PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
الجمهـوريـــة الجزائـريـــة الديـمقراطيـــة الشعبيـــة
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
وزارة التعليـم العالـي والبحـث العلمـي
HIGHER SCHOOL OF COMPUTER SCIENCE AND DIGITAL TECHNOLOGIES-BEJAIA
المـدرسة العـليا في علـوم وتكنـولوجيـات الاعـلام الالـي بجـاية

Dissertation Submitted to the Department Of Computer Science in Partial Fulfillment of the Requirements for Engineer's Degree in Computer Science

Specialty: Artificial Intelligence and Data Sciences

Submitted By:

ALMAMMA Amir

Investigation of Poisoning Effects and Defensive Measures on Reinforcement Learning-based Intrusion Response via Optimal Stopping

Supervised by:

Pr. Pierre Martin Tardif, UDES
Mrs. BOUALI Meriem, ESTIN

Members of jury:

▪ **Dr.** AMARA Karima	President	ESTIN
▪ **Dr.** CHEKLAT Lamia	Examiner	ESTIN
▪ **Mr.** Nasreddine ABDELLI	Examiner	ESTIN
▪ **Mrs.** KASSA Radia	Examiner	ESTIN

Academic year: 2023/2024

إهـداء

إلى أبي رحمه الله و أمي، صاحبي الفضل على بعد الله،

إلى إخوتي و أختيّ و جميع أفراد عائلتي،

إلى جميع اصدقائي الذين شاركوني المشوار الدراسيو شاركوني فرحة كل نجاح و حزن كل تعثر،

إلى أولائك الذين سألتهم عونهم كلما أشكل علي الأمر،و الزملاء في النوادي العلمية،

إليكم أهدي هذا العمل، أنتم من شاركتموني المشوار و ليس من أحد أحق منكم أن يشاركني فرحة النجاح.

شكر و عرفان

الحمد لله أولا أن وفقني لهذا وماكنت لأبلغه لولا أن وفقني، وأعانني على إتمامه وما كنت لأتمه لولا أن أعانني. أحمدك اللهم ملء السماوات والأرض وملء مابينهما وملء ما شئت دون ذلك، لا أحصي ثناء عليك، أنت كما أثنيت على نفسك. أشكر شكرا جزيلًا الهيئة المشرفة على هذا المشروع، طالبة الدكتوراه بوالي مريم، طالب الدكتوراه DJeff Kanda Nkashama ، و البروفيسور Pierre-Martin tardif ، الذين لولا نصحهم وتوجيههم، ما كان هذا العمل قد نجح. أشكر صبركم على أخطائي وأشكر تصويبكم إياها، وأشكركم فوق كل شيء، على ما تعلمت منكم.

أشكر أيضا والدتي التي شاركتني قلق هذا العمل. وإخوتي و أختيّ الذين ساعدوني بالنصح و التوجيه. الشكر كذلك لأولئك الذين ساعدوني ولم يبخلوا علي بمشورة أو رأي، لا لشيء سوى كرم أنفسهم وطيبة معدنهم. أحمد ياسين و وائل نجم الدين و أيوب و وجيه و عز الدين و بلال و وصال و الدكتور سفيان عيساني. والشكر لله آخرًا كما له الشكر أولا، هو الذي أنعم علي بهؤلاء جميعا، وبهذا كله فالحمد له ملء ما أنعم، والحمد له ملء ما شاء.

Abstract

As cyber infrastructures become increasingly complex and integral to critical sectors like finance, healthcare, and national security, the need for advanced security systems continues to grow. Intrusion Detection System (IDS) have become a cornerstone in detecting and mitigating a wide range of cyber threats. However, as attacks become more sophisticated, there is a pressing need for systems that not only detect intrusions but also autonomously deploy defensive actions in real-time without human intervention.

This engineering report focuses on the development of poisoning strategies aimed at compromising the training process of Reinforcement Learning (RL)-based agents in active IDS. It also proposes defensive measures to mitigate the adverse effects of such poisoning attacks, including the implementation of a dynamic reward adjustment framework through a *MetaAgent* strategy. The effectiveness of these strategies was evaluated within the Cyber Security Learning Environment (CSLE) framework, exploring the impact of poisoning on intrusion length and agent performance.

Furthermore, the report highlights the challenges faced during the experiments, particularly the limitations in computational resources, which constrained the ability to conduct large-scale experiments. Future work will explore alternative defensive strategies and further investigate methods to enhance the robustness of RL-driven IDS.

Keywords — Reinforcement Learning, Cyber Security, Intrsusion Detection Systems, Reinforcement Learning Rebustness.

Résumé

Alors que les infrastructures informatiques deviennent de plus en plus complexes et essentielles dans des secteurs critiques tels que la finance, la santé et la sécurité nationale, le besoin de systèmes de sécurité avancés ne cesse de croître. Les IDS sont devenus un outil essentiel pour la détection et la mitigation de diverses cyberattaques. Cependant, à mesure que les attaques deviennent plus sophistiquées, il est crucial de disposer de systèmes capables non seulement de détecter les intrusions, mais aussi de déployer de manière autonome des actions défensives en temps réel, sans intervention humaine.

Ce rapport de projet d'ingénieur se concentre sur le développement de stratégies d'empoisonnement visant à compromettre le processus d'entraînement des agents basés sur le RL dans des IDS actifs. Il propose également des mesures défensives pour atténuer l'impact des attaques d'empoisonnement, notamment par la mise en place d'une méthode d'ajustement dynamique des récompenses via une stratégie *MetaAgent*. L'efficacité de ces stratégies a été évaluée dans le cadre du CSLE, en explorant l'impact de l'empoisonnement sur la durée des intrusions et les performances des agents.

De plus, le rapport souligne les défis rencontrés lors des expériences, notamment les limitations en termes de puissance de calcul, qui ont restreint la possibilité de mener des expériences à grande échelle. Les travaux futurs se concentreront sur l'exploration de stratégies défensives alternatives et l'approfondissement des méthodes pour améliorer la robustesse des IDS utilisant le RL.

Mots clés — Apprentissage par renforcement, Cybersécurité, Systèmes de détection d'intrusion, Robustesse de l'apprentissage par renforcement.

ملخص

مع ازدياد تعقيد البنى التحتية السيبرانية وأهميتها في القطاعات الحيوية مثل المالية، الرعاية الصحية، والأمن الوطني، فإن الحاجة إلى أنظمة أمنية متقدمة تسمر بشكل دائم. لقد أصبحت أنظمة الكشف عن التطفل وسيلة مهمة في الكشف عن مجموعة واسعة من الهجمات السيبرانية وتخفيفها. ومع ذلك، مع تزايد تعقيد الهجمات، تبرز الحاجة إلى أنظمة لا تكتفي فقط بالكشف عن الخطر، بل تقوم أيضًا بتفعيل إجراءات دفاعية تلقائية بشكل آني دون تدخل بشري.

يركز هذا التقرير الهندسي على تطوير استراتيجيات التسميم التي تهدف إلى تعطيل عملية تدريب أنظمة التعليم المعزز المستخدمة في أنظمة الكشف عن التطفل النشطة. كما يقترح التقرير إجراءات دفاعية للتخفيف من التأثيرات السلبية لمثل هذه الهجمات التسميمية، بما في ذلك تنفيذ إطار تعديل مكافآت مرن عبر استراتيجية MetaAgent . تم تقييم فعالية هذه الاستراتيجيات باستخدام الوسيلة CSLE ، مع استكشاف تأثير التسميم على طول مدة التطفل وأداء الأنظمة الذكية.

علاوة على ذلك، يسلط التقرير الضوء على التحديات التي تمت مواجهتها خلال التجارب، خاصة القيود المتعلقة بالقدرات الحاسوبية للأجهزة المتوفرة، والتي قيدت القدرة على إجراء تجارب واسعة النطاق. ستستكشف الأبحاث المستقبلية استراتيجيات دفاعية بديلة وستحقق بشكل أعمق في الأساليب التي يمكن أن تعزز من تحمل وسائل الكشف عن التطفل المعتمدة على التعلم المعزز.

الكلمات المفتاحية ــ التعلم المعزز، الأمن السيبراني، أنظمة كشف التطفل، متانة التعلم المعزز.

Contents

Abstract ii

Résumé iii

ملخص iv

Contents viii

List of figures ix

Abbreviations x

General introduction 1

1 Background 3

1.1 Introduction 3

1.2 Reinforcement Learning 4

1.2.1 Introduction 4

1.2.2 Definition 4

1.2.3 Reinforcement Learning Framework 5

1.2.4 Markov Decision Processes 5

1.2.5 Objective of Reinforcement Learning 7

1.2.6 Policy and Value function 7

1.2.7 Taxonomy Of Reinforcement Learning Algorithms 9
1.2.8 Conclusion 10
1.3 Reinforcement Learning Robustness 10
1.3.1 Introduction 10
1.3.2 Defintion 10
1.3.3 Transition Robust 11
1.3.4 Disturbance Robust 12
1.3.5 Action Robust 12
1.3.6 Observation Robust 13
1.3.7 Conclusion 14
1.4 Cyber Attacks Types 14
1.4.1 Introduction 14
1.4.2 Conclusion 16
1.5 Intrusion Detection Systems 16
1.5.1 Introduction 16
1.5.2 Definition 17
1.5.3 Types of Intrusion Detection Systems 17
1.5.4 Conclusion 21
1.6 Conclusion 21

2 Reinforcement Learning for Intrusion Detection Systems 23

2.1 Introduction 23
2.2 Reinforcement Learning in Active Intrusion Detection Systems . . . 23
2.2.1 Case Study 24
2.2.2 Summary of RL Applications in IDS 27
2.3 Conclusion 27

3 Our Work **29**

3.1 Introduction . . . 29

3.2 Tools and Frameworks . . . 29

3.3 Approach . . . 30

3.3.1 Belief Space and Its Evolution . . . 31

3.3.2 Reward Setting and Calculation . . . 31

3.3.3 Action Space . . . 31

3.3.4 Poisoned Environment and Defensive Strategies . . . 32

3.3.5 Summary . . . 33

3.4 Implementation . . . 33

3.4.1 Poisoned Environment Wrapper . . . 34

3.4.2 MetaAgent Class . . . 36

3.4.3 Poisoning Strategy . . . 38

3.4.4 Integration into the Experiment Configuration . . . 39

3.4.5 Training the Agent with PPO . . . 40

3.5 Conclusion . . . 40

4 Evaluation and Results **41**

4.1 Introduction . . . 41

4.2 Overview of the Experiment Configuration . . . 41

4.3 Scenarios Evaluated . . . 42

4.4 Impact of Poisoning . . . 42

4.5 Effectiveness of MetaAgent's Adaptive Strategy . . . 43

4.6 Conclusion . . . 44

Discussion, Limitations, and Future Work **45**

4.7 Introduction . . . 45

4.7.1 Discussion of Results 45
4.7.2 Limitations 45
4.7.3 Future Work 46
4.8 Conclusion 46

Bibliography **47**

List of Figures

1.1 The agent–environment interaction in reinforcement learning (Sutton & Barto, 2018) . . . 6

1.2 Classification of RL Algorithms . . . 9

1.3 Transition Robustness in RL. (Moos et al., 2022) . . . 11

1.4 Disturbance Robustness in RL. (Moos et al., 2022) . . . 12

1.5 Action Robustness in RL. (Moos et al., 2022) . . . 13

1.6 Observation Robustness in RL. (Moos et al., 2022) . . . 14

1.7 Centralized IDS deployment in a network architecture (Bace, Mell, et al., 2001) . . . 17

2.1 Response Policies for Denial of Service (DoS) attacks (Phan & Bauschert, 2022) . . . 24

2.2 DeepAir framework in an Software Defined Networking (SDN) environment (Phan & Bauschert, 2022) . . . 25

2.3 The IT infrastructure and the actors in the intrusion response game (Hammar & Stadler, 2024) . . . 26

3.1 CSLE framework architecture (Limmen, 2020) . . . 30

3.2 Poisoned Environment Wrapper Architecture . . . 34

4.1 Comparison of Intrusion Length: Poisoned vs. Non-Poisoned Environments . . . 43

4.2 Comparison of Intrusion Length: With and Without MetaAgent Strategy . . . 44

Abbreviations

AC	**A**ctor-**C**ritic
CSLE	**C**yber **S**ecurity **L**earning **E**nvironment
DDoS	**D**istributed **D**enial **O**f **S**ervice
DDQN	**D**ouble **D**eep **Q**-**N**etwork
DoS	**D**enial **O**f **S**ervice
DP	**D**ynamic **P**rogramming
DQN	**D**eep **Q**-**N**etwork
Dueling DQN	**D**ueling **D**eep **Q**-**N**etwork
IDPS	**I**ntrusion **D**etection and **P**revention **S**ystem
IDS	**I**ntrusion **D**etection **S**ystem
IRS	**I**ntrusion **R**esponse **S**ystem
MC	**M**onte **C**arlo
MDP	**M**arcov **D**ecision **P**rocesse
MitM	**M**an-**I**n-**T**he-**M**iddle
ML	**M**achine **L**earning
PPO	**P**roximal **P**olicy **O**ptimization
Q-learning	**Q-LEARNING**
RL	**R**einforcement **L**earning
SARSA	**S**tate **A**ction **R**eward **S**tate **A**ction
SDN	**S**oftware **D**efined **N**etworking
SNMP	**S**imple **N**etwork **M**anagement **P**rotocol
SQL	**S**tructured **Q**uery **L**anguage
TRPO	**T**rust **R**egion **P**olicy **O**ptimization
VPN	**V**irtual **P**rivate **N**etwork
WSNs	**W**ireless **S**ensor **N**etworks
XSS	**C**ross-**S**ite **S**cripting

General introduction

In recent years, the increasing complexity of computer infrastructures has significantly heightened the challenges in securing critical systems such as finance, healthcare, and government networks. As threats become more sophisticated, the need for innovative security measures is more urgent than ever. Intrusion Detection Systems (IDS) have played a central role in identifying and mitigating cyber threats, yet they often struggle to keep up with the evolving landscape of cyber-attacks.

