The Pennsylvania State University
The J. Jeffrey and Ann Marie Fox Graduate School

ADVERSARIAL REINFORCEMENT LEARNING FOR CYBER-ATTACK PREVENTION, DETECTION, AND MITIGATION

A Dissertation in
Informatics
by
Taha Eghtesad

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2026

The dissertation of Taha Eghtesad was reviewed and approved by the following:

Aron Laszka
Assistant Professor of Information Sciences and Technology
Dissertation Advisor
Chair of Committee

Hadi Hosseini
Associate Professor of Information Sciences and Technology

Jinghui Chen
Assistant Professor of Information Sciences and Technology

Abhinav Verma
Hartz Family Career Development Assistant Professor of Computer Science and Engineering

Abstract

This doctoral dissertation addresses the evolving challenges in cyberphysical systems security, emphasizing a need for a comprehensive framework that integrates proactive prevention, effective detection, and adaptive mitigation strategies. The landscape of cybersecurity faces persistent threats from advanced persistent threats, ever-changing attack vectors, and the inevitability of human errors. Current security measures, including user management, firewalls, and secure software development life cycle practices, provide a foundational defense but fall short against sophisticated adversaries.

To bolster cybersecurity defenses, a multi-faceted approach is proposed, focusing on domain-specific prevention, detection, and mitigation of cyber threats using state-of-the-art adversarial Deep Reinforcement Learning (DRL) technologies.

Prevention of Threats using Moving Target Defense: Traditional security measures are augmented by a proactive strategy known as Moving Target Defense (MTD). MTD introduces continuous and random alterations to system configurations, making reconnaissance computationally expensive for adversaries or trapping them in exploration loops. Manual deployment of MTD configurations poses challenges, necessitating automated approaches that balance security benefits and system efficiency. The goal is to render cyber-attacks economically and logistically infeasible for adversaries.

Detection of False Data Injection in Transportation Networks: Strategic False Data Injection (FDI) attacks on navigation applications and transportation networks can lead to severe consequences, such as traffic congestion and disruption of essential services. Detecting such attacks requires automated mechanisms capable of identifying changes in traffic patterns. In the absence of public data, strategic decision-making algorithms are crucial to generating worst-case attack scenarios, informing the development of countermeasures, and enhancing detection mechanisms. The focus is on developing automated systems that can detect deviations in traffic patterns and alert authorities to disarm ongoing threats.

ICS Attack Mitigation Through Resilient Control: The remote control of Industrial Control Systems (ICS) provides efficiency but also widens the attack surface. Traditional responses involve resetting compromised software components, but this is not always feasible in case of critical infrastructure that needs to be highly available. Time gaps between detection and patching allow adversaries to exploit vulnerabilities, demanding automated mitigation strategies. The research concentrates on FDI attacks, particularly *0-stealthy* attacks, where adversaries change sensor and actuator values to destabilize physical processes. The objective is to develop automated attack-resilient control policies that minimize the worst-case impact of such attacks.

Addressing these challenges involves leveraging decision-making and especially RL algorithms, specifically tailored to the unique dynamics of each scenario. The adaptability of attackers necessitates defender strategies that can adapt and respond to changing attack tactics. Additionally, information asymmetry calls for diverse RL models to capture both attacker and defender perspectives. The scalability of decision-making problems in large-scale scenarios, such as transportation networks, demands state-of-the-art RL algorithms for efficient solutions.

This dissertation aims to contribute to the enhancement of cybersecurity by developing a comprehensive framework that addresses the dynamic challenges posed by evolving cyber threats. The proposed strategies encompass prevention, detection, and mitigation, leveraging state-of-the-art RL algorithms to adapt to the complex and ever-changing cybersecurity landscape.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acknowledgments

First and foremost, I would like to express my deepest and most sincere gratitude to my dissertation advisor, **Dr. Aron Laszka**. We have worked together for many years, and this dissertation would not have been possible without his exceptional guidance, unwavering patience, and continuous support. His mentorship, insightful feedback, and profound expertise were instrumental in shaping this research. I am incredibly fortunate to have had him as a mentor.

I am also profoundly grateful to my dissertation committee members: **Dr. Hadi Hosseini**, **Dr. Jing Yang**, **Dr. Jinghui Chen**, and **Dr. Abhinav Verma**. I thank them for their time, engaging discussions, and the invaluable insights they provided. Their constructive criticism and challenging questions significantly improved the quality and rigor of this work.

I also would like to thank the leadership of *Pennsylvania State University* and **College of Information Sciences and Technology**, specially **Dr. Sarah Rajtmajer**, as well as the academic operations staff, specifically **Jenna Hart**, for providing a calm, safe, and secure environment that was conducive to completing this dissertation.

My journey through graduate school was enriched by my lab members, who became close friends. My heartfelt thanks goes to **Dr. Afiya Ayman**, **Dr. Amutheezan Sivagnanam**, **Chaeeun Han**, and **Tinghua Chen**. I cherish the countless hours of technical discussions, collaborative problem-solving, and the supportive and vibrant lab environment we shared.

I would like to extend my gratitude to my collaborators, whose expertise was essential to the completion of the papers that form this dissertation. Specifically, I thank **Dr. Yevgeniy Vorobeychik**, **Dr. Abhishek Dubey**, **Dr. Scott Eisele**, and **Dr. Omer Akgul** for their wonderful collaboration.

This dissertation was supported in part by funding from the National Science Foundation (NSF) under grants CNS-1647015, CNS-1818901, CNS-1840052, CNS-1850510, CNS-1801545 and CNS-1952011; the US Department of Energy (DOE) under grant DE-AR0000666; and the National Institute of Standards and Technology (NIST) under grant 70NANB18H198. The findings and conclusions do not necessarily reflect the view of the funding agency.

I am deeply indebted to my peers who became dear friends along this arduous journey: **Dr. Hessam Moradi**, **Dr. Milad Heydariaan**, **Hossein Neeli**, **Dr. Alireza Ansaripour**, **Dr. Kasra Nezamadabadi**, and **Shahriar Shayesteh**. The

camaraderie, shared struggles, and mutual encouragement made the challenging moments of the Ph.D. not only bearable, but also often enjoyable.

My special thanks go to my friends outside of academia who provided a crucial anchor to the world beyond research: **Parsa Badiei**, **Hamid Shayestehmanesh**, **Pegah Kiaei Ziabari**, **Dr. Mohammad Ahmadi**, and **Alireza Ektefaei**. Their constant friendship, encouragement, and sympathy gave me the strength to persevere.

Finally, I owe my deepest gratitude to my parents. Their unconditional love, unwavering belief in me and immense sacrifices have been my greatest motivation. This achievement is as much theirs as mine.

Chapter 1
Introduction

Computer and network security typically relies on two main principles. First, security measures such as user management, firewalls, and common rule-based network and file access controls are used to prevent unauthorized access and behavior toward computing resources and software. Second, software development follows the Secure Software Development Life Cycle (SSDLC or Secure SDLC); developers and engineers regularly test the software and network stacks for security vulnerabilities and patch them accordingly. However, even with these rigorous methods in place, the absolute security of software and networked components cannot be guaranteed in light of Advanced Persistent Threats (APT) launched by sophisticated and resourceful adversaries, adaptability of attack vectors toward new software releases, and human error in system architecture.

To thwart cyber-attacks, it is important to implement **prevention** measures that make the attack infeasible and uneconomic for adversaries. One effective approach is proactively obfuscating and changing the system's attack surface, making it harder for attackers to perform reconnaissance, thus increasing their uncertainty and cost by keeping them in an infinite loop of exploration in a scheme called Moving Target Defense (MTD).

Another attempt at ensuring computer infrastructure's security and continued operation is **detection** of security breaches. System administrators use network security tools to monitor computer networks and disconnect unsafe users and connections. However, these tools fail in case of an insider attack or when facing *0-day* vulnerabilities. This calls for a domain-specific approach to detect security breaches without relying on the specifics of the network and security architecture but taking the logic of the underlying system into account. The detection mechanism should detect system deviations from its nominal and pre-specified operating points.

Moreover, there is a time gap between the initiation of a cyber-attack and its detection, during which the attacker can cause significant damage. Computer systems

like Cyber-Physical Systems (CPS) and Industrial Control System (ICS) that control critical infrastructure such as chemical and pharmaceutical manufacturing, electricity generation, or transportation systems must remain highly available, making it impossible to bring them offline in case of a security threat to prevent damage. Further, once a security breach is detected, remedial tasks such as software patching, reboots, and hardening must be implemented that prolong the time available to the adversary to cause damage. Motivated by this, there is a need for automatic domain-specific **mitigation** of threats. In this scenario, the security software can automatically adjust its behavior based on operational requirements to limit the system's deviation from its nominal working parameters, minimizing the harm.

1.1 Domain-Specific Prevention, Detection, and Mitigation

The cybersecurity landscape is evolving with threats and constant changes in attack strategies. Moreover, there exists an asymmetry of information between the attackers and the defenders on what they observe, exacerbating the difficulty of modeling and strategizing. Therefore, the paramount task is to analyze the current cybersecurity requirements in a domain-specific manner to capture and formalize the interactions between the attacker, the targeted system, and defense mechanisms. For this purpose, it is crucial to understand the specifics of the operations of the systems currently in use.

1.1.1 Attack Prevention Using Moving Target Defense

As mentioned, traditional approaches for security focus on disallowing intrusions by hardening systems to decrease the occurrence and impact of vulnerabilities using network and software security tools or on detecting and responding to intrusions, *e.g.,* restoring the configuration of compromised servers. While these passive and reactive approaches are useful, they cannot provide perfect security in practice. Further, these approaches let adversaries perform reconnaissance and planning unhindered, giving them a significant advantage in information and initiative. As adversaries are becoming more sophisticated and resourceful, it is imperative for defenders to augment traditional approaches with more proactive ones, which can give defenders the upper hand.

MTD is a proactive approach that changes the rules of the game in favor of the defenders. MTD techniques enable defenders to thwart cyber-attacks by continuously

and randomly changing the configuration of their assets, *i.e.,* networks or hosts. These changes increase the uncertainty and complexity of attacks, making them computationally expensive for the adversary [Zheng and Siami Namin, 2018] or putting the adversary in an infinite loop of exploration [Tan et al., 2019], thus **preventing** cyber-threats.

System administrators typically have to manually select MTD configurations to be deployed on their networked systems based on their previous experiences [Hu et al., 2019]. This approach has two main limitations. First, it can be very time-consuming since 1) there are constraints on data locations, so the system administrator must make sure that constraints are met before deploying MTD, 2) the physical connectivity of servers cannot be easily changed, and 3) computation resources are limited. Second, capturing the trade-off between security and efficiency is difficult since the most secure configuration is total randomization, but this has high-performance overhead [Chen et al., 2015].

In light of this, it is crucial to provide automated approaches for deploying MTD, which maximize security benefits for the protected assets while preserving the efficiency of the system. The key ingredient to automation of MTD deployment is finding a design model that reflects multiple aspects of the MTD environment [Li and Zheng, 2019, Zheng and Siami Namin, 2018, Prakash and Wellman, 2015, Albanese et al., 2019]. Further, we need a decision-making algorithm for the model to select when to deploy an MTD technique and where to deploy it [Tan et al., 2019]. Finding optimal strategies for the deployment of MTD is computationally challenging since there can be a huge number of applicable MTD deployment combinations, even with a trivial number of MTD configurations or in-control assets. Further, the adversary might adapt to these strategies.

1.1.2 Detection of False Data Injection in Transportation Networks

Drivers rely on navigation applications and online information more than before. Furthermore, the availability of social media has accelerated the spread of misinformation. A malicious actor could manipulate the drivers directly by sending malicious information through SMS messaging [Waniek et al., 2021], manipulating traffic signals [Chen et al., 2018, Levy-Bencheton and Darra, 2015, Laszka et al., 2016, Feng et al., 2018, Reilly et al., 2016], or physically changing the road signs [Eykholt et al., 2018] to interfere with drivers' route selection. With the availability of social media, the drivers can further spread this misinformation to their peers to snowball the effect of manipulation. Alternatively, the adversary can inject false information into the navigation application. For example, one can place phones in a cart and pull them on the street, tampering with the navi-

gation application to result in marking the road with heavy traffic and rerouting the drivers [Schoon, 2020].

Manipulating transit networks can lead to increased traffic congestion and devastating consequences. Modern societies heavily rely on road networks to access essential services such as education, healthcare, and emergency services. Moreover, road networks contribute to economic growth by enabling logistic movements of materials, goods, and products. Therefore, disruption of transportation networks can lead to food insecurity, job losses, or even political disarray, such as the Fort Lee scandal [Cillizza, 2015].

Efforts have been made to measure the impact of false information injection on dynamic navigation applications [Lin et al., 2018], traffic congestion [Waniek et al., 2021], and navigation applications [Raponi et al., 2022]. However, finding an optimal attack is generally computationally challenging [Waniek et al., 2021], complicating vulnerability analysis.

Understanding how strategic attacks may unfold in the domain of False Data Injection (FDI) attacks against navigation applications is crucial. In the absence of public data, sequential decision-making algorithms are required to generate worst-case strategic attacks that can inform the development and evaluation of countermeasures such as **detection** mechanisms. Further, automated detection mechanisms should be developed to detect any changes to typical traffic patterns and alert authorities of the ongoing threat.

1.1.3 ICS Attack Mitigation Through Resilient Control

Centralized and remote control of industrial processes allows more efficient, fault-tolerant, and robust control of processes. However, connecting sensors and actuators of industrial processes to computer networks for remote sensing and control also widens the attack surface of these systems. Previously, physical access was required for adversaries to sabotage industrial processes. However, with networked ICS and network-connected sensors and actuators, adversaries can exploit vulnerabilities in the underlying network to launch attacks on the ICS.

By compromising and tampering with industrial control systems, adversaries can cause financial loss, physical damage, and even bodily harm. In recent years, we have seen many attacks on ICS [Hemsley et al., 2018, Miller and Rowe, 2012]. Some of the most notable examples include the infamous Stuxnet worm [Farwell and Rohozinski, 2011], which targeted Iranian uranium enrichment facilities between 2005 and 2012, damaging one-fifth of Iran's enrichment centrifuges and leading to the resignation of the head of the Atomic Energy Organization of Iran in 2009. The BlackEnergy malware

attack on the Ukrainian power grid [Kovacs, 2016] in 2015 resulted in 250 thousand people losing electricity for 6 hours. Other examples include the 1999 attack on Gazprom infrastructure and the 2012 Flame attack [Miller and Rowe, 2012].

As discussed earlier, this problem arises since even the most secure software is prone to vulnerabilities [Robles and Choi, 2009, Andreeva et al., 2016]. When attacks are detected [Giraldo et al., 2019, Urbina et al., 2016, Paridari et al., 2018], system engineers try to reset these compromised software components to a safe state and ultimately patch the vulnerability [Foley and Hulme, 2004]. This is good practice; however, resetting might not be available remotely. Further, patching ICS components requires time since vendors usually sell these systems as bundled packages. As a result, in case of an attack, technicians operating these types of systems do not know what is inside and what needs patching to keep it safe from emerging vulnerabilities and threats [Byres, 2008]. The time gap between detecting the attack and patching the software leaves the adversary enough time to cause damage to the ICS or the physical process.

Prior research efforts have investigated the detection and mitigation of adversarial tampering with ICS in various stages, ranging from ICS software security, through attack detection mechanisms [Giraldo et al., 2019, Urbina et al., 2016, Paridari et al., 2018], to proposing fault-tolerant control policies. For example, [Kosut et al., 2010, Fawzi et al., 2014] studied the estimation and control of linear systems when an adversary corrupts some of the sensors or actuators. Another relevant field of study is the evaluation of currently existing detection schemes in different environments and types of attacks on ICS [Urbina et al., 2016]. Moreover, [Singh et al., 2020] surveyed current ICS software security tools.

The prior research literature [Fawzi et al., 2014,Kosut et al., 2010] suggests approaches for attack-detection and attack-resilient control focused on Linear Time Invariant (LTI) systems. However, most practical physical systems are non-linear. Moreover, [Paridari et al., 2018, Combita et al., 2019] used redundant trusted sensors/actuators strategically placed. However, we must always assume that even the most secure components of the ICS can be exploited. As a result, any future effort on attack detection and attack-resilient control must consider (1) any physical process, whether linear or not, and (2) compromise of any signal, regardless of whether the sensor/actuator is trusted or not.

Here, we assume that the adversary has already found vulnerabilities in a certain set of devices on the ICS network or has physical access to such devices in order to launch an attack and sabotage the physical processes controlled by ICS. We should also note that (1) the attack itself and (2) the detection of attacks are out of the scope of this

dissertation.

Our attention is on FDI attacks through which the adversary changes the values of sensor and actuator signals instead of blocking the controller's access to these signals in order to destabilize the physical process. Specifically, we focus on *0-stealthy* attacks [Teixeira et al., 2015, Pasqualetti et al., 2013], which only change these sensor/actuator values within the physical boundaries of the process. Our objective is to find an automated attack-resilient control policy that minimizes the worst-case effect of the attack.

1.2 Approach and Challenges

When faced with decision-making problems in the cyber-security domain, Reinforcement Learning (RL) algorithms can be a powerful tool [Nguyen and Reddi, 2023, Tong et al., 2020]. RL finds a sequential strategic decision-making policy to determine optimal defense mechanisms. However, the scale of the decision-making problem that RL solves can make the out-of-the-box RL algorithms infeasible for direct use. For example, when dealing with large-scale transportation networks, the defender needs to take numerous road link and intersections into account. A simple general-purpose RL algorithm might take days or weeks to learn the dynamics of the system and compute a solution. This necessitates developing state-of-the-art RL algorithms to find a solution in a time-efficient manner.

Further, in all previous scenarios, the attacker adopts a strategic approach. When the security solution attempts to thwart attacks by obfuscating networked assets, the attacker anticipates changes in these assets to extract as much information as possible. During an attack, the adversary may purposefully limit the magnitude of damage or changes made to avoid detection. In the event of a successful breach, the attacker endeavors to destabilize the system, strategically selecting a sequence of actions to maximize damage. Moreover, the adversary might adapt to the defenses and change its strategy. The adaptability of the attacker makes the defender's learned dynamics of the system obsolete in case the attacker uses a different strategy. Therefore, for the defender to remain robust, it should also strategically adapt and respond to the attackers' changing strategies. This necessitates finding optimal attack and defense strategies for both the attacker and defender.

The last obstacle that we need to overcome is the information asymmetry. For this reason, different RL algorithms and models are required to find the attacker and defender's best attack and defense strategy. For instance, in the detection of FDI in the navigation applications scenario, the defender must detect the anomaly in traffic

patterns due to the FDI and raise the alarm accordingly. Here, the attacker has full access to the state of the environment, while the defender only observes the results of false data injected by the attacker. Therefore, the defender might need to rely on past traffic pattern history as well as current understanding of the vehicles.

1.3 Organization

This dissertation proposal unfolds in several chapters. Chapter 2 delves into prior literature on Reinforcement Learning applications for cybersecurity, non-Reinforcement Learning approaches for FDI attack mitigation, and hierarchical Reinforcement Learning approaches. Chapter 3 provides the necessary mathematical foundation centered around Reinforcement Learning and game theory. In Chapter 4, we introduce our formal modeling approach and outline the challenges in depth. In the later chapers, we provide formal models for **prevention** (Chapter 5), **mitigation** (Chapter 6), and **detection** (Chapter 7 and Chapter 8), along with solutions, particular challenges of Reinforcement Learning in the application-specific domain, and current findings.

Chapter 2
Literature Review

This chapter reviews the literature relevant to applying reinforcement learning for cybersecurity, with a specific focus on defending cyber-physical systems and transportation networks. We begin by surveying the broad applications of reinforcement learning in cybersecurity. We then delve into the specific defensive paradigm of Moving Target Defense (MTD) and the role of game theory and reinforcement learning in solving MTD problems. Next, we review work on mitigating False Data Injection (FDI) attacks, a critical threat to CPS. We also explore applications of Hierarchical Reinforcement Learning, a set of techniques well-suited for the complex, multi-faceted problems in this domain. Finally, we connect these threads by examining literature on attacks against transportation networks and the use of reinforcement learning for attack detection.

2.1 Reinforcement Learning for Cybersecurity

The application of Machine Learning (ML)—especially Deep Reinforcement Learning (DRL)—for cybersecurity has gained significant attention recently. A survey of current literature explores the wide-ranging applications of DRL in cybersecurity [Nguyen and Reddi, 2023]. These applications include DRL-based security methods for CPS, autonomous intrusion detection techniques [Iannucci et al., 2019], and Multi-Agent DRL-based game-theoretic simulations for defense strategies against cyber attacks. For example, Multi-Agent DRL has been applied to network routers to throttle processing rates in order to prevent Distributed Denial of Service (DDoS) attacks [Malialis and Kudenko, 2015, Malialis et al., 2015]. In related work, a cooperative Multi-Agent Reinforcement Learning (RL) [Sutton and Barto, 2018] framework has been proposed for intelligent systems [Herrero and Corchado, 2009] to enable quick responses to threats [Bhosale et al., 2014]. Another example of Multi-Agent reinforcement learning is the fuzzy Q-Learning

approach developed for detecting and preventing intrusions in Wireless Sensor Networks (WSN) [Shamshirband et al., 2014]. Furthermore, a Multi-Agent reinforcement learning framework has been proposed for alert correlation based on Double Oracles [Tong et al., 2020].

2.1.1 Reinforcement Learning for Moving Target Defense

A common scheme for preventing reconnaissance on computer systems is the obfuscation of assets through MTD. One of the main research areas in Moving Target Defense is the modeling of interactions between adversaries and defenders. In the area of game-theoretic models for moving target defense, the work most closely related to this dissertation is a model introduced in [Prakash and Wellman, 2015], which our work also uses (see chapter 5). This model can also be used for defense against DDoS attacks [Wright et al., 2016] and for the defense of web applications [Sengupta et al., 2017]. Further, in this area, researchers have proposed MTD game models based on Stackelberg games [Li and Zheng, 2019], Markov Games [Lei et al., 2017, Tan et al., 2019], Markov Decision Process [Puterman, 1994] [Zheng and Siami Namin, 2018], and the FlipIt game [Oakley and Oprea, 2019]. For solving a game model, *i.e.,* finding the optimal playing strategies, numerous approaches such as solving a min-max problem [Li and Zheng, 2019], non-linear programming [Lei et al., 2017], the Bellman equation [Tan et al., 2019, Zheng and Siami Namin, 2018], Bayesian belief networks [Albanese et al., 2019], and reinforcement learning [Hu et al., 2019, Oakley and Oprea, 2019] have been suggested.

2.2 Model-Based Mitigation of False Data Injection in CPS

In the field of mitigating integrity attacks on control systems, *i.e.,* attacks that change the values of sensor readings, the most relevant literature is the work by [Combita et al., 2019]. They proposed a solution to integrity attacks by detecting anomalies using a Kalman filter and mitigating the change in values with analytical redundancy and Optimal Disturbances Decoupling Observer (ODDO). A strong adversary model that will always be able to bypass attack-detection mechanisms has been proposed [Urbina et al., 2016]. The performance of multiple detection techniques was compared against this type of adversary in multiple scenarios, including 1) simulations, 2) network data collected from a large Supervisory Control and Data Acquisition (SCADA), and 3) a testbed on

real-world systems [Urbina et al., 2016]. The estimation and control of linear systems when some sensors or actuators are corrupted by an attacker has been studied [Fawzi et al., 2014]. This work characterizes the physical processes based on the maximum number of sensor signals that can be compromised. It also shows how secure feedback can be used to improve resilience against attacks. Furthermore, it proposes a specific state reconstructor resilient to attacks, inspired by techniques from compressed sensing and error correction. Finally, the authors show that a principle of separation of estimation and control holds and that the design of resilient output feedback controllers can be reduced to the design of resilient state estimators [Fawzi et al., 2014].

2.3 Hierarchical Reinforcement Learning Applications

HRL has gained significant attention due to its applications and development. These methods have proven to be successful in tasks that require coordination between multiple agents, such as Unmanned Aerial Vehicle (UAV)s and autonomous vehicles, to complete objectives efficiently. For instance, a general framework for combining compound and basic tasks in robotics, such as navigation and motor functions, respectively, has been devised [Yang et al., 2018]. However, that application was limited to single-agent RL at both levels. Similarly, attention networks have been used to incorporate environmental data with steering functions of autonomous vehicles in an HRL manner so that the vehicle can safely and smoothly change lanes [Chen et al., 2019]. In UAV applications, the success of HRL in the coordination of wireless communication and data collection has been demonstrated [Zhang et al., 2021]. Although our problem is in a different domain, the fundamental ideas of these works are applicable to us since we are dealing with cooperation and coordination between adversarial agents in finding an optimal manipulation strategy in navigation applications. The study results indicate that the use of Reinforcement Learning approaches accurately modeled the effects of false information injection on navigation apps.

2.4 Attacks Against Transportation Networks

Malicious actors can exploit vulnerabilities in transportation networks to manipulate drivers' route selections. Methods for this include sending malicious SMS messages [Waniek et al., 2021], physically altering road signs [Eykholt et al., 2018], tampering with traffic signals [Ghafouri et al., 2016, Ghena et al., 2014, Laszka et al., 2016, Feng

et al., 2018], or injecting false data into crowdsourced navigation applications [Eryonucu and Papadimitratos, 2022, Schoon, 2020, Eghtesad et al., 2024, Yu et al., 2024, Yang et al., 2023]. Prior work has explored the vulnerabilities of navigation applications to adversarial manipulations and their impacts on traffic congestion [Waniek et al., 2021, Lin et al., 2018, Raponi et al., 2022, Yang et al., 2023, Yu et al., 2024].

2.5 Reinforcement Learning based Detection

Reinforcement learning is a promising tool for the detection and mitigation of attacks. Several prior works have explored methods for detecting and mitigating various types of adversarial attacks. For example, some work focuses on the detection and mitigation of false data injection (FDI) attacks in distributed control systems [Sargolzaei et al., 2020]. Network intrusion detection techniques using reinforcement learning have also been examined [Malialis and Kudenko, 2015, Malialis et al., 2015], and reinforcement learning has been leveraged to address zero-day attacks [Hu et al., 2019]. Furthermore, strategies for prioritizing relevant alerts in network intrusion detection systems using multi-agent reinforcement learning have been investigated [Tong et al., 2020].

Chapter 3
Background

This chapter endeavors to provide foundational knowledge on single-agent (Section 3.1) and multi-agent (Section 3.3) DRL, alongside prevalent Deep Learning (DL) techniques integral to DRL, such as convolutional networks (Section 3.4).

3.1 Independent Reinforcement Learning

One of the primary approaches for finding a decision-making policy is Independant Reinforcement Learning (InRL) [Sutton and Barto, 2018], which focuses on interactions of a single agent and its environment to maximize the agent's gain (represented as rewards or utilities) from the environment. Figure 3.1 shows the interactions between an InRL agent and its environment. A basic RL environment is a Partially-Observable Markov Decision Process (POMDP) [Åström, 1965], which can be represented as a tuple:

$$POMDP = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O} \rangle. \tag{3.1}$$

where $\mathcal{S}$ is the set of all possible states of the environment, $\mathcal{A}$ is the set of all possible actions by the agent, $\mathcal{T}$ is the set of stochastic transition rules, $\mathcal{R}$ is the immediate reward rule of a state transition, and $\mathcal{O}$ is the set of observation rules of the agent.

The objective of RL is to find a *policy* π, which is a mapping from observation space to action space, such that:

$$\pi(o^t) \mapsto a^t \tag{3.2}$$

$$\text{which maximizes } U_*^t = \mathbb{E}\left[\sum_{\tau=0}^{\infty} \gamma^\tau \cdot r^{t+\tau} \;\middle|\; \pi\right] \tag{3.3}$$

where $o^t \leftarrow \mathcal{O}(s^t)$ is the observation received in time step t when the agent is in state

$s^t \in \mathcal{S}$, $a^t \in \mathcal{A}$ is the action taken after that observation. Assuming $o^{t+1} \leftarrow \mathcal{O}(s^{t+1})$ is the next observation received from the environment, the next state was updated by the transition rule $s^{t+1} \leftarrow \mathcal{T}(s^t, a^t)$. Finally, $r^t \leftarrow \mathcal{R}(s^t, a^t, s^{t+1})$ is the reward received in time step t after a state transition to s^{t+1} due to action a^t. Also, the *discount factor* $\gamma \in [0, 1)$ prioritizes rewards received in the current time step over future rewards. When $\gamma = 0$, the agent cares only about the current reward, and when $\gamma = 1$, the agent cares about all future rewards equally.

