large enterprises, organisations, and governments. Network defence can be divided into three categories: prevention, detection, and reaction. In terms of prevention, penetration testing [49] describes authorised attacks that aim to evaluate the defence system by simulating attacks that malicious attackers will likely try. The white-hacker team will report problems that happen in the penetration testing process to prevent potential threats in the real working environment. The detection part of network defence is usually based on intrusion detection systems [7], which are designed to monitor the events occurring in the computer networks and systems. Any intrusion behaviours are reported to administrators or collected by the security information and event management (SIEM) system. However, the current problem of network defence is that manual prevention and detection can be inefficient and ineffective. One reason is that the number and sophistication of network offences are continually growing. Another reason is the shortage of skilled workers since information security is complex and difficult to establish systematically [14]. Thus, autonomous network defence has become a new direction in academic and industrial areas.

2.3.1 Automated Penetration Testing

The modeling approaches to penetration testing are determined by how much ethical hackers know about the target environment. If all network and system configurations in the environment are available to attackers, it is easy to build an *attack graph* [4] for the target and then find an optimal *attack path* by traditional planning algorithms. However, this modeling approach is often not practical since attackers usually cannot have the complete knowledge of the target environment. Another more abstract approach is to use an MDP model, which focuses on states and transition probabilities. Unfortunately, this approach might still be ineffective in real-world applications because it requires essential prior knowledge about the probability of successful attacks, which is hard to measure in a complicated and varied environment.

A more realistic approach is using a Partially Observable Markov Decision Process (POMDP) [33] [34] to model penetration testing. This method does not need complete knowledge of configurations and prior knowledge about probabilities of successful attacks. Its main advantage is the capability to reason about the knowledge gained from the scanning actions during the penetration test. Experiments in [33] show that this method is able to find attacks on a single machine but cannot scale due to computational problems when the state space grows.

Recent research [39] [14] [46] prefers to use RL algorithms to solve the automated penetration testing as an MDP problem with complete knowledge or sufficient prior knowledge. This is a trade-off between the modeling accuracy and the solution computability. Although the benefit of RL methods is that it has more robust capabilities to solve large-scale problems, it is still a problem to apply these algorithms in complex network scenarios. A major obstacle is the complexity of action space facing different situations. The experiments in [39] demonstrate that the size of the action space grows exponentially even as the number of hosts increases in a relatively small range.

2.3.2 Intelligent Intrusion Detection Systems

Intrusion detection systems are used to monitor and audit the behaviours in networks or systems. According to the audited data type, intrusion detection systems can be classified into host-based and network-based [7]. Host-based intrusion detection systems compare and analyse significant system files or configurations to detect any suspicious activities, while network-based intrusion detection systems monitor network traffic and analyse traffic data.

For intelligent intrusion detection systems enhanced by RL, the system call sequence data [50] can be used in an MDP to build a state-value prediction model, which can predict anomaly probabilities. Network log files [11] can also be a breakthrough since finding valuable log files with malicious behaviors is challenging. One solution is to give rewards when the log file actually contains any attack signatures such that the system will automatically select more valuable log files for analysis and tracing. Lopez-Martin et al. [24] tested four algorithm variants with two labeled intrusion datasets, and found that RL solutions, especially Double Deep Q-Network (DDQN) [47], can highly reduce the prediction time. Such that RL methods can be simple and efficient approaches in current environments where rapid responses are necessary. However, this research [24] replaced the interactive environment with a sampling function of intrusions, which is less realistic than other approaches in the literature.

Chapter 3

The CAGE Challenge in CybORG Environment

3.1 Scenario and Environment

The CybORG environment [2] is an experimental platform designed to support Autonomous Cyber Operations (ACO) research in Artificial Intelligence (AI) for Security. Its focus is not on data-driven intrusion detection approaches but on finding an optimal strategy to defend adversaries inside the network at a high level. It provides not only abstract actions in its simulator but also actual interactions with virtual machines built on Amazon Web Services (AWS), which increases its scalability and reliability.

The Cyber Autonomy Gym for Experimentation (CAGE) challenge [1] is embedded in the CybORG environment, which is under the scenario that one nation called Guilder keeps trying to perform APT attacks at military factories of another nation called Florin. The challenge aims to design an autonomous defence agent that can prevent significant operational servers from being compromised and keep the entire system runnable.

Configurations of the scenario are defined in the corresponding YAML file `Scenario1b.yaml` and can be modified for customised requirements. Most important information about hosts and servers is also specified in the scenario file, which includes operating system information (Windows or Linux), interfaces, users, groups, files, processes, sessions, and services. All these data types are instanced when the hosts are created.

The network topology under this predefined scenario will not be changed during this challenge. As illustrated in Figure 3.1, there are three subnets and thirteen hosts inside the network environment. Subnet 1 contains five user hosts. These hosts can be accessed through external networks and are not as critical as the other hosts inside the network. Subnet 2 has three enterprise servers used to support user hosts' activities in Subnet 1. Inside Subnet 3, there are three operational hosts and one operational server which is the target of adversaries. The Network Access Control List (NACL) allows all traffic in and out of Subnet 1 for users and Subnet 2 for enterprises. However, it denies the traffic from Subnet 1 to Subnet 3, which means all five hosts in the User subnet cannot directly access servers or hosts in the Operational subnet.

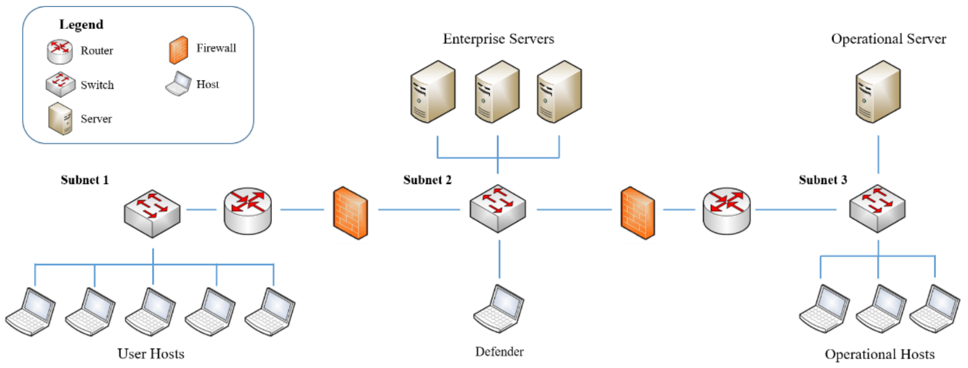

Figure 3.1: The CAGE Challenge Network Topology [25]

3.2 Agents and Actions

The CAGE challenge has three kinds of agents: blue, green, and red. Every agent will select and perform an action for each step in one episode. The execution order is that the blue agent receives all available observations from the last step and uses the chosen defensive action. Then, the green agent executes the benign behaviours, and finally the red agent performs the malicious activity [25]. For each action, there will be a reward value usually based on the significance of its influence on hosts and the entire system.

3.2.1 Red Agents as Adversaries

There are two red agents based on different attacking rules. The first one is named *B_lineAgent*, a powerful adversary who already knows the complete network topology information and the shortest attacking path to the operational server. The other red agent is *MeanderAgent*, which will execute a breadth first search on all known hosts until it reaches the final target. This adversary needs to discover helpful information during exploration, which is slower and less stealthy but more general than *B_lineAgent*. Both red agents share the same action space, which contains five high-level actions, one extra *Sleep* action, and several concrete sub-actions that will be used to support the corresponding high-level action.

The first high-level action is *DiscoverRemoteSystems*. With a subnet as input, the challenge simulator will execute a *PingSweep* sub-action and return all active IP addresses on this subnet. The second action is *DiscoverNetworkServices*. The related underline sub-action is *Portscan* to identify all available services by their port numbers on a target IP address. Both discovery actions can be classified into Reconnaissance and Lateral Movement in the APT lifecycle.

In the next stage, attack vectors will be designed according to the discovered services and then delivered and deployed later. The *ExploitRemoteService* action is provided with five low-level attacking options. In order of priority from highest to lowest, they are *HTTPSRFI*, *HTTPRFI*, *EternalBlue*, *BlueKeep*, and *SSHBruteForce*. This rate is hard-coded in the simulator, and there is a 75% chance to select the highest available option while a 25% chance to select options randomly. The next action is *PrivilegeEscalate*. This action is only allowed when the target host or server has already been exploited successfully. According to the operating system type, the Juicy Potato exploit will be used for Windows, while the V4L2 Kernel Exploit will be selected for Linux. After privilege escalation, adversaries are considered to have gained administrator access. Meanwhile, blue agents will receive continuously negative rewards (-0.1 for the user and operational hosts, and -1 for the enterprise and operational servers) in each time step until the compromised host or server is restored to its original state. Finally, if red agents have root privilege on the operational server, they will continually execute *Impact* actions that could interrupt services. Since red agents have completed their tasks, blue agents will get a -10 reward whenever a *Impact* action is executed.