A key limitation of traditional IDS is the reliance on passive detection methods, where the system identifies an intrusion but requires human intervention to respond. This delay can be costly in scenarios where immediate action is critical. As a result, cybersecurity experts have begun focusing on active IDS, which not only detect intrusions but also initiate automated defensive actions to mitigate the threat in real-time.

Reinforcement Learning (RL) offers a promising approach to addressing this challenge by enabling systems to learn and adapt through interaction with dynamic environments. By modeling cybersecurity as a decision-making problem, RL allows the creation of intelligent agents capable of responding to intrusions autonomously. These agents can be trained to deploy defensive policies that actively mitigate attacks, making them ideal for integration into active IDS.

However, RL-based IDS are susceptible to adversarial attacks such as poisoning, where attackers manipulate the learning process to degrade system performance. The focus of this report is on the investigation of poisoning strategies and the defensive measures that can be employed to safeguard RL-driven IDS.

This report is organized as follows:

- **Chapter 1: Background**
  This chapter provides a comprehensive overview of the fundamental concepts of RL and RL robustness. It also introduces key cybersecurity concepts such as cyber-attacks and IDS.
- **Chapter 2: Reinforcement Learning for Intrusion Detection Systems**
  This chapter discusses the application of RL in IDS, including case studies and a summary of its usage in active defense strategies.
- **Chapter 3: Work**
  This chapter outlines the tools and frameworks used for the project, along with an explanation of the proposed approach and its implementation.

- **Chapter 4: Results and Discussion**
  This chapter presents the experimental results of the applied poisoning strategies and defensive measures, followed by a discussion of their implications, limitations, and potential areas for future work.

In this report, we aim to explore and analyze the impact of poisoning strategies on RL-based IDS and propose potential solutions to enhance the system's resilience against such attacks.

Chapter 1

Background

1.1 Introduction

The advancement of modern technology has led to increasingly complex and interconnected digital systems. As these systems grow in importance across various sectors, their vulnerability to cyberattacks also increases. This chapter outlines key concepts that underpin the work in this report, focusing on RL, RL robustness, cyber attacks, and IDS.

First, we introduce RL, a branch of Machine Learning (ML) that focuses on training agents to make sequential decisions by interacting with an environment. RL has shown great potential in applications requiring dynamic adaptation, making it suitable for cybersecurity contexts like IDS.

Next, we discuss RL robustness, an essential aspect of deploying RL in real-world environments. This section examines the challenges of making RL systems resilient to adversarial attacks and uncertainties, which is critical when applying RL to the security domain.

We then cover the landscape of cyber attacks, highlighting different attack types, their objectives, and the methodologies used by attackers to exploit system vulnerabilities. Understanding these attacks is crucial for developing effective detection and mitigation strategies.

Finally, the concept of IDS is explored. IDS play a pivotal role in cybersecurity, acting as monitoring systems that detect suspicious activities within a network. The shift towards active IDS, which can autonomously take action to mitigate threats, is particularly relevant in the context of RL-based solutions.

1.2 Reinforcement Learning

1.2.1 Introduction

RL has emerged as an effective approach within modern ML, garnering significant interest due to its promising results and successful application in numerous real-world domains. Consequently, RL has become an important research topic in recent years, with the aim of applying its cutting-edge algorithms and techniques to practical real-world problems.

This section provides an overview of RL and its fundamental concepts, including its components, its formulation as a Marcov Decision Processe (MDP), and its objectives. Additionally, a taxonomy is proposed based on existing classifications in the literature, in order to provide an overview of the different approaches used in RL and to facilitate easy comparison between them.

1.2.2 Definition

RL is a domain that emerged from two research currents, cognitive sciences and optimal control problems research (Agostinelli et al., 2018) . The aim of Cognitive sciences is to understand the principles of human intelligence in order to simulate cognitive abilities. Cognitive science researchers found a particular interest in simulating the learning by interaction ability of the brain, were humans learn by interacting with the environment around them in the presence of a feedback and by performing sequential actions.

RL is a branch of ML were agents learns the right actions to take in order to maximize a positive reward signal by interacting with a dynamic environment (Sutton & Barto, 2018). The three classes of ML are distinguished by the type of feedback the algorithm receives after making an action or prediction. In supervised learning, the feedback indicates whether the agent has made a good or bad prediction based on labeled data. In unsupervised learning, the agent doesn't receive direct feedback since the data isn't labeled; instead, it seeks to identify patterns within the data. RL, however, doesn't rely on a static dataset; instead, it learns through trial and error by interacting with its environment and gathering experiences (Wang et al., 2020). The effectiveness of RL methods has been demonstrated in a wide range of domains, including video games such as Backgammon(Tesauro, 1995) , Chess(Silver et al., 2017) , Go(Silver et al., 2018) , and Atari(Mnih et al., 2015) . RL can also be used to develop efficient policies in various healthcare domains(Yu et al., 2021) , where the decision-making process is often characterized by a prolonged period or a sequential procedure. RL has also found numerous applications in autonomous vehicle decision-making ((Cao et al., 2022) , (Duan et al., 2020), (Kim et al., 2022)), robotics(Zhang & Mo, n.d.), task scheduling(Shyalika et al., 2020), and industry(Andersen et al., 2019).

1.2.3 Reinforcement Learning Framework

Before presenting the mathematical formulation of RL, it is essential to define its key components. According to Sutton and Barto (Sutton & Barto, 2018), an RL system primarily consists of four main elements:

Policy: The policy defines the behavior of the agent at any given time by mapping perceived states of the environment to actions to be taken. This policy can range from a simple lookup table to a complex function that requires extensive computation. Essentially, the policy is the core of the RL agent as it dictates the agent's actions in response to the state of the environment. Policies can be deterministic or stochastic, specifying probabilities for each action.

Reward Signal: The reward signal represents the objective of the reinforcement learning problem. At each time step, the environment provides the agent with a number called the reward, indicating the immediate desirability of the agent's current state and actions. The primary goal of the agent is to maximize the cumulative reward over the long term. Therefore, the reward signal serves as the basis for adjusting the policy; actions that result in low rewards may prompt the agent to modify its policy to choose different actions in similar situations in the future.

Value Function: While the reward signal provides an immediate measure of success, the value function offers a long-term perspective. The value of a state is defined as the total expected reward that an agent can accumulate starting from that state. Unlike rewards, which indicate the short-term desirability of states, values consider the potential future rewards that can be obtained from subsequent states. Accurately estimating values is crucial, as action choices are made based on these value judgments. However, determining values is more challenging than determining rewards, as values must be estimated and refined over time through the agent's experiences.

Model of the Environment: The model of the environment is an optional element that simulates the behavior of the environment, allowing the agent to make inferences about future states and rewards. A model can be used for planning, enabling the agent to consider potential future scenarios before they are actually experienced. Methods that use models for planning are referred to as model-based methods, in contrast to model-free methods that rely purely on trial-and-error learning without explicit planning.

1.2.4 Markov Decision Processes

MDP's (Puterman, 1990) are a class of stochastic sequential decision processes in which the cost and transition functions depend only on the current state of the system and the current action, this property is used in modeling RL (Sutton & Barto, 2018) and other stochastic learning problems.

An MDP is denoted by a tuple (S, A, P, R), where S represents the set of states, A denotes the set of possible actions, and $A(s)$, where $A(s) \subseteq A$, indicates the set of actions that can be applied in a particular state. $P : S \times R \times A \times S \to [0, 1]$ is a deterministic

function that represents the environment's dynamics and defines the probability of ending up in state s' after taking action a in state s and receiving reward r, denoted by $p(s', r \mid s, a)$. $R : S \times A \times S \to \mathbb{R}$ is the function that determines the rewards obtained for transitions between states, denoted by $r(s, a, s')$, and for each triplet (s, a, s'), the function R returns a real value that evaluates the established transition.

RL can be mathematically modeled as an MDP (see Figure 1.1) where the agent and the environment interact with each other at each step of a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$. At each time step t, the agent receives a representation of the current state of the environment, $s_t \in S$, and based on this, selects an action from the set of actions that can be applied in the state s_t, $a_t \in A(s_t)$. On next step $t + 1$, the agent receives a numerical reward $r_{t+1} \in \mathbb{R}$ and moves to a new state, s_{t+1} (Sutton & Barto, 2018).

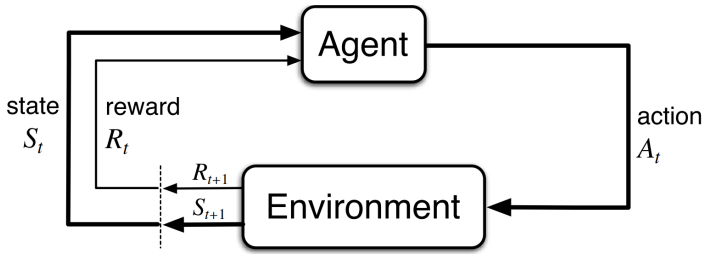

Figure 1.1: The agent–environment interaction in reinforcement learning (Sutton & Barto, 2018)

The agent and the MDP then produces a trajectory represented as follows:

$$s_0, a_0, r_1, s_1, a_1, r_2, \ldots \tag{1.1}$$

The Markov property asserts that a state S_t is a Markov state if and only if $P[S_{t+1} \mid S_t] = P[S_1, \ldots, S_t]$, which means that the future is independent of the past given the present. All states in a MDP are Markov states. This property avoids the need to refer to the agent's history every time a decision is made (Silver, 2020).

In RL, the agent-environment interaction can be divided into two types (Sutton & Barto, 2018).

- **Episodic Interaction**: Where the agent-environment interaction is broken down into subsequences such as game rounds, maze navigation, or other types of repeated interactions. Each episode ends in a terminal state, which is then reset to a standard starting state or a sample from a standard starting state distribution. Even though episodes may end in various ways, such as winning or losing a game, the next episode starts independently of how the previous one ended.
- **Continuous Interaction**: When the agent-environment interaction does not naturally break down into identifiable episodes but continues indefinitely without a fixed limit. For example, this would be the way to formulate a continuous process control task or an application to a long-lived robot.

1.2.5 Objective of Reinforcement Learning

The goal of RL is to use the interactions of the agent with its environment to derive an optimal policy to maximize the total amount of reward received by the agent (Agostinelli et al., 2018). This means that the agent must maximize long-term cumulative reward rather than immediate reward.

More formally, in RL, we aim to maximize the expected return, where the return, denoted by G_t, is defined as the cumulative reward in the simplest case.

This G_t Equation (1.2) formula makes sense in episodic interactions. However, even in episodic interactions, we use the discounted cumulative reward formula Equation (1.3) to give more importance to immediate rewards.

$$G_t = r_{t+1} + r_{t+2} + \cdots + r_T \tag{1.2}$$

To avoid infinite returns (where the agent would play indefinitely to maximize the return) in continuous episodes and to give more importance to immediate rewards than future rewards (for both episodic and continuous interactions), the return is defined as the sum of discounted rewards:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1.3}$$

where γ represents the discount factor, with $0 \leq \gamma \leq 1$, which determines the present value of future rewards. This means that a reward received k time steps in the future is worth only γ^{k-1} times what it would be if received immediately (Sutton & Barto, 2018). If $\gamma = 0$, the agent is myopic and only concerns itself with maximizing immediate rewards. In contrast, if $\gamma \approx 1$, the agent becomes more farsighted and also considers the maximization of future rewards.

1.2.6 Policy and Value function

Most RL methods involve estimating the value function of a given state s, which represents a numerical estimation of the potential future reward that the agent may receive from that state (Agostinelli et al., 2018). This reward depends primarily on the strategy followed by the agent, often referred to as the policy π.

A policy is a mapping between the states of the environment and the probabilities of selecting each possible action. Thus, at time t, if the agent follows policy π, then $\pi(a \mid s)$ is the probability that $a_t = a$ if $s_t = s$, such that $\pi(a \mid s)$ defines a probability distribution over $a \in A(s)$ for each $s \in S$ (Sutton & Barto, 2018). A policy is stochastic if, in a state s, multiple actions can be chosen with a non-zero probability.

The value function of a state s under policy π, denoted by $v_\pi(s)$, is the expected return when starting in state s and following π thereafter:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t \mid s_t = s \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \mid s_t = s \right] \end{aligned} \tag{1.4}$$

for all $s \in S$, where $\mathbb{E}_\pi[\cdot]$ denotes the expected value given that the agent follows policy π, and t is any time step. This value is always zero if the state is terminal. Similarly, the action-value function, denoted by $q_\pi(s, a)$, represents the value of taking action a in state s under policy π, and can be defined as follows:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \mid S_t = s, A_t = a \right] \end{aligned} \tag{1.5}$$

for all $s \in S$, $a \in A$.

A fundamental property of value functions is that they satisfy the following recursive relations:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right] \quad \text{for all } s \in S$$

$$q_\pi(s, a) = \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \right] \quad \text{for all } s \in S$$

These equations, known as the Bellman equations, are involved in all of RL algorithms. Indeed, many RL methods can be viewed as approximately solving the Bellman Optimality Equations.

A policy π is considered better than or equal to another policy π' if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$. Thus, an optimal policy π^* is one that maximizes the expected long-term reward and is defined as follows (Agostinelli et al., 2018):

$$\pi^* = \arg\max_\pi v_\pi(s) \quad \text{for all } s \in S \tag{1.6}$$

$$\tag{1.7}$$

Optimal policies can be multiple, but they share the same value function and action-value function, denoted as follows (Sutton & Barto, 2018):

$$v^*(s) = \max_\pi v_\pi(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v^*(s')] \quad \text{for all } s \in S \tag{1.8}$$

$$q^*(s, a) = \max_\pi q_\pi(s, a) = \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma \max_{a'} q^*(s', a')] \quad \text{for all } s \in S \tag{1.9}$$

1.2.7 Taxonomy Of Reinforcement Learning Algorithms

Several taxonomies of RL algorithms already exist in the literature. In our study, we draw inspiration from the works of (Sutton & Barto, 2018) and (Dong et al., 2020) to propose the following classification (Figure 1.2), providing an overview of typical RL algorithms.

First, we classify RL algorithms into two main categories: model-based methods and model-free methods. The first category is further subdivided based on the availability or learning of the environment model. The second category is also subdivided into three subcategories: value-based approaches, policy search approaches, and actor-critic approaches, which utilize both value functions and policy search.