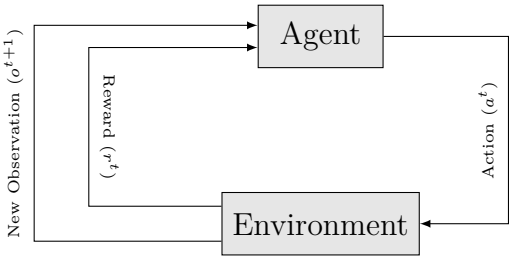


Figure 3.1. Independent Reinforcement Learning

Reinforcement learning aims to maximize the received utility of the agent U_* (eq. (3.3)) by trial and error: interacting with the environment (randomly, following heuristics, or based on the experiences that the agent has seen so far). Generally, during the training, there are two ways to find actions to be taken at each step: (1) *Exploitation*: we use the currently trained policy to choose actions, which helps the agent to more accurately find U_* values of states. (2) *Exploration*: to find actions that lead to higher utility by selecting actions at random and exploring the action/observation space.

As the result of traversing the POMDP, the RL agent collects samples of *experiences* that can be used to train its policy function. An experience $e \in \mathcal{E}$ is a tuple of:

$$e = \langle o^t, a^t, o^{t+1}, r^t \rangle \tag{3.4}$$

Algorithm 1 shows the basic workflow of an RL method.

There are two main approaches for finding an RL policy π. We will discuss these methods in the following subsections.

3.1.1 Action-Value Based Methods

Action-Value RL, also known as *Value Iteration*, methods or *on-policy* RL algorithms that learn the value of an action in each state. Thus, learning the best action that can maximize the value of subsequent states.

Algorithm 1: Off-Policy Independent Reinforcement Learning

Result: policy π
$o \leftarrow$ env.reset();
$\epsilon^t \leftarrow 1$;
for T *steps* **do**
  if $random[0,1] \leq \epsilon^t$ **then**
    $a \leftarrow$ random_action;
  else
    $a \leftarrow \pi(o)$
  end
  $(o', r, done) \leftarrow$ env.step(a);
  add $e = \langle o, a, o', r \rangle$ to E;
  update π based on $X \sim \mathcal{E}$;
  $o \leftarrow o'$;
  decay ϵ^t;
  if *done* **then**
    $o \leftarrow$ env.reset();
  end
end

3.1.1.1 Q-Learning

Q-Learning (QL) [Sutton and Barto, 2018] uses a tabular Q function to estimate the expected future utilities of an action in an observation state (eq. (3.3)):

$$Q(o^t, a^t) = U^t_*|_\pi \tag{3.5}$$

With a tabular approach of storing the Q value for each observation/action, we can find the value of the Q function by applying the Bellman optimization equation:

$$Q(o^t, a^t) = (1 - \alpha_q) \cdot Q(o^t, a^t) + \alpha_q \cdot \underbrace{(r^t + \gamma \cdot \max_{a'} Q(o^{t+1}, a'))}_{\text{TD Target}} \tag{3.6}$$

where α_q is the learning rate of the Q function. The idea for updating the Q function is that the Q function should minimize the Temporal Difference (TD) error, *i.e.,* the difference between the predicted Q value, and the actual expected utility (U^* while following π). This makes the optimal action policy π as:

$$a^t = \pi(o^t) \leftarrow \operatorname*{argmax}_{a'} Q(o^t, a') \tag{3.7}$$

One of the approaches for choosing between exploration or exploitation in action-value RL methods is the ϵ-greedy approach [Sutton and Barto, 2018], where in each step, the agent explores with probability ϵ or takes the current optimal action with probability $1 - \epsilon$.

3.1.1.2 Deep Q-Learning

When we are dealing with environments with highly dimensional action/observation spaces, using tabular-based QL is infeasible since 1) the table for storing Q-values might not fit into memory, and 2) the action and observation spaces need to be enumerated many times for the algorithm to learn an optimal policy.

To address these challenges, Deep Q-Learning (DQL) [Mnih et al., 2015], as a member of DRL algorithms family, suggests using a parameterized function to approximate the Q-value. Using parameterized functions as Q-value approximator makes the DQL approach feasible for such environments since 1) at most, thousands of parameters are stored, and 2) parameterized models can generalize the relation between observations and actions; as a result, learning agents need less time for exploring the observation/action space.

To optimize the parameters θ, we can use gradient descent to minimize the TD error of the network. With the same TD target as eq. (3.6), and taking optimal action as $\text{argmax}_{a'} Q^{\theta}(o^t, a')$, the TD target will be:

$$\forall_{x \in X \sim \mathcal{E}} : q_x^t = r_x^t + \gamma \cdot \max_{a'} Q^{\theta}(o_x^t, a') \tag{3.8}$$

Suppose we have a batch of experiences sampled from the experience replay buffer $X \sim \mathcal{E}$. Then, we can define a Bellman Mean Squared Error (BMSE) loss function to optimize the parameters:

$$L(\theta) = \frac{1}{|X|} \sum_{x}^{X} \left(q_x^t - Q^{\theta}(o_x^t, a_x^t) \right)^2 \tag{3.9}$$

Then, we can calculate the gradient of the parameters to minimize the loss, *i.e.,* making the Q function to make a more accurate estimation of the action value.

$$\theta \leftarrow \theta - \alpha_{\theta} \cdot \frac{\partial L(\theta)}{\partial \theta} \tag{3.10}$$

where α_θ is the learning rate of the Q parameters. Instead of directly applying the gradients, one can use improved optimization algorithms such as *RMSProp* [Hinton et al.,] to find optimized parameters faster.

3.1.1.3 Double Q-Learning

One drawback of DQL is that the TD target is dependent on the network parameters (eqs. (3.8) and (3.9)), making the optimization objective a moving target. Double Deep-Q-Learning (DDQL) [Van Hasselt et al., 2016], therefore, was introduced to tackle this issue by taking a parameterized function $Q^{\theta'}$ to mirror the Q-function and calculating the TD Target based on it. This makes calculating the TD target as:

$$\forall_{x \in X \sim \mathcal{E}} : q_x^t = r_x^t + \gamma \cdot \max_{a'} Q^{\theta'}(o_x^t, a') \tag{3.11}$$

The parameters of the original network are updated by calculating the TD loss (eq. (3.9)) and applying the gradients (eq. (3.10)). Then, the target function parameters can be updated by a moving average on the original Q parameters.

$$\theta' \leftarrow (1 - \tau_\theta) \cdot \theta' + \tau_\theta \cdot \theta \tag{3.12}$$

where τ_Q is the transfer rate of the target network.

3.1.1.4 Deep Deterministic Policy Gradients

While DQL is feasible for discrete action spaces where it is possible to enumerate all possible actions to calculate the TD target ($\max_{a'} Q$, eq. (3.6)) to update the policy, and the policy function ($\text{argmax}_{a'} Q$, eq. (3.7)) when taking actions, it is infeasible to use them with continuous action spaces.

This led to the emergence of policy gradient methods such as Deep Deterministic Policy Gradients (DDPG) [Lillicrap et al., 2015]. This algorithm uses a separate parameterized function μ^Θ to calculate the TD target and the optimal policy. This function is updated by gradient ascent such that it maximizes the action value, *i.e.,* performance, in each observation state:

$$J(\Theta) = \frac{1}{|X|} \sum_x^X Q^\theta \left(o_x^t, \mu^\Theta(o_x^t) \right) \tag{3.13}$$

With this, the parameters are updated by maximizing the performance of this policy J:

$$\Theta \leftarrow \Theta + \alpha_\Theta \cdot \frac{\partial J(\Theta)}{\partial \Theta} \tag{3.14}$$

Using the target policy function, similar to DDQL (section 3.1.1.3), we can calculate the TD target with:

$$\forall_{x \in X \sim \mathcal{E}} : q_x^t = r_x^t + \gamma \cdot Q^{\theta'} \left(o_x^t, \mu^{\Theta'}(o_x^t) \right) \tag{3.15}$$

The parameters of the target network can be updated similarly to the target network of DDQL (eq. (3.12)):

$$\Theta' \leftarrow (1 - \tau_\Theta) \cdot \Theta' + \tau_\Theta \cdot \Theta \tag{3.16}$$

where τ_Θ is the target transfer rate of the policy function. This makes the optimal policy:

$$a^t = \pi(o^t) \leftarrow \mu^\Theta(o^t) \approx \underset{a'}{\operatorname{argmax}}\, Q^\theta(o^t, a') \tag{3.17}$$

One challenge in DDPG is exploration. Alongside the ϵ-Greedy (section 3.1), to better sample actions that update the policy function more accurately, the exploratory policy π' can be constructed by adding a sampled noise $\mathcal{N}$ to the original policy. This noise can be sampled from a correlated distribution such as Uhlenbeck-Ornstein (OU) [Uhlenbeck and Ornstein, 1930] process.

$$a^t = \pi'(o^t) \leftarrow \pi(o^t) + \mathcal{N} \tag{3.18}$$

3.1.2 Value-Based Methods

In contrast to state-value based methods that learn a state-value function and derive a policy from it, *value based methods* directly learn the parameters of a policy. These methods are *on-policy*, meaning the agent learns and improves the same policy that it uses to interact with the environment. Therefore, these algorithms are also called *policy iteration methods*. The goal is to optimize the policy parameters by directly maximizing the expected return (performance objective).

An on-policy algorithm typically involves two main components, both of which are

represented as parameterized function approximators, such as neural networks:

- **The Policy Function (Actor):** This is a function $\pi_\theta(a|s)$, parameterized by θ, which maps a given state s to a probability distribution over actions a. This policy is inherently **stochastic**, which allows the agent to explore the environment naturally by sampling actions from this distribution.
- **The Value Function (Critic):** This is a function $V_\phi(s)$, parameterized by ϕ, which estimates the expected return (value) of being in state s and following the current policy π_θ. This component is not always required (e.g., in the REINFORCE algorithm) but is used in **Actor-Critic** architectures to reduce the variance of policy gradient estimates.

The neural network representing the policy function does not output an action directly. Instead, it outputs the parameters of a probability distribution from which an action is sampled. The choice of distribution depends on the action space of the environment:

- **Categorical Distribution:** Used for discrete action spaces. The network's final layer typically outputs logits (raw scores) for each possible action. These logits are then passed through a `softmax` function to generate a probability for each action, forming a categorical distribution.
- **Gaussian (Normal) Distribution:** Used for continuous action spaces. The network outputs the parameters of the distribution, most commonly the mean (μ) and the standard deviation (σ). The final action is then sampled from this Gaussian distribution, $\mathcal{N}(\mu, \sigma)$. The standard deviation itself can be a learned parameter or a fixed hyperparameter, influencing the degree of exploration.

The distinction between on-policy and off-policy learning is crucial and has significant practical implications:

1. **Exploration:** In on-policy methods, exploration is integrated directly into the policy. The stochastic nature of $\pi_\theta(a|s)$ ensures that the agent naturally tries different actions. In contrast, off-policy methods often use a separate behavior policy for exploration (e.g., ϵ-greedy) while learning about a different, optimal target policy.
2. **Experience Reusability:** On-policy algorithms cannot reuse experiences generated by older versions of the policy. The policy gradient calculation depends on the

actions being sampled according to the *current* policy π_θ. If experiences from an old policy $\pi_{\theta'}$ (where $\theta' \neq \theta$) were used, the gradient estimate would be incorrect, as the probability of those actions occurring would be different. This means that on-policy methods are generally less sample-efficient.

3. **Data Collection:** Consequently, on-policy methods must discard experiences after each parameter update. They typically collect a **trajectory** (a sequence of experiences $s^t, a^t, r^t, s^{t+1}, \dots$) using the current policy π_θ, use this batch of data to compute the policy gradient and update θ, and then discard the data before collecting a new trajectory with the updated policy $\pi_{\theta_{new}}$. This contrasts with off-policy methods, which can store vast amounts of past experiences in a *replay buffer* and sample from them randomly to break correlations and improve stability.

3.1.2.1 Policy Gradient Methods

Policy Gradient (PG) methods represent a class of RL algorithms that optimize the policy π_θ directly. The policy is parameterized by θ, and the algorithm performs gradient ascent on an objective function $J(\theta)$ that measures the expected return. The objective is defined as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r(s^t, a^t) \right] \tag{3.19}$$

where $\tau = (s^0, a^0, s^1, a^1, \dots)$ is a trajectory (or episode) sampled by following the stochastic policy π_θ.

The core of these methods lies in the Policy Gradient Theorem [Sutton and Barto, 2018], which provides a computable expression for the gradient of the objective function, $\nabla_\theta J(\theta)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a^t|s^t) G^t \right] \tag{3.20}$$

where $G^t = \sum_{k=t}^{T} \gamma^{k-t} r(s^k, a^k)$ is the discounted return from time step t onward. The term $\nabla_\theta \log \pi_\theta(a^t|s^t)$ is the *score function*, and G^t is the *reward-to-go*. This expression provides a way to update the policy parameters to make "good" actions (those that led to high returns) more probable.

3.1.2.2 REINFORCE: Monte Carlo Policy Gradient

The REINFORCE algorithm [Williams, 1992], also known as Monte Carlo Policy Gradient, is a direct application of Equation (3.20). As you correctly noted, it uses a Monte Carlo method to estimate the return. It operates by:

1. Sampling a complete episode τ using the current policy π_θ.
2. For each time step t in the episode, calculating the observed return G^t by summing the discounted rewards from that point to the end of the episode.
3. Updating the policy parameters θ via stochastic gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a^t|s^t) G^t \tag{3.21}$$

While this method's gradient estimate is unbiased, it suffers from **high variance**. The Monte Carlo return G^t can fluctuate dramatically between different episodes, even with small changes in the policy, which makes the learning process slow and unstable.

3.1.2.3 Actor-Critic Methods with Advantage Function

To address the high variance of REINFORCE, a common technique is to subtract a state-dependent *baseline* $b(s^t)$ from the return G^t. This operation significantly reduces variance without introducing bias, as long as the baseline does not depend on the action a^t.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a^t|s^t)(G^t - b(s^t))\right] \tag{3.22}$$

A natural and highly effective choice for the baseline is the state-value function itself, $b(s^t) = V^{\pi_\theta}(s^t) = \mathbb{E}[G^t|s^t]$. The resulting term, $G^t - V^{\pi_\theta}(s^t)$, is an estimate of the **Advantage Function**:

$$A^{\pi_\theta}(s^t, a^t) = Q^{\pi_\theta}(s^t, a^t) - V^{\pi_\theta}(s^t) \tag{3.23}$$

The advantage function $A(s^t, a^t)$ measures how much better (or worse) action a^t is compared to the average action from state s^t. Using it as the scaling factor for the gradient is more stable, as it encourages actions that are better than average and discourages those that are worse.

This formulation leads to the **Actor-Critic** architecture. As you suggested, we replace the Monte Carlo evaluation with a function approximator. In this setup, we maintain two models:

- The **Actor** ($\pi_\theta(a^t|s^t)$): The policy, which learns to select actions.
- The **Critic** ($V_\phi(s^t)$): A value function approximator (with parameters ϕ) that learns to estimate the state-value function $V^{\pi_\theta}(s^t)$.

Instead of waiting for the full return G^t, we use the Critic to bootstrap. We can estimate the advantage function using the one-step TD error, δ_t:

$$\delta^t = r^{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s^t) \tag{3.24}$$

This TD error δ^t serves as a low-variance, albeit *biased*, estimate of the true advantage $A(s^t, a^t)$.

The Critic's parameters ϕ are trained to minimize this TD error (*e.g.,* via a mean-squared error loss), which forces V_w to satisfy the Bellman equation:

$$\phi \leftarrow \phi + \beta\delta^t \nabla_\phi V_\phi(s^t) \tag{3.25}$$

Simultaneously, the Actor's parameters θ are updated using the TD error from the critic as the advantage estimate:

$$\theta \leftarrow \theta + \alpha\nabla_\theta \log \pi_\theta(a^t|s^t)\delta^t \tag{3.26}$$

This is the foundation of vanilla stochastic policy gradient algorithms. You are correct that the approximate value function V_ϕ introduces bias; however, by using it to compute the advantage (via the TD error δ^t) rather than using a raw value estimate, we achieve a much more stable learning signal that effectively balances the bias-variance trade-off.

3.1.2.3.1 Generalized Advantage Estimation The main issue with calculating the advantage $\hat{A}$ in PG is that they (eq. (3.24)) highly depend on the accuracy of the critic, which may result in poor advantage estimation. Generalized Advantage Estimation (GAE) [Schulman et al., 2016] emerged as a method to rely more on exact rewards received while traversing a trajectory and less on the value function.

$$\hat{A}^{GAE}(o^t, a^t) = \sum_{\tau=0}^{\infty} (\gamma\lambda) \cdot \delta_{t+\tau}^{V_\pi} \tag{3.27}$$

where $\lambda \in [0, 1]$ is the residual coefficient and $\delta_t^{V_\pi}$ is the TD residual of V_π:

$$\delta_t^{V_\pi} = r^t + \gamma \cdot V_\pi(o^{t+1}) - V_\pi(o^t) \tag{3.28}$$

There are two important cases of λ. When $\lambda = 0$, the GAE estimate is equivalent to eq. (3.24):

$$\hat{A}^{GAE}(o^t, a^t)|_{\lambda=0} = r^t + \gamma V_\pi(o^{t+1}) - V_\pi(o^t) \tag{3.29}$$

When $\lambda = 1$, the GAE is the discounted sum of all future rewards in the trajectory minus the value of the current state, *i.e.,* Monte-Carlo (MC) estimate:

$$\hat{A}^{GAE}(o^t, a^t)|_{\lambda=1} = \sum_{\tau=0}^{\infty} \gamma^\tau \cdot r^{t+\tau} - V_\pi(o^t) \tag{3.30}$$

In PG methods, GAE can replace the advantage estimation (eq. (3.24)) to increase the stability of the training for the actor (eq. (3.22)) by reducing bias.

3.1.2.4 Trust Region and Proximal Policy Optimization

One of the main concerns in the Stochastic Policy Gradient (SPG) [Sutton et al., 1999] is that the policy function might be updated excessively after each gradient ascent step, leading to unstable training. Algorithms such as Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] or Proximal Policy Optimization (PPO) [Schulman et al., 2017] limit the amount of policy updates by penalizing the actor performance loss proportional to the Kullback–Leibler (KL) [Kullback and Leibler, 1951] divergence of the new policy to the old one or applying clipping on the advantage, respectively. Further, instead of increasing or decreasing the raw probability of actions, they consider the *ratio* of change $r(\Theta)$:

$$r_t(\Theta) = \frac{\pi^\Theta(a^t|o^t)}{\pi^{\Theta_{old}}(a^t|o^t)} \tag{3.31}$$

where Θ_{old} is the policy parameters before the gradient ascent step. TRPO's policy performance [Schulman et al., 2015] is calculated as:

$$J(\Theta) = \frac{1}{|\tau|} \sum_t^{|\tau|} r_t(\Theta) \cdot \hat{A}(o^t, a^t) - c_1 \cdot KL\left[\pi^\Theta(\cdot|o^t) \| \pi^{\Theta_{old}}(\cdot|o^t)\right] \tag{3.32}$$

where c_1 is the KL coefficient. PPO replaces the KL penalty of TRPO with a ratio clipping. The rationale is that the KL penalty only applies to the actions, and observation states that the policy function is being trained on does not consider other actions. With this, PPO's policy performance [Schulman et al., 2017] function J will be:

$$J(\Theta) = \frac{1}{|\tau|}\sum_{t}^{|\tau|} \min\left[r_t(\Theta)\cdot \hat{A}(o^t,a^t),\, clip\left(r_t(\Theta), 1-\epsilon, 1+\epsilon\right)\cdot \hat{A}(o^t,a^t)\right] \tag{3.33}$$

where ϵ is the clipping threshold for the changes in action distribution.

Unlike off-policy algorithms (see section 3.1.1), external exploration methods such as adding noise or random actions should not be used with on-policy RL algorithms since the value function, and thus the calculated advantage, is based on policy function. To encourage exploration, an *entropy* [Shannon, 1948a, Shannon, 1948b] bonus can be applied to the performance function:

$$J^E(\Theta) = J(\Theta) + \frac{1}{|\tau|}\sum_{t}^{|\tau|} c_2 \cdot S[\pi^{\Theta}](o^t) \tag{3.34}$$

where c_2 is the entropy coefficient, and $S[\pi^{\Theta}](o^t)$ is the entropy of the policy function at o^t.

3.2 Game Theory

A Normal-Form Game (NFG) n-player game G_n can be expressed as $G_n = \langle P, \{\Pi_p\}, U_p(\boldsymbol{\pi})\rangle$ where, P is set of all players, Π_p is "pure strategy set", and U_p is the "utility profile" for each player $p \in P$. An Extensive-Form Game (EFG) allows for sequential decision making that is described as $\mathcal{G}_n = \langle P, \mathcal{T}, U_p(\boldsymbol{\pi}), \{\mathcal{O}_p\}_{p\in P}, \{\mathcal{A}_p\}_{p\in P}\rangle$, where $\mathcal{T}$ is the game tree, $\mathcal{O}_p$ is the observation rules of player p, and $\mathcal{A}_p(o_p)$ is the action set available to player p when it observes o_p.

3.2.1 Pure Strategy

A *"pure strategy"* for player p is a selection of a single strategy from their set of available strategies Π_p. In a NFG, this is a single action that the player chooses. In an EFG, this is a deterministic policy function which, given the current observation of the player from

the environment, produces an action to be taken by the player:

$$\textit{Pure Strategy}: \ \pi_p(o_p) \mapsto \mathcal{A}_p(o_p) \tag{3.35}$$

3.2.2 Utility Profile

A *utility profile* $U_p(\boldsymbol{\pi}) \mapsto \mathbb{R}$ of policies $\boldsymbol{\pi} \leftarrow \{\pi_p\}_{p \in P}$ gives the amount of "reward" or "utility" that player p receives if all players follow $\boldsymbol{\pi}$.

In a *"zero-sum game"*, all utilities received by all players will sum to zero, regardless of the policy they choose, meaning that a player's reward will come at the cost of other players:

$$\textit{Zero-Sum Game}: \ \sum_{p}^{P} U_p(\boldsymbol{\pi}) = 0 \tag{3.36}$$

in a *"general-sum game"*, this equation does not hold; therefore, with better strategies, all players can potentially increase their utility.

3.2.3 Mixed Strategy

One way to represent a stochastic policy is to have it as a probability distribution over pure strategies. A mixed strategy of agent p is a probability distribution $\sigma_p = \{\sigma_p(\pi_p)\}_{\pi_p \in \Pi_p}$ over the agent's pure strategies Π_p where $\sigma_p(\pi_p)$ is the probability that agent p, chooses policy π_p. Since a mixed strategy σ_p is a probability distribution over a finite set of pure strategies, it satisfies:

$$\forall_{\pi_p \in \Pi_p} \ 0 \leq \sigma_p(\pi_p) \leq 1 \tag{3.37}$$

$$\sum_{\pi_p \in \Pi_p} \sigma_p(\pi_p) = 1 \tag{3.38}$$

We denote Σ_p as the set of all mixed strategies available to agent p. The sum of future discounted utilities of players, when they are following mixed strategies $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma} \leftarrow \{\sigma_p\}_{p \in P}$ can be calculated as:

$$U_p(\boldsymbol{\sigma}) = \sum_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \left(\prod_{p}^{P} \sigma_p(\pi_p) \right) \cdot U_p(\boldsymbol{\pi}) \tag{3.39}$$

where $\boldsymbol{\Pi} \leftarrow (\Pi_1 \times \Pi_2 \times \cdots \times \Pi_p)$. This is directly deducted from probability rules: utility from $\boldsymbol{\pi}$ times the probability that $\boldsymbol{\pi}$ occurs summed over all strategy profiles $\boldsymbol{\pi} \in \boldsymbol{\Pi}$.

3.2.4 Best Response

We assume all players in G are rational, *i.e.,* they always choose a policy that maximizes their utility. Formally, a pure strategy Best Response (BR) for player p (π_p^*) to opponents' strategy profile $\boldsymbol{\pi_{-p}}$ is defined as:

$$\textit{Pure Strategy Best Response}: \ \pi_p^*(\boldsymbol{\pi_{-p}}) = \underset{\pi_p \in \Pi_p}{\operatorname{argmax}} U_p(\{\pi_p, \boldsymbol{\pi_{-p}}\}) \tag{3.40}$$

A similar formulation can be held for a mixed strategy BR:

$$\textit{Mixed Strategy Best Response}: \ \sigma_p^*(\boldsymbol{\sigma_{-p}}) = \underset{\sigma_p \in \Sigma_p}{\operatorname{argmax}} U_p(\{\sigma_p, \boldsymbol{\sigma_{-p}}\}) \tag{3.41}$$

3.2.5 Nash Equilibrium

A strategy profile for all players $\boldsymbol{\pi}^* \leftarrow \{\pi_p^*\}_{p \in P}$ is a Pure-Strategy Nash Equilibrium (PSNE) [Nash, 1951] *iff*:

$$\forall_{p \in P} \forall_{\pi_p \in \Pi_p}: \ U_p(\boldsymbol{\pi}^*) \geq U_p(\{\pi_p, \boldsymbol{\pi}^*_{\boldsymbol{-p}}\}) \tag{3.42}$$

A pure strategy Nash Equilibrium (NE) [Nash, 1951] is not available for all normal-form games. Similar to a pure strategy NE, a mixed strategy profile $\boldsymbol{\sigma}^* \leftarrow \{\sigma_p^*\}_{p \in P}$ is Mixed-Strategy Nash Equilibrium (MSNE) [Nash, 1951] *iff*:

$$\forall_{p \in P} \forall_{\sigma_p \in \Sigma_p}: \ U_p(\boldsymbol{\sigma}^*) \geq U_p(\{\sigma_p, \boldsymbol{\sigma}^*_{\boldsymbol{-p}}\}) \tag{3.43}$$

That is, all players are playing with their BR to all opponents' strategies, and neither player can increase their expected utility without having their opponents change their strategy. In other words, a MSNE for each player p :

$$u(\sigma_p, \boldsymbol{\sigma}^*_{\boldsymbol{-p}}) \leq u(\sigma_p^*, \boldsymbol{\sigma}^*_{\boldsymbol{-p}}) \leq u(\sigma_p^*, \boldsymbol{\sigma_{-p}}) \quad \forall_{\sigma_p \in \Sigma_p, \boldsymbol{\sigma_{-p}} \in \boldsymbol{\Sigma_{-p}}} \tag{3.44}$$

The MSNE for a zero-sum game can be calculated with a *min-max* formulation through a Linear Program (LP) in linear time. For general-sum games, the problem of finding MSNE is PPAD-Complete [Shoham and Leyton-Brown, 2008, Lanctot et al., 2017]; however, it can be calculated with approximation methods such as *ϵ-equilibrium* in polynomial time [McKelvey et al., 2006].

3.2.6 Double Oracles

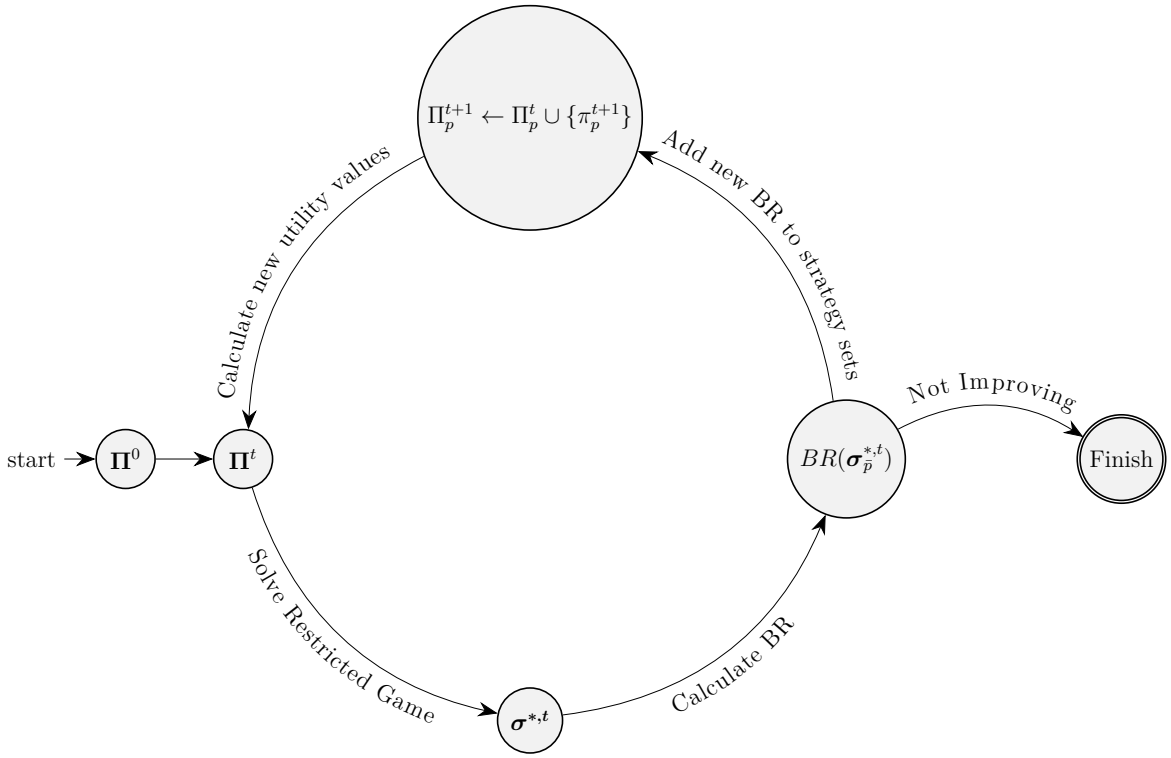


Figure 3.2. The Double Oracle Algorithm

The Double Oracles (DO) [McMahan et al., 2003] is an iterative method used to calculate the MSNE in zero-sum games with large strategy sets. Fig.3.2 shows a schema of this algorithm. The algorithm begins by initializing a small subset of the strategy set for each player p, denoted $\Pi_p^0 \subset \Pi_p$. At each iteration t, the BR algorithm computes the MSNE for each player, denoted as $\sigma_p^{*,t}$, of the subgame G^t that is defined by the current strategy sets $\Pi_p^t : \forall_{p \in P}$.