3.2.2 Green Agents as Normal Users

The green agent is designed to perform benign scanning actions. One scanning action is *GreenPingSweep*, which discovers other hosts on the same subnet. Another action is *GreenPortScan* which scans all open services on the target host. Except for these two actions, the green agent will not exploit any host, which is the main difference from the red agent. These actions and the corresponding observations can be determined as environmental noise such that the blue agent will not believe that all actions come from the red agent. However, the green agent only performs the *Sleep* action in the CAGE challenge scenario.

3.2.3 Blue Agents as Defenders

The blue agent acts as the defender in the network. There are five abstract actions, one extra *Sleep* action, and several concrete actions for the blue agent to select. The first available action is *Monitor*, where the blue agent will collect all possible malicious activities based on suspicious network connections and processes. Moreover, *Monitor* is a passive action which means it will always be executed when other actions (including the *Sleep* action) are selected in the current step. The second action is *Analyse*, where the blue agent will use tools like DensityScout and SigCheck to detect whether malware exists on the given host. Therefore, these two actions enable intrusion detection at the network and host levels.

There are two other abstract actions that the blue agent will use when dealing with the compromised hosts. One action is named *Remove*. The blue agent can remove the detected suspicious processes and files from the target host. In the simulator, this action will remove the user-level shell by killing it as a process, but it cannot remove privileged shells since these shells are considered to exist permanently [25]. To remove a privileged shell, another action named *Restore* should be used. Through this action, the blue agent will recover the target host from a benign state which is previously backed up. However, there exists punishment for this more powerful action since it will cause network and service disruptions, which undermines one of the defensive goals. Therefore, the blue agent will receive a -1 reward for executing the restore action each time.

In the extension of the CAGE challenge, the blue agent has an advanced action named *Misinform*, which can deploy decoy services on the target host. Its working principle is similar to a virtual honeypot [32] such that green agents who are as normal users will not access the decoy services. Any other access will be regarded as malicious activities created by red agents as adversaries. If the red agent exploits the decoy service, its following actions will not be successful unless another non-decoy service is chosen to exploit. The *Misinform* action delays the time adversaries will take to find valuable hosts, thus improving the chances of attacks being detected and handled.

3.3 States and Observations

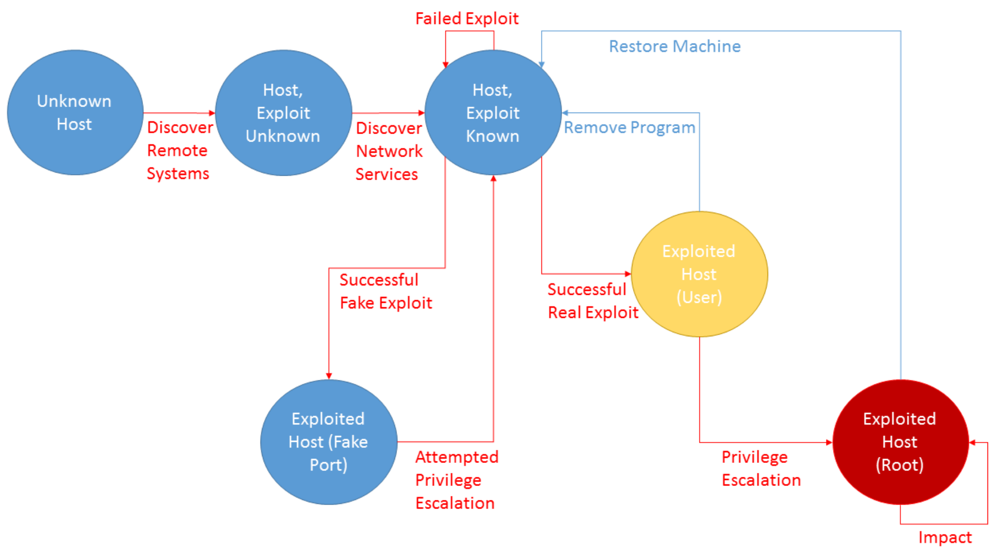

Figure 3.2: Host States [25]

The entire state transition process is illustrated in Figure 3.2. Each circle represents a specific state of the target host. Circles in blue show that the corresponding host has not been exploited. The yellow circle indicates that the host has been exploited and adversaries already have user access successfully. The red one denotes the privilege escalation is finished, and the administrator access has been achieved by attackers. Red and blue arrows are actions that can be executed by red and blue agents, as explained in Section 3.2.

To check whether the state transition is successful, observations as results should be available for both red and blue agents. For the initial state, blue agents can access the state information of the entire network, and red agents can see the information about the start point `User0` since it has already been privileged during initialisation. Additionally, blue agents will never recover this host, which aims to keep the attacking process persistent and indicate the seriousness when adversaries already have phished credentials in the environment [25].

For the same action, red and blue agents usually have different observations. Considering a successful *DiscoverRemoteSystems* action, red agents will receive a list of IP addresses in the target subnet. Although blue agents keep monitoring the entire network, no tools are given to detect this action, such that their received observation is unknown. Situations will change when performing *DiscoverNetworkServices* where both red and blue agents can observe the connections between the sender and receiver, their IP addresses, and scanned port numbers. For the *ExploitRemoteService* action, connections are still in both-side observations, but red agents will get the system and user information of the attacked host. Moreover, there will be a 5% chance that this malicious action cannot be observed by blue agents. After the *PrivilegeEscalate* action, red agents can get further information about the target host and its connected systems, while blue agents cannot detect this activity. Fortunately, blue agents can use the *Analyse* action to find suspicious files in the detected host. After analysing, blue agents can use the *Remove* or *Restore* action to defend against attacks.

Chapter 4

Autonomous Network Defence using Reinforcement Learning

4.1 Proximal Policy Optimisation

Previous research in network defence areas [39] [14] [46] preferred to use the DQN algorithm as the core RL method. The DQN algorithm is a value-based method that uses Q-value functions to estimate the goodness of the selected action under a specific state. The possible reason for using this method is that it is suitable for the discrete action space in the network defence environment (e.g., CybORG) and can be a baseline for this new direction. The problem is that value-based methods are unsuitable for high-dimensional action spaces. The DQN algorithm tries to mitigate this problem by using deep neural network to approximate the Q-value function [12]. However, its efficiency is still not satisfactory for large-scale action spaces, and it is relatively difficult to be stable and unsuitable for learning stochastic policies. Therefore, policy-based approaches might be a better solution

The size of the blue action space is 54 in CAGE challenge version 1 and 158 in challenge version 2, and the size of the red action space is 69 in version 1 and 888 in version 2. Moreover, observation spaces in both versions are also huge, with lengths over 11,000. Under this scenario, policy-based RL (e.g., Policy Gradient) might have better convergence properties and can be effective in such a high-dimensional action space [41]. However, policy gradient methods still have problems since they typically converge to a local optimum, and evaluating a policy is inefficient because of high gradient variance [41]. The potential solution is to combine the benefits of both value-based and policy-based methods, which is called Actor-Critic. In this hybrid approach, the policy-based actor controls the agent's behaviours, and the value-based critic evaluates the goodness of selected actions.