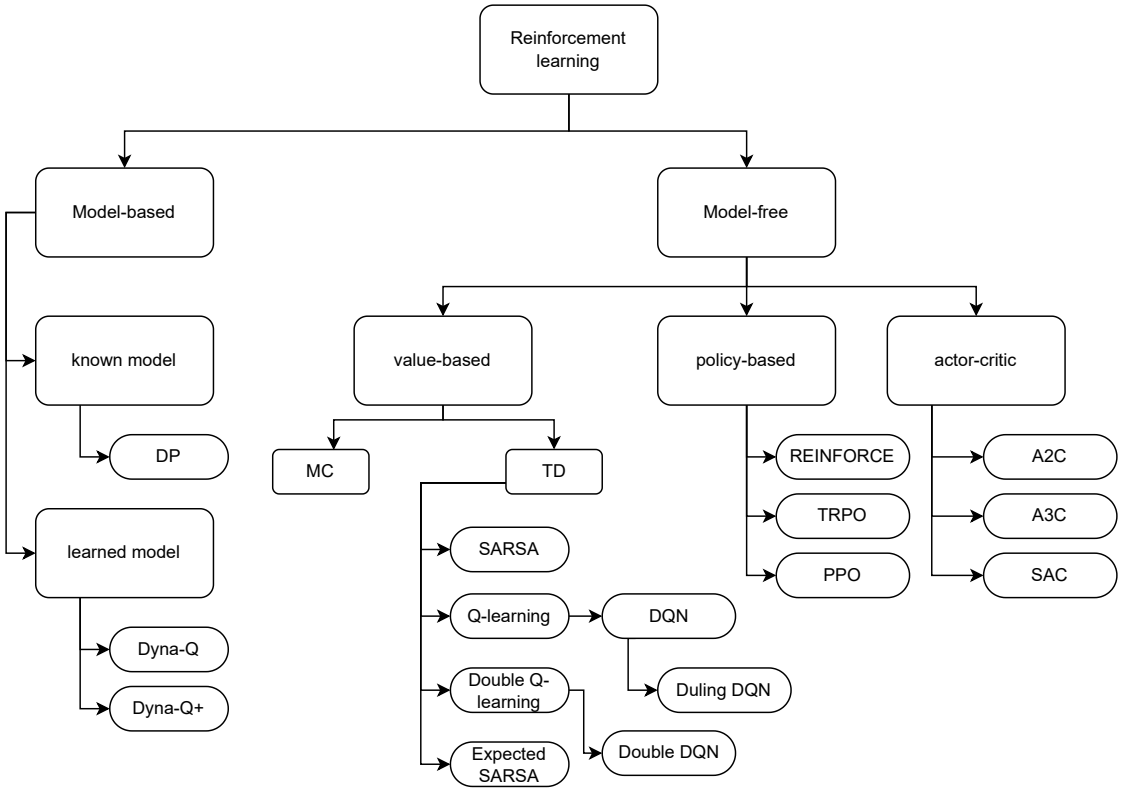

Figure 1.2: Classification of RL Algorithms

The application of RL methods depends on the specific nature of the problem being addressed. Although no definitive comparative study has been conducted, the general properties of these methods can suggest which algorithms may yield more promising results given certain characteristics of the problem, such as the type of episodic interaction (finite or infinite), reward structure, and state space. Table 1.1 summarizes some of the key advantages and disadvantages of the most commonly used RL methods in the literature.

Due to its stability and general applicability across various RL problems, PPO has been widely used as a baseline in numerous research studies involving RL. In the following section, we will provide a brief description of the PPO algorithm.

1.2.7.1 PPO

PPO is a family of policy gradient methods proposed by (Schulman et al., 2017) that alternate between sampling data through interaction with the environment and optimizing a "surrogate" objective function using stochastic gradient ascent. PPO is designed to improve the data efficiency and reliable performance of Trust Region Policy Optimization (TRPO) while being simpler to implement. PPO achieves this by using a clipped probability ratio to form a pessimistic estimate of policy performance, which enables multiple epochs of minibatch updates. PPO has been shown to outperform other online policy gradient methods in tasks like simulated robotic locomotion and Atari game playing, making it a state-of-the-art algorithm for continuous and discrete action spaces.

1.2.8 Conclusion

In this section, we discussed RL, a promising approach in ML that has gained significant success in current research by illustrating its fundamental concepts and presenting a set of its algorithms while providing a comparative study between them to clearly identify their use cases.

Moreover, RL has received much attention in recent years due to its successful application in various fields and real-world scenarios, particularly in Cyber Security to develop defensive policies, which will be the focus of this project.

1.3 Reinforcement Learning Robustness

1.3.1 Introduction

Reinforcement Learning (RL) poisoning is a significant threat to machine learning systems, where adversaries exploit vulnerabilities within RL frameworks to manipulate outcomes. Understanding and mitigating these attacks is crucial for the deployment of RL in critical areas like cybersecurity and autonomous systems.

In this section, we will highlight four types of robustness in reinforcement learning: transition robustness, disturbance robustness, action robustness, and observation robustness. We will explain how each of these approaches works to enhance the security and reliability of RL systems against adversarial attacks.

1.3.2 Defintion

Robust Reinforcement Learning (RL) is a field of study focused on enhancing the resilience of RL algorithms to various forms of uncertainty and adversarial attacks. Robust RL methods aim to improve the stability and performance of RL agents by considering worst-case scenarios during the learning process, thereby ensuring that the learned policies are

effective even in the presence of disturbances or manipulations in the environment (Moos et al., 2022).

1.3.3 Transition Robust

Transition robustness addresses uncertainties in the transition dynamics of Markov Decision Processes (MDPs). The transition function in an MDP defines the probability of moving from one state to another given a specific action. Adversaries can manipulate this transition function, introducing uncertainty that degrades the performance of the RL agent.

To counteract this, transition robust methods define an uncertainty set of possible transition functions. This approach involves creating a model that considers various potential transitions instead of relying on a single, potentially compromised transition function. The key challenge here is balancing robustness with computational tractability. Traditional methods often require the assumption of rectangularity in the uncertainty set, which can lead to overly conservative or pessimistic policies.

Modern approaches aim to relax this assumption, proposing more realistic non-rectangular uncertainty sets or incorporating distributional robustness, where the adversary selects a worst-case distribution over transition functions rather than a single worst-case function. This additional layer of uncertainty helps prevent the agent from adopting overly pessimistic strategies while still maintaining robustness against a wide range of adversarial scenarios. The interatction between the adversary and the protagonist is illustrated in Figure 1.3:

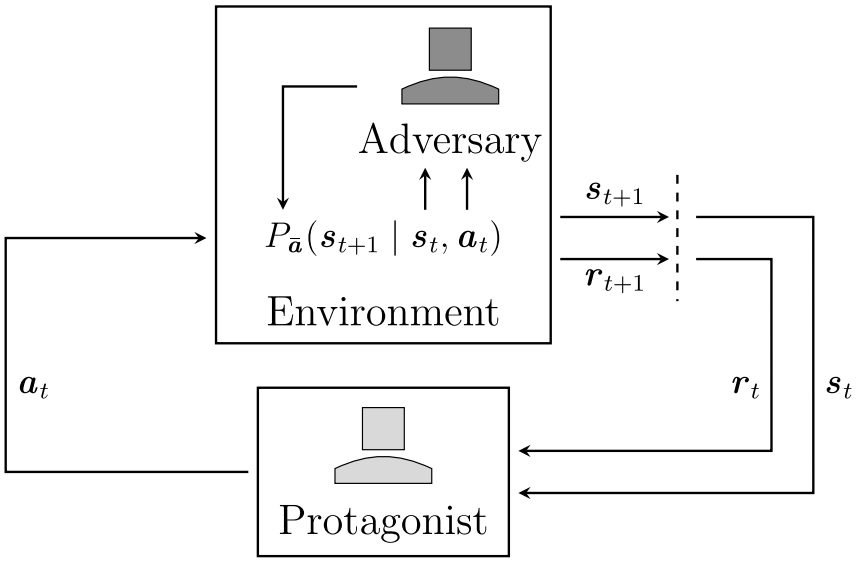

Figure 1.3: Transition Robustness in RL. (Moos et al., 2022)

1.3.4 Disturbance Robust

Disturbance robustness addresses the uncertainty introduced by external forces that affect the system dynamics within an MDP. Rather than focusing on the transition function itself, disturbance robustness considers how unexpected disturbances can impact the environment and the agent's interactions with it.

In this approach, the uncertainty is modeled as external forces acting on the system, which can disrupt the expected behavior of the environment (see Figure 1.4). These forces can represent anything from physical disturbances in robotics to unexpected changes in network conditions in cybersecurity.

The key advantage of disturbance robustness is that it eliminates the need for explicitly defined uncertainty sets, making the modeling process more flexible. However, many of the recent contributions in this area are primarily empirical, lacking strong mathematical guarantees. This limitation suggests that while disturbance robustness offers a useful framework for dealing with external uncertainties, further research is needed to establish more rigorous theoretical foundations and to expand its applicability to a broader range of RL scenarios.

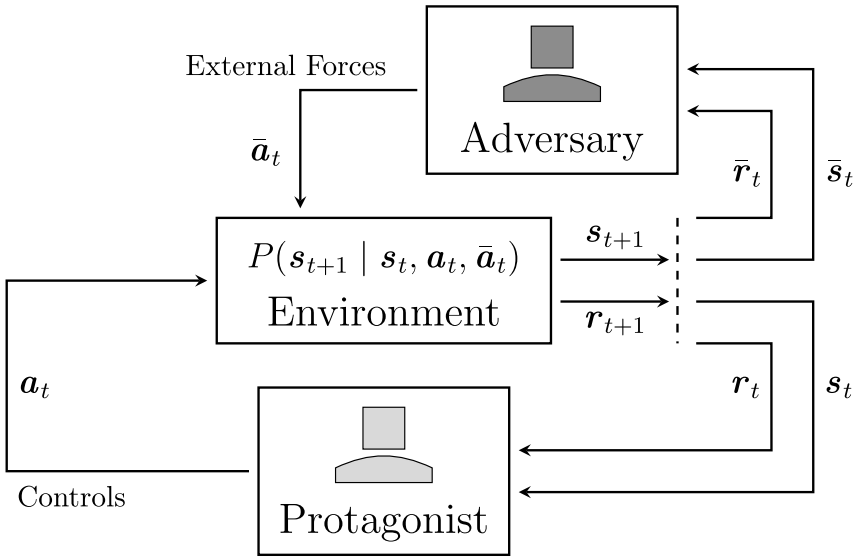

Figure 1.4: Disturbance Robustness in RL. (Moos et al., 2022)

1.3.5 Action Robust

Action robustness focuses on the potential disturbances that can affect an agent's actions within an MDP. These disturbances can take the form of sudden, catastrophic events or continuous perturbations that alter the agent's intended actions.

There are two primary variations of action robustness: probabilistic action robustness and noisy action robustness. Probabilistic action robustness deals with rare but severe disruptions, such as those that might occur in autonomous driving scenarios where a sudden obstacle or system failure could have catastrophic consequences. Noisy action

robustness, on the other hand, considers continuous disturbances that might arise from changes in physical parameters, such as sensor noise or actuator errors.

In both cases, the robustness strategy typically involves defining a joint policy that balances the protagonist's intended actions with the potential for adversarial disruptions. Recent research has extended this concept by adopting techniques from adversarial machine learning, where an adversary intentionally perturbs the actions to degrade performance (see Figure 1.5). These advances in action robustness aim to create agents that can withstand both random and intentional disruptions to their actions, thereby improving their reliability in uncertain environments.

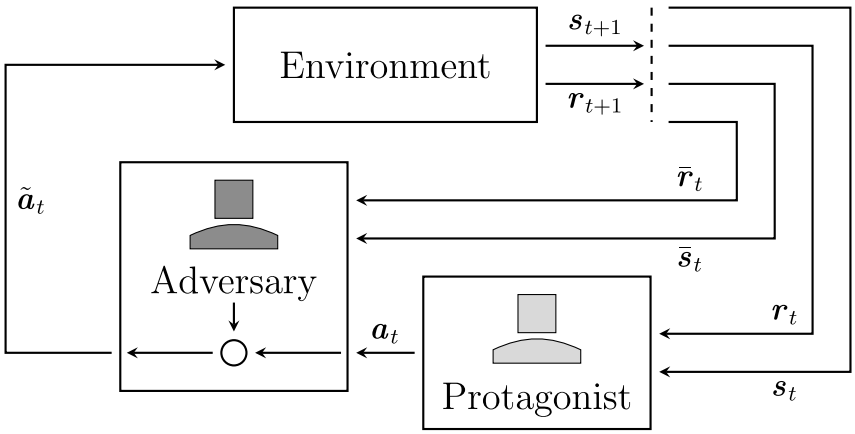

Figure 1.5: Action Robustness in RL. (Moos et al., 2022)

1.3.6 Observation Robust

Observation robustness targets the vulnerabilities in an agent's perception of its environment within an MDP. Since the agent's decisions are based on its observations of the state, any distortion in these observations can lead to incorrect or suboptimal actions.

Adversaries can exploit this vulnerability by manipulating the observations received by the agent, effectively steering it towards unfavorable decisions (see Figure 1.6). This type of robustness is crucial because even minor perturbations in the observations can accumulate over time, leading to significant deviations from the optimal policy.

Traditional approaches to observation robustness often involve directly optimizing the observation or state space to mitigate the impact of adversarial perturbations. Instead of using adversarial RL formulations, some methods derive robust policies using classical RL algorithms, which are then tested against adversarial scenarios to ensure their resilience. Recent work in this area also focuses on providing robustness guarantees and certification bounds, ensuring that the agent can maintain reliable performance even when its observations are under attack.