Once the MSNE is computed, each player calculates their BR pure strategy $\pi_p^{t+1} = \text{argmax}_{\pi_p \in \Pi_p} u(\pi_p, \boldsymbol{\sigma}_{-p}^{*,t})$ which is the pure strategy that maximizes their utility given the mixed strategies $\boldsymbol{\sigma}_{-p}^{*,t}$ of the opponent players, respectively. This best response strategy is added to the player's strategy set, expanding the game:

$$\Pi_p^{t+1} \leftarrow \Pi_p^t \cup \{\pi_p^{t+1}\} \tag{3.45}$$

The process is repeated iteratively, updating the strategy sets in each iteration. The algorithm continues until the MSNE utility of the subgames U_{G^t} converges [McMahan

et al., 2003] to the MSNE of the game G:

$$\forall_{p \in P}: \quad \sigma_p^* = \lim_{t \to \infty} \sigma_p^{*,t} \tag{3.46}$$

$$U_p(\boldsymbol{\sigma}^*) = \lim_{t \to \infty} U_p^t(\boldsymbol{\sigma}^{*,t}). \tag{3.47}$$

3.3 Multi-Agent Reinforcement Learning

A foundational assumption of single-agent RL is that the environment, modeled as an POMDP, is **stationary**. This means that the transition probabilities ($\mathcal{T}$) and reward function ($\mathcal{R}$) are fixed, even if stochastic. However, in a multi-agent environment, this assumption is fundamentally violated. From the perspective of a single agent, the other learning agents are part of the environment. As these agents update their own policies, the environment's effective transition and reward rules change, rendering the environment non-stationary. An agent's past experience quickly becomes obsolete, and conventional RL approaches, such as independent Q-learning, will fail to converge.

Formally, a Multi-Agent Reinforcement Learning (MARL) environment with n agents is described by a Multi-Agent Partially-Observable Markov Decision Process (MAPOMDP) [Albrecht et al., 2024], which is an extension of a POMDP. A MAPOMDP can be viewed as a relaxed version of an EFG (see Section 3.2), where the game tree is relaxed to a transition rule with the Markovian property. It is defined as a tuple:

$$MAPOMDP = \langle P, \mathcal{S}, \{\mathcal{A}_p\}_{p \in P}, \mathcal{T}, \{\mathcal{R}_p\}_{p \in P}, \{\mathcal{O}_p\}_{p \in P} \rangle \tag{3.48}$$

where P is the set of agents, $\mathcal{S}$ is the set of states, $\mathcal{A}_p$ is the action space for agent p, and $\mathcal{O}_p$ is the observation rule for agent p. The state transition rule $\mathcal{T}$ is conditioned on the joint action of all agents: $s^{t+1} \leftarrow \mathcal{T}(s^t, \{a_p^t | a_p^t \in \mathcal{A}_p, p \in P\})$. Similarly, each agent p receives a reward $r_p^t \leftarrow \mathcal{R}_p(s^t, \{a_p^t | a_p^t \in \mathcal{A}_p, p \in P\}, s^{t+1})$ based on the joint action and resulting state. Finally, each agent receives a private observation $o_p^t \leftarrow \mathcal{O}_p(s^t)$.

This MAPOMDP framework gives rise to distinct challenges depending on the nature of the agents' reward functions.

3.3.1 Cooperative MARL and the Credit Assignment Problem

In a fully cooperative setting, all agents share a single team reward function, $\mathcal{R}_p = \mathcal{R}$ for all $p \in P$. While this simplifies the objective, it introduces a significant hurdle known as the **structural credit assignment problem** [Agogino and Tumer, 2004]. When

the team receives a joint reward, it is non-trivial to determine the contribution of each individual agent's action to that outcome. An agent cannot easily deduce whether a good reward resulted from its own "correct" action or from the good actions of others. This problem is studied extensively in MARL literature, with a notable solution being the Counterfactual Multi-Agent Policy Gradient (COMA) [Foerster et al., 2018], which uses a counterfactual baseline to marginalize out a single agent's action.

A popular approach to solving the credit assignment problem is through value function decomposition.

3.3.1.1 Value Decomposition Networks

In a cooperative setting, Value Decomposition Network (VDN) [Sunehag et al., 2018] is used to train agents simultaneously by learning a joint action-value function Q_{tot} that is additively decomposed into individual agents' action-value functions:

$$Q_{tot}^{\boldsymbol{\theta}}(\boldsymbol{o}^t, \boldsymbol{a}^t) = \sum_{p}^{P} Q^{\theta_p}(o_p^t, a_p^t) \tag{3.49}$$

where $\boldsymbol{o}^t$ is the vector of individual observations and $\boldsymbol{a}^t$ is the vector of joint actions. The total Q-function is trained to minimize the standard TD loss:

$$L(\boldsymbol{\theta}) = \frac{1}{|X|} \sum_{x}^{|X|} \left(q_{x,tot}^t - Q_{tot}^{\boldsymbol{\theta}}(\boldsymbol{o}^t, \boldsymbol{a}^t) \right)^2 \tag{3.50}$$

This decomposition allows each agent p to select its action greedily with respect to its own Q^{θ_p}, which simplifies execution:

$$\pi_p(o_p^t) \leftarrow \underset{a'}{\operatorname{argmax}}\, Q^{\theta_p}(o_p^t, a') \tag{3.51}$$

3.3.1.2 QMix

A limitation of VDN is that the simple summation in Equation (3.49) restricts the complexity of the joint value functions that can be represented. QMix [Rashid et al., 2020] improves on this by replacing the linear sum with a parametric, non-linear mixing network. This network takes the joint state (or joint observations) as input and generates the weights for combining the individual Q^{θ_p} values. QMix ensures that the joint Q_{tot} is **monotonic** with respect to each individual Q^{θ_p}, i.e., $\frac{\partial Q_{tot}}{\partial Q_p} \geq 0$. This monotonicity guarantees that performing a greedy action selection on each Q^{θ_p} individually is equivalent

to performing a greedy selection on Q_{tot}, thus ensuring consistency between the centralized training objective and decentralized execution.

3.3.2 Competitive MARL and Centralized Training

In competitive or mixed-motive settings, agents have differing or opposing reward functions. Here, the non-stationarity problem is paramount. A powerful paradigm to address this is **Centralized Training with Decentralized Execution (CTDE)**.

3.3.2.1 Multi-Agent Deep Deterministic Policy Gradients

Multi-Agent Deep Deterministic Policy Gradients (MADDPG) [Lowe et al., 2017] is a prominent Centralized Training Decentralized Execution (CTDE) algorithm suitable for both competitive and cooperative settings. It extends the DDPG algorithm (see Section 3.1.1.4) to the multi-agent domain. The core idea of CTDE is to leverage extra information during training that will not be available during execution. In MADDPG, each agent p learns its own actor (policy) μ^{Θ_p} and a centralized critic Q^{θ_p}.

To solve the non-stationarity problem, the critic Q^{θ_p} is provided with the observations and actions of *all* agents. The critic for agent p thus learns an action-value function of the form:

$$\textit{Action Value Function} : Q^{\theta_p}\left(\boldsymbol{x}^t, \{a_j^t\}_{j\in P}\right) \tag{3.52}$$

where $\boldsymbol{x}$ is a joint state representation and $\{a_j^t\}_{j\in P}$ is the set of actions from all agents. By conditioning on the actions of all agents, the environment becomes stationary from the critic's point of view; an opponent changing its policy does not change the critic's input-output mapping. This stable, centralized critic can then be used to train the agent's decentralized policy $\mu^{\Theta_p}(o_p^t)$.

During *decentralized execution*, each agent p only uses its local observation o_p^t to select an action via its actor:

$$\textit{Policy Function} : \mu^{\Theta_p}(o_p^t) \tag{3.53}$$

To further stabilize training, MADDPG may also learn approximations $\hat{\mu}^{\phi_p^j}$ of other agents' policies, which are used to estimate the next-state value for the critic update.

3.3.3 Game-Theoretic Solutions to Non-Stationarity

An alternative paradigm tackles the non-stationarity problem by framing MARL as a game and explicitly solving for a NE.

3.3.3.1 Policy Space Response Oracles

The Policy Space Response Oracles (PSRO) [Lanctot et al., 2017] algorithm fixes the non-stationarity problem by solving the game using a NE concept. It recasts the MARL problem as finding a NE in an N-player Normal Form Game (NFG), where the "strategy" for each player is not a single action, but an entire policy $\pi \in \Pi$. When other players' parameters (policies) are fixed to a Mixed Strategy Nash Equilibrium (MSNE), the environment becomes stationary for the remaining agent. This insight allows PSRO to cast the problem of multi-agent RL into a series of independent, single-agent RL problems.

PSRO uses the DO method (see Section 3.2.6). Instead of enumerating all possible strategies, which is intractable, PSRO maintains a set of policies for each agent. In each iteration, it computes a MSNE over the current restricted game (using only the policies discovered so far). Then, each agent p trains a new policy π^+ that is an approximate BR to the MSNE policy mixture of its opponents. This BR is found using standard single-agent DRL, which is feasible because the opponents' mixed strategy is fixed, making the environment stationary. The new BR policy is added to the agent's policy set, and the process repeats. This approach is applicable to both competitive and cooperative games. Algorithm 2 shows a pseudocode for this method.

3.4 Convolutional Neural Networks

A Convolutional Neural Networks (CNN) [Venkatesan and Li, 2017] is used for summarizing information from a high-dimensional image. It uses local connections of [Li et al., 2022] each image pixel instead of global connections to share weight between local pixels. Finally, a CNN down-samples image scale using a *pooling layer* [Fukushima, 1980]. A basic *Convolution* operator $\otimes$ is for input image f and convolution kernel h is:

$$[f \otimes h]_{m,n} = \sum_j \sum_k h_{j,k} f_{m-j,n-k} \tag{3.54}$$

Then, a pooling layer summarizes the information from adjacent feature outputs of

Algorithm 2: Policy Space Response Oracles

Require: Initial strategy sets $\mathbf{\Pi}$;
Compute *Expected Utilities* $U^{\boldsymbol{\pi}}$ for each strategy profile $\boldsymbol{\pi} \in \mathbf{\Pi}$;
Compute MSNE of $\mathbf{\Pi}$ as $\boldsymbol{\sigma}^*$;
for *many epochs* **do**
  for *each player* p **do**
    for *many episodes* **do**
      Sample $\boldsymbol{\pi}^*_{-p} \sim \boldsymbol{\sigma}^*_{-p}$;
      Train $\pi^+_p(\boldsymbol{\pi}_{-p})$ using independent RL;
    end
    $\Pi_p \leftarrow \Pi_p \cup \{\pi^+_p\}$;
  end
  Compute $U^{\boldsymbol{\pi}}$ for new strategies ;
  Compute MSNE of $\mathbf{\Pi}$ as $\boldsymbol{\sigma}^*$;
end
Output current solution $\boldsymbol{\sigma}^*$;

the convolution to reduce feature size. For example, a *maxpool* with size 2×2 selects the maximum value of each 2×2 adjacent cell from $[f \otimes h]$ into one cell of output.

3.4.1 Graph Convolutional Networks

Similar to CNN (Section 3.4), a Graph Convolutional Network (GCN) [Kipf and Welling, 2016, Morris et al., 2019] summarizes information from graph nodes and edges using DL. In its simplest form, a GCN layer can be expressed as:

$$H(X) = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW\right) \tag{3.55}$$

where $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with self-connections (I_N is the identity matrix), $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is a diagonal matrix that sums the row i of the adjacency matrix, W are the parameters of this layer, and σ is the activation function of this layer. In short, a GCN averages information from neighboring nodes in a graph with trainable weights into feature vectors. With the help of pooling layers, such as max pooling over neighbors, a GCN can reduce the dimensionality of feature vectors.

Chapter 4
Research Method

To solve the problems of prevention, detection, and mitigation of security threats, first, we model the interactions between the defender, adversary, and the environment as a MAPOMDP (see Section 3.3). Further, using the PSRO (see Section 3.3.3.1), we can cast the problem of MARL into iterations of InRL based on a **black-box** model where neither the adversary nor the defender has access to the policy of the opponent and only observe the result of opponent affecting the environment. The final challenge is finding or designing an appropriate InRL algorithm to find a BR (see Section 3.2.4).

4.1 Strategic Defense as a MAPOMDP

In each scenario, the game model should represent interactions between the adversary and the defender and their joint environment. Each player will receive rewards based on achieving their respective objectives. In the **prevention** scenario, the attacker aims to gain as much knowledge as possible regarding target assets, and the defender tries to obfuscate assets in control to prevent reconnaissance. Thus, the defender can gain a reward based on the information not exposed to the adversary. In the **mitigation** scenario, the attacker aims to inject false data into a CPS, and the defender aims to prevent deviations of the CPS from its nominal point. Here, we can form a MAPOMDP where the attacker's reward is proportional to the system's deviation from its nominal point, and the defender will receive a negative of that amount. Finally, in the **detection**, the adversary aims to inject false data into a transportation network to increase traffic falsely and reroute the vehicles. The defender aims to detect it immediately without raising false alarms. In this case, the attacker will receive a reward proportional to the amount of extended travel time as a result of FDI, and the defender will receive a reward equivalent to the negative of total travel time for all vehicles. The internal system dictates

the rules of the system transition. For example, in the mitigation scheme against FDI in CPS, the system transition rules can be expressed as differential equations pertaining to the system in control. In the detection of FDI in transportation networks, we can assume that the transportation network graph is known, and vehicles choose their path based on the perceived shortest path to their destination. Figure 4.1 shows the interactions between the attacker and the defender with respect to their joint environment.

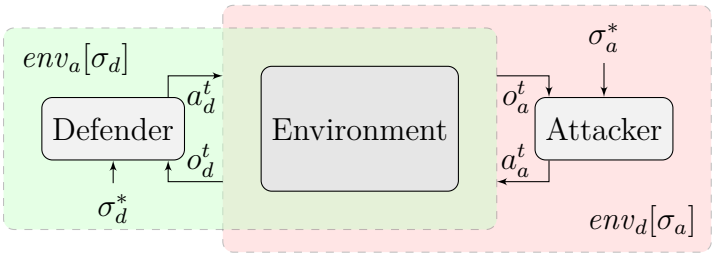


Figure 4.1. A two-player game between the attacker and the defender as a MAPOMDP.

4.2 Two Player Game Solution Concept

The aim of both agents is to maximize their reward. As we are considering a rational adversary and a defender, we can assume that they always pick a strategy that maximizes their own utility. Each agent's sequential decision-making process can be modeled using RL [Sutton and Barto, 2018]. However, training RL agents simultaneously is challenging because the environment is non-stationary from each agent's perspective; as one agent updates its policy, the other agent's optimal strategy changes [Lanctot et al., 2017]. This violates the stationarity assumption required for many RL algorithms. To address this, we model the interaction as an EFG and apply the solution concept of MSNE [Nash, 1951].

A *best response* mixed strategy $\sigma_p^*(\sigma_{-p})$ (see Section 3.2.4) provides maximum utility for agent p given that its opponent $-p$ is using mixed strategy σ_{-p}. Formally, if the opponent $-p$ is using a mixed strategy σ^{-p}, then player p's best response σ_p^* is

$$\sigma_p^*(\sigma_{-p}) = \underset{\sigma_p}{\operatorname{argmax}}\, U_p(\sigma_p, \sigma_{-p}). \tag{4.1}$$

We find each player's optimal strategy, assuming that its opponent always uses a best-response strategy. This formulation is, in fact, equivalent to finding a MSNE of the players' policy spaces Π_a and Π_d. Formally, a profile of mixed strategies (σ_a^*, σ_d^*) is a

MSNE *iff*

$$\forall_{p\in\{a,d\}}\forall_{\sigma_p\in\Sigma_p} : U_p(\sigma_p^*, \sigma_{-p}^*) \geq U_p(\sigma_p, \sigma_{-p}^*) \tag{4.2}$$

That is, neither player can increase its expected utility by unilaterally changing its strategy.

In the cybersecurity domain, the space of all possible policies Π_p is intractably large, making it infeasible to enumerate in search of an MSNE. To overcome this, we leverage the PSRO algorithm [Lanctot et al., 2017], which extends the DO algorithm [McMahan et al., 2003] by using DRL as an approximate best-response oracle. In this framework, we model the game such that a player's pure strategy is the set of parameters $\boldsymbol{\theta}$ (i.e., the neural network weights) that define its DRL policy $\pi_{\boldsymbol{\theta}}$. The utility $U_p(\boldsymbol{\theta}_p, \boldsymbol{\theta}_{-p})$ of a strategy is the expected discounted cumulative reward the agent receives when its policy is parameterized by $\boldsymbol{\theta}_p$ and its opponent's by $\boldsymbol{\theta}_{-p}$. A key insight of this approach is that when one player p commits to a strategy (i.e., fixes its parameters $\boldsymbol{\theta}_p$), the environment for the opponent $-p$ becomes a stationary MDP (or POMDP). The opponent's problem is thus reduced to a single-agent InRL problem, where the goal is to find the best-response policy π_{-p}^+ that maximizes its utility against $\pi_{\boldsymbol{\theta}_p}$. PSRO proceeds iteratively: it computes an approximate best response for one player against the opponent's current MSNE (using DRL as the oracle), adds this new policy (i.e., new set of parameters) to the set of known strategies, and recomputes the MSNE of the restricted game. This process continues until the strategies converge to an approximate MSNE of the full game [McMahan et al., 2003].

Given the different action/observation space types in the environment for each player, *i.e.,* discrete, continuous, or allocation, different InRL algorithms (see Section 3.1) can be used. However, an out-of-the-box general purpose InRL algorithm can not always be used as it might be infeasible to find a BR with that algorithm. For example, in highly-dimensional continuous applications, a simple DDPG (see Section 3.1.1.4) might require millions of experience samples to converge at a solution. The need for calculating a BR multiple times might exacerbate the difficulty of working with these algorithms. This makes the need for development of new InRL algorithms

Algorithm 3: Adaptive Solver

Result: set of pure policies Π^a and Π^d
$\Pi_a \leftarrow$ attacker heuristics;
$\Pi_d \leftarrow$ defener heuristics;
while $U_p(\sigma_p, \sigma_{-p})$ *not converged* **do**
  $\sigma_a, \sigma_d \leftarrow$ solve_MSNE(Π_a, Π_d);
  $\theta \leftarrow$ random;
  $\pi_a^+ \leftarrow$ train($T \cdot N_e, env_a[\sigma_d], \theta$);
  $\Pi_a \leftarrow \Pi_a \cup \pi_a^+$;
  assess π_a^+;
  $\sigma_a, \sigma_d \leftarrow$ solve_MSNE(Π_a, Π_d);
  $\theta \leftarrow$ random;
  $\pi_d^+ \leftarrow$ train($T \cdot N_e, env_d[\sigma_a], \theta$);
  $\Pi_d \leftarrow \Pi_d \cup \pi_d^+$;
  assess π_d^+;
end

4.3 Optimal Defense Decision-Making Framework

We propose our framework to find the MSNE of the game and, therefore, the optimal decision-making policy for the adversary and the defender. Algorithm 3 shows a pseudo-code of our framework.

We start by initializing the adversary's and defender's strategy sets Π_a^0 and Π_d^0 with arbitrary heuristic policies. From this stage, we proceed in iterations. In each iteration, first, we compute an MSNE (σ_a^*, σ_d^*) of the game restricted to the current strategy sets Π_a and Π_d, take the adversary's equilibrium mixed strategy σ_a^* and train an approximate best-response policy $(\pi_d^+(\sigma_a))$ for the defender with a InRL algorithm assuming that the adversary uses σ_a. Next, we add this new policy to the defender's set of policies $(\Pi_d \leftarrow \Pi_d \cup \{\pi_d^+\})$. Then, we do the same for the adversary. First, find the MSNE strategy of the defender (σ_d), and train an approximate best-response policy $(\pi_a^+(\sigma_d))$ for the adversary, assuming that the defender uses σ_d. Then, we add this new policy to the adversary's set of policies $(\Pi_a \leftarrow \Pi_a \cup \{\pi_a^+\})$. Each invocation of InRL, denoted as `train()` in Algorithm 3, receives the limit on the number of training steps T of training and initial parameters $\boldsymbol{\theta}$. Moreover, we let $env_p[\sigma_{-p}]$ denote the InRL environment for player p when its opponent plays a mixed strategy σ_{-p}. During each time step of the InRL training, both players need to decide on an action. The learning agent either chooses an action randomly (*i.e.,* exploration), or follows its current policy. The opponent,

whose strategy is a fixed mixed strategy σ_{-p}, refers to a pure strategy $\pi_{-p} \in \Pi_{-p}$ with probability distribution σ_{-p} and follows that policy. The MSNE payoff evolves over the iterations of the DO algorithm: whenever we add a new policy for a player, which is trained against its opponent's best-mixed strategy so far, the MSNE changes in favor of the player. We continue these iterations until the MSNE payoff of the defender and the adversary ($U_p(\sigma_p^*, \sigma_{-p}^*)$) converges (see Figure 3.2). Formally, we say that the MSNE is converged for both players *iff*

$$\forall_{p \in \{a,d\}} : U_p(\pi_p^+, \sigma_{-p}^*) \leq U_p(\sigma_p^*, \sigma_{-p}^*) \tag{4.3}$$

where σ_p is player p's current MSNE strategy and π_p^+ is the approximate best response found for player p against its opponent's current MSNE strategy. Convergence of the algorithm means that neither the adversary nor the defender could perform better by introducing a new policy.

4.4 Challenges

The first and foremost difficult challenge in finding the best prevention, detection, or mitigation strategy is, in fact, designing a game model. In the prevention problem (see Chapter 5), we used the model of [Prakash and Wellman, 2015]. For the mitigation scheme (see Chapter 6), we developed a practical, though simplistic, model of a CPS. In the detection scheme (Chapter 7), we used the model of data from [Transportation Networks for Research Core Team, 2020] to simulate a transportation network and devised a threat and defense model based on FDI. The second issue that needs to be resolved is selecting or developing an InRL method (`train` function in Algorithm 3). In the [Prakash and Wellman, 2015] model for prevention, the action space for both players is discrete, so it is easy to select DQL (see Section 3.1.1.2) as an InRL algorithm for BR oracle. Similarly, in the mitigation method, as we are dealing with low-dimensional continuous observation/action spaces, a DDPG (see Section 3.1.1.4) algorithm will be the appropriate BR oracle for both players. In the detection method, the defender has a simple discrete action space, either raising or not raising the alarm. Thus, a DQL algorithm will be the BR oracle. However, an out-of-the-box DDPG will not be feasible for the attacker as it has a highly dimensional constrained continuous action space. Hence, we sought to develop a Hierarchical Multi-Agent Reinforcement Learning (HMARL) InRL algorithm as the attacker's BR oracle.

Chapter 5
Attack Prevention Using Moving Target Defense

Moving Target Defense (MTD) is a proactive defense approach that aims to thwart attacks by continuously changing the attack surface of a system, *e.g.,* changing host or network configurations, thereby increasing the adversary's uncertainty and attack cost. To maximize the impact of MTD, a defender must strategically choose when and what changes to make, taking into account both the characteristics of its system as well as the adversary's observed activities. Finding an optimal strategy for MTD presents a significant challenge, especially when facing a resourceful and determined adversary who may respond to the defender's actions. In this section, we propose a multi-agent partially-observable MDP model of MTD based on the model of [Prakash and Wellman, 2015] and formulate a two-player general-sum game between the adversary and the defender. To solve this game, we used the framework of Section 4.3. Finally, we provide experimental results to demonstrate the effectiveness of our framework in finding optimal policies. The findings described here are published at *2020 Conference on Decision and Game Theory for Security* [Eghtesad et al., 2020].

5.1 Model

In this adversarial model, two players, a defender and an adversary ($p = a$ and $p = d$, resp.), compete for control over a set of servers. At the beginning of the game, all servers are under the control of the defender. To take control of a server, the adversary can launch a *"probe"* against the server at any time, which either compromises the server or increases the success probability of subsequent probes. To keep the servers safe, the defender can *"reimage"* a server at any time, which takes the server offline for some time,

but cancels the adversary's progress and control. The goal of the defender is to keep servers uncompromised, *i.e.,* under the defender's control, and available, *i.e.,* online. The goal of the adversary is to compromise the servers or make them unavailable.

5.1.1 Environment and Players

There are M *servers* and two players, a *defender* and an *adversary*. The servers are independent of each other in the sense that they are independently attacked, defended, and controlled. The game environment is explained in detail in the following subsections.

5.1.2 State

Time is discrete, and in a given time step τ, the state of each server i is defined by tuple $s_i^\tau = \langle \rho, \chi, \upsilon \rangle$ where

- $\rho \in \mathbb{N}_0$ represents the number of probes lunched against server i since the last reimage,
- $\chi \in \{adv, def\}$ represents the player controlling the server, and
- $\upsilon \in \{up\} \cup \mathbb{N}_0$ represents if the server is online (*i.e.,* up) or if it is offline (*i.e.,* down) with the identifier of the time step in which the server was reimaged.

5.1.3 Actions

In each time step, a player may take either a single action or no action at all. The adversary's action is to select a server and *probe* it. Probing a server takes control of it with probability $1 - e^{-\alpha \cdot (\rho+1)}$ where ρ is the number of previous probes and α is a constant that determines how fast the probability of compromise grows with each additional probe, which captures how much information (or progress) the adversary gains from each probe. Also, by probing a server, the adversary learns whether it is up or down.

The defender's action is to select a server and *reimage* it. Reimaging a server takes the server offline for a fixed number Δ of time steps, after which the server goes online under the control of the defender and with the adversary's progress (*i.e.,* number of previous probes ρ) against that server erased (*i.e.,* reset to zero).

5.1.4 Rewards

Prakash and Wellman [Prakash and Wellman, 2015] define a family of utility functions. The exact utility function can be chosen by setting the values of preference parameters, which specify the goal of each player. The value of player p's utility function u^p at a particular, as described by Equations (5.1) and (5.2), depends on the number of servers in control of player p and the number of servers offline. Note that the exact relation depends on the scenario (*e.g.,* whether the primary goal is confidentiality or integrity), but in general, a higher number of controlled servers yields a higher utility.

$$u_p(n_p^c, n^d) = w_p \cdot f\left(\frac{n_p^c}{M}, \theta_p\right) + (1 - w_p) \cdot f\left(\frac{n_p^c + n^d}{M}, \theta^p\right) \tag{5.1}$$

where n_p^c is the number of servers that are up and in control of player p, n^d is the number of unavailable (down) servers, and f is a sigmoid function with parameters $\theta_p \leftarrow (\theta_p^{sl}, \theta_p^{th})$:

$$f(x, \theta_p) = \frac{1}{\mathrm{e}^{-\theta_p^{sl} \cdot (x - \theta_p^{th})}} \tag{5.2}$$

Where θ^{sl} and θ^{th} control the slope and position of the sigmoid's inflection point, respectively. Please note that the value of variables used for computation of the utility function (n_p^c, n^d), and therefore, the utility function depends on the time step. However, in writing, the time step is removed explicitly from the formulation since the time step can be understood from the context.