Proximal Policy Optimisation (PPO) [38] is a powerful Actor-Critic method that has become the default algorithm choice in many RL projects [36]. Its advantages include more straightforward implementation, simpler tuning, and better performance than most algorithms. The main idea of PPO is to solve efficiency and reliability problems in Trust Region Policy Optimisation (TRPO) [37]. TRPO tried to solve large policy change problems in policy gradient methods by designing a trust region that limits the distance between old and new policies via Kullback–Leibler (KL) divergence. By denoting the probability ratio between old and new policies as

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{4.1}$$

the objective function of TRPO can be represented as

$$L^{TRPO}(\theta) = \hat{\mathbb{E}}_t\Big[r_t(\theta)\hat{A}_t\Big] \tag{4.2}$$

where $\hat{A}_t$ is the estimated advantage. TRPO aims to maximise the objective function in Equation 4.2 but not exceed the trust region with the parameter δ:

$$\hat{\mathbb{E}}_t\Big[\mathrm{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\Big] \leq \delta \tag{4.3}$$

However, TRPO is relatively complicated and it is expensive to implement the constraint [38]. Therefore, PPO imports a clipping function to simplify the algorithm and keep a similar performance. It enforces the probability ratio $r_t(\theta)$ fall inside the interval $[1-\epsilon, 1+\epsilon]$ where ϵ is a hyperparameter. Then, the PPO objective function is

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min\Big(r_t(\theta)\hat{A}_t,\ \mathrm{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t\Big)\Big] \tag{4.4}$$

which causes that the advantage function will be clipped if $r_t(\theta)$ falls outside the range $[1-\epsilon, 1+\epsilon]$. As illustrated in Figure 4.1, the probability ratio $r_t(\theta)$ is clipped at $1+\epsilon$ (the left sub-figure) or $1-\epsilon$ (the right sub-figure), which is determined by whether the values of the advantage function are positive or negative. For this reason, PPO with clipped surrogate objective will reduce occurrences of large policy updates.

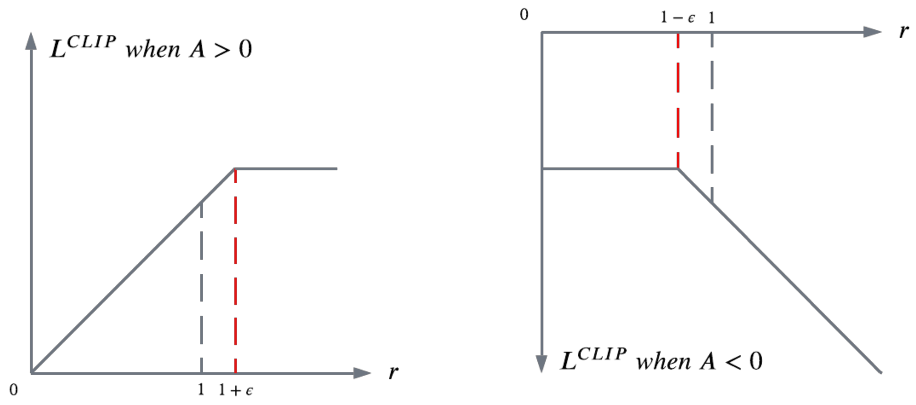

Figure 4.1: Clipped Surrogate Objective Function

Moreover, PPO uses the minibatch stochastic gradient descent (SGD) to make its computations more tractable and efficient. In conclusion, the pseudocode for PPO is presented in Algorithm 1. The code implementation of the standard PPO algorithm is created by RLLib [23], and it is the version used in this research.

Algorithm 1 PPO with Clipped Surrogate Objective Function

```
1: Input: initial policy parameter $\theta_0$, and initial clipping hyperparameter $\epsilon$
2: for k = 0, 1, 2, ... do
3:     Collect set of partial trajectories $\mathcal{D}_k$ by running policy $\pi_\theta = \pi(\theta_k)$ in the environment
4:     Estimate advantages $\hat{A}_t^{\pi_k}$ based on the current value function
5:     Update the policy by maximising the objective function
           $$\theta_{k+1} = \underset{\theta}{\operatorname{argmax}}\ L_{\theta_k}^{CLIP}(\theta)$$
       by taking $K$ steps of minibatch SGD, where
           $$L_{\theta_k}^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min\Big(r_t(\theta)\hat{A}_t^{\pi_k},\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t^{\pi_k}\Big)\Big]$$
6: end for
```

4.2 Intrinsic Curiosity Module

For training blue agents, some other challenges need to be addressed. The CybORG is a sparse reward environment where blue agents do not always receive rewards after actions. The rewards from the environment to blue agents are decided by both red and blue actions. When considering the red action space, the *DiscoverRemoteSystems*, *DiscoverNetworkServices*, and *ExploitRemote-Service* do not have direct punishments (negative reward) for the blue agent. The *Impact* action will only be executed after the target operational server has been privileged, which is still rare in the entire process. The significant red action that always gives punishments to blue agents is *PrivilegeEscalate*, but it might not be enough for training. For the blue action space, the situation is similar. Most actions, including *Monitor*, *Analyse*, *Remove*, and *Misinform*, do not have rewards after their executions. The *Restore* is even a deceptive action that will directly give a negative reward to blue agents though it gives benefits in the long term. Therefore, sparse rewards are a significant challenge in the CybORG environment.

Traditional RL algorithms choose actions by mimicking the reward system in human psychology, and this makes the training process difficult in a sparse reward environment. However, humans can still keep learning in such situations. Recent research that tries to overcome sparse rewards can be divided into three categories: Reward Shaping [28], Curriculum Learning [8], and Hierarchical Reinforcement Learning [22].

Curiosity is one research direction that belongs to Reward Shaping. It imports an internal reward type that guides agents explore the environment and learn more knowledge that might be useful in future interactions. Pathak et al. [30] categorised observation sources into three types: (a) can be controlled by the current agent; (b) cannot be controlled by the current agent but have influences; (c) cannot be controlled by the current agent and have no impact (i.e., random environmental noise). They also found that a good curiosity feature space should model (a) and (b) and are unaffected by (c). The CybORG environment perfectly meets these requirements: the blue agent can control its actions but cannot control the actions from the red agent, and the green agent's *Sleep* actions do not have influences on the blue agent's decisions. This indicates that the Intrinsic Curiosity Module (ICM) might be helpful here. To apply the ICM for the CAGE challenge, its implementation from RLLib [23] is used for training blue agents.

After adding the ICM into interactions between the agent and environment, one state and one action will lead to two rewards r_t^e and r_t^i at time step t: r_t^e is given by the environment while r_t^i is created by the curiosity. This intrinsic reward signal is generated by

$$r_t^i = \frac{\eta}{2} \, || \, \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \, ||_2^2 \tag{4.5}$$

which is represented as a prediction error between $\hat{\phi}(s_{t+1})$ and $\phi(s_{t+1})$. The representation $\hat{\phi}(s_{t+1})$ is the output of a forward model with action a_t and feature $\phi(s_t)$. More specifically, features $\phi(s_t)$ and $\phi(s_{t+1})$ are encoded by the ICM through two separate neural networks using states s_t and s_{t+1}. According to this definition, the agent will be more willing to explore unknown states when the prediction error is larger. However, there exists unrelated information in the state space which will disrupt agent decisions. To mitigate this problem, the algorithm uses an inverse model to estimate an action $\hat{a}_t$ using features $\phi(s_t)$ and $\phi(s_{t+1})$. If a_t and $\hat{a}_t$ is related, then the used information is also related. To conclude, the entire working process is illustrated in Figure 4.2.

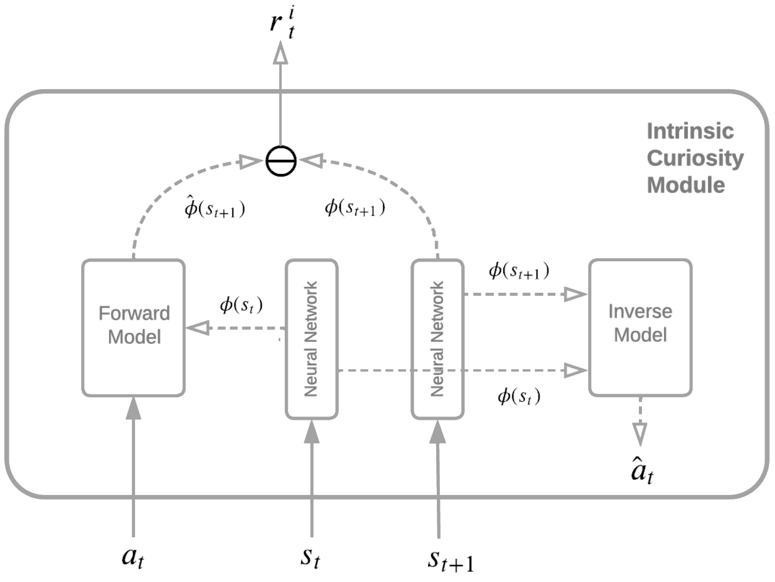

Figure 4.2: Intrinsic Curiosity Module

4.3 Hierarchical Architecture

As explained in Section 4.2, the simulated CybORG environment has sparse rewards for training blue agents. Moreover, several red agents act as APT adversaries and perform persistent behaviours in the environment. The challenge evaluation will test the algorithm performance up to 100 steps in each episode. According to previous research, hierarchical RL can be a mitigation approach to deal with the sparse reward [22] and the long-horizon [29] problems, which indicates that hierarchical RL methods should work well here.