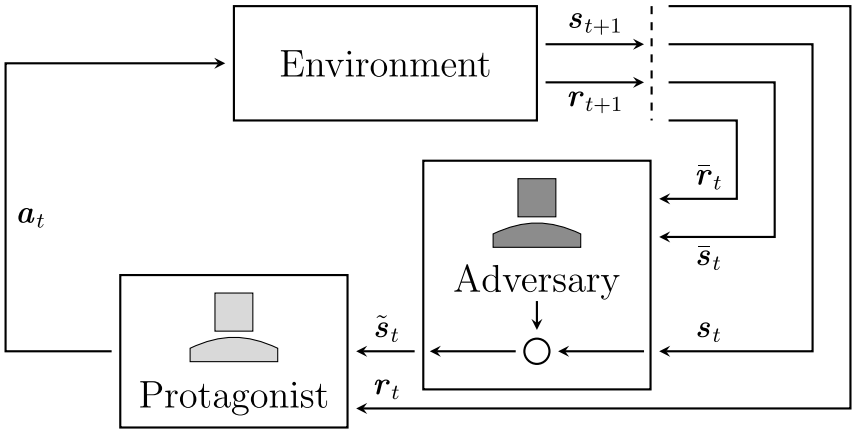

Figure 1.6: Observation Robustness in RL. (Moos et al., 2022)

1.3.7 Conclusion

In this section, we explored the various types of robustness in Reinforcement Learning and how they relate to poisoning attacks. Addressing these challenges is essential for deploying RL systems in environments where security is paramount. Future work may focus on improving these robustness techniques to ensure RL systems remain resilient against increasingly sophisticated attacks.

1.4 Cyber Attacks Types

1.4.1 Introduction

In the ever-evolving landscape of cyber threats, understanding the various types of cyber attacks is crucial for the development of robust defensive strategies. Cyber attacks are malicious attempts to breach the information systems of individuals or organizations. These attacks can have severe consequences, including data theft, financial loss, and disruption of services. This section provides a brief overview of the most common classes of cyber attacks, illustrating the diverse range of threats that cybersecurity professionals must contend with.

1.4.1.0.1 DoS and Distributed Denial of Service (DDoS) Attacks: A DoS attack overwhelms system resources, making them unavailable to users. In a DDoS attack, multiple compromised machines flood the target, causing service disruption. These attacks are challenging to prevent due to the difficulty of distinguishing between legitimate and malicious traffic. Ensuring that the system includes an IDS and DDoS protection products is recommended to mitigate these attacks (Biju et al., 2019).

1.4.1.0.2 Man-in-the-Middle (MitM) Attack: A MitM attack occurs when a third party intercepts communication between a client and server. The attacker impersonates both parties, gaining unauthorized access to sensitive information. Using secure connections, such as a Virtual Private Network (VPN), can help prevent MitM attacks by adding an additional layer of security (Biju et al., 2019).

1.4.1.0.3 Phishing Attacks: Phishing attacks involve sending fraudulent emails that appear to come from trusted sources, aiming to steal personal and credential information. This type of attack is a form of social engineering that tricks users into revealing sensitive data. Awareness and critical thinking are key in reducing the risk of phishing attacks (Biju et al., 2019).

1.4.1.0.4 Drive-by Download Attack: A drive-by download attack occurs when a user inadvertently downloads malicious software simply by visiting a compromised website. To prevent this, it is important to regularly update software, use firewalls, and limit browsing with privileged accounts (Biju et al., 2019).

1.4.1.0.5 Password Attack: Password attacks are common methods used by attackers to gain unauthorized access by obtaining or decrypting user passwords. Techniques include brute force, dictionary attacks, and sniffing. Strong password practices, such as using complex passwords and changing them frequently, can help defend against these attacks (Biju et al., 2019).

1.4.1.0.6 Structured Query Language (SQL) Injection Attack: SQL injection attacks exploit vulnerabilities in database queries to gain unauthorized access to sensitive information. Input validation and the use of web application firewalls are effective in preventing SQL injection attacks (Biju et al., 2019).

1.4.1.0.7 Cross-Site Scripting (XSS) Attack: XSS attacks involve injecting malicious scripts into trusted websites, which then execute in the user's browser. These attacks can be mitigated through encoding and validation techniques that prevent the execution of malicious code (Biju et al., 2019).

1.4.1.0.8 Eavesdropping Attack: Eavesdropping, or sniffing, involves intercepting data as it is transmitted over a network. This attack is difficult to detect and can be prevented by using encryption and avoiding insecure networks for transmitting sensitive information (Biju et al., 2019).

1.4.1.0.9 Birthday Attack: A birthday attack is a type of cryptographic attack that exploits the probability theory concept known as the birthday problem. It targets hash functions to find collisions and compromise the integrity of digital signatures (Biju et al., 2019).

1.4.1.0.10 Malware Attack: Malware attacks involve installing malicious software on a user's device without their consent. Types of malware include viruses, worms, trojans, ransomware, and spyware, each with different methods of compromising user data and system integrity. Regular updates, antivirus software, and cautious online behavior are essential for preventing malware attacks (Biju et al., 2019).

1.4.2 Conclusion

In this section, we have provided an overview of the most common types of cyber attacks, highlighting the wide array of threats that cybersecurity professionals must address. Understanding these attacks is fundamental for developing effective defense mechanisms. The next section will focus on IDS, which play a crucial role in detecting and mitigating these cyber threats.

1.5 Intrusion Detection Systems

1.5.1 Introduction

To detect intrusions, security experts conventionally need to observe and examine audit data, e.g., application traces, network traffic flow, and user command data, to differentiate between normal and abnormal behaviors. However, the volume of audit data surges rapidly when the network size is enlarged. This makes manual detection difficult or even impossible. An IDS is a software or hardware platform installed on host computers or network equipment to detect and report to the administrator abnormal or malicious activities by analysing the audit data. An intrusion detection and prevention system may be able to take appropriate actions immediately to reduce impacts of the malevolent activities (Nguyen & Reddi, 2021).

The most common way to classify IDSs is to group them by information source. Some IDSs analyze network packets, captured from network backbones or LAN segments, to find attackers. Other IDSs analyze information sources generated by the operating system or application software for signs of intrusion (Figure 1.7).

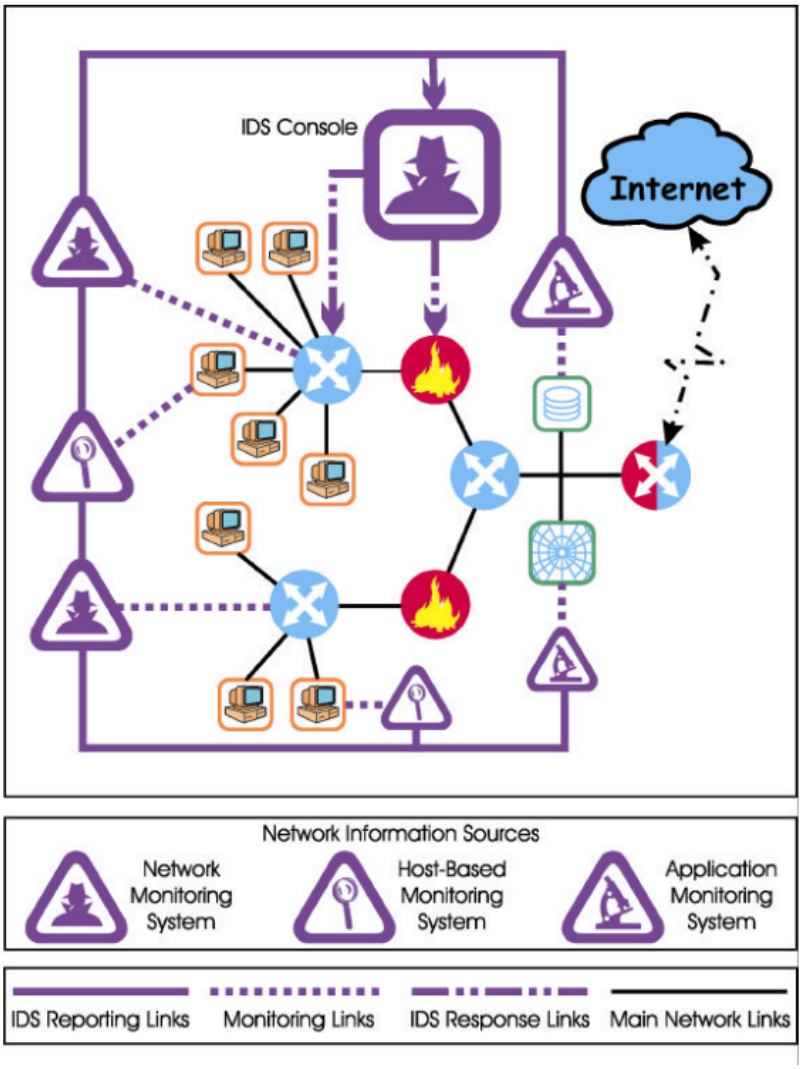

Figure 1.7: Centralized IDS deployment in a network architecture (Bace, Mell, et al., 2001)

In this section se will introduce the diffrent types of IDS and talk about the advantage and disadvatage of each type.

1.5.2 Definition

Intrusion Detection Systems are tools that automate the process of analyzing the infrastructure activity to find possible threats caused by attackers trying to access resources they are not authorized to, or harm the system.(Bace, Mell, et al., 2001)

1.5.3 Types of Intrusion Detection Systems

We can distinguish different types of IDS by there source of information, the analysis type and the response.

1.5.3.1 Classifying IDS by Source of events

1.5.3.1.1 Network-based Most IDS operate by analysing events comming from the network, we call those tools Network-based IDS. Listening on a network segment or switch, one network-based IDS can monitor the network traffic affecting multiple hosts that are connected to the network segment, thereby protecting those hosts. Network-based IDSs can be in the form of sensors or hosts deployed in the network, they observe the network activity looking for possible intrusion traces. As the sensors are designed to be less exposed to attacks.

Network-based IDSs offer several advantages for monitoring large networks. They can effectively cover extensive areas of the network with only a few strategically placed devices. Since they are typically passive, they don't interfere with the network's regular operation, making it easy to integrate them into existing infrastructure with minimal disruption. Additionally, these systems can be highly secured, often remaining undetectable to attackers, which enhances their effectiveness in protecting the network (Bace, Mell, et al., 2001).

On the other hand, network-based IDSs face several challenges, particularly in handling large volumes of network traffic, which may cause them to miss attacks during periods of high activity. While some vendors are addressing this with hardware-based solutions, this approach may still limit the range of attacks detected. Furthermore, modern switch-based networks reduce the effectiveness of network-based IDSs, as they limit monitoring capabilities to specific segments or hosts. These systems also cannot analyze encrypted data, a growing concern with the increased use of VPNs. Additionally, network-based IDSs often cannot confirm the success of an attack, requiring manual follow-up. Packet fragmentation can also cause instability in some IDSs, leading to crashes (Bace, Mell, et al., 2001).

1.5.3.1.2 Host-based Host-based IDSs operate on information collected from within an individual computer system, providing a reliable and precise analysis of activities by determining the specific processes and users involved in an attack on the operating system. Unlike notwork-based IDSs, host-based IDSs can observe the outcome of an attempted attack as they can directly monitor data files and system processes. They utilize information sources such as operating system audit trails and system logs, with audit trails offering detailed and protected data, while system logs are smaller and easier to comprehend.

The primary advantages of host-based IDSs include their ability to detect attacks that network-based IDSs cannot, operate in encrypted environments, and remain unaffected by switched networks. Additionally, when using operating system audit trails, they can detect integrity breaches, such as Trojan Horse attacks.

However, host-based IDSs are more challenging to manage since they require configuration for each monitored host. They can also be targeted and disabled during an attack, struggle to detect network-wide scans, and can be disabled by certain DoSs attacks. Furthermore, using operating system audit trails can lead to immense data storage needs, and host-based IDSs can impact the performance of the systems they monitor.

1.5.3.1.2.1 Application-based Application-based IDS are a specialized subset of host-based IDSs designed to analyze events occurring within a specific software application, typically using the application's transaction log files as a primary information source. These systems have the advantage of interfacing directly with the application, allowing for significant domain or application-specific knowledge to be incorporated into the analysis. This enables them to detect suspicious behavior, especially when authorized users exceed their permissions, by monitoring the interaction between the user, data, and the application.

One of the main advantages of application-based IDSs is their ability to monitor user-application interactions closely, which can help trace unauthorized activities back to individual users. Additionally, they can function effectively in encrypted environments since they operate at transaction endpoints, where data is presented in an unencrypted form, ensuring that they can still perform their monitoring tasks without being hindered by encryption.

However, there are also notable disadvantages to application-based IDSs. They may be more vulnerable to attacks compared to host-based IDSs because application logs are generally less protected than the operating system audit trails used by host-based systems. Additionally, because they monitor events at the user level of abstraction, application-based IDSs typically cannot detect software tampering attacks like Trojan Horses. Therefore, it is recommended to use them in conjunction with host-based and/or network-based IDSs to ensure comprehensive protection.

1.5.3.2 Event Analysis

We can distinguish event analysis approach in IDSs to anomaly-based and signature-based. Anomaly detection focuses on detecting abnormal behaviors in the system, while signature-based analysis compares the current activity of the system to a predefined set of activities that are considered as malicious.

1.5.3.2.1 Anomaly-based Anomaly detectors identify unusual behavior, or anomalies, within a host or network by assuming that such deviations signify potential attacks. These systems work by constructing profiles of normal behavior from historical data, and then using various measures, such as threshold detection and statistical analysis, to identify deviations from these norms.

One of the key advantages of anomaly detection is its ability to detect new or unknown attack forms that signature-based systems might miss. However, this method often generates a large number of false alarms due to the variability in what constitutes "normal" behavior. Despite this drawback, anomaly detection is a valuable tool for identifying new threats and can complement misuse detectors by providing them with data that can be incorporated into detection signatures.

While anomaly detection is not widely used in commercial IDSs, it is an active area of research and may become more prominent in future systems. Currently, it is mostly employed in detecting network or port scanning, but its potential for broader application

continues to be explored.

1.5.3.2.2 Signature-based Signature-based detectors, or misuse detectors, analyze system activities by searching for specific patterns that match known attacks, known as signatures. These detectors are efficient in identifying attacks, as they quickly diagnose the use of specific tools or techniques. This feature allows security managers to prioritize and respond effectively to threats, even if they have limited security expertise.