Table 5.1. Utility Environments in MTD Game Model

	Utility Environment	w_a	w_d
0	control / availability	1	1
1	control / confidentiality	1	0
2	disrupt / availability	0	1
3	disrupt / confidentiality	0	0

Reward weight (w_p) specifies the goal of each player. As described by [Prakash and Wellman, 2015], there can be four extreme combinations of this parameter, which are summarized in Table 5.1. For example, in *"control / availability"*, both players gain reward by having the servers up and in their control. Or in *"disrupt / availability"*, which is the most interesting case, the defender gains reward by having the servers up and in its control. In contrast, the adversary gains reward by bringing the servers down or having

them in its control.

The utility function implicitly defines the defender's cost of action. In other words, the cost of reimaging a server comes from not receiving a reward for the time steps when the server is "down." In contrast, the adversary's reward accounts for the cost of probing (C_A), which is a fixed cost that can be avoided by not taking any action.

The reward given to the adversary (r_a^τ) and defender (r_d^τ) at time τ is defined by:

$$r_d^\tau = u_d, \qquad r_a^\tau = \begin{cases} u_a(n_a^c, n^d) - C_A & \text{adversary probed a server at } \tau \\ u_a(n_a^c, n^d) & \text{adversary did nothing at } \tau \end{cases} \tag{5.3}$$

5.1.5 Observations

A key aspect of the model is the players' uncertainty regarding the state of the servers. The defender does not know which servers have been compromised by the adversary. Further, the defender observes a probe only with a fixed probability $1-\nu$ (with probability ν, the probe is undetected). Consequently, the defender can only estimate the number of probes against a server and whether a server is compromised. However, the defender knows the status of all servers, *i.e.,* whether the server is up or down, and if it is down, how many time steps it requires to be back up again).

The adversary always observes when the defender reimages a compromised server, but cannot observe reimaging an uncompromised server without probing it. Consequently, the adversary knows with certainty only which servers are compromised.

Observation of a player p is represented as a vector of tuples o_p^i, where o_p^i corresponds to player p's observation of server i:

$$\boldsymbol{o}_p = \langle o_p^0, o_p^1, \cdots, o_p^{M-1} \rangle \tag{5.4}$$

The adversary knows which servers are compromised and knows how many probes it has initiated on each server.The adversary's observation of server i is defined as a tuple o_a^i:

$$\forall_{0 \le i < M} : \qquad o_a^i = \langle \tilde{\rho}_a, \chi, \tilde{v}_a \rangle \tag{5.5}$$

Where $\tilde{\rho}_a$ is the number of probes launched by the adversary since the last *observed* reimaging, χ is the player controlling the server (always known by the adversary), and $\tilde{v}_a \in \{up, down\}$ is the observed state of the server.

Unlike the adversary, the defender does not know who controls the servers. Further, if ν is greater than 0, the defender can only estimate the number of probes. The observation state of the defender of each server i can be modeled with a tuple o_d^i:

$$\forall_{0 \leq i < M} : \qquad o_d^i = \langle \tilde{\rho}_d, v \rangle \tag{5.6}$$

where $\tilde{\rho}^d$ is the number of probes *observed* since the last reimaging, and $v \in \{up\} \cup \mathbb{N}_0$ is the state of the server (always known by the defender).

5.2 Challenges

Solving the MAPOMDP model of Section 5.1 with the DO algorithm is not straightforward. In the following paragraphs, we discuss the issues faced while solving the MAPOMDP model and propose approaches for resolving these issues.

5.2.1 Partial Observability

For both players, the state is only partially observable. This can pose a significant challenge for the defender, who does not even know whether a server is compromised or not. Consider, for example, the defender observing that a particular server has been probed only a few times: this may mean that the server is safe since it has not been probed enough times, but it may also mean that the adversary is not probing it because the server is already compromised. We can try to address this limitation by allowing the defender's policy to consider a long history of preceding observations; however, this poses computational challenges since the size of the policy's effective state space explodes.

Since partial observability poses a challenge for the defender, we let the defender's policy use information from preceding observations. To avoid state-space explosion, we feed this information into the policy in a compact form. In particular, we extend the observed state of each server, *i.e.,* the number of observed probes and whether the server is online, with (a) the amount of time since the last reimaging r (always known by the defender) and (b) the amount of time since the last observed probe $\tilde{p}_d$. So, the actual state representation of the defender will be:

$$\forall_{0 \leq i < M} : \qquad o_d^i = \langle \tilde{\rho}_d, v, \tilde{p}_d, r \rangle \tag{5.7}$$

where $\tilde{p}_d$ is the time since the last *observed* probe of the server, and r is the time since

the last reimage of the server.

Further, the adversary needs to make sure that the progress of the probes on the servers is not reset. Therefore, it is important that the adversary knows the amount of time since the last probe p of servers when deciding which server to probe. Hence, the observation state of the adversary becomes:

$$\forall_{0 \leq i < M} : \qquad o_a^i = \langle \tilde{\rho}_a, \chi, \tilde{v}_a, p \rangle \tag{5.8}$$

5.2.2 Complexity of MSNE Computation

In zero-sum games, computation of MSNE can be done in polynomial time, *e.g.,* linear programming. However, in general-sum games, the problem of finding the MSNE of given strategy sets of players is PPAD-complete [Shoham and Leyton-Brown, 2008], which makes the computation of true MSNE infeasible for a game of non-trivial size. Therefore, we use an ϵ-*equilibrium* solver, which produces an approximate correct result. One such solver is the Global Newton solver [Govindan and Wilson, 2003].

5.2.3 Equilibrium Selection

Typically, the DO algorithm is used with zero-sum games, where all equilibria of a game yield the same expected payoffs. However, in general-sum games, multiple equilibria may exist with significantly different payoffs. The DO algorithm in general-sum games converges to only one of these equilibria. The exact equilibrium to which the DO algorithm converges depends on the players' initial strategy sets and the output of the best-response oracle. However, in our experiments (Section 5.3.3), we show that in our game, this problem is not significant in practice, *i.e.,* all equilibria yield almost the same expected payoffs (Figure 5.3) regardless of the initial strategy sets.

5.2.4 Model Complexity

Due to the complexity of our MAPOMDP model, computation of best response using tabular RL approaches (*e.g., Q*L see section 3.1.1.3) is computationally infeasible. For example, the size of state space for the defender is $(2T^3)^M$ since each of $\hat{\rho}_d, \hat{p}_d$, and r can take any value between 0 and T, and v can only take two values. Although we expect that the values of $\hat{\rho}_d, \hat{p}_d$, and r be much smaller than T due to the dynamics of the game, it is still infeasible to explore each state even once or store a tabular policy in memory for a game of non-trivial size on a conventional computer. To address this challenge,

we use computationally feasible *approximate best-response oracles* to find approximate best-response strategies instead of best responses. [Lanctot et al., 2017] show that deep reinforcement learning can be used as an approximate best-response oracle. However, convergence guarantees are lost when approximate best responses are used instead of true best responses. Our experiments show that this algorithm converges in only a few iterations (see Figure 5.1).

5.2.5 Short-term Losses vs. Long-term Rewards

For both players, taking an action has a negative short-term impact: for the defender, reimaging a server results in lower rewards while the server is offline; for the adversary, probing incurs a cost. While these actions can have positive long-term impact, benefits may not be experienced until much later: for the defender, a reimaged server remains offline for a long period of time; for the attacker, many probes may be needed until a server is finally compromised.

As a result, with typical temporal discount factors, *e.g.,* $\gamma = 0.9$, it may be an optimal policy for a player never to take any action since the short-term negative impact outweighs the long-term benefit. In light of this, we use higher temporal discount factors, *e.g.,* $\gamma = 0.99$. However, such high values can pose challenges for deep reinforcement learning since convergence will be much slower and less stable.

5.3 Evaluation

In this section, we first describe the heuristic strategies of the MTD game (Section 5.3.1) proposed by [Prakash and Wellman, 2015]. These heuristics can be used as our initial strategies for each player in the DO algorithm (see Section 4.3 and Algorithm 3). Next, we discuss our implementation of the framework (Section 5.3.2). Finally, we present the numerical results (Section 5.3.3).

5.3.1 Baseline Heuristic Strategies

[Prakash and Wellman, 2015] proposed a set of heuristic strategies for both the adversary and the defender that we will describe.

5.3.1.1 Adversary's Heuristic Strategies

- *Uniform-Uncompromised:* Adversary launches a probe every P_A time steps, always selecting the target server uniformly at random from the servers under the defender's control.
- *MaxProbe-Uncompromised:* Adversary launches a probe every P_A time steps, always targeting the server under the defender's control that has been probed the most since the last reimage (breaking ties uniformly at random).
- *Control-Threshold:* Adversary launches a probe if the adversary controls less than a threshold τ fraction of the servers, always targeting the server under the defender's control that has been probed the most since the last reimage (breaking ties uniformly at random).
- *No-Op:* Adversary never launches a probe.

5.3.1.2 Defender's Heuristic Strategies

- *Uniform:* Defender reimages a server every P_D time steps, always selecting a server that is up uniformly at random.
- *MaxProbe:* Defender reimages a server every P_D time steps, always selecting the server that has been probed the most (as observed by the defender) since the last reimage (breaking ties uniformly at random).
- *Probe-Count-or-Period (PCP):* Defender reimages a server which has not been probed in the last P time steps or has been probed more than π times (selecting uniformly at random if there are multiple such servers).
- *Control-Threshold:* Defender assumes that all of the observed probes on a server except the last one were unsuccessful. Then, it calculates the probability of a server being compromised by the last probe as $1 - e^{-\alpha \cdot (\rho+1)}$. Finally, if the expected number of servers in its control is below $\tau \cdot M$ and it has not reimaged any servers in P_D, then it reimages the server with the highest probability of being compromised (breaking ties uniformly at random). In other words, it reimages a server *iff* the last reimage was at least P_D time steps ago and $\mathbb{E}[n_d^c] \leq M \cdot \tau$.
- *No-Op:* Defender never reimages any servers.

5.3.2 Implementation

We implemented the MAPOMDP of Section 5.1 as an Open AI Gym [Brockman et al., 2016] environment. We used Stable-Baselines' [Hill et al., 2018] Deep Q-Network (DQN) implementation. Stable-Baselines internally uses TensorFlow [Abadi et al., 2016] as the neural network framework. For the artificial neural network as our Q approximator, we used a feed-forward network with two hidden layers of size 32 and *tanh* as our activation function. The rest of the parameters are described in Table 5.3. We implemented the remainder of our framework in Python, including the double oracle algorithm. For the computation of the mixed-strategy ϵ-equilibrium of a general-sum game, we used the Gambit-Project's [McKelvey et al., 2006] Global Newton implementation.

We run the experiments on a computer cluster, where each node has two 14-core 2.4 GHz Intel Xeon CPUs and two NVIDIA P100 GPUs. Each node is capable of running ≈ 85 steps of DQL per second, which results in about 1.5 hours of running time per each invocation of the best-response oracle, *i.e.,* DQL training for the adversary or defender. Note that the DQL algorithm is not distributed, so we use only one core of the CPU and one GPU. It is important to note that, in practice, optimal policies can be pre-computed and then executed to mitigate attacks when needed. When policies are executed, inference takes only milliseconds.

5.3.3 Numerical Results

For acquiring the following results, our MTD model is always instantiated using baseline parameters from Table 5.3, unless explicitly specified otherwise.

5.3.3.1 DQL Convergence and Stability

Figure 5.2 shows the learning curve of the agents for their first iteration of the DO algorithm (Iteration 1 and 2). On average, the DQL algorithm converges in $3.88 \cdot 10^5$ steps (49.11 minutes) for the adversary and $1.10 \cdot 10^5$ steps (18.13 minutes) for the defender. We can see that over the iterations of the DO algorithm, the speed of the training decreases. For example, in the first iteration of adversary training, DQL's speed is 131.67 steps per second, while for the first iteration of defender training, DQL's speed reduces to 101.12 steps per second. Further, in the fourth training of the adversary, training speed is decreased furthermore to 52.34 iterations per second.

Since over the iterations of the DO algorithm, the fraction of DQL strategies in both players' MSNE increase (0% vs 51% for the first training), and inference from a DQL

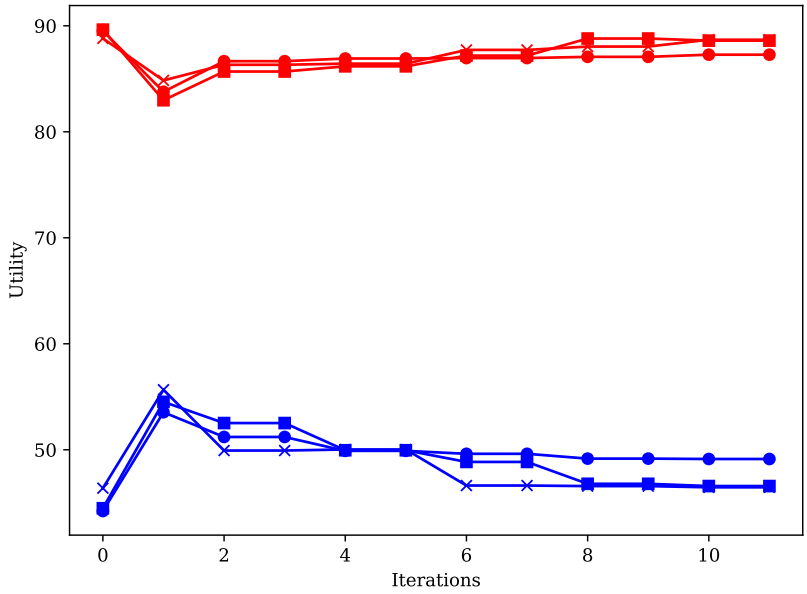


Figure 5.1. Evolution of payoff in MTD's DO iterations. Iteration 0 shows the MSNE payoff of the heuristics while each D*Q*N training for adversary and defender happens at odd and even iterations, respectively.

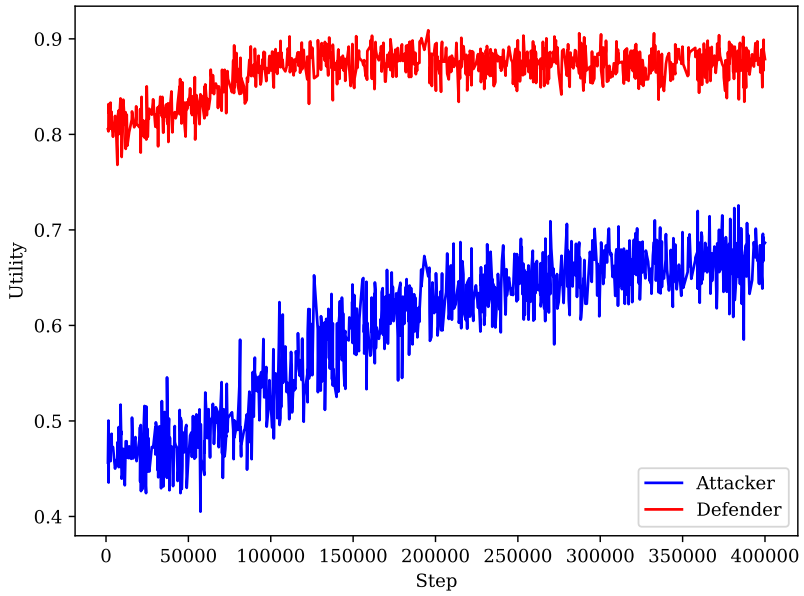


Figure 5.2. The learning curve of the players in the first iteration of DO algorithm in MTD. The blue and red plots show the smoothed episodic reward over the D*Q*L training steps for adversary and defender, respectively.

Table 5.2. Payoff Table for Heuristic and Reinforcement Learning Based Strategies

Adversary \ Defender	No-OP	ControlThreshold	PCP	Uniform	MaxProbe	Mixed-Strategy DQL
No-OP	98.20 / 26.89	98.20 / 26.89	98.20 / 26.89	95.83 / 46.03	98.20 / 26.89	97.47 / 33.23
MaxProbe	47.69 / 78.66	49.62 / 75.67	93.01 / 36.58	67.12 / 64.56	86.82 / 41.99	87.84 / 45.87
Uniform	46.74 / 79.08	51.58 / 70.97	89.48 / 44.43	76.23 / 56.83	75.21 / 57.14	88.16 / 45.91
ControlThreshold	85.98 / 63.64	85.35 / 65.58	88.81 / 46.38	81.32 / 59.54	80.09 / 60.43	87.91 / 45.91
Mixed-Strategy DQL	72.29 / 62.78	82.45 / 58.31	91.32 / 45.76	87.10 / 55.31	91.32 / 44.57	92.38 / 45.23

policy, which requires matrix multiplications, is slower than inference from a heuristic strategy, which requires only a few operations, we can conclude that DQL policies will be more dominant over the iterations. This means that DQL policies are performing better than heuristics over the iterations.

To measure the stability of the DQL algorithm, we extracted the first DQL training for both players. The DQL algorithm converges to almost the same expected cumulative reward with a mean and standard deviation of 0.672 and 0.021 for the adversary and 0.878 and 0.011 for the defender. Table 5.2, which we will discuss in detail later, shows that these policies are significantly better than the heuristics.

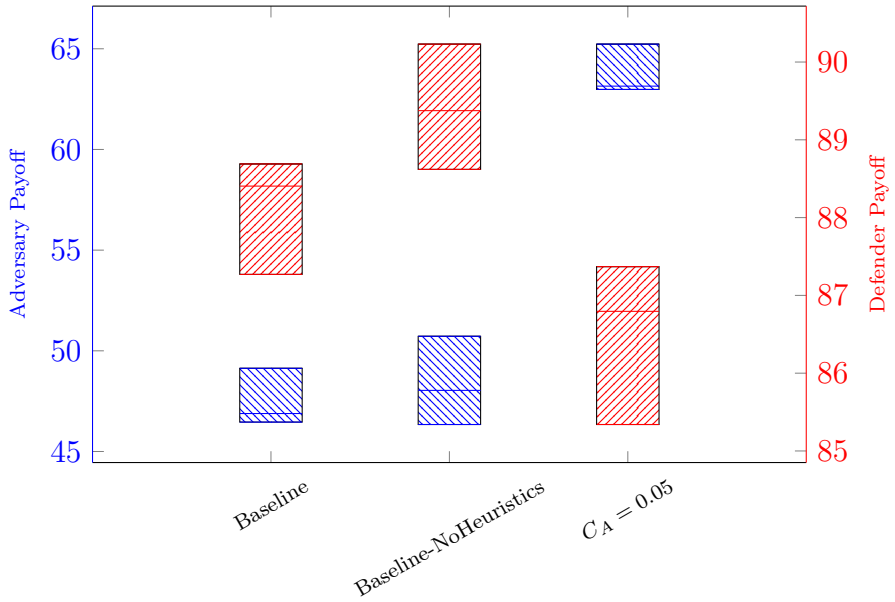


Figure 5.3. Comparison of stability for different configurations in MTD. The blue and red boxes show the adversary's and defender's payoff, respectively. Each box shows the result of eight runs.

5.3.3.2 DO Convergence and Stability

Table 5.3. List of Symbols and Experimental Values for MTD

Symbol	Description	Baseline Value
	Environment, Agents, Actions	
M	number of servers	10
Δ	number of time steps for which a server is unavailable after reimaging	7
ν	probability of the defender not observing a probe	0
α_θ	knowledge gain of each probe	0.05
C_A	attack (*probe*) cost	0.20
θ_p^{sl}	slope of reward function for player p	5
θ_p^{th}	steep point threshold of reward function for player p	0.2
w_p	weighting of reward for having servers up and in control for player p	0 / 1
r_p^τ	reward of player p in time step τ	
	Heuristic Strategies	
P_D	period for defender's periodic strategies	4
P_A	period for adversary's periodic strategies	1
π	threshold of number of probes on a server for *PCP* defender	7
τ	threshold for adversary's / defender's *Control-Threshold* strategy	0.5 / 0.8
	Reinforcement Learning	
T	length of the game (number of time steps)	1000
γ	temporal discount factor	0.99
ϵ_p	exploration fraction	0.2
ϵ_f	final exploration value	0.02
α_t	learning rate	0.0005
$\|E\|$	experience replay buffer size	5000
$\|X\|$	training batch size	32
N_e	number of training episodes	500

Figure 5.1 shows the evolution of MSNE payoff over the iterations of the DO algorithm over three experiments with baseline values of Table 5.3. In this figure, each training for the adversary and defender happens at odd and even iterations, respectively, while iteration 0 is the equilibrium of heuristic policies. The figure shows that the DO algorithm indeed converges with ≈4 training for each player, *i.e.,* 6 hours of training in total. Comparing multiple runs with the same configuration shows that the DO algorithm with multiple approximations (*e.g.,* approximation with deep networks, approximation on equilibrium computation) is stable since the average and standard deviation of the MSNE payoff is 47.81 and 1.77 for the adversary and 88.92 and 1.04 for the defender. For different configurations, the difference between final expected payoffs of eight DO runs is described in Figure 5.3.

5.3.3.3 Equilibrium Selection

To analyze the impact of equilibrium selection on the MSNE payoff of the game, we executed Algorithm 3, but without heuristics as initial strategy sets. Instead, the initial strategy sets are set to only NoOP adversary and NoOP defender. As we can see in Figure 5.3, regardless of the initial strategy sets, the resulting policies always converge to an MSNE with the same payoffs for both players. As a result, equilibrium selection is not an issue in practice since we always end with approximately the same equilibrium.

5.3.3.4 Heuristic Strategies

Table 5.2 shows the utilities for all combinations of the heuristic defender and adversary strategies with baseline parameters. The optimal strategy given only heuristics as players' strategy sets are PCP for the defender and control threshold for the adversary. This table also compares these heuristic strategies to mixed-strategy policies computed using DO and DQL. We can see that it is optimal for both players to commit to the mixed strategy DQL, since no player can receive more utility by committing to another policy. In contrast, the opponent still commits to the DQL policy.

5.3.3.5 Resiliency to Under/Over Estimation

One interesting observation of the DQL policies is their resiliency to under/overestimation of the opponent. As a showcase for when the defender underestimates the adversary, assume a defender who has trained with $C_A = 0.2$ plays with an adversary who is trained with $C_A = 0.05$. They received 88.18 and 61.93 utility, respectively. For the defender, this utility is the same as when it correctly predicts the cost of attack (Figure 5.3).

Chapter 6
Mitigation of False Data Injection in Industrial Control Systems

The escalating threat of cyber-attacks poses a significant challenge to the security of critical infrastructure, necessitating comprehensive strategies for prevention and mitigation. Traditional approaches, focused on securing compromised components, are often time-consuming and may leave systems vulnerable during the intervening period. This section introduces a pioneering solution: employing multi-agent reinforcement learning to formulate resilient control policies. These policies are designed to be promptly deployed, enabling the immediate mitigation of cyber-attacks and minimizing their instantaneous impact, even before fully securing compromised components.

6.1 Model

We devised a threat model for a 0-stealthy FDI adversary ($p = a$) and a model for the mitigation agent, hereon referred to as a defender ($p = d$), to reduce the effects of the attack in different scenarios of different sensor or actuator signals are compromised.

6.1.1 System Model

A physical/chemical/biochemical process receives a set of inputs, processes them, and produces a set of outputs. Figure 6.1 shows a simple feedback control system. The state of the system at a continuous time t can be expressed as a vector of variables $x(t)$. The dynamics of the process are usually described as differential equations:

$$\dot{x} = \frac{dx(t)}{dt} = f(x(t), u(t)) \tag{6.1}$$

Table 6.1. List of Symbols and Experimental Values for the Resilient Control Framework.

Symbol	Description	Baseline Value
	Environment	
$u[t]$	vector of actuation signals at time t	
n	dimension of actuation signals	2
$y[t]$	vector of sensor signals at time t	
m	dimension of sensor signals	2
P	Maximum change ratio to each compromised signal.	0.3
μ	noise mean	0.03
σ^2	variance of noise	0.07
x	state of the system	
$\tilde{x}$	desired state of the system	
C^u, C^y	probability of compromisation of actuation/sensor signals, resp	0.5
S^y, S^u	set of sensor and actuator signals, resp.	
$\tilde{S}$	set of compromised signals (scenario)	
ϵ	closeness condition	0.01
$L^C[t]$, $L^A[t]$	Loss of controller and adversary at time t, resp	
	Agents	
$u_a^y[t]$, $u_a^u[t]$	the amount of change to observation/actuation, resp.	
$y_d[t]$, $u_d[t]$	controller's sensor reading/actuation signal at time t	
$U_d[t]$, $U_d[t]$	utility of attacker and controller at time t, resp.	
π^A, π^C	pure policy of adversary and controller, resp.	
Π_a, Π_d	set of all pure policies available to the adversary and controller, resp.	
σ_a, σ_d	mixed strategy of adversary and controller, resp.	
$\pi_*(\sigma)$	best response pure policy to strategy σ	
PT	payoff table (refered to as a matrix)	
	Reinforcement Learning	
T	total number of steps of DDPG training	$5 \cdot 10^5$
T_{epoch}	Maximum number of steps for each epoch	200
γ	discount factor	0.90
α_μ	learning rate of actor	10^{-4}
α_Q	learning rate of critic	10^{-3}
e	probability of random action in each step	0.1
$\|R\|$	size of replay buffer	$5 \cdot 10^4$
$\|X\|$	neural network fitting batch size	128
θ^Q, θ^μ	parameters of the Q and μ functions, resp	

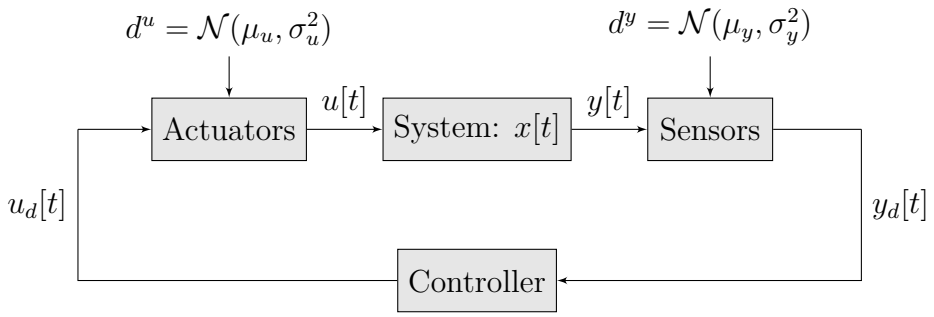


Figure 6.1. A feedback control system. $x[t]$ is the vector process variables, $u_d[t]$ is the vector of actuator signals of the system, and $y_d[t]$ is the vector of sensor signals.

$$y(t) = g(x(t), u(t)) \tag{6.2}$$

where u is the actuator signal of the system, and y is the sensor readings. However, as we are dealing with discrete-time models, we convert these equations to discrete time:

$$\dot{x} = \frac{dx(t)}{dt} \mapsto \Delta x[t] = x[t+1] - x[t] \tag{6.3}$$

$$\text{eqs. (6.1) and (6.2)} \Rightarrow \begin{cases} \Delta x[t] & = f(x[t], u[t]) \\ y[t] & = g(x[t], u[t]) \end{cases} \tag{6.4}$$

The controller's goal is to keep the system's state at a point $\tilde{x}$. As a result, to measure the performance of the controller, we assume a loss function depending on the distance of the current state of the system to the desired state:

$$L_d[t] = \|x[t] - \tilde{x}\| \tag{6.5}$$

While we are using the Euclidean norm, and we assume only one desired point, any other loss metric, *e.g.,* absolute distance or distance from two points, is also acceptable.

The state of the system is not necessarily observable, so the controller relies on the m sensor signals :

$$y_d[t] \in \mathbb{R}^m \tag{6.6}$$

To change the state of the system toward the desired point or keep it at that point, the controller sends n signals to the actuators:

$$u_d[t] \in \mathbb{R}^n \tag{6.7}$$

Note that sensor and actuator signals are not necessarily accurate, *i.e.,* each sensor and actuator signal has normally distributed noise $d = \mathcal{N}(\mu, \sigma^2)$.

6.1.2 Threat Model

By finding vulnerabilities in the ICS communication network or software or by gaining physical access to ICS devices, an adversary can exploit the ICS to sabotage the physical process by changing the controller's sensor readings or actuation commands. Figure 6.2 shows the closed loop feedback control system of Figure 6.1 when an adversary has compromised the sensor/actuator signals of the controller.

In our threat model, we assume that the adversary has already compromised the set of actuation signals (S^u) and/or the set of sensor signals (S^y) and intends to change them. We denote the compromised signals as a *scenario* $\tilde{S} \subseteq S^u \cup S^y$.