The idea of *Options* was proposed by Sutton et al. in 1999 [45], where the *Options* can be regarded as temporal abstractions of actions. Because the challenge assumes that there is only one kind of red agent in every episode, the entire policy can be divided into several sub-policies in different temporal processes. Therefore, the hierarchical architecture developed for the CAGE challenge in CybORG environment contains several lower-level blue subagents and an upper-level controller. For each red agent, including the provided agents in Subsection 3.2.1 and advanced agents in Section 5.1, a corresponding blue subagent will be trained using the PPO with ICM algorithm. During the blue subagent training processes, the environment and agents are reset to their initial states after each episode until the total training time steps are reached or the performance has converged. For the agent controller, it aims to distinguish which kind of red agents it currently defends. To achieve this goal, the controller will load all trained blue subagents and randomly select a subagent to defend against the current red agent in one episode. Then, the controller will perform the action provided by the blue subagent in each time step. This indicates that observations and rewards the controller will receive are results of subagents' actions. Similarly, the PPO algorithm is selected as the training method for the controller. Finally, the controller will distinguish the type of adversary and select the most suitable subagent to minimise the impacts of these APT adversaries. This hierarchical model can be represented in Figure 4.3.

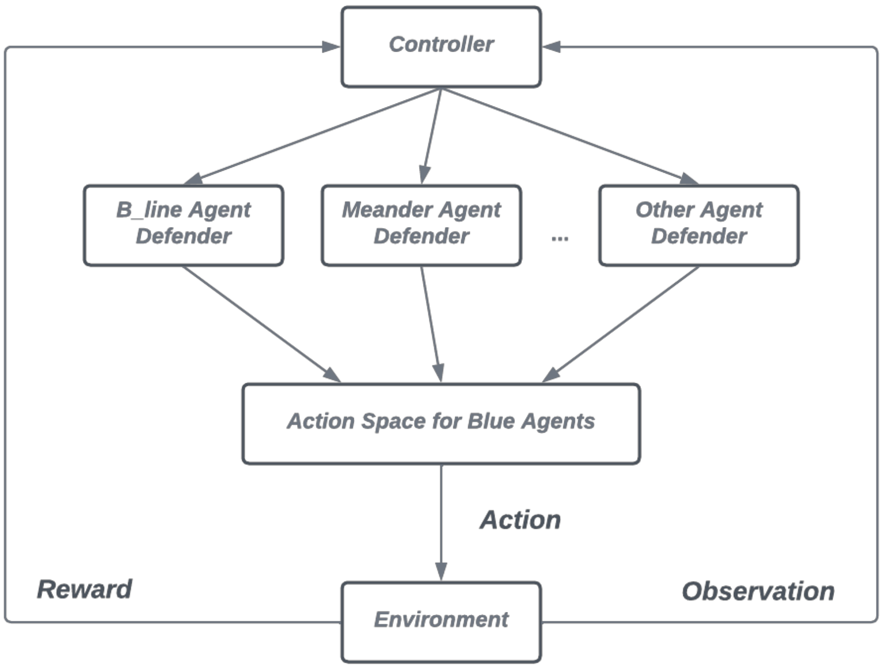

Figure 4.3: Hierarchical Architecture

Chapter 5

Extensions and Experiments

5.1 Extended Adversarial Strategies

From the ATT&CK Matrix [43], adversaries select different kinds of techniques and vulnerabilities to achieve their attacking goals. To investigate the effectiveness and robustness of the algorithm designed in Chapter 4, we extend the CAGE challenge by implementing a new action inspired by the ATT&CK Matrix, randomising exploration paths, and training several red agents against two heuristic blue agents in this section.

5.1.1 Defence Evasion

The CAGE challenge developers [1] use the ATT&CK Matrix tactics and techniques to design abstract and concrete actions in their red action space. For example, the *DiscoverNetworkServices* action is the ATT&CK Technique T1210, and the *PrivilegeEscalate* action is the ATT&CK Tactic TA0004. The newly implemented action *DefenceEvade* is inspired by the ATT&CK Tactic TA0005, which aims to avoid detection during the compromising process. Defence evasion is a necessary step in the lifecycle of APT attacks. Adversaries can use this action in the final phase to hide their attacking tracks and maintain potential accesses in the future, which can be regarded as the basis of the *Persistent* property of typical APT attacks.

This *DefenceEvade* action will be added into the *B_lineAgent* to establish another red agent with a more advanced strategy, which is named *CovertAgent.* This agent will perform the defence evasion action immediately after it gets the administrator access to the target machine except for the operational server (instead preferring to execute the *Impact* action for the operational server). This newly added action will use a hostname as its input and only gives the success or failure observation for the red agent.

Similar to the other red actions, the *DefenceEvade* action has one abstract action and one concrete action called *HideArtifacts* (ATT&CK Technique T1564). The abstract action will detect the existence of the relevant session. The concrete action will be performed only if the relevant session exists; otherwise, the abstract action will return an observation that this action failed. Therefore, if the current machine is restored by blue agents promptly after the privilege escalation, this defence evasion action will not be successful since the necessary session has been removed.

The *HideArtifacts* sub-action will hide artifacts (e.g., sessions, processes, and files) associated with the attacking tracks to evade the detection from the blue agent. For instance, the blue agent will no longer obtain valuable observations from the *Analyse* action after the *DefenceEvade* action. Moreover, the *DefenceEvade* action will set a flag to decrease the exploit detection rate from 0.95 to 0.25 to help adversaries covertly exploit the compromised machine again. This flag can only be unset by the *Restore* action from the blue agent, which aims to show the power of this hidden behaviour and the significance of the credential access (ATT&CK Technique T1212). When considering the action rewards, this *DefenceEvade* action inherits the continuous reward from the *PrivilegeEscalate* action because of the implementation method. The red agent will receive an instant reward for this action as well due to its important influence and the compensation for an extra step compared with the original *B_lineAgent*.

Finally, the *CovertAgent* also has the recovery function such that it can find the correct position to attack again if it fails in an action (including the added *DefenceEvade* action) or is interrupted by the blue agent. The implementation method is to use a dictionary data structure to cache the step number and the selected action. If the red agent needs to recover, there is an array to guide which step it should jump to in each of the different attack phases.

5.1.2 Randomised Attacking Paths

In addition to adding new actions, we also investigate potential flaws of existing agents to improve their performance. The main problem is that random elements are not enough in the environment, which has a significant impact on the robustness of the designed algorithm. One source of non-determinism is the intrusion detection rate which is designed to be 0.95. However, its goal is to ensure that the red agent can eventually access the operational server to complete its APT attacks if one episode has enough steps for the red agent. Another non-determinism occurs when choosing the most valuable exploit vulnerability. Adversaries will have a 25% chance to select a random exploit approach rather than the one with the highest priority. The weights of the provided exploit options are chosen based on how common the corresponding vulnerable service was in the network; therefore, frequently changing their rates might not be an acceptable option.

A significant deficiency in randomness can be found in the fixed attacking paths after analysing the action histories of the two red agents: *B_lineAgent* and *MeanderAgent*. Since the *B_lineAgent* has complete knowledge of the network topology, its attacking path is shortest. This assumption is reasonable since there will exist at least one optimal path in different network structures, so behaviours of the *B_lineAgent* should not be changed. However, the *MeanderAgent* explores the network environment using breadth first search, which also leads to a fixed attacking path in a static network topology. Therefore, the second adversarial strategy we design in this thesis is to use randomised attacking paths in the environment, and the red agent who will apply this strategy is named *RandomAgent*.

The essential problem is that red actions are inherently causally-linked. One action should use the information from the observation result of previous actions, or one action is the prerequisite for the following action. For example, the *ExploitRemoteService* should use the IP address discovered by the *DiscoverRemoteSystems* and port numbers scanned by the *DiscoverNetworkServices*, and each machine can only be privilege escalated after it has been exploited. Additionally, the NACL should also be considered such that devices in the User Subnet cannot directly access machines in the Operational Subnet. In the action implementation, these illogical actions will be considered as *InvalidAction* or their performing results will always be unsuccessful. Therefore, keeping the correct order of each APT attack lifecycle is necessary.