However, one of the key disadvantages of signature-based detection is its inability to identify new or unknown attacks. To remain effective, these detectors require constant updates with the latest attack signatures. Additionally, many commercial misuse detectors use highly specific signatures, limiting their ability to detect variations of known attacks. Although state-based detection techniques can address this limitation, they are not widely used in commercial products.

Signature-based detection is reliable for recognizing known threats but is limited by its reliance on predefined signatures, necessitating frequent updates and sometimes missing new or modified attack methods.

1.5.3.3 Response

After analyzing event data, IDSs generate responses, which can range from reporting findings to taking automated actions. The effectiveness of these response functions is crucial, even if they are sometimes undervalued. Commercial IDSs typically offer a mix of active and passive responses to appropriately address detected threats.

1.5.3.3.1 Active Active IDS responses involve automated actions when specific types of intrusions are detected. These responses fall into three main categories (Bace, Mell, et al., 2001):

- **Collecting Additional Information:** This response gathers more data to clarify whether an attack is occurring. For example, increasing the sensitivity of system logs or network monitors can provide critical details, aiding in diagnosing the attack and potentially supporting legal actions.
- **Changing the Environment:** This includes halting ongoing attacks and preventing further access by blocking the attacker's IP address, re-configuring routers and firewalls, or severing connections to stop the intrusion.
- **Taking Action Against the Intruder:** Although some advocate for counterattacks, this approach is risky and potentially illegal. It can escalate the situation and harm innocent parties due to the common use of false network addresses by attackers. Legal advice is recommended before considering such actions.

1.5.3.3.2 Passive Passive IDS responses involve providing information to system users, allowing them to take necessary actions based on the data provided. Many commercial IDS primarily rely on passive responses, which are often categorized into alarms, notifications, and SNMP traps or plug-ins.

- **Alarms and Notifications:** IDS generate alarms and notifications to inform users of detected attacks. These alarms can vary from simple onscreen alerts to detailed messages containing the IP addresses of the attacker and target, the attack method used, and the result. Organizations can also configure IDS to send alerts to remote devices such as cell phones or pagers of the incident response team, though email notifications are less advisable due to the risk of attackers intercepting them.
- **Simple Network Management Protocol (SNMP) Traps and Plug-ins:** Some commercial IDS utilize SNMP traps to report alarms and alerts to network management systems, which can be beneficial for adapting the network infrastructure to respond to detected attacks. This approach allows for the processing load of active responses to be handled by systems other than the one targeted, using common communication channels.

1.5.3.3.3 Reporting and Archiving Many, if not all, commercial IDS provide capabilities to generate routine reports and other detailed documents. Some can output reports of system events and intrusions detected over a particular reporting period (for example, a week or a month). Some also provide statistics or logs generated by the IDS in formats suitable for inclusion in database systems or for use in report-generating packages (such as Crystal Reports).

1.5.4 Conclusion

Conclusion: This section has outlined the various types of IDS and their importance in Cyber Security. We discussed how IDSs automate the process of monitoring and analyzing infrastructure activity to detect potential threats. The distinction between different types of IDSs—whether network-based, host-based, or application-based—demonstrates the diverse approaches available for securing systems. In our study, the focus on active intrusion detection systems is particularly important as these systems not only detect intrusions but also autonomously deploy defensive actions in real-time, which is crucial for responding to sophisticated and rapidly evolving cyber threats.

1.6 Conclusion

In this section, we explored the various types of robustness in RL and how they relate to poisoning attacks. Addressing these challenges is essential for deploying RL systems in environments where security is paramount. Future work may focus on improving these robustness techniques to ensure RL systems remain resilient against increasingly sophisticated attacks.

Algorithm	Advantages	Disadvantages
DP	• Requires a model to calculate value functions. • Provides an exact solution for problems with a known model.	• Needs a complete model of the environment, which may not always be available. • Computationally expensive for large state spaces.
MC	• Does not require a model of the environment. • Learns from experience directly by sampling.	• Has to wait for the entire episode to finish before learning. • High variance in value estimates.
SARSA	• Learns on-policy, meaning the agent learns values for the policy it follows. • Adapts to changes in the environment.	• Sensitive to exploration strategy (e.g., ε-greedy). • Can result in suboptimal policies if not properly tuned.
Q-learning	• Off-policy algorithm, learns optimal policy regardless of the agent's actions. • Can be applied to various problems without requiring a model.	• Struggles with high-dimensional state spaces. • Can overestimate the value function.
DQN	• Handles high-dimensional state spaces using neural networks. • Achieves better generalization compared to traditional Q-learning.	• Overestimates action values due to the use of a single network for action selection and evaluation.
DDQN	• Solves the overestimation problem in DQN by separating action selection and evaluation.	• Does not work well in environments with continuous action spaces.
Dueling DQN	• Separates state-value and action-advantage functions, allowing the model to distinguish between useful and irrelevant actions.	• Can still struggle with continuous action spaces. • Requires more computational resources.
PPO	• State-of-the-art algorithm for reinforcement learning. • Optimizes a policy directly, making it highly effective for continuous and discrete action spaces.	• Requires careful tuning of hyperparameters. • Slower training compared to simpler algorithms.
AC	• Combines policy optimization and value function learning. • Outputs both policy and value function, improving stability.	• More complex to implement than value-based methods. • Prone to instability if not tuned properly.

Table 1.1: General Comparision of RL Algorithms

Chapter 2

Reinforcement Learning for Intrusion Detection Systems

2.1 Introduction

In the previous chapter, we provided a background on reinforcement learning as a method within machine learning used to train policies that generate sequential actions. Due to this particular property, it has garnered interest from machine learning researchers in the field of cybersecurity.

In this chapter, we will explore the litterature on a specific area of cybersecurity aimed at securing large networks from intrusions automatically, which involves the development of IDS. We will cover its definition, types, and under each type of IDS, we will mention some reinforcement learning applications from the literature.

2.2 Reinforcement Learning in Active Intrusion Detection Systems

Researchers are increasingly focused on developing active IDS that can automate responses to common attacks, such as DoS or DDoS attacks (Phan & Bauschert, 2022). The development of active IDS requires optimizing decision-making processes based on observations of network events, ensuring that actions taken do not compromise efficiency or introduce new vulnerabilities.

Reinforcement Learning (RL) has been widely applied to enhance the capabilities of Intrusion Detection Systems (IDS) in cybersecurity. Traditional RL methods, such as Q-learning and SARSA, have been used to address various aspects of IDS, from host-based to network-based systems. Recent advancements in Deep Reinforcement Learning (DRL) have further propelled the effectiveness of IDS by combining the strengths of deep learning with RL, allowing for better handling of complex and high-dimensional environments. For instance, (**Malialis2015**) introduced a multi-agent RL approach to mitigate

Distributed Denial of Service (DDoS) attacks by learning optimal traffic throttling strategies. Additionally, (**Bhosale2017**) proposed a multi-agent intelligent system using RL to respond to sophisticated network attacks quickly. More recent work, such as that by (**Shamshirband2020**), has employed DRL to enhance detection accuracy in wireless sensor networks by integrating game theory and fuzzy logic into the learning process. Overall, the integration of RL and DRL into IDS has shown significant promise in improving the detection and response capabilities of cybersecurity systems, although challenges such as scalability and adaptability remain areas of ongoing research. In the following section we will do a case study where we will explain two approachs from the literature of modeling cybersecurity problems for reinforcement learning.

2.2.1 Case Study

A key aspect of designing defensive policies using RL lies in accurately modeling the attack scenario within the network infrastructure to effectively train the agent. In this section, we introduce two approaches that utilize RL to enhance network security. The first approach, DeepAir (Phan & Bauschert, 2022), presents a framework combining RL and SDN to enhance the efficiency of the Intrusion Response System (IRS). The second approach, presented by (Hammar & Stadler, 2024), explores a game-theoretic model that enables the development of defender strategies, leveraging signals from an Intrusion Detection and Prevention System (IDPS) to defend against dynamic attackers. We will briefly discuss their contributions in the following sections.

2.2.1.1 DeepAir: RL for Software-Defined Networks

In his work, the authors designed a deep-reinforcement learning agent assisting a machine learning based IRS to choose defensive actions from the list in 2.1

Policy	Notation	Meaning
blockIP-30secs	a_1	Drop all incoming packets with the attacker's IP address for 30 seconds
blockIP-1min	a_2	Drop all incoming packets with the attacker's IP address for 1 minute
blockIP-3min	a_3	Drop all incoming packets with the attacker's IP address for 3 minutes
blockIP-5min	a_4	Drop all incoming packets with the attacker's IP address for 5 minutes
limitRate-25%	a_5	Reduce the rate of incoming packets from the attacker by 25% compared to the current rate
limitRate-50%	a_6	Reduce the rate of incoming packets from the attacker by 50% compared to the current rate
limitRate-75%	a_7	Reduce the rate of incoming packets from the attacker by 75% compared to the current rate
removeMaliciousFlows	a_8	Remove all abnormal traffic flow rules at SDN switches on the attack path
toHoneyPot	a_9	Redirect the attack traffic flows to a predefined honeypot
reRoute	a_{10}	Redirect the attack traffic flows to another path in the data plane
doNothing	a_{11}	Do nothing

Figure 2.1: Response Policies for DoS attacks (Phan & Bauschert, 2022)

this action is then passed to the response policy placement entity that will deploy a new flow policy in the network, this new flow rule aims to maximize the attack defense performance while minimizing the negative impact on the benign traffic forwarding performance and the policy deployment cost.

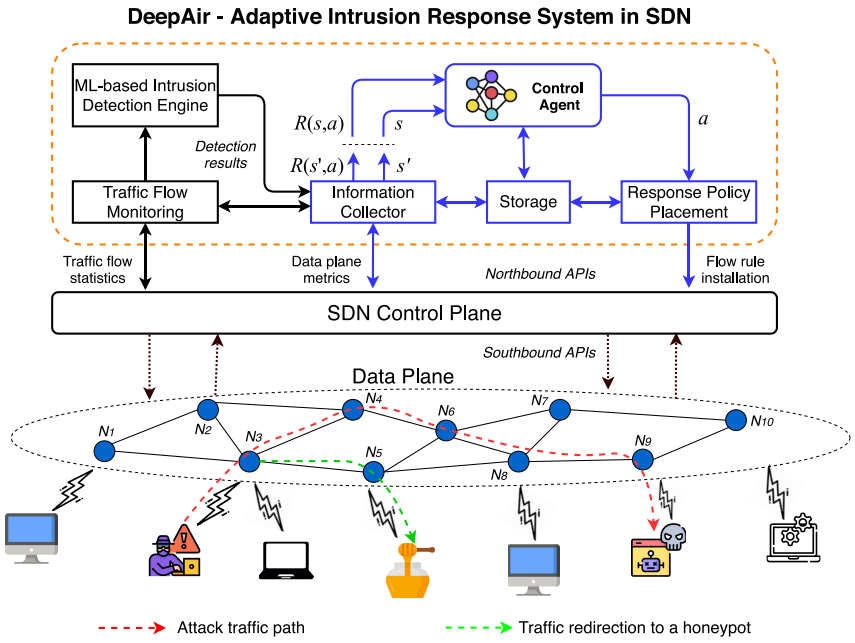

Figure 2.2: DeepAir framework in an SDN environment (Phan & Bauschert, 2022)

Figure 2.2 Illustrates 2 traffic flow options, the attack traffic path passes by multiple nodes in the network before reaching the target victim, causing exhaustion of the network traffic and damage to a critical end node. While the traffic redirection to a honeypot path is the new flow rule deployed by the agent as a defensive strategy after observing the attack damage, this results in less traffic on the network and forwards the attack to a honeypot (Phan & Bauschert, 2022).

2.2.1.2 Game-Theoretic RL for Dynamic Attackers

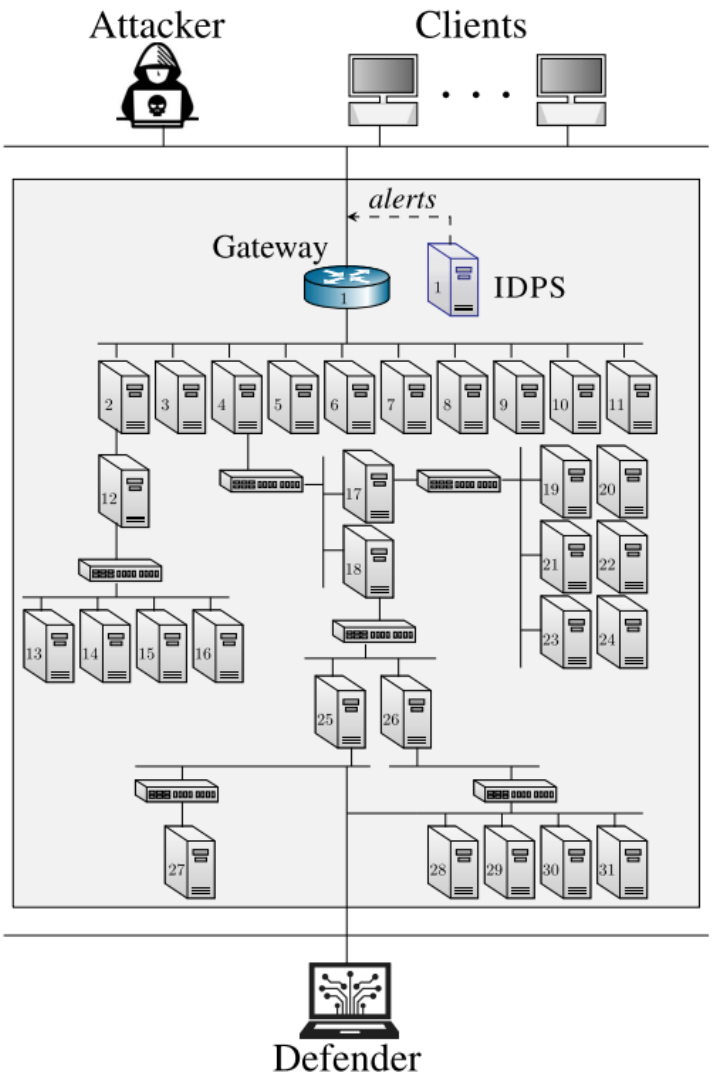

Figure 2.3: The IT infrastructure and the actors in the intrusion response game (Hammar & Stadler, 2024)

In this use case, the defender, responsible for the IT infrastructure of an organization, must protect servers while maintaining services to clients. An attacker, with the goal of compromising servers, launches intrusions by exploiting vulnerabilities while trying to evade detection. The defender uses a response system to monitor and respond to potential threats. Defensive actions, such as blocking traffic or limiting access, are taken based on the severity of IDPS alerts. The defender's strategy must balance two goals: maintaining service availability and stopping attacks at the lowest possible cost. The key challenge is identifying the right moment to initiate defensive measures without causing unnecessary disruption to client services (Hammar & Stadler, 2024).