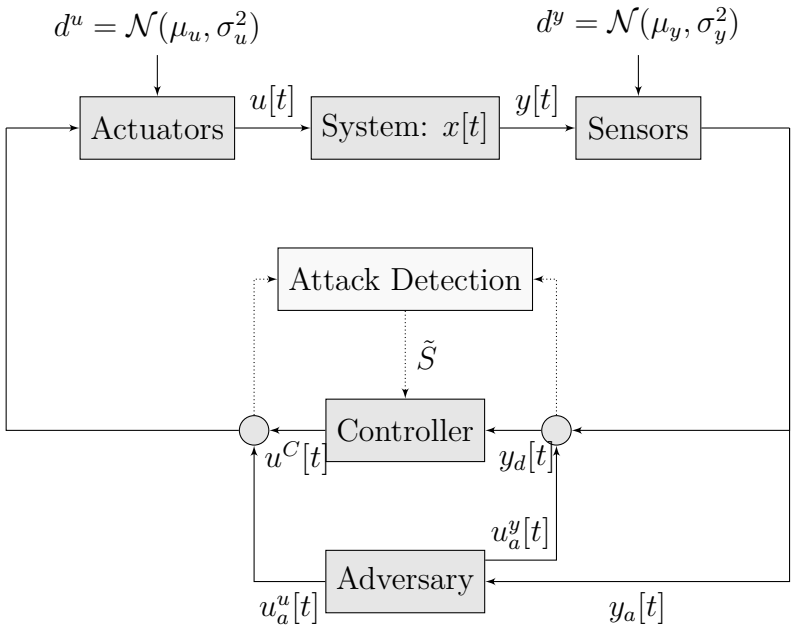


Figure 6.2. Controlling a physical system when an adversary is present. y_a and y_d are the observations of the adversary and controller, resp. $u_a^u \cup u_a^y$ is the action of the adversary. u_d is the action of the controller. $\tilde{S}$ is the set of compromised signals.

The adversary observes all of the sensor signals of the system. Also, the adversary knows the scenario of the attack. We denote the observation of adversary at time t as $o_a[t]$:

$$o_a[t] = \langle y_a[t], \tilde{S} \rangle \tag{6.8}$$

$$y_a[t] = y[t] + d^y[t]$$

where y_a is the sensor readings of the system, and $d^y = \mathcal{N}(\mu, \sigma^2)$ is the normal noise of the sensors. Also, the scenario for the attack ($\tilde{S}$) can be represented as a two-dimensional vector ($\tilde{S} \in \{0,1\}^2$), where $\tilde{S}_0$ shows whether the sensor signals are compromised, and $\tilde{S}_1$ shows whether actuator signals are compromised.

For each sensor or actuator signal that the adversary has compromised, the adversary will choose a change fraction between $-P$ and P to alter that signal. The value of the altered signal will be $v \cdot (1+p)$ where v is the original value, and $p \in [-P, P]$ is the signal change chosen by the adversary.

Formally, the action space of the adversary is:

$$\begin{aligned} u_a[t] &= \langle u_a^y[t], u_a^u[t] \rangle \\ u_a^y[t] &= \begin{cases} [-P,P]^m & \text{if } S^y \subset \tilde{S} \\ [0]^n & \text{if } S^y \not\subset \tilde{S} \end{cases} \\ u_a^u[t] &= \begin{cases} [-P,P]^n & \text{if } S^u \subset \tilde{S} \\ [0]^n & \text{if } S^u \not\subset \tilde{S} \end{cases} \end{aligned} \tag{6.9}$$

where u_a^u is the fraction of change to the controller's actuation signals, and u_a^y is the fraction of change to the controller's sensor signals.

The goal of the adversary is to deviate the state of the system from the controller's desired state. To measure the performance of the adversary based on this model, we assume a loss function for the adversary based on the loss of the controller:

$$L_a[t] = -L_d[t] = \|x[t] - \tilde{x}\| \tag{6.10}$$

This is a conservative assumption since we are assuming a worst-case attack (given a set of compromised components). Again, we should note that our proposed framework (Section 4.3) would work with other reasonable distance metrics as well.

6.1.3 Controller Model

As mentioned above, we assume that there exists an attack detection scheme [Giraldo et al., 2019, Paridari et al., 2018, Urbina et al., 2016] which tells the controller the scenario for the attack ($\tilde{S}$), *i.e.,* which set of signals are compromised. Further, the controller

can observe the changed sensor readings of the system. We denote the observation of controller at time t as $o_d[t]$:

$$o_d[t] = \langle y_d[t], \tilde{S} \rangle \tag{6.11}$$
$$y_d[t] = (y[t] + d^y[t]) \odot (1 + u_a^y[t])$$

where element-wise multiplication of vectors is denoted by $\odot$. Again, the scenario for the attack, *i.e.,* set of compromised signals, can be represented as a two-dimensional vector ($\tilde{S} \in \{0,1\}^2$) where $\tilde{S}_0$ shows whether the observation signals are compromised, and $\tilde{S}_1$ shows whether actuation signals are compromised. Further, when the adversary has compromised the actuator signals, the actuators receive these values:

$$u[t] = (u_c[t] \odot (1 + u_a^u[t])) + d^u[t] \tag{6.12}$$

6.2 Preliminary Results

This section describes the technical details of how we implemented our framework (Section 6.2.1). Then, to demonstrate the effectiveness of the proposed solution, we test our framework on two physical systems with known differential equations: a bioreactor (Section 6.2.2.1) and a three coupled tanks system (Section 6.2.2.2).

6.2.1 Implementation

Table 6.2. Rest points of the bioreactor with differential equations of Equation (6.13) and parameters of Table 6.3.

x_1^*	x_2^*	
0	4	Stable
0.995103	1.512243	Unstable
1.530163	0.174593	Stable

We converted the dynamics of the physical processes to discrete-time based on Equation (6.3) and implemented them as OpenAI Gym [Brockman et al., 2016] environments. We used Stable-Baselines' DDPG [Hill et al., 2018] as the implementation of our single-agent learning. Stable-Baselines internally uses TensorFlow [Abadi et al., 2016] as the neural network framework. We implemented the remainder of our framework in Python, including the double oracle algorithm and equilibrium computation. In our case, as

we have a zero-sum game, we formulated the problem of finding mixed-strategy Nash equilibrium as a linear program [Shoham and Leyton-Brown, 2008], and solved it with the Python library *scipy.optimize.linprog*.

As the critic (Q) and actor (μ) approximators, we used multilayer-perceptron neural networks with 4 hidden layers, each having 64 neurons with Exponential Linear Unit (elu) activation function (since our utilities are negative for the controller) and the parameters from Table 6.1.

For helping with the convergence of the μ values, we added a +100 "extra" utility to the controller when the distance of the state of the system to the desired state is less than ϵ (i.e., when $\|x[t] - \tilde{x}\| \leq \epsilon$). This extra utility is also included in the reported utilities of agents when evaluating them.

6.2.2 Test Systems

Here, we describe our two test systems, a bioreactor (Section 6.2.2.1) and a three-tanks system (Section 6.2.2.2), and show the effectiveness of our proposed solution. For each physical process, we present its functionality, describe the dynamics of the process, and show numerical results from each step of our framework. All parameters for training are instantiated based on Table 6.1.

6.2.2.1 Bioreactor

A bioreactor is a system designed to provide some environmental conditions that are required to carry out a biochemical process. For example, a bioreactor might be used for processing pharmaceuticals or food that involves the use of microorganisms or substances derived from them. Particularly, processes focused on the growth of organisms (also called biomass) should provide a batch of organisms with food in order to promote population growth.

For the experiments, we have used the bioreactor system model that was introduced by [Barreto et al., 2013]. This bioreactor has two inputs, the dilution rate (D) and the substrate feed (x_{2f}), and two state variables, x_1 and x_2. Further we assume that the state of the system is fully observable, *i.e.,* there are sensors for both the biomass and substrate amount placed in the bioreactor.

Table 6.3. List of Parameters for the Bioreactor

Symbol	Unit	Description	Value
Parameters			
μ_{max}	hr^{-1}		0.53
k_m	$g/liter$		0.12
k_1	$liter/g$		0.4545
Y	ratio	ratio of the change in the population mass and the change in the substrate mass	0.4
Nominal Input / Output			
D	g/s	substrate feed (or food income)	0.3
x_{2f}	g/s	output biomass flow	4
State Variables			
x_1	g	amount of biomass	
x_2	g	amount of substrate	

This biochemical process can be modeled with:

$$\Delta x = x[t+1] - x[t] = \begin{cases} \Delta x_1 & = (\mu[t] - D) \cdot x_1[t] \\ \Delta x_2 & = D \cdot (x_{2f} - x_2[t]) - \frac{\mu[t]}{Y} x_1[t] \end{cases}$$
$$\mu[t] = \mu_{max} \cdot \left(\frac{x_2[t]}{k_m + x_2[t] + k_1 \cdot x_2[t]^2} \right) \tag{6.13}$$

The parameters of the bioreactor are presented in Table 6.3. With nominal dilution rate $D = 0.3$, and substrate feed $x_{2f} = 4.0$, the bioreactor has three rest points, one of which is unstable. These rest points are shown in Table 6.2. We can form a game where the goal of the controller is to keep the system at the unstable rest point $\tilde{x} = \langle 0.995, 1.512 \rangle$, and the adversary tries to deviate the state of the system from this point by changing the sensor and/or actuator signals. Further, we assume that the actuators of this system will accept any positive real number.

In Figure 6.3, we show that DDPG performs very well in finding best-response policies for our proposed framework since after only $3 \cdot 10^5$ steps, it learns how to keep the state of the system, *i.e.,* keep the distance of the state of the system less than ϵ from the desired state, where no attack is present.

Figure 6.4 shows the evolution of mixed strategy equilibrium payoff ($U^C(\sigma_*^C, \sigma_*^A)$) over steps of the double-oracle algorithm. As we can see, the mixed strategy equilibrium has converged in 12 iterations or less, meaning that the algorithm has already explored all sets of possible attack/control policies. We run the experiments on a workstation with

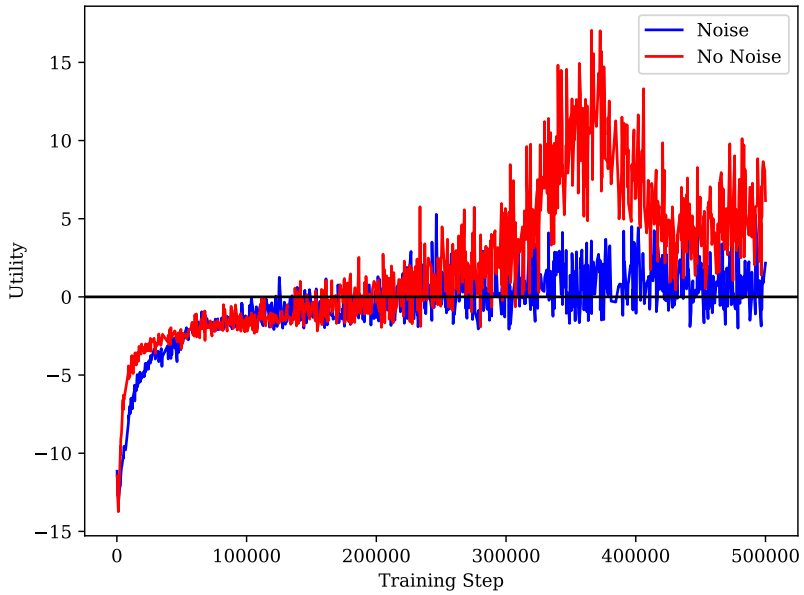


Figure 6.3. Learning curve of the controller in bioreactor system.

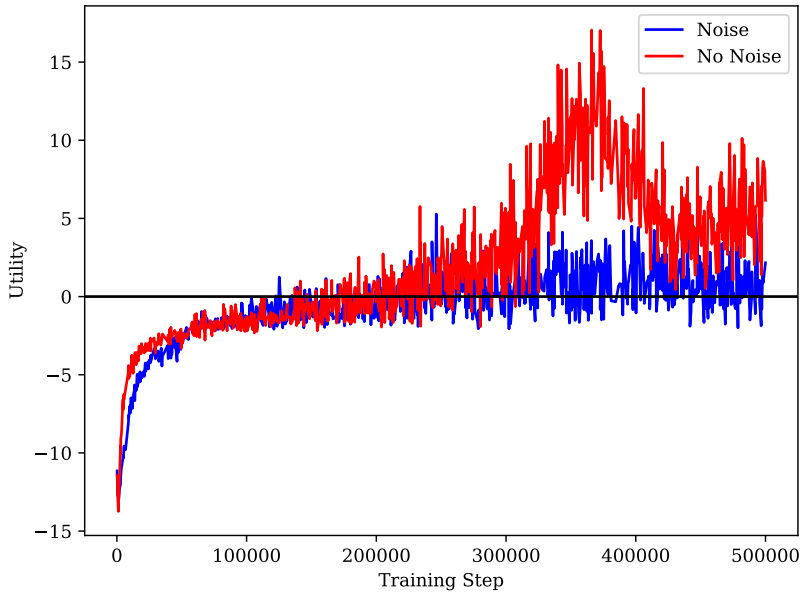


Figure 6.4. Evolution of the bioreactor system controller's equilibrium payoff ($U_d(\sigma_d^*, \sigma_a^*)$) over iterations of the double oracle. Simple means when the adversary has compromised only the actuation signals (*i.e.*, $\tilde{s} = S^u$). In Multi Scenario, $C^y = C^u = 0.5$.

20 Core 2.4 GHz Intel®Xeon®CPU and NVIDIA®Titan RTX™GPU. This workstation is capable of running ≈ 140 steps of DDPG per second, which yields to 2 hours per each iteration of the double oracle algorithm. Further, the DDPG algorithm is not distributed, so we only need to use one core of the CPU. This paves the way for multiple DDPGs to run at the same time. We also need to mention that, set of policies only needs to be pre-computed. After the attack, these policies are only deployed without any additional computation.

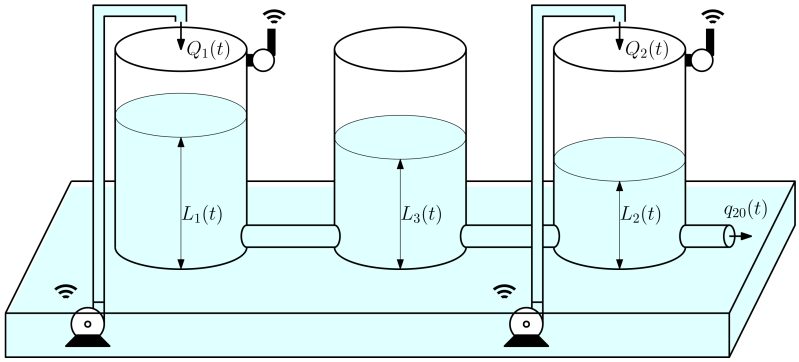


Figure 6.5. Schema of the three tanks system. Diagram taken from [Combita et al., 2019].

To show the effectiveness of the framework in mitigating adversarial tampering, we have compared a non-resilient policy (π_d^0) with a resilient policy (σ_d^*) in Figure 6.6. As you can see, the resilient control policy (mixed-strategy equilibrium of the controller) is able to keep the state of the system close to the desired state while an adversary is present.

Table 6.4. Payoff Table of Bioreactor's Defender

		Attacker Policy Generated in Iteration														
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Defender Policy Generated in Iteration	0	-1062.5278	-60.7144	-133.2385	-135.3946	-223.8062	-667.9065	-106.8297	-80.9442	-115.3957	-116.9765	-83.9764	-122.2647	-40.4468	-103.4953	-169.0730	-130.0481
	1	-37.2163	-754.5798	-1893.1854	-1956.2957	-37.5236	-48.8347	-81.3956	-81.6147	-1790.0800	-1123.2222	-159.1248	-1477.9965	-890.1891	-1143.9990	-1706.2990	-67.2577
	2	-367.1605	14.0463	-48.4147	-37.5867	-1000.5052	-1263.1415	-38.6504	15.6583	-61.5425	-34.8168	-134.8090	-34.4913	-90.7459	31.6843	-30.4202	-215.4627
	3	-413.8956	89.0763	86.2773	84.4174	-790.2409	-909.6236	-87.1434	-115.0079	89.3830	90.7394	67.0225	-69.6258	-52.9248	-35.1016	78.6845	-231.5823
	4	-1926.4229	-20.7814	-17.9397	-14.4014	-1904.1029	-1908.2865	-96.1622	-120.1073	-43.7463	-20.2101	-29.1217	-22.7784	2.4853	4.2434	-27.6401	-235.5610
	5	-68.0506	-129.8021	-159.5376	-155.0538	3.8488	-52.8699	-180.1517	-187.3842	-157.3390	-139.1703	-78.9755	-144.7794	-105.2223	-146.3316	-154.7047	-68.3322
	6	-107.1579	-1628.9873	-1656.9748	-2038.9951	12.4141	-54.9482	-2470.8423	-2325.1877	-2081.0609	-2252.1969	-1715.3721	-2035.9816	-1431.1542	-1650.1598	-1749.4343	-1821.2800
	7	-95.1620	-1146.7708	-2167.7975	-2382.9580	90.2800	-0.9087	-1280.4411	-1244.6693	-1602.8200	-2380.2582	-1635.2055	-2377.3544	-2377.8033	-2132.5308	-1818.9371	-1753.0500
	8	-91.2814	-1736.0559	-2368.7374	-2147.3210	12.1659	-50.6632	-1907.0340	-1242.4375	-1529.3686	-2154.1480	-1976.4891	-2366.4138	-2129.1564	-2092.4342	-1810.7624	-1487.4191
	9	-76.5098	-189.7340	-231.4714	-226.2535	91.2756	-23.0871	-109.9416	-76.4330	-226.8518	-220.7357	-200.5393	-228.7897	-97.5295	-224.1690	-232.1124	-60.0890
	10	-120.5652	-121.5800	-233.1025	-212.2048	89.4213	-35.5731	-184.2321	-142.8168	-238.3319	-158.6177	-126.6337	-198.3248	-102.5119	-205.6803	-218.7598	-43.2830
	11	-79.3808	-178.3255	-189.1195	-191.3982	52.0420	15.1543	-116.8980	-83.8733	-195.9938	-198.1456	-197.3299	-182.4141	-137.2828	-202.9991	-187.9795	-68.5034
	12	-69.2062	-138.9754	-380.4202	-209.2603	87.0969	19.3129	-115.8547	-89.7844	-200.4015	-329.7046	-506.6450	-190.2273	-673.3889	-192.2152	-213.4839	-86.5613
	13	-112.7862	-225.4718	-1325.4476	-653.5512	-35.8491	-47.5547	-2197.2286	-196.0760	-1081.9079	-235.6581	-235.2000	-876.6508	-291.5608	-202.3613	-430.4403	-286.0190
	14	-57.6284	-126.6927	-198.8452	-195.9530	42.6847	-41.0893	-121.5689	-61.9094	-195.0703	-196.3210	-191.0047	-279.9479	-94.7251	-168.4408	-269.1706	-72.3701
	15	-64.1027	-189.0744	-161.5968	-154.8786	80.4412	25.1787	-179.2542	-111.9868	-154.9419	-167.8632	-199.7539	-155.9191	-329.3882	-147.8971	-150.6461	-76.2398
	16	-250.4163	-1806.4048	-2179.6851	-1493.0887	-274.1134	-489.8190	-579.3843	-1248.6494	-1499.9731	-1589.2914	-1660.9171	-2005.2848	-1809.5406	-1136.3496	-1264.5406	-1357.2924

Furthermore, Table 6.4 shows the payoff table for all the controller and adversary policies that have been generated by the double oracle algorithm over its iterations. In each iteration, one new row (controller's policy), and one new column (adversary's policy)

is added to the table. This newly added policy is the best response against the opponent's previously generated strategies. Each cell (i, j) of the payoff table shows the utility of the controller when the controller uses the pure approximate strategy found at the i-th iteration of the double oracle algorithm, and the adversary uses the j-th iteration policy. Figure 6.4 and Table 6.4 tell the same story: At first, when the controller has not been trained against any attack ($PT_{0,0}$), it receives a huge negative utility when it encounters an attack. In the first iteration ($PT_{1,0}$), the adversary does not know that the controller is mitigating the attacks, so it receives a small positive utility. After 12 iterations, the adversary and controller reach a payoff equilibrium where neither player can improve its own utility by finding another policy (Figure 6.4).

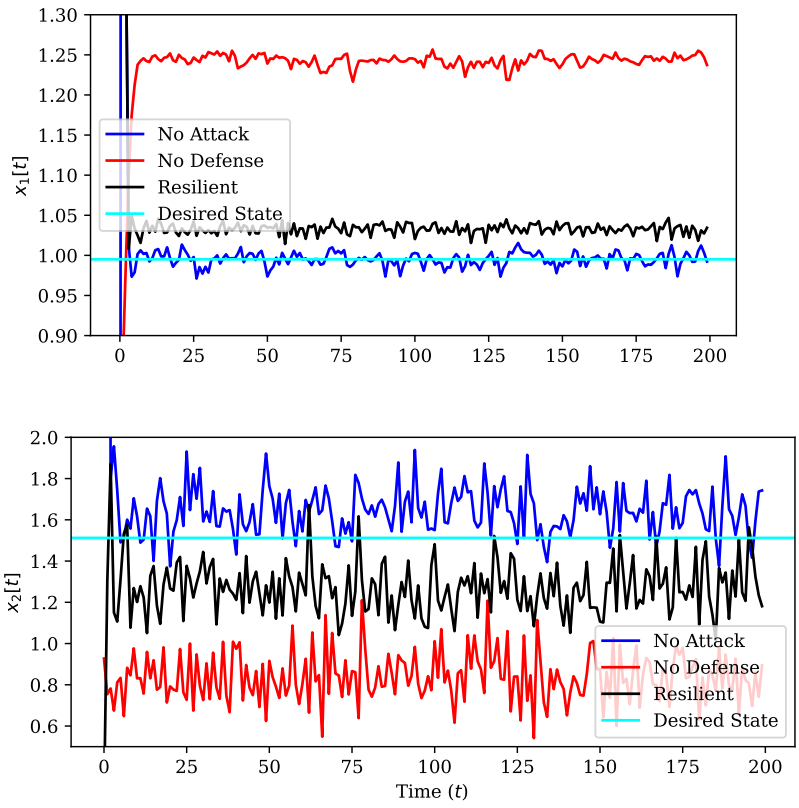


Figure 6.6. Comparison of attacks and resilient control policies on the state of the bioreactor. *No Attack* refers to the base controller policy (π_d^0) when no attacker is present. *No Defense* is the same controller policy against the mixed strategy equilibrium of the adversary (σ_a^*). *With Defense* shows the state of the system when both adversary and controller choose policies based on the mixed strategy equilibrium (σ_d^* vs. σ_a^*). These measurements were performed in a noisy environment with scenario $\tilde{S} = \{S^y, S^u\}$.

6.2.2.2 Three Tanks

Our second test system is the three-tanks system introduced by [Combita et al., 2019], which consists of three intercoupled water tanks. The controller can change the input flow of the first and second tanks. There are sensors measuring the water level of the first and second tanks. Figure 6.5 shows the schema of this system.

Table 6.5. List of Parameters for the Three Tanks System

Symbol	Unit	Description	Value
Parameters			
S	m^2	Tank cross section area	0.0154
S_n	m^2	Pipe cross section area	5×10^{-5}
μ_{13}	ratio	Outflow coefficient	0.5
μ_{32}	ratio	Outflow coefficient	0.5
μ_{20}	ratio	Outflow coefficient	0.6
$\max_{Q_i} \mid i \in \{1,2\}$	m^3/s	Maximum flow rate	1.5×10^{-4}
$\max_{L_j} \mid j \in \{1,2,3\}$	m	Maximum tank level	0.62
Nominal Input / Output			
Q_1	m^3/s	Tank 1 input flow	3.5×10^{-5}
Q_2	m^3/s	Tank 2 output flow	3.75×10^{-5}
State Variables			
$L_i \mid i \in \{1,2,3\}$	m	Tank level	

The nonlinear dynamics of this system are obtained using first-principles [Combita et al., 2019]. The approach of first-principles is based on the use of physical laws to describe the dynamic evolution of a system. In this specific case, a balance of mass is used to obtain the differential equations which are the model of the system. The dynamics of this system are as follows:

$$
\begin{aligned}
S\Delta L_1[t] &= Q_1[t] - q_{13}[t] \\
S\Delta L_2[t] &= Q_2[t] + q_{32}[t] - q_{20}[t] \\
S\Delta L_3[t] &= q_{13}[t] - q_{32}[t] \\
\frac{1}{\mu_{13} S_n} q_{13} &= sgn[L_1[t] - L_3[t]] \sqrt{2g|L_1[t] - L_3[t]|} \\
\frac{1}{\mu_{32} S_n} q_{32} &= sgn[L_3[t] - L_2[t]] \sqrt{2g|L_3[t] - L_2[t]|} \\
\frac{1}{\mu_{20} S_n} q_{20} &= \sqrt{2gL_2[t]}
\end{aligned}
\tag{6.14}
$$

where the parameter values are shown in Table 6.5, and $g \approx 9.80665\ m/s^2$ is the standard

acceleration of gravity. If we fix the nominal intake flows of the system as $Q_1 = 3.5 \times 10^{-5}$ and $Q_2 = 3.75 \times 10^{-5}$ (Table 6.5), we obtain the operation point of the system as $L_1 = 0.4$, $L_2 = 0.2$, and $L_3 = 0.3$. We assume the controller tries to keep the system stable in this state $\tilde{x} = \langle 0.4, 0.2, 0.3 \rangle$.

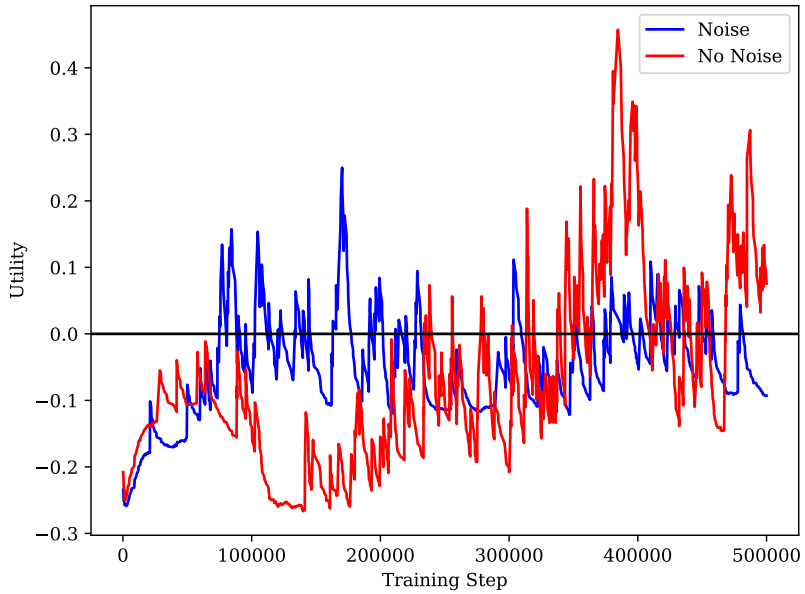


Figure 6.7. Learning curve of the controller in three tanks system.

The learning curve of the DDPG for the controller in the three-tank system is presented in Section 6.2.2.2 when no attack is ongoing. Again, to show the effectiveness of the resilient control policies, we compare three cases in Figure 6.8: 1) the state of the system when no adversary is present, 2) the state of the system when the controller chooses a control policy that considered no attack (π_d^0) versus the equilibrium strategy of the adversary (σ_a^*), and (3) state of the system with resilient control policies, *i.e.,* both the adversary and controller are choose policies based on the mixed-strategy equilibrium (σ_d^* vs. σ_a^*). We can see that 1) the worst case adversary (σ_a^*) can deviate the state of the system from the desired point with a norm of 0.126, while an attack resilient controller only lets deviations with a norm of 0.047. This is a 62% improvement.

The evolution of the controller's mixed-strategy equilibrium payoff over iterations of the double oracle algorithm is shown in Figure 6.9.