The approach that our *RandomAgent* applies is to shuffle the choice of target machines but not break the inherently interdependent attacking chain at each machine. To be specific, after the port scanning behaviour on one host, the red agent might choose to privilege escalate another exploited device. Furthermore, although the *RandomAgent* still uses a breadth first search of the network, the order in each layer is also shuffled to implement randomness and mislead the blue agent. Finally, the *RandomAgent* also inherits the recovery function from the *MeanderAgent* to keep the attacking process alive.

5.1.3 Autonomous Network Attackers

Although the two originally provided red agents (*B_lineAgent* and *MeanderAgent*) and two our newly designed red agents (*CovertAgent* and *RandomAgent*) are heuristic, it is also possible to use the algorithm in Chapter 4 to train the autonomous red agents. The CybORG environment provides two blue agents used to test the functionality of *Remove* and *Restore* actions: *BlueReactRemoveAgent* and *BlueReactRestoreAgent*. Therefore, two autonomous APT adversaries can be trained against the corresponding blue agent.

The challenge should be reversed such that there already exists a blue agent in the environment, and the trained red agent will be the challenger to achieve higher rewards. However, similar to training blue agents, rewards for red agents are also sparse in the environment. The red action space has only two actions that will provide the positive reward, which are *PrivilegeEscalate* and *Impact*. As explained in Subsection 5.1.2, both actions should use the information collected in previous actions or meet some specific prerequisites. If the red agent breaks the inherently dependent attacking chain, it will receive a -0.1 reward for each invalid action. Moreover, inside the implementation of the *Impact* action, this action can only be successful when it is used on the operational server. Therefore, the trained red agent needs to explore valid actions without rewards before they can execute two actions with rewards, and it is necessary to find a suitable position to recover if it is interrupted by blue agents.

5.2 Training Details

5.2.1 Hardware and Software Environment

All experiments were conducted on a personal laptop which has a 12th Gen Intel Core i9-12900H Process and 32 GB of RAM, where all computation tasks were finished by the CPU without any GPU acceleration. The running environment is the Ubuntu 20.04 operating system built on Windows Subsystem for Linux version 2 (WSL2). The challenge environment is `cage-challenge-1` with several modifications for bug fixes. All RL algorithms are implemented by the open source library RLLib in Python3 (3.8.10). More details about the environment and related dependencies can be accessed through the given GitHub Code Repository in Appendix A.

5.2.2 Training Settings

The training procedure assumes that each red agent will exist in the environment for at least 100 time steps in one episode before being cleaned by blue agents. For the training process, there will be 10 million time steps for blue agents and 20 million for red agents, which appears to be a sufficient time period for the agents' training to converge. Moreover, there are several differences between each training procedure. When training red agents, the agents select actions from the original red action space, which means that the *DefenceEvade* action is not included. When training blue agents, the algorithm PPO with ICM will be used for each subagent, while training hierarchy controller will only use the PPO algorithm.

Each training process needs a selection of various hyperparameter values. The hyperparameters for the *B_lineAgent* and *MeanderAgent* are selected after a grid search [13], and the *CovertAgent* and *RandomAgent* use the same values since they are modified from the original agents. The values for training red agents are similar but encourage more chances to explore the red action space. The controller training prefers to use the default values provided by the PPO algorithm. Please see Appendix B for the detailed hyperparameter values.

5.3 Evaluation and Comparison

5.3.1 Performance of Trained Red Agents

The red agent learning processes are shown in Figure 5.1 with the help of Tensorboard. It can be seen that there are several stages in their learning processes that need to be explained with the assistance of their action histories.

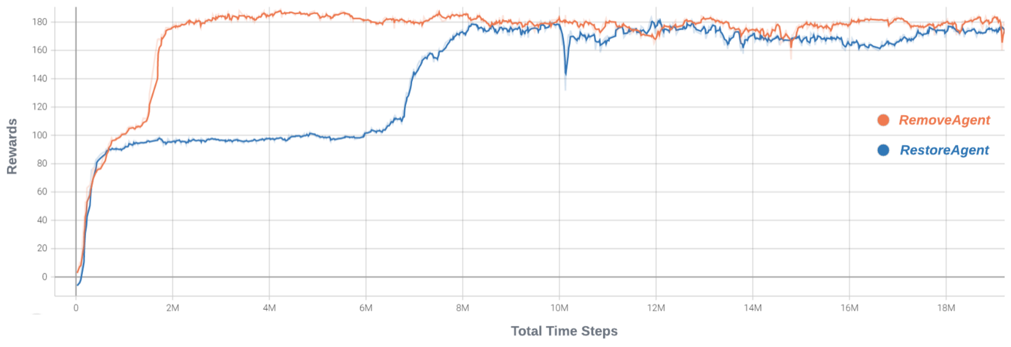

Figure 5.1: Training Processes for Red Agents

There are two planes that divide the entire learning process into four stages. In the first increasing phase, two red agents will learn what kinds of actions are valid, and they must find the logical chain from discovery and exploitation to privilege escalation. The next is a plain phase which indicates that red agents are defended by the corresponding blue agent using the *Remove* or *Restore* action. During this phase, more exploration over the action space should be encouraged; otherwise, the red agent will converge into a low-level local optimum. In the second increasing stage, the red agents find a way to recover after the *Remove* or *Restore* action because these actions will be executed when the malicious activities are detected by the *Monitor* action. The last stage is when two red agents and the corresponding blue agent fall into a defence-recovery circle, so the maximal time step value limits their rewards. Figure 5.2 shows the cumulative rewards of two autonomous red agents under different total time steps.

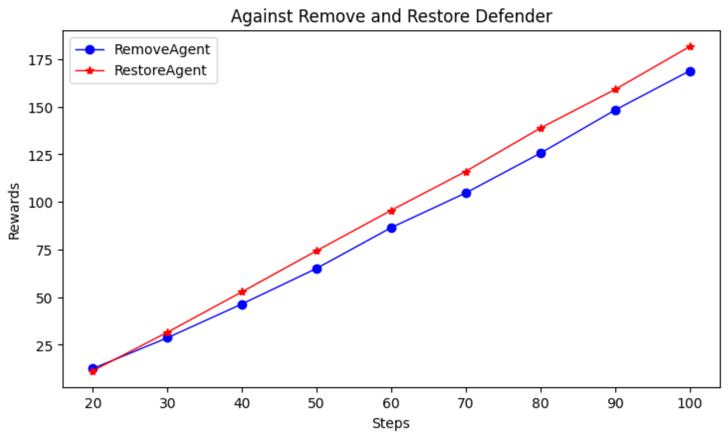

Figure 5.2: Performances of Our Trained Red Agents against the Remove and Restore Defenders

Moreover, since the *Restore* action is more powerful than the *Remove* action, the trained red *RestoreAgent* will spend more time steps learning the recovery method from the defending action. Meanwhile, the trained *RestoreAgent* can also be used to defend against the blue agent operating the *Remove* action with higher performance but not vice versa, as shown in Table 5.1.

Table 5.1: Performance of Each Trained Red Agent after 100 Time Steps

Blue Agent	*RemoveAgent*	*RestoreAgent*
BlueRemoveAgent	168.886	182.956
BlueRestoreAgent	149.180	181.619

5.3.2 Performance of Trained Blue Agents

The evaluation uses the provided method in the CybORG environment with a few modifications. To guarantee the accuracy of evaluation results, each experiment is run over 1000 episodes for each time step selection. Before evaluating the performance of the trained blue agent, the capability of the extended adversarial strategies should be tested as the baseline. The first experiment is to attack the environment protected by the vanilla PPO algorithm, which is already included in the CAGE challenge. The involved red agents are *B_lineAgent*, *CovertAgent*, *MeanderAgent*, *RandomAgent*, and *SleepAgent*. The two red trained agents will not participate due to compatibility issues in the CybORG environment. The results for the first experiment are shown in Figure 5.3.