2.2.2 Summary of RL Applications in IDS

This table 2.1 summarizes various applications of Reinforcement Learning (RL) in Intrusion Detection Systems (IDS), detailing the goals/objectives, the algorithms employed, the states, actions, and rewards associated with each application. The information is gathered from various studies in the literature, and it highlights the diverse approaches used to enhance the effectiveness and robustness of IDS.

Applications	Goals/Objectives	Algorithms	States	Actions	Rewards
Distributed IDPS using Multi-agent RL (Malialis & Kudenko, 2015)	Mitigate DDoS attacks by optimizing traffic throttling strategies	SARSA	Network traffic patterns	Adjust traffic thresholds	Reduction in attack traffic
Multi-agent RL for IDS (Bhosale et al., 2014)	Quick response to sophisticated network attacks	Q-learning	Network state	Deploy defensive actions	Successful attack mitigation
Game theoretic approach for IDPS in WSNs (Shamshirband et al., 2014)	Enhance detection accuracy using game theory and fuzzy logic	DRL	Sensor readings	Adjust sensor parameters	Improved detection rate
DeepAir (Phan & Bauschert, 2022)	Optimize IRS in SDN environments	DRL	Network topology	Modify network flow rules	Minimized attack impact
Optimal Stopping for Intrusion Response (Hammar & Stadler, 2024)	Determine optimal time to deploy defensive actions	DRL	Network state	Trigger defensive action	Maximized service continuity

Table 2.1: Summary of Reinforcement Learning Applications in Intrusion Detection Systems

2.3 Conclusion

As we have explored throughout this chapter, reinforcement learning (RL) presents a significant advantage for enhancing cybersecurity measures. By employing RL, cybersecurity systems gain the ability to adapt dynamically to evolving threats, making them exceptionally effective against complex and ever-changing cyber attacks. The applications of RL in developing sophisticated Intrusion Detection Systems (IDS) exemplify its potential

to not only detect threats but also to learn and improve response strategies over time.

Further research and development in this area promise to yield more robust cybersecurity tools that can predict, detect, and respond to threats with minimal human intervention. The intersection of RL and cybersecurity is still a burgeoning field, offering ample opportunities for innovation and improvement. As threats become more sophisticated, the integration of advanced machine learning techniques like RL into cybersecurity solutions will be crucial for safeguarding sensitive information and maintaining the integrity of network infrastructures.

Chapter 3

Our Work

3.1 Introduction

In this chapter, we detail the work conducted throughout the internship, covering the tools and frameworks used, the proposed approach, and the implementation of the system.

The first section outlines the tools and technologies employed for the development of the solution, such as Python, VS Code, and the CSLE framework. These tools formed the backbone of our system, providing the necessary infrastructure for experimentation and training.

The second section delves into the proposed approach, explaining the structure of the environment, the reward setup, and the reinforcement learning (RL) specifications. We present the defensive and poisoning strategies used to modify the environment and guide the RL agent during training.

Lastly, the implementation section covers the codebase and the integration of the strategies into the CSLE framework. This includes a detailed explanation of the *PoisonedEnvWrapper*, *MetaAgent*, and the different poisoning strategies that were implemented. We also demonstrate how these strategies were integrated into the experimentation pipeline using the PPO agent for training and evaluation.

3.2 Tools and Frameworks

In this project, several tools and frameworks were employed to implement and evaluate the reinforcement learning (RL) policies in the context of cybersecurity. Below is a summary of the key tools:

- **Python:** Python was used as the primary programming language due to its wide range of libraries and frameworks that are well-suited for RL applications. Its rich ecosystem, particularly libraries like TensorFlow, PyTorch, and Gym, provided the

necessary tools for both policy training and evaluation.

- **Visual Studio Code:** For the development environment, Visual Studio Code (VS Code) was used. Its integrated debugging, Git management, and extensions for Python programming made it an ideal editor for the project's development cycle.
- **CSLE:** The CSLE (Limmen, 2020) was the core framework used for training the policies. CSLE is a modular framework designed specifically for reinforcement learning in cybersecurity contexts (see Figure 3.1). It includes built-in components that facilitate the creation of simulation environments for cybersecurity attack-defense scenarios. Additionally, CSLE provides a management system that supports model training, logging, and evaluation. This system was crucial for managing experiments, tracking performance metrics, and visualizing the learning progress of the trained agents.

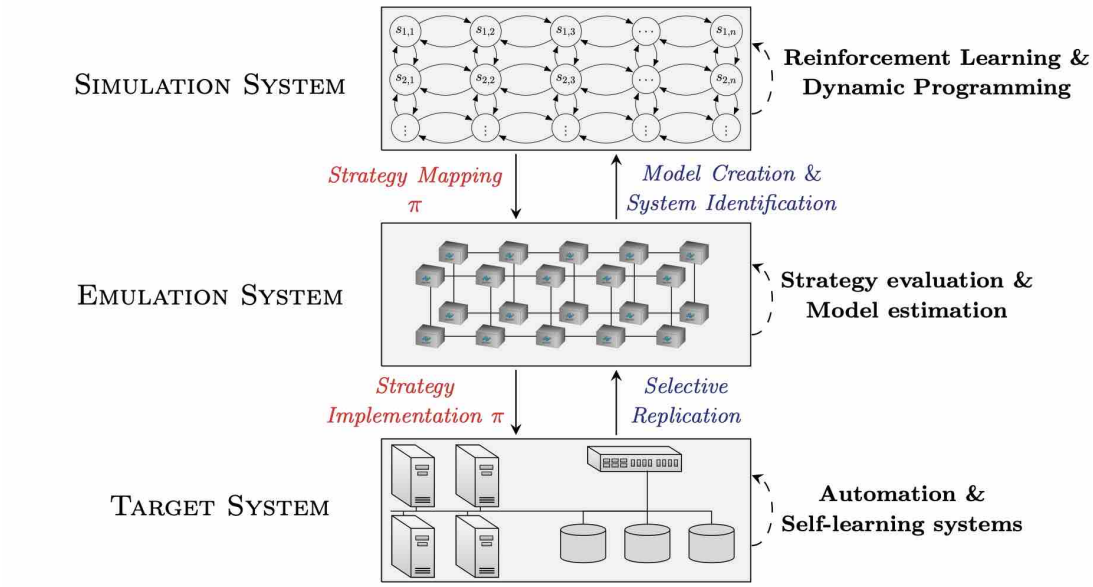

Figure 3.1: CSLE framework architecture (Limmen, 2020)

3.3 Approach

In this section, we present the details of the approach proposed for enhancing the robustness and effectiveness of RL-based intrusion detection systems (IDS). This approach integrates dynamic belief state modeling, reward setting, and action spaces with novel poisoning and defensive strategies. The core of this work revolves around the integration of *Poisoned Environment Wrapper* and *MetaAgent Defensive Strategy* within the CSLE environment.

3.3.1 Belief Space and Its Evolution

The belief space in the CSLE framework represents the defender's belief about the current game state. This belief is a probabilistic value, b_t, that indicates the defender's level of certainty regarding whether the system is under attack. The belief space B is defined as the interval $[0, 1]$, where:

$$b_t(1) = P[S_t = 1 \mid H(D)_t], \quad b_t(0) = 1 - b_t(1)$$

Here, S_t denotes the current state (0: safe, 1: under attack) and $H(D)_t$ represents the defender's history of observations and actions until time step t. The evolution of the belief state, b_t, is governed by the update equation:

$$\begin{aligned} b_t(s_t) = C \sum_{s_{t-1} \in S} \sum_{a_{t-1}^{(A)} \in A_A} & b_{t-1}(s_{t-1}) \pi_A \left(a_{t-1}^{(A)} \mid s_{t-1}, b_{t-1} \right) \\ & \cdot \mathcal{Z} \left(o_t, s_t, \left(a_{t-1}^{(D)}, a_{t-1}^{(A)} \right) \right) \\ & \cdot \mathcal{T} \left(s_t \mid s_{t-1}, \left(a_{t-1}^{(D)}, a_{t-1}^{(A)} \right) \right) \end{aligned} \tag{3.1}$$

where C is a normalization constant ensuring that the belief remains a valid probability, $\mathcal{Z}$ represents the observation operator and $\mathcal{T}$ the transition operator. The defender's belief is updated based on past actions, observations, and the defender's and attacker's strategies. This belief update process is critical as it influences the decisions made by the defender (the RL agent) in response to perceived threats.

3.3.2 Reward Setting and Calculation

The reward setting is central to training a robust defense policy. In the CSLE environment, the defender's objective is to minimize the length of successful intrusions while maintaining the integrity and availability of the system. The reward vector $\vec{R}$ is structured as follows:

$$\vec{R} = [R_{\text{INT}}, R_{\text{COST}}, R_{\text{SLA}}, R_{\text{ST}}]$$

- R_{INT}: Penalty for successful intrusions, - R_{COST}: Cost of executing defensive actions, - R_{SLA}: Penalty for service degradation, - R_{ST}: Reward for successfully stopping an attack.

This reward structure allows the defender to balance multiple objectives: preventing intrusions, minimizing operational costs, and maintaining service-level agreements SLA. The reward values are adjusted during training based on the evolving performance of the defender, specifically intrusion length metrics.

3.3.3 Action Space

The defender's actions are limited to two key options:

- **Stop**: The defender initiates actions to stop the intrusion, which might involve cutting off network access, blocking malicious traffic, or taking other preventive measures.
- **Continue**: The defender decides to monitor the situation without taking immediate actions, allowing the intrusion to evolve while gathering more data for future defensive measures.

The simplicity of this action space reflects the importance of timing in defense decisions. The defender must weigh the costs and potential risks of acting too soon or waiting too long.

3.3.4 Poisoned Environment and Defensive Strategies

3.3.4.1 PoisonedEnvWrapper

The `PoisonedEnvWrapper` class is a key component of our approach, enabling the environment to introduce adversarial conditions (poisoning) into the defender's learning process. This wrapper class extends the base environment provided by CSLE, and integrates a poisoning strategy through an `EnvPoisoning` interface.

The poisoning strategies distort the defender's observations, rewards, and belief state during interactions with the environment. The `PoisonedEnvWrapper` ensures that during each environment step, the poisoning strategy can influence the outcome, increasing the challenge faced by the defender during training. The wrapper tracks iterations and passes control to the poisoning strategy at each time step.

3.3.4.2 Defensive Strategies: MetaAgent

The defender's strategy is encapsulated in the `MetaAgent`, which dynamically adjusts the reward vector $\vec{R}$ based on performance metrics. The defender's goal is to reduce the average intrusion length, a critical performance indicator. The `MetaAgent` monitors the progress of the defender's strategy using an `IntrusionLengthTracker` and updates the reward structure every N iterations.

Two specific defensive strategies are explored:

- **AdaptiveRewardMetaAgent**: Adjusts the reward vector based on the absolute change in intrusion lengths. If intrusion lengths are decreasing, the reward for stopping attacks increases.
- **DiscountedAdaptiveRewardMetaAgent**: Introduces a discount factor c into the reward updates, balancing the trade-off between immediate and long-term rewards based on changes in intrusion lengths.

These strategies provide the defender with a mechanism to adjust its behavior during training, leading to more refined and effective policies.

3.3.4.3 Poisoning Strategies

Poisoning strategies are implemented through the `EnvPoisoning` class, which provides adversarial modifications to the environment. We explore two poisoning strategies:

- **DummyThresholdPoisoning**: A baseline strategy that allows the defender to train without significant poisoning but serves as a reference point for the more sophisticated strategies.
- **RandomBeliefPadding**: This strategy randomly increases the defender's belief in the presence of an attack, distorting the belief state and potentially leading to suboptimal defensive actions.

By integrating these poisoning strategies, we simulate a more hostile training environment, where the defender must adapt to unpredictable and deceptive conditions.

3.3.5 Summary

In this section, we detailed the core elements of our proposed approach, including the belief space, reward structure, action space, and the integration of poisoning and defensive strategies. These components work together to train a robust defender that can operate in dynamic and adversarial environments. The following section will focus on the implementation details of these strategies within the CSLE framework.

3.4 Implementation

In this section, we will discuss the detailed implementation of the proposed method for integrating poisoning strategies and dynamic defensive mechanisms into reinforcement learning-based Intrusion Detection Systems (IDS). The implementation focuses on two key components: the `PoisonedEnvironmentWrapper`, which introduces adversarial perturbations into the environment, and the `MetaAgent`, responsible for adapting the reward function based on the observed progress in intrusion lengths.

We will walk through the construction of these components, including the development of the poisoning and defensive strategies, and explain how they are integrated into the overall training process. Additionally, we will describe the training setup using the PPO algorithm and how the configuration parameters were tailored to evaluate the effectiveness of the proposed strategies. Through this implementation, we aim to create a flexible and robust framework for enhancing the security of active IDS.

3.4.1 Poisoned Environment Wrapper

The `PoisonedEnvironmentWrapper` class is designed to wrap a base gym environment and apply a poisoning strategy during interactions. It takes a base environment, a poisoning strategy, and a meta agent to monitor and adjust the reward settings during training. This wrapper interacts with the environment while potentially modifying the observations, rewards, and actions based on the strategy being used.