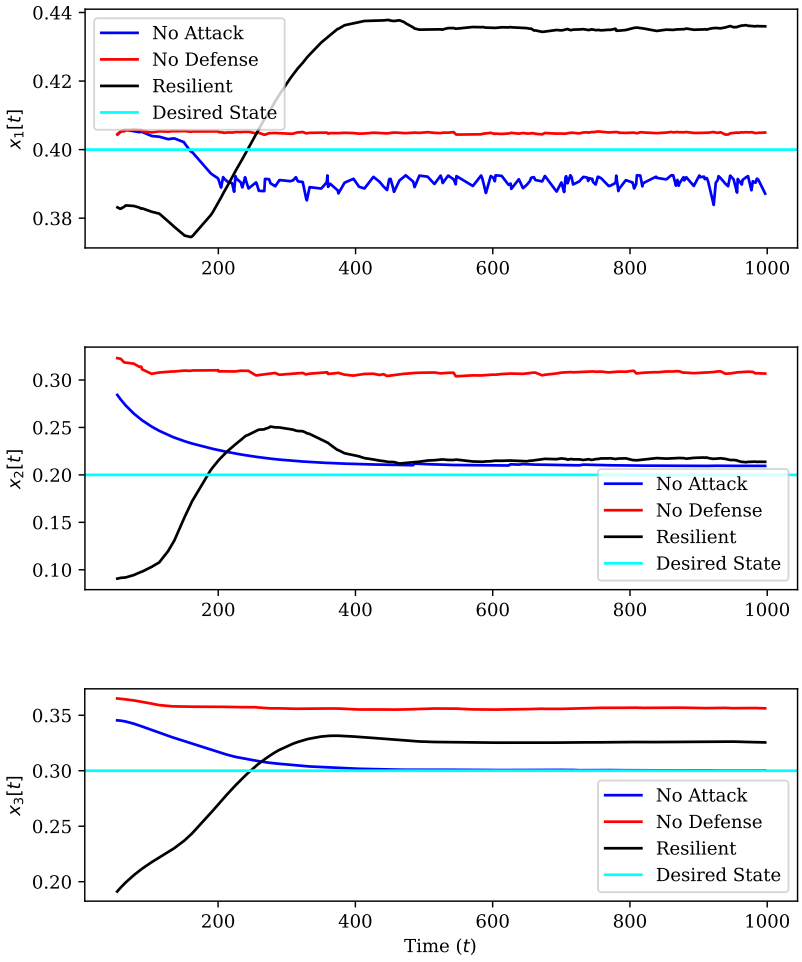


Figure 6.8. Comparison of non-resilient and resilient control policies based on the state of the three tanks. *No Attack* refers to the base controller policy (π_d^0) when the controller was trained, not to expect an attack. *No Defense* is the same controller policy against the mixed-strategy equilibrium of the adversary (σ_a^*). *With Defense* shows the state of the system when both adversary and controller choose policies based on the mixed strategy equilibrium (σ_d^* vs. σ_a^*) with scenario $\tilde{S} = \{S^y, S^u\}$.

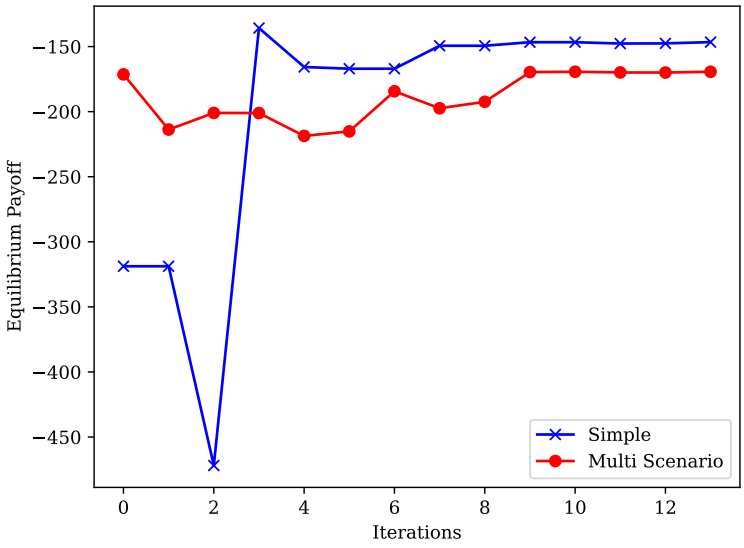


Figure 6.9. Evolution of the controller's equilibrium payoff ($U_d(\sigma_d^*, \sigma_a^*)$) over iterations of the double oracle algorithm in three tanks system. In multi scenario, $C^y = C^u = 0.5$. For simple scenario: $\tilde{S} = S^u$

6.2.3 Discussion

There are major differences between the three-tank system and the bioreactor. First and the most important one is that in bioreactor, the state of the system is fully observable, however, in three tanks system, only two out of three state variables are observable. This reduces the reliability of the DDPG algorithm since Q-values are not a direct function of observation (Section 6.2.2.2 vs. Figure 6.3).

Second, the action space of the controller in the bioreactor is much bigger than the action space of the controller in three tanks. This results in (1) the number of steps for the controller in three tanks to converge to the desired state is much bigger than the bioreactor's controller, and (2) control of three tanks will be much smoother without any jitter compared to the bioreactor (Figure 6.6 vs Figure 6.8).

Also, to determine the number of steps T for each execution of DDPG, we rely on the *TD Error* of the Q network. DDPG can be terminated when the TD Error converges to a sufficiently low value. However, it is a challenge to detect the optimal value for TD Error, since it depends on the stochastic rules of the environment and the discount factor (γ). As a result, we decided on a static number of $T = 5 \cdot 10^5$ steps. In our experiments (Section 6.2.2.2 and Figure 6.3), we show that the decided T is, in fact, a good choice.

We also experimented on training different policies for different scenarios, *i.e.,* simple, instead of combining them in one policy, *i.e.,* multi-scenario. The results show that generalizing over scenarios decreases the policy's gained utilities by less than 4%. This measurement was done by comparing simple MSNE strategies with multi-scenario MSNE strategies against the opponent's simple MSNE strategy for different scenarios and averaging them with the probability of occurrence of scenario (Equation (3.38)).

Chapter 7
Assessment of False Information Injection in Navigation Application

The increasing reliance of drivers on navigation applications has made transportation networks more susceptible to data manipulation attacks by malicious actors. Adversaries may exploit vulnerabilities in the data collection or processing of navigation services to inject false information, thus interfering with the drivers' route selection. Such attacks can significantly increase traffic congestion, resulting in a substantial waste of time and resources, and may even disrupt essential services that rely on road networks. We introduce a computational framework based on single-agent RL to assess the threat posed by such attacks to find worst-case data-injection attacks against transportation networks. This framework is then used as a BR oracle for the attacker to find the best defense strategy for the defender. For this purpose, first, we devise an adversarial model with a threat actor who can manipulate drivers by increasing the travel times that they perceive on certain roads. Then, we employ hierarchical multi-agent reinforcement learning to find an approximate optimal adversarial strategy for data manipulation. We demonstrate the applicability of our approach by simulating attacks on the Sioux Falls, SD, network topology. The findings here are accepted for publication at *The 23rd International Conference on Autonomous Agents and Multi-Agent Systems* [Eghtesad et al., 2024]. Using our HMARL algorithm as a BR attacker oracle for our adaptive solver framework (see Section 4.3) to devise a FDI detection algorithm.

7.1 Model

In this section, we devise and formalize our threat model with respect to a transportation network environment where the adversarial agent ($p = a$) injects false traffic information with a restricted budget with the aim of increasing the total travel time of vehicles traveling in this network. Further, we devised a detection model for a defender ($p = d$) to detect such FDI.

7.1.1 Environment

The traffic model is defined by a *road network* $G = (V, E)$, where V is a set of nodes representing road intersections, and E is a set of directed edges representing road segments between the intersections. Each edge $e \in E$ is associated with a tuple $e = \langle t_e, b_e, c_e, p_e \rangle$, where t_e is the free flow time of the edge, c_e is the capacity of the edge, and b_e and p_e are the parameters for the edge to calculate actual edge travel time $W_e(n_e)$ given the congestion of the network, where n_e is the number of vehicles currently traveling along the edge [Transportation Networks for Research Core Team, 2020]. Specifically, we use the following function for $W_e(n_e)$:

$$W_e(n) = t_e \times \left(1 + b_e \left(\frac{n_e}{c_e}\right)^{p_e}\right) \tag{7.1}$$

The *set of vehicle trips* are given with R, where each trip $r \in R$ is a tuple $\langle o_r, d_r, s_r \rangle$, with $o_r \in V$ and $d_r \in V$ the origin and destination of the trip, respectively, and s_r the number of vehicles traveling between the origin-destination pair $\langle o_r, d_r \rangle$.

7.1.2 State Transition

For each vehicle trip $r \in R$ at each time step $t \in \mathbb{N}$, *vehicle location* $l_r^t \in V \cup (E \times \mathbb{N})$ represents the location of vehicle r at the end of time step t, where the location is either a node in V or a tuple consisting of an edge in E and a number in $\mathbb{N}$, which represents the number of timesteps left to traverse the edge.

Each vehicle trip begins at its origin; hence $l_r^0 = o_r$. At each timestep $t \in \mathbb{N}$, for each vehicle trip r that $l_r^{t-1} \in V \setminus \{d_r\}$, i.e., the vehicle trip is at a node but has not reached its destination yet, let $\oslash_r^{t-1} = (l_r^{t-1}, e_1, v_1, e_2, v_2, \ldots, e_k, d_r)$ be a shortest path from l_r^{t-1} to d_r considering congested travel times $\boldsymbol{w}^t$ as edge weights. Then $l_r^t = \langle e_1, \lfloor w_e^{t-1} \rceil \rangle$, where

the travel time of edge e is

$$w_e^t = W_e \left(\sum_{\left\{ r \in R \,\middle|\, l_r^{t-1} = \langle e, \cdot \rangle \right\}} r_s \right). \tag{7.2}$$

Thus, for a trip r with $l_r^{t-1} = \langle e, n \rangle$, i.e., the vehicle is traveling along an edge, if $n = 1$, that is, the vehicle is one time step from reaching the next intersection, $l_r^t = v_1$. Otherwise, $l_r^t = \langle e, n-1 \rangle$.

7.1.3 Attacker Model

At the high level, our attack model involves adversarial perturbations to *observed* (rather than actual) travel times along edges e, subject to a perturbation budget constraint $B \in \mathbb{N}$. Let $a_e^t \in \mathbb{R}$ denote the adversarial perturbation to the observed travel time over the edge e. The budget constraint is then modeled as $\|\boldsymbol{a}^t\|_1 \leq B$, where $\boldsymbol{a}^t$ combines all perturbations over individual edges into a vector. The observed travel time over an edge e is then

$$\hat{w}_e^t = w_e^t + a_e^t. \tag{7.3}$$

It is this observed travel time that is then used by the vehicles to calculate their shortest paths from their current positions in the traffic network to their respective destinations. Since we aim to develop a defense that is robust to informational assumptions about the adversary, we assume that the attacker completely observes the environment at each time step t, including the structure of the transit network G, all of the trips R, and the current state of each trip l_r.

The attacker's goal is to maximize the total vehicle travel times, which we formalize as the following optimization problem:

$$\max_{\{\boldsymbol{a}^1, \boldsymbol{a}^2, \ldots\} : \forall t (||\boldsymbol{a}^t||_1 = B)} \sum_{t=0}^{\infty} \gamma^t \cdot \sum_{\{r \in R \,|\, l_r^t \neq d_r\}} r_s, \tag{7.4}$$

where $\gamma \in (0, 1)$ is a temporal discount factor.

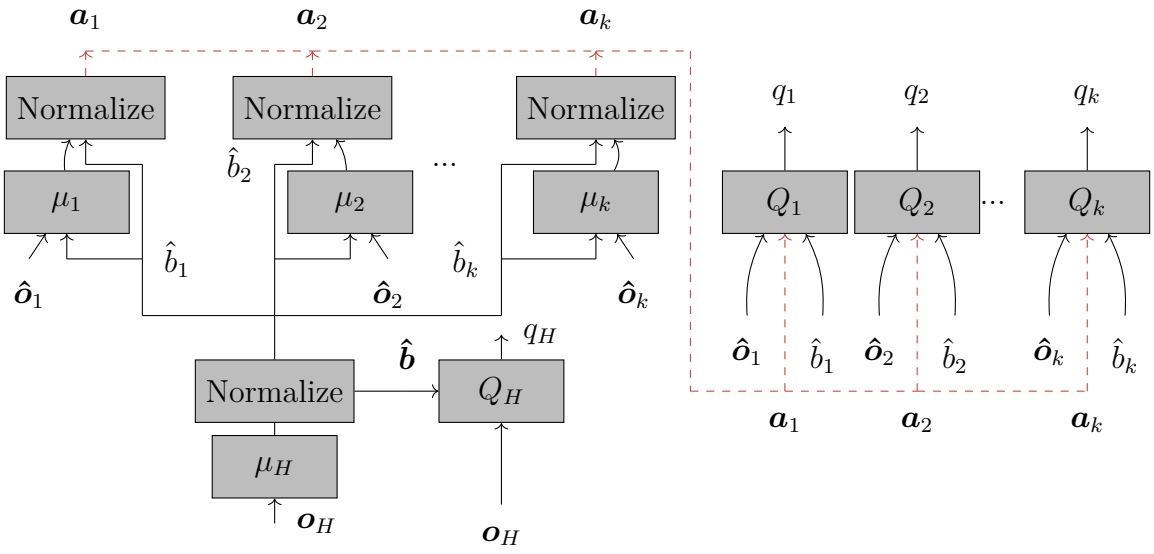


Figure 7.1. Hierarchical Multi-Agent Deep Reinforcement Learning Architecture. μ_H and Q_H are the high-level agent's actor and critic function approximators, respectively. μ_k and Q_k are the actor and critic function approximators of low-level agent k, respectively. $\boldsymbol{a} = \langle \boldsymbol{a_1} \times \hat{b}_1, \boldsymbol{a_2} \times \hat{b}_2, \cdots \boldsymbol{a_k} \times \hat{b}_k \rangle$ is the perturbations of all edges of the transit graph G where a_k is the perturbations of edges in component k. $\boldsymbol{o_k}$ and $\hat{b}_k$ are the observation of the k-th agent from its component and the proportion of budget allocated to it, respectively. The *Normalize* layer can be constructed using the *Softmax* function or the 1-norm normalization of ReLU-activated actor outputs.

7.1.4 Defender Model

The defender observes the edge travel times $\hat{\boldsymbol{w}}^t$ at each time t, and aims to learn a detector $D(\hat{\boldsymbol{w}}^t)$ that takes observed travel times as input, and returns a prediction whether or not these are due to an adversarial perturbation. If the defender identifies an ongoing attack at time t, all future perturbations are thereby prevented, i.e., $\boldsymbol{a}^\tau = 0$ for all $\tau \geq t$. Failure to detect attacks entails direct consequences in terms of increased travel times as formalized in the attack model in Equation (7.4) (which the defender aims to minimize). On the other hand, false positives (alerts triggered when no attack is taking place) incur a fixed cost c. The complete defense model and the training algorithm are presented in section 8.2.

7.2 Challenges

At each timestep of the game, the adversary needs to find the approximately optimal perturbations to all the edges in a city network G.

The action space for the low-level agent is $|E|$-dimensional. Given a moderate-sized city such as Anaheim, CA or Chicago, IL, that has 914 and 2950 road links, respectively [Transportation Networks for Research Core Team, 2020], it is infeasible for a Single-Agent RL algorithm to learn the optimal budget allocation strategy.

This requires that the transit network be broken down into components. Then, an RL agent will be responsible for the edges in the component, observing the information pertaining to the component and only finding the optimal perturbations for that component.

Approaches such as MADDPG will fail in this scenario as the agents will compete over the budget, making the MDP difficult to learn. This makes the need to devise a two-level hierarchical multi-agent reinforcement learning algorithm where the purpose of the high-level agent is to allocate the budget to the components, eliminating the competition over budget, and the purpose of the low-level agent, which itself is comprised of component agents, is to further allocate the perturbation budget between the edges in their component constrained to the allocated budget to the component by the high-level agent.

7.3 Hierarchical Multi-Agent Reinforcement Learning

To form our HMARL framework, first, we need to divide the transportation system into sub-components; then, for these components, we devise a MARL framework based on Multi-Agent Deep Deterministic Policy Gradients (Section 3.3.2.1). Finally, a high-level agent based on Deep Deterministic Policy Gradients (Section 3.1.1.4) will coordinate the component agents. Algorithm 5 shows the training workflow of the HMARL.

7.3.1 K-Means Node Clustering

First, these components can be formed by applying a K-Means clustering algorithm, assuming the distance between two nodes is the shortest path distance given edge weights $w_e = t_e$. Then, each edge $e = uv$ is assigned to the component of its source node u. Algorithm 4 shows a pseudocode for the K-means clustering algorithm. Figure 7.2 shows the decomposition of the Sioux Falls, SD transportation network with K-Means clustering into four components.

Algorithm 4: K-Means Graph Clustering

Result: c_e for all $e \in E$; The centroid node for all edges.
Calculate all-pairs shortest path distance $d_{u,v} : \forall_{u,v \in V}.$;
Select $|K|$ initial nodes as component centers arbitrarily and call them $c_k \in K$;
for n_iterations **do**
  $c_u \leftarrow \text{argmin}_v \, d_{u,v} : \forall_{v \in C}.$;
  $c_k \leftarrow \text{argmin}_{c_{k'}} \text{argmax}_u \, d_{u,c_{k'}}$ such that $c_u = c_k.$;
end
$c_e = \langle uv \rangle \leftarrow c_u \forall e \in E$

7.3.2 High and Low-Level DRL Agents

We assume that the adversarial agent has access to all features of the transportation network G and all rider information l_r^t at all time steps. Thus, it can summarize the information into features that can be used to train the high and low-level agents. Figure 7.1 summarizes our HMARL architecture.

7.3.2.1 Low-Level Multi-Agent MADDPG

When graph G is broken down into $|K|$ components, the agent supervising component $k = \hat{G}(\hat{V}_k, \hat{E}_k) \subset G(V, E)$ observes a feature vector of $\hat{\boldsymbol{o}}_k^t = \langle \langle s_e, n_e, \hat{s}_e, m_e, \tilde{s}_e \rangle : \forall_{e \in \hat{E}_k} \rangle$, where $s_e = \sum \{ s_r | l_r^t \in V \wedge e \in \oslash_r^t : \forall_{r \in R} \}$ is the number of vehicles that are currently at a node with an unperturbed shortest path to the destination passing through e, $\hat{s}_e = \sum \{ s_r | l_r^t \in V \wedge e = \oslash_r^t(e_1) : \forall_{r \in R} \}$ is the number of such vehicles that will immediately take e, $m_e = \sum \{ s_r \cdot n | l_r^t = \langle e', n \rangle \wedge e' = e : \forall_{r \in R} \}$ is the sum of required timesteps for vehicles traveling e to arrive at its endpoint, and $\hat{s}_e = \sum \{ s_r | e \in \oslash_r^{t-1} : \forall_{r \in R} \}$ is the number of all vehicles taking e as their shortest path at some timestep assuming the perceived travel times to remain unchanged. The agent then outputs a vector of perturbations $\boldsymbol{a}_k^t = \langle a_e^t | e \in \hat{E}_k \rangle$ to perturb all the components' edges. This agent would receive a reward $r_k^t = \sum \{ s_r | l_r^t \in \hat{G}(\hat{V}_k \hat{E}_k) : \forall_{r \in R} \}$ as the number of vehicles in its component.

As the low-level agents participate in a cooperative setting with our hierarchical approach, they do not need to see other agents' actions to train their critics. The Q function for each agent can be constructed with $Q^k(\hat{\boldsymbol{o}}_k^t, \hat{\boldsymbol{b}}_k^t, \boldsymbol{a}_k^t)$ with a Multi-Layer Perceptron (MLP) such that its output is activated with a Rectified Linear Unit (ReLU) as the reward for each component is non-negative, i.e., the number of vehicles in the component. The agent's action function $\mu_k(\hat{\boldsymbol{o}}_k^t, \hat{\boldsymbol{b}}_k^t) \mapsto \boldsymbol{a}_k^t$ can be constructed using

an MLP. As the output of the actor function of k-th low-level agent needs to sum to $\hat{b}_k$ to satisfy the budget and allocation constraint, it needs a normalizing function that can be either a Softmax function or 1-norm normalizer. The final perturbations can then be drawn by multiplying budget of the component to its action output $\boldsymbol{a} = \langle \boldsymbol{a_1} \times \hat{b}_1, \boldsymbol{a_2} \times \hat{b}_2, \cdots \boldsymbol{a_k} \times \hat{b}_k \rangle$. The training of μ_k and Q_k functions can be performed according to the MADDPG algorithm (see section 3.3.2.1).

Algorithm 5: Hierarchical Multi-Agent Reinforcement Learning

Require: A road network graph $G = (V, E)$; Set of Riders R;;
Initialize environment *env*;
Initialize Replay Buffer E.;
Run K-Means Clustering to acquire components;
$s \leftarrow env.reset()$;
for *step* $\Rightarrow$ *total steps* **do**
  $\hat{\boldsymbol{b}} \leftarrow \mu_H(\hat{s})$;
  $\boldsymbol{a} \leftarrow \odot\mu_k(\hat{b}, s) + \mathcal{N}$;
  Normalize $\boldsymbol{a}$;
  $s', r \leftarrow env.step(\boldsymbol{a})$;
  $E \leftarrow E \cup \langle s, s', \boldsymbol{a}, r \rangle$;
  $s \leftarrow s'$;
  if $env.done()$ **then**
    $s \leftarrow env.reset()$;
  end
  Sample $\hat{E} \sim E$;
  Update Q_H, μ_H, $Q_k, \mu_k \forall_{k \in K}$ with $\hat{E}$;
end

7.3.2.2 High-Level DDPG Agent

The high-level agent H observes an aggregated observation of the components at time t, specifically the number of vehicles in the component and number of vehicles that are making a decision in that component $\boldsymbol{o}_H^t = \langle \langle \sum_e^{\hat{E}_k} n_e, \sum_e^{\hat{E}_k} \hat{s}_e \rangle : \forall_{k \in K} \rangle$, and outputs $\hat{\boldsymbol{b}} \in [0, B]^{|K|}$ such that $\|\hat{\boldsymbol{b}}^t\|_1 = B$ the portion of the budget allocated to each component. The high-level agent is rewarded by the total number of the vehicles in the network $r_H^t = \sum_{\{r \in R | l_r^t \neq d_r\}} s_r$.

Similar to the low-level agent actors, the output of the high-level actor is normalized with either Softmax or 1-norm and then multiplied by the total budget B to allocate each budget to the component. The training of the high level μ_H and Q_H can be performed

according to the DDPG algorithm (Section 3.1.1.4).

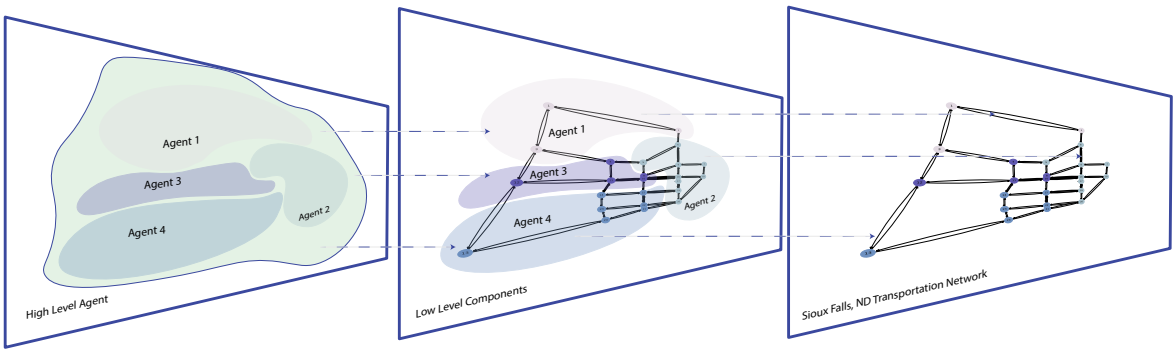


Figure 7.2. Decomposition of Sioux Falls, ND transportation network into four components, where one low-level agent is responsible for adding perturbation to edges in each component, and one high-level agent is responsible for allocating budget B to each low-level agent. Edge width represents the density of vehicles moving over the edge without any attacker perturbation added.

Table 7.1. Neural Network Architecture for HMARL

Hyperparameter	Value
High-Level	
Actor μ_H	
Number of hidden layers	2
Sizes of layers	[256, 128]
Activation function	ReLU
Optimizer	Adam
Critic Q_H	
Number of hidden layers	2
Sizes of hidden layers	[128, 128]
Activation function	ReLU
Optimizer	Adam
Low-Level	
Actor μ_k	
Number of hidden layers	2
Sizes of hidden layers	[512, 512]
Activation function	[ReLU, ReLU, Sigmoid]
Optimizer	Adam
Critic Q_k	
Number of hidden layers	2
Sizes of hidden layers	[128, 128]
Activation function	ReLU
Optimizer	Adam

7.4 Evaluation

We simulated the framework using benchmark data from [Transportation Networks for Research Core Team, 2020] and evaluated the effectiveness of HMARL for finding an optimal strategy for false information injection on the Sioux Falls, ND testbed.

7.4.1 Experimental Setup

Table 7.2. List of Hyperparameters for the HMARL

Hyperparameter	Value
Environment	
Training Horizon	400
Evaluation Horizon	50
$\|K\|$ Number of Components	4
Total Training Steps	200,000
Randomizing Factor of Number of Vehicles	0.05
Common	
τ_H, τ_k target network transfer rate	0.001
Training Batch Size	64
Experience Replay Buffer Size	50,000
Stand Alone High Level	
μ_k Learning Rate	0.00005
Q_k Learning Rate	0.01
γ_H	0.99
τ_H	0.001
Noise Decay Steps	10,000
Stand Alone Low Level	
μ_k Learning Rate	0.00005
Q_k Learning Rate	0.01
γ_k	0.99
Noise Decay Steps	30,000
Hierarchical	
Low-Level	
μ_k Learning Rate	0.00005
Q_k Learning Rate	0.01
γ_k	0.9
Noise Decay Steps	10,000
High-Level	
μ_H Learning Rate	0.00001
Q_H Learning Rate	0.001
γ_H	0.99
τ_H	0.001
Noise Decay Steps	30,000
Standalone DDPG	
μ Learning Rate	0.00001
Q Learning Rate	0.001
γ	0.99
Noise Decay Steps	30,000

To make the environment non-deterministic, we randomly increased or decreased r_s by 5% at each training episode's beginning. We simulate the environment by following a vehicle-based simulation based on the state transition rules of Section 7.1.2.

7.4.1.1 Hardware and Software Stack

The experiments, including the neural network operations, are done on an Apple MacBook Pro 2021 with an M1 Pro SoC with eight processing cores and 16GB of RAM. None of the experiments, including the neural operations, have been done on the Metal Performance Shaders. The simulation of the environment has been implemented using Python. For neural network operations, we used PyTorch [Paszke et al., 2019]. We used NumPy [Harris et al., 2020] as our scientific computing library.

7.4.1.2 Seeds and Hyperparameters

To make sure that the results presented in this article are reproducible, we initialized the random seeds of Numpy, PyTorch, and Python to zero. The hyperparameters used for the simulation and the training of high and low-level agents are presented in Table 7.2 and the neural network architectures are presented in Table 7.1.