We find that the *MeanderAgent* and the *RandomAgent* receive similar rewards in every stage. This result indicates that randomised behaviours without breaking the inherently dependent attacking chain does not influence the performance compared with using the breadth first search method. Furthermore, the *CovertAgent* receives fewer rewards (i.e., lower performance) than the

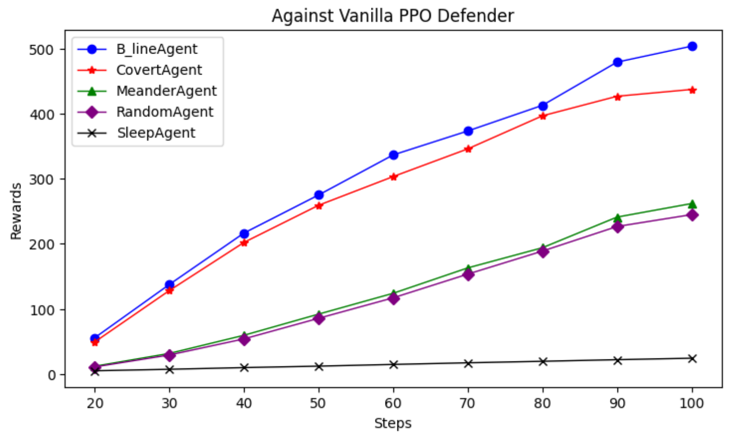

Figure 5.3: Performance of Red Agents against the Vanilla PPO Defender

B_lineAgent. One potential reason is that *CovertAgent* needs to perform an extra action *DefenceEvade* after it successfully privilege escalates a machine, which will delay its executions of other actions with rewards. Finally, four red agents (except the *SleepAgent*) can effectively impact the operational server in almost every episode after checking their printed action histories.

The second experiment is completed with the trained hierarchical defender provided by Mindrake, a team who participated the CAGE challenge [13]. This defender consists of one controller and two subagent defenders for the *B_lineAgent* and *MeanderAgent*. However, this defender has no knowledge about the red agents with extended strategies. Therefore, it is necessary to investigate the performance of this unmodified defender against the two known and two unknown red agents. The results of this experiment are illustrated in Figure 5.4.

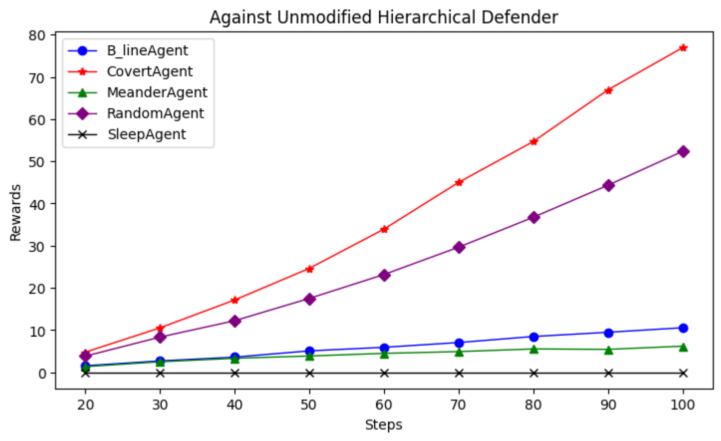

Figure 5.4: Performance of Red Agents against the Unmodified Hierarchical Defender

The results indicate that this trained defender can effectively defend against two known red agents: *B_lineAgent* and *MeanderAgent*. Their received rewards are limited to a range of around 10 which demonstrates that the operational server will never be compromised by these two red agents. However, the extended adversarial strategies work well in this protected environment. The cumulative rewards for the *RandomAgent* reach about 50 after 100 time steps. In its action history, the *RandomAgent* has a 5‰ (5 in 1000) probability that it can successfully impact the operational server. Furthermore, the *CovertAgent* has about 80 cumulative reward after 100 time steps and has more records for the *Impact* action (higher probability) in its action history. According to the performance of the *CovertAgent*, the defence evasion strategy with artifact hiding is more lethal for the unmodified defender who prefers to use *Misinform* greedily during its network defence process. Therefore, training an updated hierarchical defender is necessary for defending against the two red agents with extended adversarial strategies. With the help of Tensorboard, the blue subagents' training processes are shown in Figure 5.5 (deep blue for the *B_lineAgent*, red for the *CovertAgent*, orange for the *MeanderAgent*, and light blue for the *RandomAgent*), and the training process of the controller for four red agents are displayed in Figure 5.6.

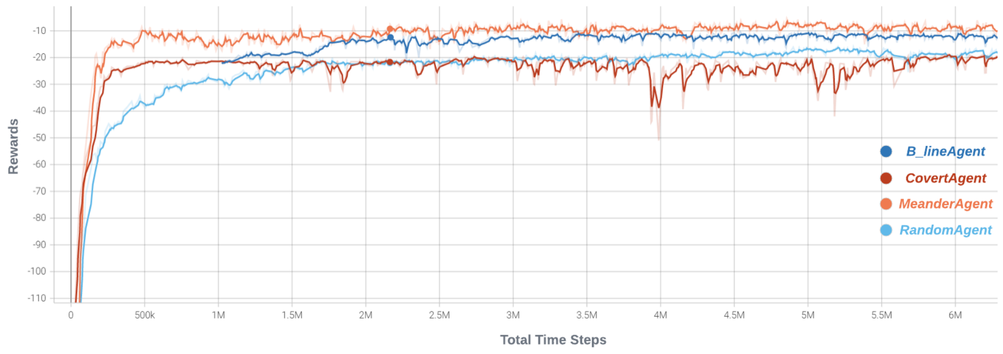

Figure 5.5: Training Processes for Blue Subagents

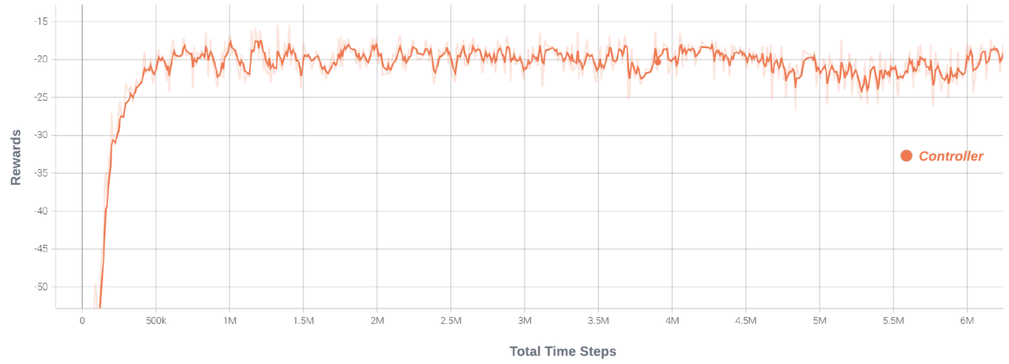

Figure 5.6: Training Process for Blue Controller

These training processes demonstrate that all defenders converge in performance after 2 million steps, which costs around one and a half hours using the hardware and software environment given in Subsection 5.2.1. To be specific, the defenders for *B_lineAgent* and *MenaderAgent* converge to a value of approximately 10 for the reward, while the defenders for *CovertAgent* and *RandomAgent* and the defence controller converge to approximately 20.

Figure 5.7 shows the performance of five red agents against our updated hierarchical defender. The received rewards for the *B_lineAgent* and *MeanderAgent* have few difference from the unupdated one. The *RandomAgent* is expected to have a similar performance to the *MeanderAgent*, but its randomised attacking paths might increase the difficulty of learning a general defending policy, and this defender might converge to a local optimum. After investigating its action history, there is a 1‰ (1 in 1000) chances that the *RandomAgent* can impact the operational server in 100 time steps, which shows that the updated defender is better than the unmodified one.

The *CovertAgent* is a special case whose received rewards are maintained at around 18. After investigating its action history, the trained defender greedily uses the *Remove* action in the first stage rather than the *Misinform* action, which is the main difference from defenders against the other red agents. Moreover, this hierarchical defender does not have the capability to prevent the *CovertAgent* from reaching and privilege escalating the operational server. A possible explanation is the newly designed *DefenceEvade* action exposes less information to the defender. Fortunately, the defender successfully learns to restore the operational server before each time the red agent tries to impact it, which is the second stage of the defender's behaviours. Then, the red agent will exploit the operational server again to continue this kind of iterated strategy, and this is the main reason for staying at a value of 18 for the reward.