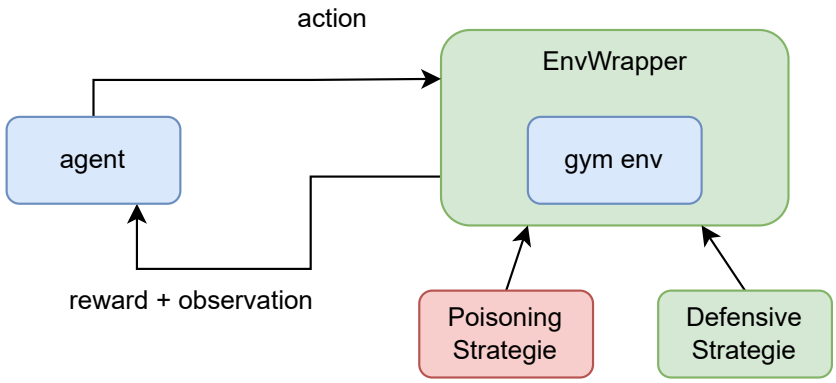

Figure 3.2: Poisoned Environment Wrapper Architecture

In this wrapper, the base environment remains intact, but its interactions are influenced by both the poisoning strategy and the meta agent's dynamic reward updates. Here's a simplified view of the class structure:

```
class PoisonedEnvWrapper(BaseEnv):
    def __init__(self, env: BaseEnv, poisoning_strategy: EnvPoisoning,
                 meta_agent: MetaAgent):
        super().__init__()
        self.env = env
        self.poisoning_strategy = poisoning_strategy
        self.iteration = 0
        self.meta_agent = meta_agent

        # Copying the attributes from the original environment
        self.action_space = env.action_space
        self.observation_space = env.observation_space
        self.reward_range = env.reward_range
        self.metadata = env.metadata

    def step(self, action, attack_mode=True):
        observation, reward, done, _, info = self.env.step(action)
        if attack_mode:
            observation, reward, done, _, info = \
                self.poisoning_strategy.attack_callback(observation, \
                    reward, done, done, info, self.iteration\
                        )
        self.iteration += 1

        if done:
            self.meta_agent.intrusion_length_tracker.\
                add_intrusion_length(\
```

```
                length=info['intrusion_end']\
                    )

        if self.iteration % self.meta_agent.reward_vector_update_freq == 0:
            reward_tensor = self.meta_agent.update_reward_vector()
            self.config.stopping_game_config.R = list(
                StoppingGameUtil.reward_tensor(
                    R_INT=reward_tensor[0],
                    R_COST=reward_tensor[1],
                    R_SLA=reward_tensor[2],
                    R_ST=reward_tensor[3],
                    L=self.config.stopping_game_config.L
                )
            )
        return observation, reward, done, done, info

    def reset(self, **kwargs):
        self.iteration = 0
        return self.env.reset(**kwargs)

    def save_to_json(self, filename='rewards.json'):
        self.meta_agent.record_rewards_to_file(filename=filename)
        self.meta_agent.reset()
        self.poisoning_strategy.reset()

    def render(self, mode='human'):
        return self.env.render(mode)

    def close(self):
        return self.env.close()

    def seed(self, seed=None):
        return self.env.seed(seed)

    def __getattr__(self, name):
        return getattr(self.env, name)

    def get_traces(self) -> List[SimulationTrace]:
        return self.env.get_traces()

    def reset_traces(self) -> None:
        self.traces = []

    def manual_play(self) -> None:
        self.env.manual_play()

    def set_model(self, model: Any) -> None:
        self.env.set_model(model)

    def set_state(self, state: Any) -> None:
        self.env.set_state(state)
```

Code block 3.1: Poisoned Environment Wrapper

3.4.2 MetaAgent Class

The `MetaAgent` class is an abstract class that is responsible for tracking the intrusion length during the game episodes and updating the reward vector based on the intrusion progress. It also records and resets reward vectors across episodes.

The abstract class defines the structure, while the actual behavior is implemented by specific subclasses such as `AdaptiveRewardMetaAgent` and `AdaptiveRewardDiscountedMetaAgent`.

```
from abc import ABC, abstractmethod
from collections import deque
import os
import json

class IntrusionLengthTracker:
    def __init__(self, window_size=100):
        self.window_size = window_size
        self.intrusion_lengths = deque(maxlen=window_size)

    def add_intrusion_length(self, length):
        self.intrusion_lengths.append(length)

    def add_intrusion_length_vec(self, lengths):
        self.intrusion_lengths.extend(lengths)

    def get_average_intrusion_length(self):
        return sum(self.intrusion_lengths) / len(self.intrusion_lengths)

    def intrusion_length_absolute_progress(self):
        return abs(self.intrusion_lengths[-1] - \
                   self.intrusion_lengths[0])

    def smooth_intrusion_length_absolute_progress(self,
                                                  smoothing_window=10):
        recent = list(self.intrusion_lengths)[-smoothing_window:]
        past = list(self.intrusion_lengths)[:smoothing_window]
        return abs(sum(recent) / smoothing_window - \
                   sum(past) / smoothing_window)

class MetaAgent(ABC):
    def __init__(self,
                 initial_reward_vector,
                 save_directory='./experiments',
                 learning_rate=0.1,
                 intrusion_length_tracker=None,
                 reward_vector_update_freq=20
                 ):
        self.reward_vector = initial_reward_vector
        self.initial_reward_vector = initial_reward_vector
        self.learning_rate = learning_rate
        self.intrusion_length_tracker = \
            intrusion_length_tracker or IntrusionLengthTracker()
        self.reward_vector_update_freq = reward_vector_update_freq
        self.reward_vector_record = deque([list(initial_reward_vector)])
        self.save_directory = save_directory

    @abstractmethod
```

```
    def update_reward_vector(self):
        pass

    def reset(self):
        self.reward_vector = self.initial_reward_vector
        self.reward_vector_record = deque(
            [
                list(self.initial_reward_vector)
            ]
            )

    def get_recent_rewards(self, n):
        return list(self.reward_vector_record)[-n:]

    def record_rewards_to_file(self, filename='rewards.json'):
        filepath = os.path.join(self.save_directory, filename)
        with open(filepath, 'w') as file:
            json.dump(list(self.reward_vector_record), file, indent=4)
```

Code block 3.2: MetaAgent abstract class

3.4.2.0.1 AdaptiveRewardMetaAgent: This class dynamically adjusts the reward vector based on the observed progress in intrusion length, increasing or decreasing the reward values to improve the performance of the agent. The reward vector is updated periodically to reflect changes in the learning process.

```
class AdaptiveRewardMetaAgent(MetaAgent):
    def update_reward_vector(self):
        """
        Update the reward vector based on the intrusion length change.
        """
        intrusion_length_change = \
            self.intrusion_length_tracker.\
                smooth_intrusion_length_absolute_progress()
        for i in range(len(self.reward_vector)):
            if self.reward_vector[i] < 0:
                self.reward_vector[i] -= \
                    self.learning_rate * intrusion_length_change
            else:
                self.reward_vector[i] += \
                    self.learning_rate * intrusion_length_change

        self.reward_vector_record.append(list(self.reward_vector))
        return self.reward_vector
```

Code block 3.3: AdaptiveRewardMetaAgent

3.4.2.0.2 AdaptiveRewardDiscountedMetaAgent: This variant of the MetaAgent uses a discounting factor c to smooth the updates to the reward vector. The smaller the change in intrusion length, the smaller the reward update, ensuring smoother progression during training.

```
class AdaptiveRewardDiscountedMetaAgent(MetaAgent):
    def __init__(self, initial_reward_vector,
                 save_directory='./experiements',
```

```
                 learning_rate=0.1, intrusion_length_tracker=None,
                 reward_vector_update_freq=20, c=0.5):
        super().__init__(initial_reward_vector,
                         save_directory, learning_rate,
                         intrusion_length_tracker,
                         reward_vector_update_freq)
        self.c = c

    def update_reward_vector(self):
        """
        Update the reward vector based on the intrusion length change,
        applying a discount factor.
        """
        intrusion_length_change = \
            self.intrusion_length_tracker.\
                smooth_intrusion_length_absolute_progress()
        for i in range(len(self.reward_vector)):
            if self.reward_vector[i] < 0:
                self.reward_vector[i] -= \
                    self.learning_rate * 1/\
                        (self.c + intrusion_length_change)
            else:
                self.reward_vector[i] += self.learning_rate * 1/ \
                    (self.c + intrusion_length_change)

        # Record the updated reward vector
        self.reward_vector_record.append(list(self.reward_vector))
        return self.reward_vector
```

Code block 3.4: AdaptiveRewardDiscountedMetaAgent

3.4.3 Poisoning Strategy

The `EnvPoisoning` class defines an abstract interface for various poisoning strategies. These strategies affect the interaction between the agent and the environment by modifying the observation, reward, or action at every step.

In this project, we implemented two strategies: - `DummyThresholdPoisoning`, which acts as a placeholder strategy and leaves the environment interaction mostly unaltered. - `RandomBeliefPadding`, which alters the belief value of the agent randomly within a defined ratio and padding.

```
class EnvPoisoning(ABC):
    @abstractmethod
    def attack_callback(self, observation: List[float],
                        reward: float, done: bool, done_: bool,
                        info: dict, iteration: int\
                            ):
        pass

    @abstractmethod
    def reset(self):
        pass

    def env_wrapper(self, env: BaseEnv, meta_agent: MetaAgent):
```

```
        return PoisonedEnvWrapper(env, self, meta_agent)


class RandomBeliefPadding(EnvPoisoning):
    def __init__(self, random_ratio = 0.2, padding = 5) -> None:
        super().__init__()

        self.random_ratio = random_ratio
        self.padding = padding

    def shift_belief(self, belief: float):
        return max(0, min(1, belief + self.padding))

    def attack_callback(self, observation: List[float],
                        reward: float, done: bool,
                        done_: bool, info: dict, iteration: int):
        if random.random() < self.random_ratio:
            observation[1] = self.shift_belief(belief=observation[1])
        return observation, reward, done, done_, info

    def reset(self):
        return super().reset()
```

Code block 3.5: Poisoning Strategies implementation

3.4.4 Integration into the Experiment Configuration

To integrate these components into the experiment configuration, we first initialize the strategies and meta-agent with appropriate parameters. Then, these components are passed to the experiment configuration, which handles the training process.

```
intrusion_length_tracker = IntrusionLengthTracker(window_size=5)
meta_agent = AdaptiveRewardMetaAgent(
    initial_reward_vector=[-10, -10, 0, 20],
    reward_vector_update_freq=10,
    save_directory=experiment_folder,
    learning_rate=0.01,
    intrusion_length_tracker=intrusion_length_tracker
)

dummy_threashold_poisoning = DummyThresholdPoisoning()

experiment_config = ExperimentConfig(
    meta_agent=meta_agent,
    poisoning_strategie=dummy_threashold_poisoning
    ...
)
```

Code block 3.6: Initializing Strategies and MetaAgent

3.4.5 Training the Agent with PPO

For this project, the PPO algorithm was chosen due to its stability in convergence when training the defensive agent. The agent is trained in the poisoned environment, and the meta-agent continuously updates the reward settings.

```
agent = PPOAgent(
    emulation_env_config=emulation_env_config,
    simulation_env_config=simulation_env_config,
    experiment_config=experiment_config
)
s
experiment_execution = agent.train()
```

Code block 3.7: Integrating strategies into PPO agent training

The PPO algorithm offers stability, making it easier to observe the effects of the poisoning and defensive strategies on the agent's performance. After training, the metrics are saved for evaluation, including the intrusion length metric and the progression of the reward vector.

3.5 Conclusion

In this section, we have implemented a robust framework for training reinforcement learning agents in the context of Intrusion Detection Systems. The core elements of the implementation are the `PoisonedEnvironmentWrapper`, which facilitates the application of various poisoning strategies during agent-environment interactions, and the `MetaAgent`, which dynamically adjusts the reward function based on the progress in intrusion lengths. These components were designed to ensure adaptability and efficiency during the training process.

We employed two specific poisoning strategies and implemented them in conjunction with adaptive meta-agents to modify the learning environment and reward structure dynamically. The integration of these strategies into the experiment configuration, and their application using the stable PPO algorithm, provided a solid basis for conducting extensive experiments.

The implementation laid a solid foundation for future experimentation and analysis. In the next chapter we will provide the evaluation and results of applying these methods.

Chapter 4

Evaluation and Results

4.1 Introduction

In this chapter, we present the results of the experiments conducted in the context of training a reinforcement learning (RL) agent for active intrusion detection using the CSLE framework. The primary metric analyzed during the experiments is the **intrusion length**, which serves as the key indicator of the agent's effectiveness in mitigating network intrusions.

The experiments were divided into two main phases: evaluating the effect of poisoning strategies on the RL agent's performance and testing the potential of the *MetaAgent* strategy as a defensive approach. While the poisoning strategy significantly impacted the agent's training, the MetaAgent strategy did not improve convergence as anticipated. This section will highlight both the successes and challenges encountered during the implementation and testing of these approaches.

Additionally, we will discuss the limitations faced during the experimentation process, particularly the constraints imposed by limited computational resources. Due to these limitations, some experiments relied on emulation results provided by the original CSLE framework authors, which were gathered over a period of six months, far beyond the time frame available for this internship. Finally, we will conclude with an analysis of the results and potential directions for future work.

4.2 Overview of the Experiment Configuration

The experiments conducted during this study were based on the CSLE framework for simulating and evaluating intrusion detection strategies. For the experiments, we utilized the PPO agent implementation due to its stability in convergence, specifically for reducing the intrusion length over the training phase. The training was executed by integrating various defensive and poisoning strategies, with an emphasis on assessing the robustness of the agent's learning process.

The experiment setup included the integration of both defensive and poisoning strategies, which were passed into the environment configuration before launching the PPO agent training. The metrics collected during training were primarily focused on intrusion length, which serves as a direct indicator of the agent's effectiveness in counteracting poisoning attacks.

4.3 Scenarios Evaluated

The experiments were divided into four primary scenarios to assess the influence of the defensive and poisoning strategies on the learning agent. Each scenario is detailed below:

Poisoning with MetaAgent: In this scenario, the PPO agent was trained under a poisoning attack while using the *MetaAgent* strategy for defense. The learning rate (lr) was set to 0.01, and the update frequency was fixed at 20. This means the *MetaAgent* adjusted the reward vector every 20 interactions based on the intrusion length observed during training. The goal was to observe if dynamically adjusting the reward could reduce the impact of the poisoning attack and stabilize the training process.