7.4.2 Heuristics

We used a *Greedy* heuristic as our baseline strategy. In the greedy approach, the adversarial agent counts the number of vehicles s_e passing through each edge e as their unperturbed shortest path to their destination. Then its applied perturbation will be

$$\boldsymbol{a} = \frac{\langle s_e : \forall e \in E \rangle}{\sum_{e \in E} s_e} \times B.$$

When running the ablation study (see Figure 7.3) and testing the high-level and low-level agents separately, we replaced the high-level with a *proportional allocation*, meaning that each component agent gets a proportion of the budget relative to the number of vehicles making a decision in that component. Further, the low-level agent can be replaced with a local greedy that perturbs the edges in its component relative to the number of vehicles passing through the edges:

$$a_k = \frac{\langle s_e : \forall e \in E_k \rangle}{\sum_{e \in E} s_e} \times \hat{b}_k.$$

7.4.3 Numerical Evaluation

After the initialization of the environment, as the HMARL is off-policy, it can draw experiences of states, actions, next states, and rewards from the environment by taking

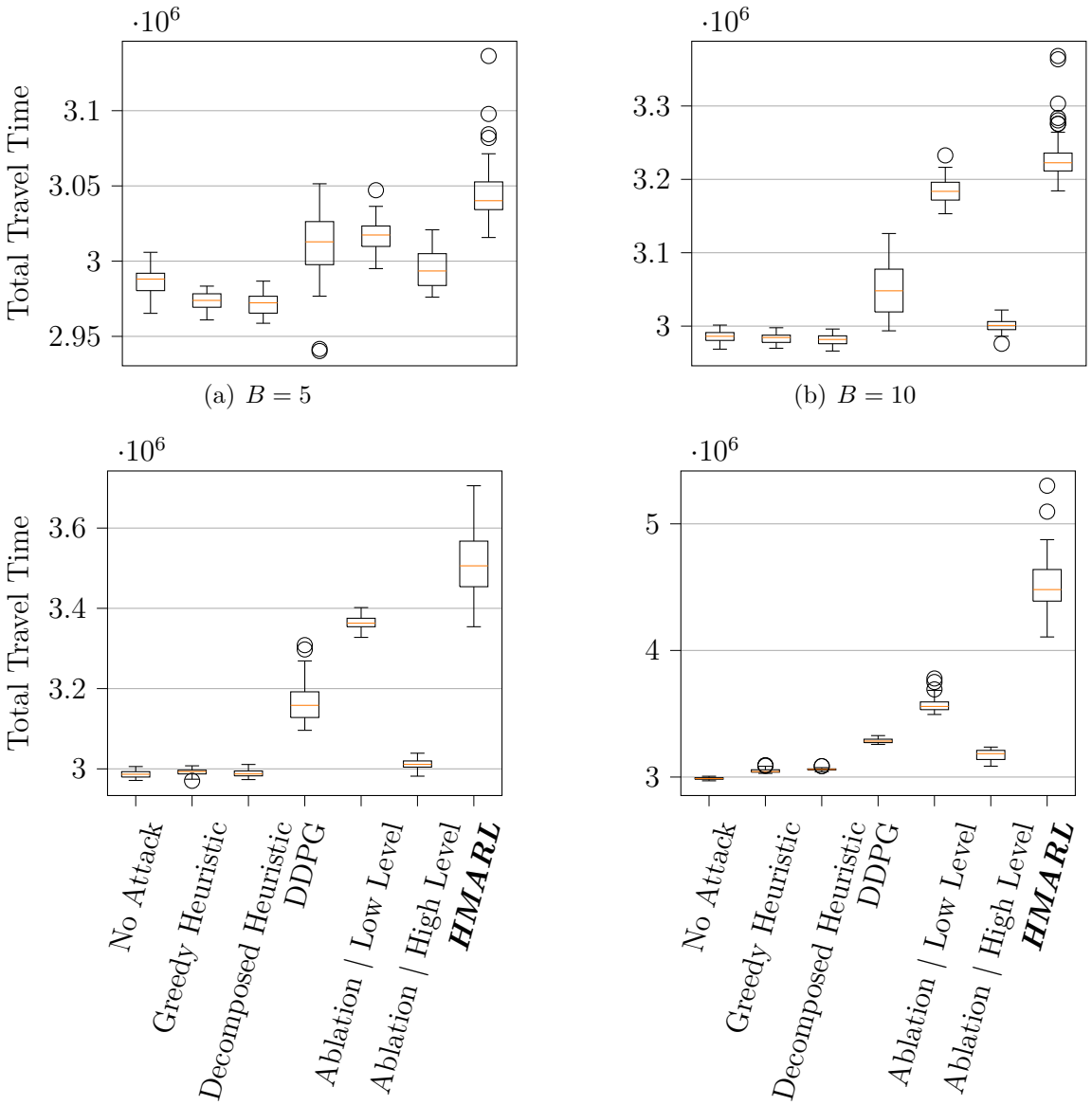


Figure 7.3. Ablation study of HMARL on the **Sioux Falls** network. *"No Attack"* pertains to no attack on the network. *"Greedy Heuristic"* is a network greedy (see Section 7.4.2) attack. *"DDPG"* applies the general-purpose DDPG algorithm network-wide. In the remaining columns, the network is divided into **four** components. In *"Decomposed Heuristic,"* the low-level actors are low-level greedy agents, with the high-level being a proportional allocation to the number of vehicles in each component. In *"Ablation | Low Level,"* the high-level agent is the proportional allocation heuristic, while its low-level is the MADDPG approach. In *"Ablation | High Level,"* the low-level is the greedy heuristic, while the high-level is a DDPG allocator RL agent. ***"HMARL"*** is our HMARL approach. Here, the low-level MADDPG and high-level DDPG components have been trained simultaneously.

either random actions or by taking OU noise added to actions outputted by the low-level agent. Using these experiences, all actors and critics can be updated simultaneously.

When agents are trained simultaneously, the low-level agent should have lower learning rates as it needs the high-level agent to learn its behavior but should account for more steps in the future with a higher discount factor γ.

Figure 7.3 shows the result of the training with an ablation study on the Sioux Falls, SD transportation network [Transportation Networks for Research Core Team, 2020]. This network has 24 nodes and 76 edge links. We ran HMARL with different attack budgets. As expected, the HMARL performs better by 10-50% depending on the budget, making it a viable solution to the scalability of Deep Reinforcement Algorithms.

Chapter 8
Detection of False Data Injection Attacks in Vehicoular Routing

8.1 System and Threat Model

This section introduces the transportation network system model and formalizes the threat model, where an adversarial agent aims to increase total travel time of vehicles by injecting false traffic information.

Our system model uses the "Transportation Networks for Core Research" dataset [2020] as a foundation. Consistent with prior work [Eghtesad et al., 2024], we employ a dynamic, step-wise simulation where vehicles make routing decisions sequentially over time, a departure from classic static network-flow problems. Our primary contribution, however, is a modified threat model that diverges from Eghtesad *et al.* by removing the explicit attack budget constraint. We argue this constraint is superfluous because a rational attacker, seeking to maximize expected impact, is already incentivized to self-limit their attack magnitude to balance effectiveness with the inherent risk of detection. Increase in the total magnitude of the attack increases the likelihood of the attack being detected and thwarted.

8.1.1 Environment

The road network is represented as a directed graph $G = (V, E)$, where nodes denote intersections and edges denote road segments. Each edge $e = (u, v) \in E$ and $u, v \in V$ is characterized by a **free-flow travel time** t_e (the time to traverse the edge without traffic), a **capacity** parameter c_e (the maximum flow rate of the segment), and two **congestion parameters**, b_e and p_e, which control the function's shape. When n_e

vehicles are traveling on edge e, the travel time $W_e(n_e)$ is calculated using the Bureau of Public Roads (BPR) function [Transportation Networks for Research Core Team, 2020, United States. Bureau of Public Roads, 1964, Gore et al., 2023]:

$$W_e(n_e) = t_e \cdot \left(1 + b_e \left(\frac{n_e}{c_e}\right)^{p_e}\right). \tag{8.1}$$

The travel demand within the network is defined by a set of trips, R. Each trip $r \in R$ is a tuple $r = \langle o_r, d_r, s_r \rangle$, representing a demand of s_r vehicles traveling from the same origin node $o_r \in V$ to the same destination node $d_r \in V$. By simulation, we can calculate the number of timesteps for vehicle trip r to reach its destination T_r.

8.1.2 States and Transitions

The system evolves in discrete time steps t. The **state** is the set of all trip locations $\{l_r^t \in V \cup (E \times \mathbb{N})\}$, where a location can be a node $v \in V$ or a position on an edge $\langle e, \tau \rangle$ with $\tau \in \mathbb{N}$ time remaining.

When a trip r is at a node ($l_r^t = u$), it dynamically routes based on a **softmin** policy. It calculates the cost-to-go via each adjacent node $v \in N(u)$ as $C_v = w_{(u,v)}^t + d(v, d_r)$, where $d(v, d_r)$ is the shortest path distance based on travel times w_e^t. The probability of choosing edge (u, v) follows a Boltzmann distribution:

$$P(l_r^{t+1} = \{(u, v), \lfloor w_{(u,v)}^t \rceil\}) = \frac{e^{-\theta C_v}}{\sum_{v' \in N(u)} e^{-\theta C_{v'}}}, \tag{8.2}$$

where the parameter $\theta \geq 0$ controls rationality; as $\theta \to \infty$, the policy approaches a deterministic shortest path.

Then, the locations are updated for $t + 1$. A trip at a node moves onto its chosen edge. A trip on an edge $\langle e, \tau \rangle$ updates its state: if $\tau > 1$, its new state is $l_r^{t+1} = \langle e, \tau - 1 \rangle$; if $\tau = 1$, it arrives at the edge's destination node, $l_r^{t+1} = u$.

8.1.3 Threat Model

The attacker perturbs the observed travel times of the edges. Let $\boldsymbol{a}^t = \{a_e^t \in \mathbb{R}\}_{\forall_{e \in E}}$ denote the perturbation on edge e. The observed travel time that is used to choose the path (eq. (8.2)) is

$$\hat{w}_e^t = w_e^t + a_e^t. \tag{8.3}$$

During an FDI attack, vehicles use $\hat{w}_e^t$ to compute shortest paths. The attacker, assumed to fully observe the environment–including the network topology, vehicle trips origin, destination, and location–seeks to maximize total vehicle travel time.

$$\max_{\{\boldsymbol{a}^1, \boldsymbol{a}^2, \ldots\}} \underbrace{\sum_{r \in R} s_r \cdot T_r}_{u_a} . \tag{8.4}$$

8.2 Game Theoretic Threat Detection

In this section, we present a model for detecting abnormal traffic patterns by formulating the system as a Partially Observable MDP (POMDP) based on the observed travel times $\hat{\boldsymbol{w}}$. Building on this, we extend the model to capture the interaction between an attacker attempting to manipulate traffic patterns by injecting perturbations into the navigation application and a detector aiming to identify such anomalies. This interaction is formulated as a two-player strategic zero-sum game, capturing the adversarial dynamics. To determine the optimal strategies for both players, we leverage the PSRO (Section 3.3.3.1) algorithm to compute the MSNE of the game.

8.2.1 Defense Model

The defender observes the edge travel times $\hat{\boldsymbol{w}}^t$ at each time t, and makes a decision to raise an alert or not $d^t \in \{0, 1\}$. If the defender correctly identifies an ongoing attack at time t and raises the alert, all future perturbations are thereby prevented, i.e., $\boldsymbol{a}^\tau = 0$ for all $\tau \geq t$. Failure to detect attacks entails direct consequences in terms of increased travel times as formalized in the attack model in eq. (8.4) (which the defender aims to minimize). Further, false positives (alerts triggered when no attack is taking place) incur a fixed cost of c.

By *detecting an attack correctly at timestep* t, the defender prevents further perturbations of the edge travel times

$$\forall_{\tau \geq t} : \boldsymbol{a}^\tau = 0 \tag{8.5}$$

We can formulate the defender's objective to minimize the total travel time while

minimizing the false alarms:

$$\min_{\{d^1,d^2,\cdots\}} \underbrace{\sum_{r\in R} s_r \cdot T_r + C_f \cdot |F|}_{u_d}, \tag{8.6}$$

where $|F|$ is the number of false positive alerts raised by the defender before an attack is correctly detected, and C_f is the cost of false positive alerts.

Unlike static anomaly detectors that learn a fixed baseline of normal behavior, our defender is trained against an adaptive attacker, learning to identify sophisticated and evolving attack patterns by observing the history of travel times ($\hat{w}^t$), which reflect both the attacker's direct perturbations and their indirect, cascading effects on traffic congestion.

The challenge is further amplified because observed traffic deviates from normal patterns in two ways. First, deviations arise directly from the attacker's perturbations, which alter the travel time data vehicles use for routing. Second, these manipulated observations cause vehicles to reroute, leading to indirect, cascading effects that produce real, yet anomalous, traffic congestion. The defender must therefore untangle anomalies caused by both the attacker's false data and the real-world consequences of those manipulations.

8.2.2 The Detection Game

We model the interaction between the attacker and the defender as a two-player game. The attacker maximizes its utility, which is the total travel, while the defender aims to minimize it by correctly detecting adversarial perturbations while reducing the false positive alarm rate. This game is not strictly zero-sum due to the defender's *false alarm penalty* (u_d, Eq.8.6). However, the defender's false alarm penalty depends only on its own strategy and is independent of the attacker's actions. Apart from this penalty, the players' goals (u_a vs. u_d) are directly opposing: any gain for the attacker results in an equivalent loss for the defender in terms of total travel time. As such, the interaction is strategically equivalent to a zero-sum game. To solve the detection game, we employ the policy space response oracles (PSRO) (Sec.3.3.3.1). The PSRO framework iteratively updates the policy space of both players by identifying the best response of one player to the current MSNE strategy of the other. This approach ensures convergence to a Nash equilibrium.

8.2.3 Attack Oracle

The attacker's oracle finds an attack policy that perturbs the traffic flow observations in a way that reroutes vehicles, increasing the overall travel time. This is achieved using a reinforcement learning algorithm designed to compute a best-response attack policy.

While continuous action DRL algorithms such as Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015] (Section 3.1.1.4) or Proximal Policy Optimization (PPO) [Schulman et al., 2017] (Section 3.1.2.4) are suitable for this task, scalability to larger networks is an important consideration. For larger and more complexnetwork instances, we can used the HMARL framework that we developed in section 7.3.

The DRL attack oracle leverages a set of features as its observation regarding each edge in the transportation network. Five of these features—the number of vehicles on any node traversing the edge with their unperturbed shortest path (s_e^t), those immediately taking the edge ($\hat{s}_e^t$), vehicles currently on the edge (m_e^t), and vehicles on any edge whose shortest path includes this edge ($\tilde{s}_e^t$), along with the current number of vehicles on the edge (n_e^t)—are adopted from section 7.3. Additionally, two new edge-specific features from the dataset are incorporated: the **edge capacity** (c_e) and **free-flow time** (t_e), collectively forming the attack oracle's observation vector:

$$\mathbf{o}_a^t = \left\langle \langle s_e^t, \hat{s}_e^t, m_e^t, \tilde{s}_e^t, n_e^t, c_e, t_e \rangle : \quad \forall_{e \in E} \right\rangle. \tag{8.7}$$

The attacker's action is the direct perturbation applied to each edge e of the network:

$$a_a^t = \langle a_e^t | a_e^t >= 0 : \forall_{e \in E} \rangle. \tag{8.8}$$

As a result of this action, vehicles will be directed on suboptimal routes and will increase the total travel time (T_r) of vehicle trip r. To translate the attacker's objective (eq. (8.10)) into a stepwise reward signal, we can reward adversarial DRL agent by the number of vehicles that are still traveling through the network:

$$r_a^t = \sum \{ s_r | l_r^t \neq d_r : \forall_{r \in R} \}. \tag{8.9}$$

8.2.4 Defense Oracle

On the defense side, the oracle is simpler in comparison, as it outputs a single decision: whether to raise an alarm or not. Consequently, RL algorithms such as Deep Q-Networks (DQN) [Mnih et al., 2015] (Section 3.1.1.2) or PPO are sufficient to approximate the

defender's best response. The reduced complexity of the defender's decision space ensures that these methods are computationally efficient and effective in identifying optimal defense policies.

The reinforcement learning algorithm pertaining to the defense side observes the perturbed edge travel times of the network reported by the vehicles:

$$\boldsymbol{o}_d^t = \hat{\boldsymbol{w}}^t = \left\langle \hat{w}_e^t : \forall_{e \in E} \right\rangle. \tag{8.10}$$

The defense oracle operates under a partially observable MDP; to improve its efficiency, we provide the defender with a history of observations h, where $h_t = \langle o_{t-|H|}, \cdots, o_{t-1}, o_t \rangle$. We set $|H| = 5$ in our experiments.

The defense oracle outputs a binary showing the detection decision of time step t:

$$a_d^t \in \{0, 1\} \tag{8.11}$$

In addition, the defense oracle receives a reward based on the negative number of vehicles on the network and a penalty for a false alarm since reinforcement learning is used to maximize future discounted rewards.

$$r_d^t = -\sum \left\{ r_s | l_r^t \neq d_r : \forall_{r \in R} \right\} - p \tag{8.12}$$

$$p = \begin{cases} C_f \quad if \quad \text{there was no attack at timestep } t. \\ 0 \quad if \quad \text{attack correctly detected} \end{cases}$$

8.3 Solving the Strategic Detection Game

By iteratively generating the best responses using reinforcement learning for both players, the double oracle algorithm refines the strategy sets until convergence. While the DO algorithm are proven to converge to the MSNE of the game [McMahan et al., 2003], these guarantees are lost when an *approximate* best-response algorithm is used instead of a true BR algorithm. However, we empirically show that our attack and defense oracles are general and effective enough to achieve convergence in relatively few iterations. Specifically, our results indicate that the strategy sets for both players converge in six iterations. This efficiency demonstrates that the algorithm converges rapidly, thereby showing the feasibility of the approach. This expedited convergence highlights the practicality of the proposed approach for real-world traffic monitoring systems, where timely adaptation

to adversarial behavior is crucial. By ensuring convergence within a limited number of iterations, our method strikes a balance between computational feasibility and strategic depth, making it suitable for dynamic and large-scale environments.

Further, we generated two experimental networks using the Grid model with Random Edges (GRE) [Peng et al., 2014]. GRE stochastically generates a random graph with similar characteristics of a real-world transportation network. For this network, we also generated the edge attributes, e.g., capacity and free flow time, based on the same distribution of Sioux Falls, SD [Transportation Networks for Research Core Team, 2020].

In terms of game theory, the Nash equilibrium guarantees that any other policy, aside from the one we have identified, will result in suboptimal outcomes. Specifically, any attacker strategy other than the optimal one will lead to lower total travel time, while any defense strategy that deviates from the equilibrium will increase total travel time.

Finally, the false positive cost (C_f) ensures that the defense oracle automatically manages the trade-off between false positive rate and total travel time. This mechanism allows for the dynamic adjustment of strategies to maintain a balance between defense effectiveness and false alarm cost.

8.4 Experimental Setup

8.4.1 Hardware Configuration

All experiments, including neural network training, are performed on a workstation with two AMD® EPYC™ 7763 each with 64 cores and 1TB of RAM and NVIDIA®RTX™ A5000 with 24GB of VRAM. The CPU is capable of executing 128 concurrent environment instances. Since the models are relatively small, only 64GB of RAM is required for simulation and inference, and 2GB of VRAM for model training.

8.4.2 Software Configuration

The simulation is based on the source code of section 7.1. We developed the environments for the attack and defense oracles using Python in the standard format of Gymnasium [Towers et al., 2024]. For single-agent DRL algorithm, we used stable-baselines3 [Raffin et al., 2021], which internally uses PyTorch [Paszke et al., 2019] as its neural operations framework. In addition, we used Scikit-Learn [Lisa and Bot, 2017] to solve the linear program that provides the equilibrium of the current restricted game [Shoham and Leyton-Brown, 2008].

When training a model, multiple trajectories or experiences are drawn from concurrently running environments. These trajectories are moved to the GPU for training. When running the PSRO algorithm, obtaining one experience requires executing the other agent's previously trained policy. This policy is duplicated and loaded for each instance of the environment on the main RAM and executed on the CPU.

Each independent DRL algorithm can be executed for simulation, data collection, and training at 879, 281, and 198 steps per second, on average, for the 3x2, 5x4, and Sioux Falls networks, respectively. The substantial difference comes from the computational cost of simulating vehicle decisions and movements.

8.4.3 Hyperparameters

The hyperparameters for the Proximal Policy Optimization (PPO) algorithm, used for both the attack and defense oracles, were adopted from the default configurations of the Stable Baselines3 [Raffin et al., 2021] library, which themselves are based on the well-established defaults from OpenAI Baselines [Dhariwal et al., 2017]. This decision was informed by the comprehensive large-scale empirical studies [Andrychowicz et al., 2020]. Their findings provide strong evidence that these default parameters represent a highly competitive and robust baseline, making them a suitable and valid choice for our experiments. A detailed breakdown of these parameters is provided in Table 8.1.

8.4.4 Seeds

For managing stochasticity, we adhered to the default random number generators provided by Python, NumPy [Harris et al., 2020], and PyTorch [Paszke et al., 2019] without setting explicit global seeds. This approach ensures that the inherent randomness in the training process, such as weight initialization and environment interactions, is handled by the standard, well-vetted procedures of these libraries. This allows our results to reflect the general performance of the methodology rather than being tied to a specific, potentially fortunate, random seed.

8.4.5 Statistical Tests

To validate the significance of our experimental outcomes, we employed a permutation test to compare the performance of our equilibrium strategies against the baselines. For each comparison, we collected 64 episodic rewards from both our trained agent and the

Table 8.1. All the hyperparameters used for the attack oracle, defense oracle, and the Double Oracle (DO) training process.

Component	Hyperparameter	Value
Attack Oracle Hyperparameters		
Training	Total Training Timesteps	5,000,000
	Algorithm	Proximal Policy Optimization (PPO)
	Learning Rate	0.0003 (default)
	PPO Number of Steps (n_steps)	50
	Batch Size	64 (default)
	PPO Number of Epochs (n_epochs)	10 (default)
	Discount Factor (Gamma)	0.99 (default)
	GAE Lambda	0.95 (default)
	Clip Range	0.2 (default)
	Entropy Coefficient (ent_coef)	0.01
	Value Function Coefficient (vf_coef)	0.5 (default)
	Max Gradient Norm	0.5 (default)
	Normalize Advantage	True (default)
Neural Network	Policy	MlpPolicy
	Network Architecture	`dict(pi=[64, 64], vf=[64, 64])` (default)
	Activation Function	Tanh (default)
Defense Oracle Hyperparameters		
Training	Total Training Timesteps	2,000,000
	Algorithm	Proximal Policy Optimization (PPO)
	Learning Rate	0.0003 (default)
	PPO Number of Steps (n_steps)	50
	Batch Size	64 (default)
	PPO Number of Epochs (n_epochs)	10 (default)
	Discount Factor (Gamma)	0.99 (default)
	GAE Lambda	0.95 (default)
	Clip Range	0.2 (default)
	Entropy Coefficient (ent_coef)	0.01
	Value Function Coefficient (vf_coef)	0.5 (default)
	Max Gradient Norm	0.5 (default)
	Normalize Advantage	True (default)
Neural Network	Policy	MlpPolicy
	Network Architecture	`dict(pi=[64, 64], vf=[64, 64])` (default)
	Activation Function	Tanh (default)
Double Oracle (DO) Parameters		
DO Parameters	DO Iterations	10
	Environment Horizon	50
	Number of Parallel Environments (n_envs)	128
	Evaluation Episodes	50
	Post-Training Testing (Numerical Report) Epochs	64
Environment Hyperparameters		
	Softmin Policy (θ)	1.0
	K-Means Graph Clustering (n_components)	4
	False Positive Cost (C_f)	1.0
	History Size	5
	GRE generation parameter (p, q)	0.6057, 0.3162

baseline agent. The permutation test then assesses the null hypothesis that the two sets of rewards are drawn from the same distribution. This is achieved by calculating the difference in means between the two groups and then repeatedly shuffling the labels of the data points and re-calculating the difference in means to create a distribution of possible differences under the null hypothesis. The p-value, representing the probability of observing a difference as large as the one in our actual data if the null hypothesis were true, was then determined. We utilized the robust implementation of this statistical test provided by the SciPy [Virtanen et al., 2020] library. This non-parametric approach is particularly well-suited for this analysis as it does not rely on assumptions about the underlying distribution of the rewards, which is often unknown in reinforcement learning contexts.

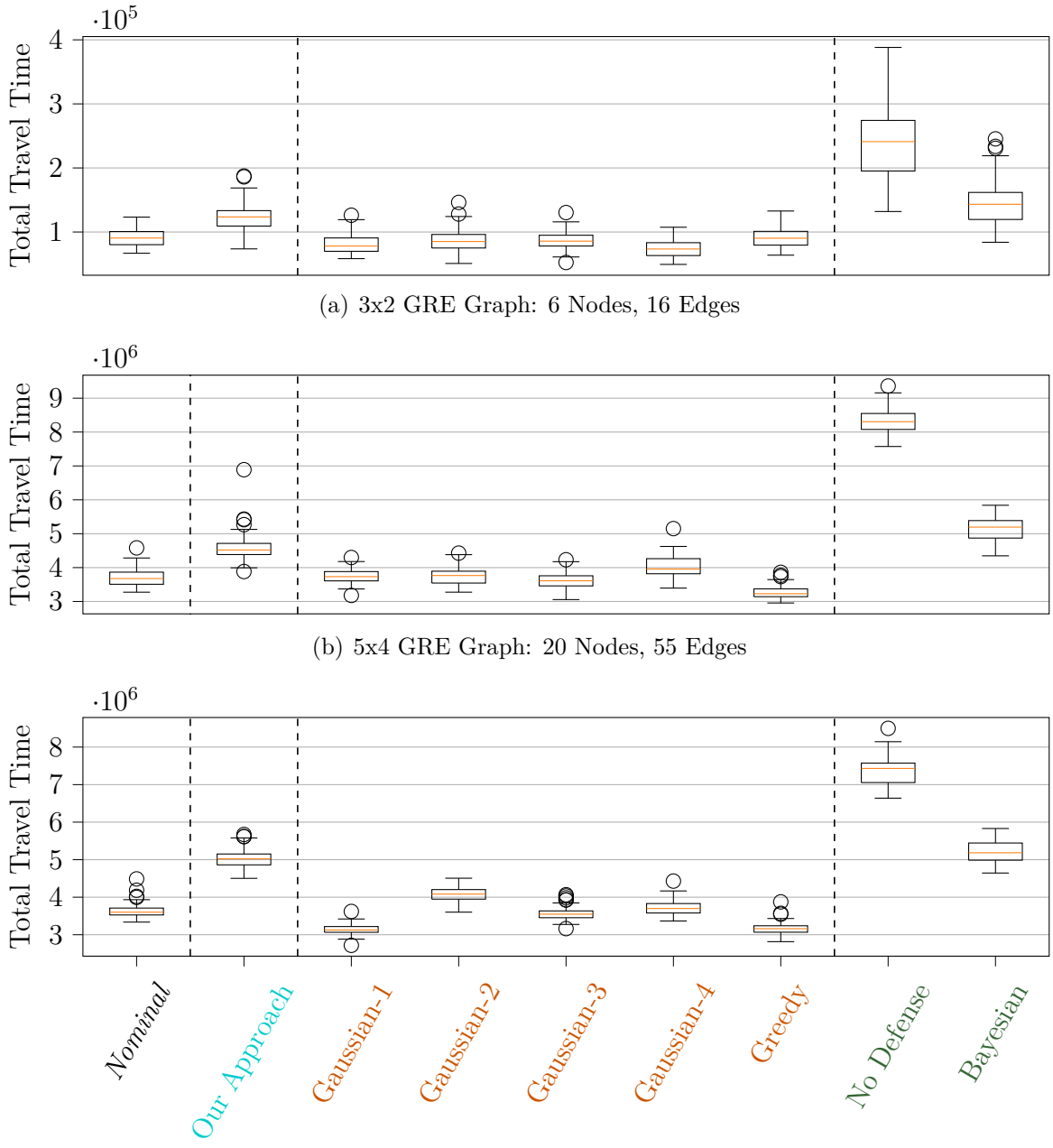
8.5 Experimental Analysis

As established in our methodology (Section 8.2.3 and Section 8.2.4), we employ Proximal Policy Optimization (PPO) [Schulman et al., 2017] as the DRL algorithm for both the attack and defense best-response oracles. For the attack oracle, PPO optimizes a multivariate Gaussian distribution with diagonal covariance matrix to output continuous perturbation values, which are afterwards scaled to be positive. For the defense oracle, PPO optimizes a Bernoulli distribution for the binary decision to raise an alarm. Both of these choices are default in Stable Baselines 3 [Raffin et al., 2021].

We used the default hyperparameters from Stable Baselines 3, which are validated by prior work [Andrychowicz et al., 2020], and left random seeds to their default values in Numpy [Harris et al., 2020] and PyTorch [Paszke et al., 2019], as our algorithm exhibits low sensitivity to random initialization.

To show the significance of our results, after the nash equilibrium model is trained, we collect 64 episodic rewards and run permutation statistical test on them while comparing the means of the distributions.

The vehicle data, such as source, destination, and count, is drawn from data provided by Transportation Networks for Research Core Team [2020]; the vehicle counts are randomized by a $\pm 0.05\%$. The initial strategy set for players contains only one policy of No Attack and No Defense, where neither player is taking any action and the environment operates under ***nominal*** conditions. In each iteration of PSRO, first an adversarial DRL, then a detection DRL is trained.



(a) 3x2 GRE Graph: 6 Nodes, 16 Edges

(b) 5x4 GRE Graph: 20 Nodes, 55 Edges

(c) Sioux Falls, SD: 24 Nodes, 76 Edges

Figure 8.1. Evaluation of our approach against alternative strategies according to Eq. 3.43. First, various baseline attack strategies are tested against the resulted equilibrium defender, where a higher total travel time indicates a more effective attack (**higher is better**). Second, different defense strategies are evaluated against the resulted equilibrium attacker, where a lower total travel time signifies a superior defense (**lower is better**). The results show our approach outperforms the alternatives, inducing higher travel times than other attacks and securing lower travel times than other defenses. The results show our approach outperforms the alternatives in both roles; crucially, our equilibrium-based defender is robust against these alternative attacks without prior training on them, which demonstrates the success of our algorithm.