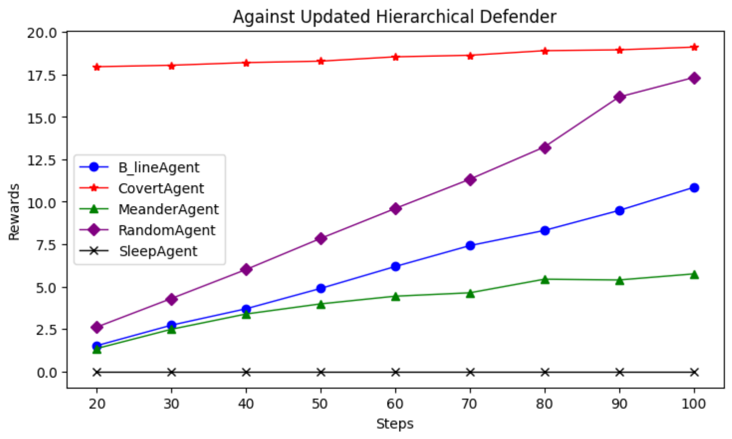

Figure 5.7: Performance of Red Agents against Updated Hierarchical Defender

Finally, Table 5.2 concludes the comparison between three trained blue agents who will defend against four different red agents. The hierarchical blue agent using PPO with ICM algorithm does better than using the vanilla PPO algorithm. Moreover, the blue agent using this newly designed algorithm can adapt to red agents with extended adversarial strategies, and its final score is at a higher level. Therefore, this algorithm successfully defends against multiple APT adversaries over different lengths of time, such that it can provide a more generalised defensive capability with great effectiveness and robustness.

Table 5.2: Performance of Each Blue Agent after 100 Time Steps

Blue Agent	*B_lineAgent*	*CovertAgent*	*MeanderAgent*	*RandomAgent*	Avg.
Vanilla PPO	-503.439	-436.970	-261.677	-244.698	-368.585
Hierarchical-Unmodified	-10.542	-76.914	-6.175	-52.363	-36.980
Hierarchical-Updated	-10.844	-19.115	-5.745	-17.323	-13.613

Chapter 6

Discussion and Conclusion

6.1 Contributions

This research aims to investigate the robustness of hierarchical RL enhanced agents which can autonomously defend different kinds of APT adversaries in a competitive network environment. To achieve this goal, this complicated problem is broken into three sub-directions which we explore:

(a) What are the types of adversarial strategies APT attackers might employ?

(b) How can the defender defend against a single ATP attacker?

(c) How general is the resulting algorithm when it is used to protect against different attackers?

To answer question (a), the entire lifecycle of APT attacks is surveyed through the ATT&CK Matrix. After comparing with the already provided red actions, the abstract action *DefenceEvade* and concrete action *HideArtifacts* serve as a supplement to formulate a more advanced adversarial strategy, which is used to design the *CovertAgent*. Moreover, some shortcomings about randomness are found in the provided red agents *B_lineAgent* and *MeanderAgent*, so the randomised attacking path is the strategy that the *RandomAgent* will apply. Finally, the designed algorithm is also used to train autonomous red agents for the potential applications of RL methods in network attacking. These trained red agents autonomously attack the environment protected by blue agents who will perform *Remove* and *Restore* actions when finding malicious activities.

Before training the corresponding blue agents, the capabilities of these red agents with extended adversarial strategies is tested. The evaluation results for varying lengths of time are shown in Figure 5.2, Figure 5.3, and Figure 5.4, and each experiment is executed over 1000 episodes to guarantee accuracy. The code implementations are shown in Appendix A, along with several additional technical contributions about bugs fixed and issues raised on the corresponding CAGE challenge repositories. The hyperparameter settings for autonomous red agents are listed in Appendix B.1.

For questions (b) and (c), the evaluation results of the blue subagents and controller provide a reliable answer. Each blue subagent is trained to defend against one specific red agent with its own adversarial strategy. As shown in Figure 5.5, all defenders converged into an acceptable level of performance after carefully tuning the hyperparameters, as listed in Appendix B.2. Moreover, the trained controller can successfully distinguish the type of red agent it currently defends against and then will choose the correct actions to mitigate the APT attack. Therefore, as illustrated in Table 5.2, the defender using the hierarchical RL algorithm overcomes the overfitting problem in defending against one particular adversarial strategy and provides a high and more generalised performance for autonomous network defence.

6.2 Limitations and Future Works

Although this research currently has successfully designed some extended adversarial strategies and measured the robustness of hierarchical RL algorithm, there are still some limitations in the designed experiments.

The first one is that there are some compatibility issues with the Wrappers in the CybORG environment, such that it cannot support the performance comparisons between two trained red agents and other code-designed red agents. For the same reason, the environment also does not enable training of autonomous red agents for the hierarchical RL enhanced defender. These issues limit the possibility of training more advanced autonomous red agents. Therefore, one of the future works is to overwrite the original wrappers given by the CybORG or implement a custom one. Then, the capabilities of autonomous red agents can be measured, and more advanced red agents can be trained to attack the defender using the algorithm described in Chapter 3. Moreover, these processes can be repeated to find a more generalised attacking or defending strategy in this interactive cyber environment.

More actions can also be added into the CybORG environment to design different kinds of adversarial attacking strategies. One that should be considered is that actions from green agents need to be included into the environment as noise, which is what the project description file claims. After some preliminary modifications and testing, the currently trained blue agents are likely to overreact if green agents perform actions (e.g., port scanning) with significant observations. This kind of overreacting behaviour will seriously reduce the performance (the total rewards) of the hierarchical defender, which means the defender used in this research should be trained again for the environment with any significant noise. Another potential improvement is to enhance the simulation of the file system. The current file system is used to add and detect malicious files, which is different from the real-world environment where these files usually hide inside huge numbers of benign files. It is possible to dynamically create or delete files with varying levels of threat during the process such that it will be more challenging for defenders to find valuable information from a larger observation space. Finally, the virtual environment should not be the final target, and all these functions should be realised using virtual machines running on Amazon Web Services as before to test the authenticity.

Not restricted to the environment implementation, the algorithm scalability should also be tested in different network topologies and sizes. The current network contains only 13 machines distributed in three subnets with basic NACL rules. However, there are countless network topologies in the real world, and the decreasing performance of the *RandomAgent* exacerbates the concern about the scalability of the developed algorithm. Moreover, the increasing number of machines in the network will lead to the exponential growth in the size of both red and blue action spaces [39] [46]. Although developers might be able to spend more time on training to deal with the large network size, this method does not meet the quick response requirement in industry if the blue agents should be retrained (e.g., APT attackers can find other state-of-the-art strategies). Fortunately, the CybORG environment provides a scenario file that can help the researchers to define customised network topology with different numbers of machines inside. Therefore, the work for the next stage will test the performance of the designed algorithm in as many network scenarios as possible, and the potential solution is to implement a generator of randomised network topologies for the CybORG environment.

Finally, the explainability of RL algorithms is a frontier and promising direction in the academic community. Although this research tries to explain the learning process of trained agents via their action histories, it still needs reliable quantification methods to formalise their attacking or defending principles. This future plan will help the researchers in the Artificial Intelligence area comprehend and retrace the process of arriving at results, and then design more accurate models and more advanced algorithms to improve their performances and trustworthiness.

6.3 Closing Remarks

Network attacking and defending scenarios might still be a cat-and-mouse game. Network attacks have become more complex in recent years, and adversaries will take more advanced strategies to perform attacks. This research investigates the robustness of the RL approaches when applied to autonomous network defenders. Although there are still some future works for limitations to be completed, the results in Chapter 5 indicate that the designed algorithm in Chapter 4 can deal with adversaries using different adversarial strategies over varying lengths of time with outstanding performance and robustness. Currently, academic interest in autonomous network defence is rising, involving automated penetrating testing, intelligent intrusion detection systems, and more sophisticated techniques in a highly competitive environment like the CAGE challenge. This research might help alleviate the defence dilemma in large-scale and complex network environments.

Bibliography

[1] *CAGE Challenge 1*. arXiv, 2021.

[2] *CybORG: A Gym for the Development of Autonomous Cyber Agents*. arXiv, 2021.

[3] Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, Moutaz Alazab, et al. Zero-day malware detection based on supervised learning algorithms of api call signatures. 2010.

[4] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, 2002.

[5] CVE Numbering Authorities. CVE-2014-0322. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0322.

[6] CVE Numbering Authorities. CVE-2017-0144. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0144.

[7] Rebecca Gurley Bace, Peter Mell, et al. Intrusion detection systems. 2001.

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[9] Sergio Caltagirone, Andrew Pendergast, and Christopher Betz. The diamond model of intrusion analysis. Technical report, Center For Cyber Intelligence Analysis and Threat Research Hanover Md, 2013.

[10] Ping Chen, Lieven Desmet, and Christophe Huygens. A study on advanced persistent threats. In *IFIP International Conference on Communications and Multimedia Security*, pages 63–72. Springer, 2014.

[11] Bhagyashree Deokar and Ambarish Hazarnis. Intrusion detection system using log files and reinforcement learning. *International Journal of Computer Applications*, 45(19):28–35, 2012.