Poisoning with Discounted MetaAgent: Here, the *Discounted MetaAgent* was employed as the defensive strategy under poisoning conditions. Similar to the first scenario, the update frequency was set at 20 interactions. However, the key difference lies in the fact that after each update, the effect of adjusting the reward vector was discounted, reducing the strength of subsequent changes. This strategy was designed to introduce a degree of gradual adaptation and resilience in the reward system over time.

Poisoning without MetaAgent: In this scenario, poisoning attacks were applied to the agent without the presence of any defensive strategies. The goal of this scenario was to serve as a baseline to understand the full impact of poisoning on the agent's performance without any mitigation efforts, highlighting the vulnerability of the system under attack conditions.

No Poisoning and No MetaAgent: This control scenario involved no poisoning and no defensive strategies applied to the PPO agent. The purpose of this configuration was to observe the natural convergence behavior of the agent without external interference and provide a comparison against the poisoned environments to assess the effect of poisoning strategies on the learning process.

4.4 Impact of Poisoning

The primary metric for evaluating the effectiveness of the training process was the intrusion length, which measures the time an agent took to stop an intrusion during the training phase. In the poisoned environment, this metric served as an indicator of how well the agent could handle adversarial scenarios.

The figure below (Figure 4.1) compares the performance of the agent under poisoned

conditions with no defensive strategies against a scenario where no poisoning was applied. It is evident that poisoning has a significant negative impact on the agent's ability to reduce the intrusion length, demonstrating the effectiveness of the poisoning strategy in disrupting the agent's learning process.

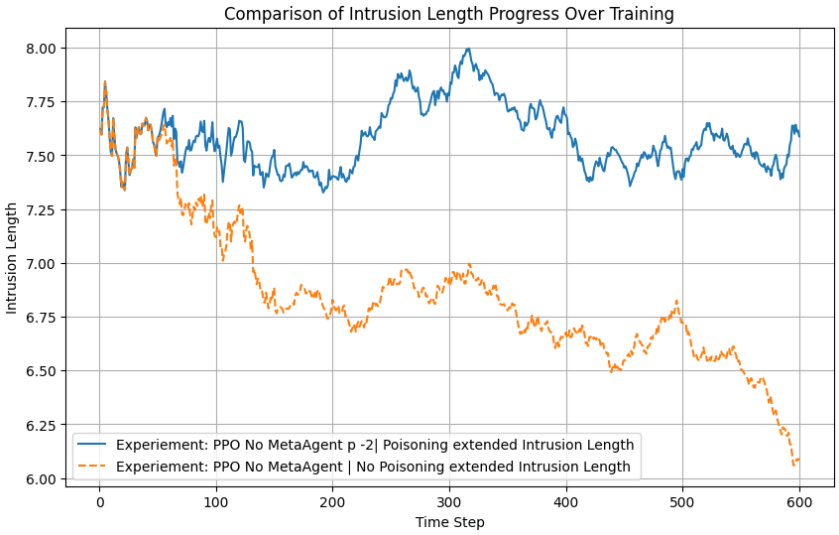

Figure 4.1: Comparison of Intrusion Length: Poisoned vs. Non-Poisoned Environments

The impact of poisoning on the intrusion length during training can be observed clearly in Figure 4.1. The poisoned agent took significantly longer to stop the intrusion compared to the non-poisoned agent. This effect was primarily achieved by manipulating the belief state and distorting the observations passed to the learning agent, leading to suboptimal decisions during training.

In the absence of defensive strategies, the agent failed to adjust to the adversarial inputs effectively, leading to an increased intrusion length. This highlights the importance of defensive mechanisms, which we further explore in the next section. Despite efforts to mitigate this using the *MetaAgent* and *Discounted MetaAgent*, the success of these strategies in reducing the impact of poisoning was limited, as discussed in later sections.

4.5 Effectiveness of MetaAgent's Adaptive Strategy

In our experiments, we tested the *MetaAgent*'s adaptive strategy to see whether it could mitigate the impact of poisoning on the agent's performance. The core idea behind the *MetaAgent* was to dynamically adjust the reward vector based on the agent's performance, specifically using the intrusion length as a key metric.

Although the *MetaAgent* was able to track the agent's progress and update the reward vector accordingly, the overall success in reducing the poisoning effect was limited. Despite our efforts to fine-tune the learning rate and reward update frequency, the agent still struggled to cope with adversarial manipulations during training.

The plot in Figure 4.2 shows the performance of the agent with the adaptive *MetaAgent* strategy compared to the baseline (non-poisoned) scenario. As can be seen, the *MetaAgent* did succeed in partially reducing the intrusion length, but it did not entirely negate the poisoning effect.

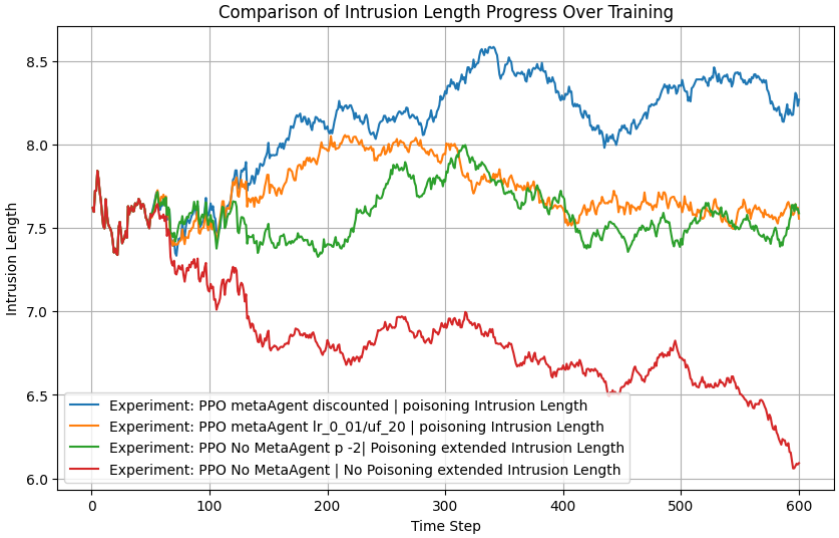

Figure 4.2: Comparison of Intrusion Length: With and Without MetaAgent Strategy

4.6 Conclusion

Despite these efforts, the convergence of the intrusion length during training remained suboptimal compared to the non-poisoned scenario. The results indicate that while the adaptive reward strategy was somewhat effective, more robust mechanisms are needed to fully counteract the effects of poisoning. Future work could explore alternative defensive strategies or improvements to the MetaAgent approach to enhance resilience against adversarial attacks. In the following chapter, we will discuss the results in more detail, highlight the limitations encountered during our experiments, and outline directions for future work.

Discussion, Limitations, and Future Work

4.7 Introduction

In the previous chapter we provided details about the scenrio of our experiementation process. In this chapter, we will discuss the outcomes of our experiments, highlight the limitations encountered, and outline directions for future research.

4.7.1 Discussion of Results

Our experiments focused on evaluating the impact of poisoning attacks on the training process of reinforcement learning agents in the context of intrusion detection systems. The key metric we used to assess the agent's performance was the *intrusion length*, which serves as a proxy for how quickly the agent can detect and respond to intrusions.

The results clearly demonstrated that poisoning has a significant negative impact on the agent's performance. As shown in Figure 4.1, the intrusion length increases considerably when poisoning is introduced, confirming the vulnerability of the agent to adversarial attacks during training. Despite attempts to mitigate these effects with the *MetaAgent*, our adaptive strategies did not fully negate the impact of the poisoning, though there were small improvements in specific cases.

The *MetaAgent* adaptive reward strategy did manage to reduce the poisoning effect to some extent, but its overall effectiveness remained limited. This indicates that while our approach had potential, further refinement of the defensive strategy is necessary to achieve more robust performance against poisoning attacks.

4.7.2 Limitations

Several limitations affected the scope and outcomes of this research. The primary limitation was the computational power available for the experiments. Due to resource constraints, we were unable to execute large-scale experiments with diverse attacking scenarios. Instead, we relied on emulation results provided by the authors of the CSLE

framework, which required six months to generate. Reproducing these results within the time frame of the internship was not feasible.

Additionally, the implementation of the *MetaAgent* strategy was intended to adjust the reward vector dynamically during training to guide the agent toward mitigating the impact of poisoning. However, despite experimenting with different approaches and hyperparameters, the strategy did not significantly improve the agent's performance. This outcome highlights the need for further research into more effective defensive strategies that could better withstand poisoning attacks.

4.7.3 Future Work

Moving forward, several areas of future research can be explored to improve upon the limitations and findings of this work:

- **Exploring alternative defensive strategies**: While the *MetaAgent* provided some benefits, future research could investigate other reward adjustment techniques or defensive strategies that may be more effective in mitigating poisoning effects.
- **Experimentation with different poisoning strategies**: The scope of this research was limited to specific poisoning strategies. Expanding the range of poisoning techniques and testing their effects on various defensive approaches could yield deeper insights into the strengths and weaknesses of reinforcement learning in adversarial environments.
- **Improved computational resources**: Access to greater computational power would enable more extensive experimentation, including training agents in real-time simulations and with diverse attack scenarios. This would allow for a more comprehensive evaluation of the strategies discussed in this work.
- **Collaboration with emulation environments**: Collaborating with other researchers or institutions to gain access to emulation environments, like those used in CSLE, could expedite the evaluation process and allow for more sophisticated testing of defensive strategies.

4.8 Conclusion

In conclusion, while the results of this research provide valuable insights into the vulnerabilities of reinforcement learning in active intrusion detection systems, the limitations encountered highlight the need for continued exploration and innovation in this field.

Bibliography

Agostinelli, F., Hocquet, G., Singh, S., & Baldi, P. (2018). From reinforcement learning to deep reinforcement learning: an overview [Accessed: 2024-08-10]. https://cse.sc.edu/~foresta/assets/files/Agostinelli2018_Chapter_FromReinforcementLearningToDee.pdf

Andersen, P.-A., Goodwin, M., & Granmo, O.-C. (2019). Towards model-based reinforcement learning for industry-near environments. *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. https://doi.org/10.1007/978-3-030-34885-4_3

Bace, R. G., Mell, P., et al. (2001). Intrusion detection systems.

Bhosale, R., Mahajan, S., & Kulkarni, P. (2014). Cooperative machine learning for intrusion detection system. *International Journal of Scientific and Engineering Research*, *5*(1), 1780–1785.

Biju, J. M., Gopal, N., & Prakash, A. J. (2019). Cyber attacks and its different types. *International Research Journal of Engineering and Technology*, *6*(3), 4849–4852.

Cao, Z., Xu, S., Peng, H., Yang, D., & Zidek, R. (2022). Confidence-aware reinforcement learning for self-driving cars. *IEEE Transactions on Intelligent Transportation Systems*, *23*(7), 7419–7430. https://doi.org/10.1109/TITS.2021.3069497

Dong, H., Ding, Z., & Zhang, S. (2020). Deep reinforcement learning fundamentals, research and applications. *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. https://doi.org/10.1007/978-981-15-4095-0

Duan, J., Li, S. E., Yang Guan, Q. S., & Cheng, B. (2020). Hierarchical reinforcement learning for self-driving decision-making without reliance on labeled driving data. *IET Intelligent Transport Systems, 2020, 14(5): 297-305*. https://doi.org/10.48550/arXiv.2001.09816

Hammar, K., & Stadler, R. (2024). Learning near-optimal intrusion responses against dynamic attackers. *IEEE Transactions on Network and Service Management*, *21*(1), 1158–1177. https://doi.org/10.1109/TNSM.2023.3293413

Kim, M.-S., Eoh, G., & Park, T.-H. (2022). Decision making for self-driving vehicles in unexpected environments using efficient reinforcement learning methods. https://doi.org/10.3390/electronics11111685

Limmen, D. (2020). Cyber security learning environment (csle) [https://limmen.dev/csle/]. *limmen.dev*.

Malialis, K., & Kudenko, D. (2015). Distributed response to network intrusions using multiagent reinforcement learning. *Engineering Applications of Artificial Intelligence*, *41*, 270–284.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C.,

Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., & Peters, J. (2022). Robust reinforcement learning: a review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, *4*(1), 276–315.

Nguyen, T. T., & Reddi, V. J. (2021). Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, *34*(8), 3779–3795.

Phan, T. V., & Bauschert, T. (2022). Deepair: deep reinforcement learning for adaptive intrusion response in software-defined networks. *IEEE Transactions on Network and Service Management*, *19*(3), 2207–2218.

Puterman, M. L. (1990). Chapter 8 markov decision processes. In *Stochastic models* (pp. 331–434, Vol. 2). Elsevier. https://doi.org/https://doi.org/10.1016/S0927-0507(05)80172-0

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shamshirband, S., Patel, A., Anuar, N. B., Kiah, M. L. M., & Abraham, A. (2014). Cooperative game theoretic approach using fuzzy q-learning for detecting and preventing intrusions in wireless sensor networks. *Engineering Applications of Artificial Intelligence*, *32*, 228–241. https://doi.org/https://doi.org/10.1016/j.engappai.2014.02.001

Shyalika, C., Silva, T., & Karunananda, A. (2020). Reinforcement learning in dynamic task scheduling: a review. https://doi.org/10.1007/s42979-020-00326-5

Silver, D. (2020). Markov decision processes [Accessed: 2024-08-28]. https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*. https://arxiv.org/abs/1712.01815

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144. https://doi.org/10.1126/science.aar6404

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction* (2nd). MIT Press.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, *38*(3), 58–68. https://doi.org/10.1145/203330.203343

Wang, C., Zhang, Q., Tian, Q., Li, S., Wang, X., Lane, D., Petillot, Y., & Wang, S. (2020). Learning mobile manipulation through deep reinforcement learning. *Sensors*, *20*(3), 939. https://doi.org/10.3390/s20030939

Yu, C., Liu, J., Nemati, S., & Yin, G. (2021). Reinforcement learning in healthcare: a survey. *ACM Computing Surveys*, *55*(1), 5:1–5:36. https://doi.org/10.1145/3477600

Zhang, T., & Mo, H. (n.d.). Reinforcement learning for robot research: a comprehensive review and open issues. https://doi.org/10.1177/1729881421100730