8.5.1 Baselines

On these two network models, and the Sioux Falls, SD model, we compare our solution approach to attack baselines such as greedy attack heuristic (Section 7.4.2), and a Gaussian [Yu et al., 2024] attack. We also compare our defense approach to state-of-the-art anomaly detection [Laszka et al., 2019].

8.5.1.1 Attack Baselines

In the **Greedy** ▮ (see Section 7.4.2), the adversarial agent counts the number of vehicles s_e^t passing through each edge e as their unperturbed shortest path to their destination at the time of execution t. Then it divides the budget B as proportional to the number of vehicles:

$$\boldsymbol{a}^t = \frac{\langle s_e^t : \forall e \in E \rangle}{\sum_{e \in E} s_e^t} \cdot B. \tag{8.13}$$

In the **Gaussian** ▮ [Yu et al., 2024], the attacker divides the environment into multiple subcomponents; we used the K-Means graph clustering [Eghtesad et al., 2024] to divide the environment into sub-components. Then, the attacker applies a normal Gaussian perturbation to that sub-component i, dissuading vehicles from passing through i:

$$\boldsymbol{a}^t = \left\langle a_e^t = \begin{cases} 0 & if \quad e \notin i \\ \sim \mathcal{N}(\hat{B} \cdot c_e, \frac{1}{10} \cdot c_e) & \text{otherwise} \end{cases} : \forall_{e \in E} \right\rangle,$$

where $\hat{B}$ is the budget applied to each edge in sub-component i.

8.5.1.2 Defense Baselines

We can define an anomaly detection algorithm based on a **Bayesian** ▮ process similar to ML based anomaly detection [Laszka et al., 2019]. In anomaly detection, a machine learning model is trained based on the nominal operations of the system. Then, if the likelihood of a state (in our case, perceived congestion) happening based on the model is below a threshold, the model will raise an alarm.

We can collect nominal, i.e., without attack, baseline experiences x. The observed travel time at each edge $\hat{w}_e^t$ becomes one random variable for $t \in |H|$ where $|H|$ is the length of the observation history for the defense, so that the defense is comparable to

our approach. The mean and covariance of these variables can then be calculated over the trajectories to form a multivariate normal distribution:

$$F_{\hat{w}} \sim \begin{cases} \text{Mean}(\hat{w}_e^t) & = \mathbb{E}_x[\hat{w}_e^t] \\ \text{Cov}(\hat{w}_{e^i}^{t_i}, \hat{w}_{e^j}^{t_j}) & = \mathbb{E}_x[(\hat{w}_{e^i}^{t_i} - \mathbb{E}_x[\hat{w}_{e^i}^{t_i}]) \cdot (\hat{w}_{e^j}^{t_j} - \mathbb{E}_x[\hat{w}_{e^j}^{t_j}])] \end{cases} \tag{8.14}$$

Then, the detection decision is based on the similarity of a history of observations $\hat{\boldsymbol{w}}_{\boldsymbol{H}}$ to this distribution based on a threshold τ:

$$a_d^t = \begin{cases} 0 & if \quad F_{\hat{w}}(\hat{\boldsymbol{w}}_H^t) > \tau \\ 1 & \text{otherwise} \end{cases} \tag{8.15}$$

8.5.2 Numerical Results

To demonstrate the effectiveness of our approach, we need to show that the calculated equilibrium attack and defense policies satisfy eq. (3.43). In other words, 1) when the equilibrium attacker is playing against an alternative defense policy, it achieves a higher total travel time, and 2) when the equilibrium defender is playing against an alternative attack policy, it achieves a lower travel time.

Figure 8.1 illustrates the numerical results of our equilibrium strategies against the baseline attacks and defenses. *Nominal* shows the normal total travel time of vehicles without any attacks. Comparison of *Nominal* to our approach shows that our equilibrium defender limits total travel time deviations by 35%, 24%, and 38% against the worst-cast attacker in the 3x2 GRE, 5x4 GRE, and Sioux Falls, SD network, respectively.

Our equilibrium attack strategy is 19%, 11%, and 22% (episode samples=64, p-value=0.0002) more effective compared to the best (i.e., highest) alternative attack baseline in 3x2 GRE graph, 5x4 GRE graph, and Sioux Falls, SD network, respectively. Our equilibrium defense strategy is 4%, 34%, and 14% (episode samples=64, p-value=0.0002) more robust compared to the best (i.e., lowest) alternative defense baseline.

In this figure, The No Defense scenario describes the situation where the attacker executes its equilibrium policy (i.e., a worst-case attacker), but no detection mechanism is present. A maximum achievable total travel time exists because of the simulation's finite time horizon.

Bibliography

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., and others (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283.

[Agogino and Tumer, 2004] Agogino, A. K. and Tumer, K. (2004). Unifying temporal and structural credit assignment problems. In *Autonomous agents and multi-agent systems conference*.

[Albanese et al., 2019] Albanese, M., Connell, W., Venkatesan, S., and Cybenko, G. (2019). Moving Target Defense Quantification. In *Adversarial and Uncertain Reasoning for Adaptive Cyber Defense*, pages 94–111. Springer.

[Albrecht et al., 2024] Albrecht, S. V., Christianos, F., and Schäfer, L. (2024). *Multi-Agent Reinforcement Learning Foundations and Modern Approaches*. MIT Press.

[Andreeva et al., 2016] Andreeva, O., Gordeychik, S., Gritsai, G., Kochetova, O., Potseluevskaya, E., Sidorov, S. I., and Timorin, A. A. (2016). Industrial control systems vulnerabilities statistics. *Kaspersky Lab, Report*.

[Andrychowicz et al., 2020] Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., and Bachem, O. (2020). What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study. *arXiv preprint arXiv:2006.05990*.

[Åström, 1965] Åström, K. (1965). Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205.

[Barreto et al., 2013] Barreto, C., Cárdenas, A. A., and Quijano, N. (2013). Controllability of Dynamical Systems: Threat Models and Reactive Security. pages 45–64.

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

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym.

[Byres, 2008] Byres, E. (2008). Hidden Vulnerabilities in SCADA and Critical Infrastructure Systems.

[Chen et al., 2015] Chen, P., Xu, J., Lin, Z., Xu, D., Mao, B., and Liu, P. (2015). A Practical Approach for Adaptive Data Structure Layout Randomization. In *20th European Symposium on Research in Computer Security (ESORICS)*, pages 69–89.

[Chen et al., 2018] Chen, Q. A., Yin, Y., Feng, Y., Mao, Z. M., and Liu, H. X. (2018). Exposing Congestion Attack on Emerging Connected Vehicle based Traffic Signal Control. In *Proceedings 2018 Network and Distributed System Security Symposium*, Reston, VA. Internet Society.

[Chen et al., 2019] Chen, Y., Dong, C., Palanisamy, P., Mudalige, P., Muelling, K., and Dolan, J. M. (2019). Attention-Based Hierarchical Deep Reinforcement Learning for Lane Change Behaviors in Autonomous Driving. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1326–1334. IEEE.

[Cillizza, 2015] Cillizza, C. (2015). Bridgegate is still a BIG problem for Chris Christie. *The Washington Post*.

[Combita et al., 2019] Combita, L. F., Cardenas, A., and Quijano, N. (2019). Mitigating Sensor Attacks Against Industrial Control Systems. *IEEE Access*, 7:92444–92455.

[Dhariwal et al., 2017] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). OpenAI Baselines. https://github.com/openai/baselines.

[Eghtesad et al., 2024] Eghtesad, T., Li, S., Vorobeychik, Y., and Laszka, A. (2024). Multi-Agent Reinforcement Learning for Assessing False-Data Injection Attacks on Transportation Networks. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, pages 508–515, Auckland, NZ. IFAAMAS.

[Eghtesad et al., 2020] Eghtesad, T., Vorobeychik, Y., and Laszka, A. (2020). Adversarial Deep Reinforcement Learning Based Adaptive Moving Target Defense. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12513 LNCS, pages 58–79.

[Eryonucu and Papadimitratos, 2022] Eryonucu, C. and Papadimitratos, P. (2022). Sybil-Based Attacks on Google Maps or How to Forge the Image of City Life. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 73–84, New York, NY, USA. ACM.

[Eykholt et al., 2018] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018). Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634.

[Farwell and Rohozinski, 2011] Farwell, J. P. and Rohozinski, R. (2011). Stuxnet and the Future of Cyber War. *Survival*, 53(1):23–40.

[Fawzi et al., 2014] Fawzi, H., Tabuada, P., and Diggavi, S. (2014). Secure Estimation and Control for Cyber-Physical Systems Under Adversarial Attacks. *IEEE Transactions on Automatic Control*, 59(6):1454–1467.

[Feng et al., 2018] Feng, Y., Huang, S., Chen, Q. A., Liu, H. X., and Mao, Z. M. (2018). Vulnerability of Traffic Control System Under Cyberattacks with Falsified Data. *Transportation Research Record: Journal of the Transportation Research Board*, 2672(1):1–11.

[Foerster et al., 2018] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual Multi-Agent Policy Gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

[Foley and Hulme, 2004] Foley, J. and Hulme, G. V. (2004). Get ready to patch. *InformationWeek*, (1003):18–20.

[Fukushima, 1980] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202.

[Ghafouri et al., 2016] Ghafouri, A., Abbas, W., Vorobeychik, Y., and Koutsoukos, X. (2016). Vulnerability of fixed-time control of signalized intersections to cyber-tampering. In *2016 Resilience Week (RWS)*, pages 130–135. IEEE.

[Ghena et al., 2014] Ghena, B., Beyer, W., Hillaker, A., Pevarnek, J., and Halderman, J. A. (2014). Green Lights Forever: Analyzing the Security of Traffic Infrastructure. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, San Diego, CA. USENIX Association.

[Giraldo et al., 2019] Giraldo, J., Urbina, D., Cardenas, A., Valente, J., Faisal, M., Ruths, J., Tippenhauer, N. O., Sandberg, H., and Candell, R. (2019). A Survey of Physics-Based Attack Detection in Cyber-Physical Systems. *ACM Computing Surveys*, 51(4):1–36.

[Gore et al., 2023] Gore, N., Arkatkar, S., Joshi, G., and Antoniou, C. (2023). Modified Bureau of Public Roads Link Function. *Transportation Research Record: Journal of the Transportation Research Board*, 2677(5):966–990.

[Govindan and Wilson, 2003] Govindan, S. and Wilson, R. (2003). A global Newton method to compute Nash equilibria. *Journal of Economic Theory*, 110(1):65–86.

[Harris et al., 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F.,

Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[Hemsley et al., 2018] Hemsley, K. E., Fisher, E., and others (2018). History of industrial control system cyber incidents. Technical report, Idaho National Lab.(INL), Idaho Falls, ID (United States).

[Herrero and Corchado, 2009] Herrero, A. and Corchado, E. (2009). Multiagent Systems for Network Intrusion Detection: A Review. In *Computational Intelligence in Security for Information Systems*, pages 143–154. Springer.

[Hill et al., 2018] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable Baselines. https://github.com/hill-a/stable-baselines.

[Hinton et al.,] Hinton, G., Srivastava, N., and Swersky, K. Neural Networks for Machine Learning.

[Hu et al., 2019] Hu, Z., Chen, P., Zhu, M., and Liu, P. (2019). Reinforcement Learning for Adaptive Cyber Defense Against Zero-Day Attacks. In *Adversarial and Uncertain Reasoning for Adaptive Cyber Defense*, pages 54–93. Springer.

[Iannucci et al., 2019] Iannucci, S., Barba, O. D., Cardellini, V., and Banicescu, I. (2019). A Performance Evaluation of Deep Reinforcement Learning for Model-Based Intrusion Response. In *4th IEEE International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 158–163.

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-Supervised Classification with Graph Convolutional Networks.

[Kosut et al., 2010] Kosut, O., Jia, L., Thomas, R. J., and Tong, L. (2010). Malicious Data Attacks on Smart Grid State Estimation: Attack Strategies and Countermeasures. In *2010 First IEEE International Conference on Smart Grid Communications*, pages 220–225. IEEE.

[Kovacs, 2016] Kovacs, E. (2016). BlackEnergy malware used in Ukraine power grid attacks.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

[Lanctot et al., 2017] Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., Silver, D., and Graepel, T. (2017). A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30.

[Laszka et al., 2019] Laszka, A., Abbas, W., Vorobeychik, Y., and Koutsoukos, X. (2019). Detection and mitigation of attacks on transportation networks as a multi-stage security game. *Computers & Security*, 87:101576.

[Laszka et al., 2016] Laszka, A., Potteiger, B., Vorobeychik, Y., Amin, S., and Koutsoukos, X. (2016). Vulnerability of Transportation Networks to Traffic-Signal Tampering. In *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, pages 1–10. IEEE.

[Lei et al., 2017] Lei, C., Ma, D.-H., and Zhang, H.-Q. (2017). Optimal Strategy Selection for Moving Target Defense Based on Markov Game. *IEEE Access*, 5:156–169.

[Levy-Bencheton and Darra, 2015] Levy-Bencheton, C. and Darra, E. (2015). Cyber security and resilience of intelligent public transport: good practices and recommendations.

[Li and Zheng, 2019] Li, H. and Zheng, Z. (2019). Optimal Timing of Moving Target Defense: A Stackelberg Game Model. In *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, pages 1–6. IEEE.

[Li et al., 2022] Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. (2022). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Lin et al., 2018] Lin, J., Yu, W., Zhang, N., Yang, X., and Ge, L. (2018). Data Integrity Attacks Against Dynamic Route Guidance in Transportation-Based Cyber-Physical Systems: Modeling, Analysis, and Defense. *IEEE Transactions on Vehicular Technology*, 67(9):8738–8753.

[Lisa and Bot, 2017] Lisa, M. and Bot, H. (2017). My Research Software.

[Lowe et al., 2017] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6382–6393, Red Hook, NY, USA. Curran Associates Inc.

[Malialis et al., 2015] Malialis, K., Devlin, S., and Kudenko, D. (2015). Distributed reinforcement learning for adaptive and robust network intrusion response. *Connection Science*, 27(3):234–252.

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

[McKelvey et al., 2006] McKelvey, R. D., McLennan, A. M., and Turocy, T. L. (2006). Gambit: Software tools for game theory.

[McMahan et al., 2003] McMahan, H. B., Gordon, G. J., and Blum, A. (2003). Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 536–543.

[Miller and Rowe, 2012] Miller, B. and Rowe, D. (2012). A survey SCADA of and critical infrastructure incidents. In *Proceedings of the 1st Annual conference on Research in information technology*, pages 51–56, New York, NY, USA. ACM.

[Mnih et al., 2015] 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., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Morris et al., 2019] Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4602–4609.

[Nash, 1951] Nash, J. (1951). Non-Cooperative Games. *The Annals of Mathematics*, 54(2):286.

[Nguyen and Reddi, 2023] Nguyen, T. T. and Reddi, V. J. (2023). Deep Reinforcement Learning for Cyber Security. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3779–3795.

[Oakley and Oprea, 2019] Oakley, L. and Oprea, A. (2019). $$\mathsf {QFlip}$$: An Adaptive Reinforcement Learning Strategy for the $$\mathsf {FlipIt}$$ Security Game. pages 364–384.

[Paridari et al., 2018] Paridari, K., O'Mahony, N., El-Din Mady, A., Chabukswar, R., Boubekeur, M., and Sandberg, H. (2018). A Framework for Attack-Resilient Industrial Control Systems: Attack Detection and Controller Reconfiguration. *Proceedings of the IEEE*, 106(1):113–128.

[Pasqualetti et al., 2013] Pasqualetti, F., Dorfler, F., and Bullo, F. (2013). Attack Detection and Identification in Cyber-Physical Systems. *IEEE Transactions on Automatic Control*, 58(11):2715–2729.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., and others (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

[Peng et al., 2014] Peng, W., Dong, G., Yang, K., and Su, J. (2014). A Random Road Network Model and Its Effects on Topological Characteristics of Mobile Delay-Tolerant Networks. *IEEE Transactions on Mobile Computing*, 13(12):2706–2718.

[Prakash and Wellman, 2015] Prakash, A. and Wellman, M. P. (2015). Empirical Game-Theoretic Analysis for Moving Target Defense. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, pages 57–65, New York, NY, USA. ACM.

[Puterman, 1994] Puterman, M. L. (1994). *Markov Decision Processes*. Wiley Series in Probability and Statistics. Wiley.

[Raffin et al., 2021] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8.

[Raponi et al., 2022] Raponi, S., Sciancalepore, S., Oligeri, G., and Di Pietro, R. (2022). Road Traffic Poisoning of Navigation Apps: Threats and Countermeasures. *IEEE Security & Privacy*, 20(3):71–79.

[Rashid et al., 2020] Rashid, T., Samvelyan, M., Schroeder De Witt, C., Farquhar, G., Foerster, J., and Whiteson, S. (2020). Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *Journal of Machine Learning Research*, 21:1–51.

[Reilly et al., 2016] Reilly, J., Martin, S., Payer, M., and Bayen, A. M. (2016). Creating complex congestion patterns via multi-objective optimal freeway traffic control with application to cyber-security. *Transportation Research Part B: Methodological*, 91:366–382.

[Robles and Choi, 2009] Robles, R. J. and Choi, M.-k. (2009). Assessment of the vulnerabilities of SCADA, control systems and critical infrastructure systems. *Assessment*, 2(2):27–34.

[Sargolzaei et al., 2020] Sargolzaei, A., Yazdani, K., Abbaspour, A., Crane, C. D., and Dixon, W. E. (2020). Detection and Mitigation of False Data Injection Attacks in Networked Control Systems. *IEEE Transactions on Industrial Informatics*, 16(6):4281–4292.

[Schoon, 2020] Schoon, B. (2020). Google Maps 'hack' uses 99 smartphones to create virtual traffic jams. https://9to5google.com/2020/02/04/google-maps-hack-virtual-traffic-jam/.

[Schulman et al., 2015] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust Region Policy Optimization. *arXiv preprint arXiv:1502.05477*.

[Schulman et al., 2016] Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv preprint arXiv:1506.02438v6*.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

[Sengupta et al., 2017] Sengupta, S., Vadlamudi, S. G., Kambhampati, S., Doupé, A., Zhao, Z., Taguinod, M., and Ahn, G.-J. (2017). A game theoretic approach to strategy generation for moving target defense in web applications. In *16th Conference on Autonomous Agents and Multiagent Systems*, pages 178–186.

[Shamshirband et al., 2014] Shamshirband, S., Patel, A., Anuar, N. B., Kiah, M. L. M., and 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.

[Shannon, 1948a] Shannon, C. E. (1948a). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423.

[Shannon, 1948b] Shannon, C. E. (1948b). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(4):623–656.

[Shoham and Leyton-Brown, 2008] Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

[Singh et al., 2020] Singh, S., Karimipour, H., HaddadPajouh, H., and Dehghantanha, A. (2020). Artificial Intelligence and Security of Industrial Control Systems. In *Handbook of Big Data Privacy*, pages 121–164. Springer International Publishing, Cham.

[Sunehag et al., 2018] Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. (2018). Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pages 2085–2087, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Sutton and Barto, 2018] Sutton, R. and Barto, A. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition.

[Sutton et al., 1999] Sutton, R. S., Mcallester, D., Singh, S., and Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. Technical report, Neural Information Processing Symposium.

[Tan et al., 2019] Tan, J.-l., Lei, C., Zhang, H.-q., and Cheng, Y.-q. (2019). Optimal strategy selection approach to moving target defense based on Markov robust game. *Computers & Security*, 85(5):63–76.

[Teixeira et al., 2015] Teixeira, A., Shames, I., Sandberg, H., and Johansson, K. H. (2015). A secure control framework for resource-limited adversaries. *Automatica*, 51:135–148.

[Tong et al., 2020] Tong, L., Laszka, A., Yan, C., Zhang, N., and Vorobeychik, Y. (2020). Finding Needles in a Moving Haystack: Prioritizing Alerts with Adversarial Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):946–953.

[Towers et al., 2024] Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., and Younis, O. G. (2024). Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032*.

[Transportation Networks for Research Core Team, 2020] Transportation Networks for Research Core Team (2020). Transportation Networks for Research. https://github.com/bstabler/TransportationNetworks/.

[Uhlenbeck and Ornstein, 1930] Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the Theory of the Brownian Motion. *Physical Review*, 36(5):823–841.

[United States. Bureau of Public Roads, 1964] United States. Bureau of Public Roads (1964). *Traffic Assignment Manual for Application with a Large, High Speed Computer*. Number v. 2 in Traffic Assignment Manual for Application with a Large, High Speed Computer. U.S. Department of Commerce, Bureau of Public Roads, Office of Planning, Urban Planning Division.

[Urbina et al., 2016] Urbina, D. I., Giraldo, J. A., Cardenas, A. A., Tippenhauer, N. O., Valente, J., Faisal, M., Ruths, J., Candell, R., and Sandberg, H. (2016). Limiting the Impact of Stealthy Attacks on Industrial Control Systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1092–1105, New York, NY, USA. ACM.

[Van Hasselt et al., 2016] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

[Venkatesan and Li, 2017] Venkatesan, R. and Li, B. (2017). *Convolutional neural networks in visual computing: a concise guide*. CRC Press.

[Virtanen et al., 2020] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, I., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., Vijaykumar, A., Bardelli, A. P., Rothberg, A., Hilboll, A., Kloeckner, A., Scopatz, A., Lee, A., Rokem, A., Woods, C. N., Fulton, C., Masson, C., Haggstrom, C., Fitzgerald, C., Nicholson, D. A., Hagen, D. R., Pasechnik, D. V., Olivetti, E., Martin, E., Wieser, E., Silva, F., Lenders, F., Wilhelm, F., Young, G., Price, G. A., Ingold, G.-L., Allen,

G. E., Lee, G. R., Audren, H., Probst, I., Dietrich, J. P., Silterra, J., Webber, J. T., Slavic, J., Nothman, J., Buchner, J., Kulick, J., Schonberger, J. L., de Miranda Cardoso, J. V., Reimer, J., Harrington, J., Rodríguez, J. L. C., Nunez-Iglesias, J., Kuczynski, J., Tritz, K., Thoma, M., Newville, M., Kummerer, M., Bolingbroke, M., Tartre, M., Pak, M., Smith, N. J., Nowaczyk, N., Shebanov, N., Pavlyk, O., Brodtkorb, P. A., Lee, P., McGibbon, R. T., Feldbauer, R., Lewis, S., Tygier, S., Sievert, S., Vigna, S., Peterson, S., More, S., Pudlik, T., Oshima, T., Pingel, T. J., Robitaille, T. P., Spura, T., Jones, T. R., Cera, T., Leslie, T., Zito, T., Krauss, T., Upadhyay, U., Halchenko, Y. O., and Vazquez-Baeza, Y. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272.

[Waniek et al., 2021] Waniek, M., Raman, G., AlShebli, B., Peng, J. C.-H., and Rahwan, T. (2021). Traffic networks are vulnerable to disinformation attacks. *Scientific Reports*, 11(1):5329.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

[Wright et al., 2016] Wright, M., Venkatesan, S., Albanese, M., and Wellman, M. P. (2016). Moving Target Defense against DDoS Attacks. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pages 93–104, New York, NY, USA. ACM.

[Yang et al., 2023] Yang, Y.-T., Lei, H., and Zhu, Q. (2023). Strategic Information Attacks on Incentive-Compatible Navigational Recommendations in Intelligent Transportation Systems. *arXiv preprint arXiv:2310.01646*.

[Yang et al., 2018] Yang, Z., Merrick, K., Jin, L., and Abbass, H. A. (2018). Hierarchical Deep Reinforcement Learning for Continuous Action Control. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5174–5184.

[Yu et al., 2024] Yu, Y., Thorpe, A. J., Milzman, J., Fridovich-Keil, D., and Topcu, U. (2024). Sensing Resource Allocation Against Data-Poisoning Attacks in Traffic Routing. *arXiv preprint arXiv:2404.02876*.

[Zhang et al., 2021] Zhang, Y., Mou, Z., Gao, F., Xing, L., Jiang, J., and Han, Z. (2021). Hierarchical Deep Reinforcement Learning for Backscattering Data Collection With Multiple UAVs. *IEEE Internet of Things Journal*, 8(5):3786–3800.

[Zheng and Siami Namin, 2018] Zheng, J. and Siami Namin, A. (2018). A Markov Decision Process to Determine Optimal Policies in Moving Target. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2321–2323, New York, NY, USA. ACM.

Vita

Taha Eghtesad

Taha Eghtesad

tahaeghtesad@psu.edu | *tahaeghtesad.com*
https://github.com/tahaeghtesad
https://www.linkedin.com/in/tahaeghtesad
https://scholar.google.com/citations?user=lburGhYAAAAJ

Education

- **Doctor of Philosophy in Informatics** Aug 2022 – Present
  Pennsylvania State University, University Park, PA, USA
  - Advisor: Dr. Aron Laszka
  - Dissertation: Adversarial Reinforcement Learning for Cyber Attack Prevention, Detection, and Mitigation
- **Master of Science in Computer Science** Sep 2018 – May 2022
  University of Houston, Houston, TX, USA
  - Advisor: Dr. Aron Laszka
  - Thesis: Adversarial Deep Reinforcement Learning for Moving Target Defense Automatic Decision-making
- **Bachelor of Science in Computer Engineering** Sep 2013 – Feb 2018
  Shahid Beheshti University, Tehran, Iran

Experience

- **Software Engineer** Dec 2025 – Present
  Google, Mountain View, CA, USA
- **Research Assistant** Aug 2022 – Dec 2025
  Pennsylvania State University, University Park, PA, USA
- **Teaching Assistant** Jan 2025 – May 2025
  Pennsylvania State University, University Park, PA, USA
  - ETI 461: Database Management and Administration
  - IST 597: Deep Reinforcement Learning

- **Artificial Intelligence and Machine Learning Co-Op** July 2024 – Aug 2024
  Triconex at Schneider Electric, Lake Forest, CA, USA
- **Research Assistant** Aug 2018 – Aug 2022
  University of Houston, Houston, TX, USA
- **Software Engineer** Sep 2017 – July 2018
  Hamisystem Sharif, Tehran, Iran

Awards

- **Distinguished Paper Award** Aug 2023
  USENIX Security 2023

Publications

- T. Eghtesad, Y. Vorobeychik, A. Laszka. (2026). **Adversarial Reinforcement Learning for Detecting False Data Injection Attacks in Vehicular Routing.** In *17th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS).*
- T. Eghtesad, S. Li, Y. Vorobeychik, A. Laszka. (2024). **Multi-Agent Reinforcement Learning for Assessing False-Data Injection Attacks on Transportation Networks.** In *23rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).*
- O. Akgul, T. Eghtesad, A. Elazari, O. Gnawali, M. Mazurek, J. Grossklags, D. Votipka, A. Laszka. (2023). **Bug Hunters' Perspectives on the Challenges and Benefits of the Bug Bounty Ecosystem.** In *32nd USENIX Security Symposium (USENIX Security).*
- S. Eisele, M. Wilbur, K. Silvergold, F. Eisele, A. Mukhopadhyay, T. Eghtesad, A. Laszka, A. Dubey. (2022). **Decentralized Computation Market for Stream Processing Applications.** In *10th IEEE International Conference on Cloud Engineering (IC2E).*
- S. Eisele, T. Eghtesad, K. Campanelli, P. Agrawal, A. Laszka, A. Dubey. (2020). **Safe and Private Forward-Trading Platform for Transactive Microgrids.** In *ACM Transactions on Cyber-Physical Systems (TCPS).*
- T. Eghtesad, Y. Vorobeychik, A. Laszka. (2020). **Adversarial Deep Reinforcement Learning based Adaptive Moving Target Defense.** In *11th Conference on Decision and Game Theory for Security (GameSec).*

- S. Eisele, C. Barreto, A. Dubey, X. Koutsoukos, T. Eghtesad, A. Laszka, A. Mavridou. (2020). **Blockchains for Transactive Energy Systems: Opportunities, Challenges, and Approaches.** In *IEEE Computer Magazine*.

- S. Eisele, T. Eghtesad, N. Troutman, A. Laszka, A. Dubey. (2020). **Mechanisms for Outsourcing Computation via a Decentralized Market.** In *14th ACM International Conference on Distributed and Event-based Systems (DEBS)*.

- C. Barreto, T. Eghtesad, S. Eisele, A. Laszka, A. Dubey, X. Koutsoukos. (2020). **Cyber-attacks and mitigation in blockchain based transactive energy systems.** In *IEEE Conference on Industrial Cyberphysical Systems (ICPS)*.

- O. Akgul, T. Eghtesad, A. Elazari, O. Gnawali, J. Grossklags, D. Votipka, A. Laszka. (2020). **The Hackers' Viewpoint: Exploring Challenges and Benefits of Bug-Bounty Programs.** In *6th Workshop on Security Information Workers (WSIW)*.