[12] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In Alexandre M. Bayen, Ali Jadbabaie, George Pappas, Pablo A. Parrilo, Benjamin Recht, Claire Tomlin, and Melanie Zeilinger, editors, *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pages 486–489. PMLR, 10–11 Jun 2020.

[13] Myles Foley, Chris Hicks, Kate Highnam, and Vasilios Mavroudis. Autonomous network defence using reinforcement learning. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 1252–1254, New York, NY, USA, 2022. Association for Computing Machinery.

[14] Zhenguo Hu, Razvan Beuran, and Yasuo Tan. Automated penetration testing using deep reinforcement learning. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 2–10. IEEE, 2020.

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

[16] FireEye Inc. APT28: A Window into Russia'S Cyberespionage Operations? https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-apt28.pdf, 2010.

[17] FireEye Inc. APT41: A Dual Espionage and Cyber Crime. https://content.fireeye.com/apt-41/rpt-apt41, 2019.

[18] FireEye Inc. M-Trends 2021: Cyber Security Insights. https://vision.fireeye.com/editions/11/11-m-trends.html, 2021.

[19] Mandiant Inc. Darwin's Favorite APT Group. https://www.mandiant.com/resources/darwins-favorite-apt-group-2.

[20] Mandiant Inc. APT1: Exposing One of China's Cyber Espionage Units. https://www.mandiant.com/sites/default/files/2021-09/mandiant-apt1-report.pdf, 2021.

[21] Katharina Krombholz, Heidelinde Hobel, Markus Huber, and Edgar Weippl. Advanced social engineering attacks. *Journal of Information Security and applications*, 22:113–122, 2015.

[22] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[23] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR, 10–15 Jul 2018.

[24] Manuel Lopez-Martin, Belen Carro, and Antonio Sanchez-Esguevillas. Application of deep reinforcement learning to intrusion detection for supervised problems. *Expert Systems with Applications*, 141:112963, 2020.

[25] Son Hoang Toby Richer Martin Lucas Maxwell Standen, David Bowman and Richard Van Tassel. Cyber autonomy gym for experimentation challenge 1. https://github.com/cage-challenge/cage-challenge-1, 2021.

[26] MITRE. ATT&CK Matrix for Enterprise. https://attack.mitre.org.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[28] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[29] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

[30] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[31] Julien Perolat, Bart de Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *arXiv preprint arXiv:2206.15378*, 2022.

[32] Niels Provos et al. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, pages 1–14, 2004.

[33] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. Pomdps make better hackers: Accounting for uncertainty in penetration testing. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[34] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. Penetration testing== pomdp solving? *arXiv preprint arXiv:1306.4714*, 2013.

[35] Robert Schmidt, Gregory J Rattray, and Christopher J Fogle. Methods and apparatus for developing cyber defense processes and a cadre of expertise, July 10 2008. US Patent App. 11/947,655.

[36] John Schulman. Proximal policy optimization, Sep 2020.

[37] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

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

[39] Jonathon Schwartz and Hanna Kurniawati. Autonomous penetration testing using reinforcement learning. *arXiv preprint arXiv:1905.05965*, 2019.

[40] David Silver. Notes for Lecture 2 Markov Decision Processes in COMPGI13, 2015.

[41] David Silver. Notes for Lecture 7 Policy Gradient Methods in COMPGI13, 2015.

[42] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[43] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck®: Design and philosophy. 2020.

[44] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction.* MIT press, 2018.

[45] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

[46] Khuong Tran, Ashlesha Akella, Maxwell Standen, Junae Kim, David Bowman, Toby Richer, and Chin-Teng Lin. Deep hierarchical reinforcement agents for automated penetration testing. *arXiv preprint arXiv:2109.06449*, 2021.

[47] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[48] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[49] Andrew Whitaker and Daniel P Newman. *Penetration Testing and Network Defense: Penetration Testing _1.* Cisco Press, 2005.

[50] Xin Xu and Tao Xie. A reinforcement learning approach for host-based intrusion detection using sequences of system calls. In *International Conference on Intelligent Computing*, pages 995–1003. Springer, 2005.

Appendix A

Code Repository and Technical Contributions

A.1 Code Repository

All implementation code and evaluation results can be found at the following GitHub Repo: `https://github.com/ZzPoLariszZ/Autonomous-Network-Defence`

It contains three main modules:

- `cage-challenge-1`: the CAGE challenge contains the extended adversarial strategies
- `cage-challenge-1-unmodified`: the unmodified algorithm and evaluation method
- `cage-challenge-1-updated`: the updated algorithm, evaluation method and results

Details about installation, evaluation, and training are illustrated in `README.md`

A.2 Technical Contributions

Pull Requests

Fix the issue that the *Restore* action cannot correctly remove malicious files on hosts

- `https://github.com/cage-challenge/cage-challenge-2/pull/1` (Merged)
- `https://github.com/cage-challenge/cage-challenge-1/pull/16` (Merged)

Fix the issue that the *Impact* action cannot correctly give a -10 reward to blue agents

- `https://github.com/cage-challenge/cage-challenge-1/pull/17` (Merged)

Appendix B

List of Hyperparameters

B.1 Hyperparameters for Autonomous Red Agents

Table B.1: List of Hyperparameters for Autonomous Red Agents

Hyperparameter	*Remove Attacker*	*Restore Attacker*
Steps in each Episode	100	100
Maximum Training Steps	20,000,000	20,000,000
CPU Parallelism	0	0
PPO: Learning Rate	0.0005	0.0005
PPO: vf_share_layers	True	True
PPO: vf_loss_coeff	0.05	0.05
PPO: clip_param	0.75	0.50
PPO: vf_clip_param	2.5	5.0
ICM: Weight for Intrinsic Rewards	1.5	1.0
ICM: Learning Rate	0.01	0.01
ICM: Feature Dimensionality	288	288
ICM: Hidden Layers of Inverse Model	256	256
ICM: Hidden Layers of Forward Model	256	256
ICM: Weight for Forward Loss	0.1	0.2
ICM: Exploration Type	Stochastic Sampling	Stochastic Sampling

B.2 Hyperparameters for Autonomous Blue Agents

Table B.2: List of Hyperparameters for *B_lineAgent* and *CovertAgent* Defenders

Hyperparameter	*B_lineAgent* *Sub-Defender*	*CovertAgent* *Sub-Defender*
Steps in each Episode	100	100
Maximum Training Steps	10,000,000	10,000,000
CPU Parallelism	0	0
PPO: Learning Rate	0.0001	0.0001
PPO: vf_share_layers	True	True
PPO: vf_loss_coeff	0.01	0.01
PPO: clip_param	0.5	0.5
PPO: vf_clip_param	5.0	5.0
ICM: Weight for Intrinsic Rewards	1.0	1.0
ICM: Learning Rate	0.001	0.001
ICM: Feature Dimensionality	288	288
ICM: Hidden Layers of Inverse Model	256	256
ICM: Hidden Layers of Forward Model	256	256
ICM: Weight for Forward Loss	0.2	0.2
ICM: Exploration Type	Stochastic Sampling	Stochastic Sampling

Table B.3: List of Hyperparameters for *MeanderAgent* and *RandomAgent* Defenders

Hyperparameter	*MeanderAgent* *Sub-defender*	*RandomAgent* *Sub-defender*
Steps in each Episode	100	100
Maximum Training Steps	10,000,000	10,000,000
CPU Parallelism	0	0
PPO: Learning Rate	0.0001	0.0001
PPO: vf_share_layers	False	False
PPO: vf_loss_coeff	0.01	0.01
PPO: clip_param	0.5	0.5
PPO: vf_clip_param	5.0	5.0
ICM: Weight for Intrinsic Rewards	1.0	1.0
ICM: Learning Rate	0.001	0.001
ICM: Feature Dimensionality	288	288
ICM: Hidden Layers of Inverse Model	256	256
ICM: Hidden Layers of Forward Model	256	256
ICM: Weight for Forward Loss	0.2	0.2
ICM: Exploration Type	Stochastic Sampling	Stochastic Sampling

Table B.4: List of Hyperparameters for Controller

Hyperparameter	*Controller*
Steps in each Episode	100
Maximum Training Steps	10,000,000
CPU Parallelism	16
PPO: Learning Rate	0.0001
PPO: vf_share_layers	True
PPO: vf_loss_coeff	0.005
PPO: clip_param	0.3
PPO: vf_clip_param	10.0