Targeted Poisoning of Reinforcement Learning Agents

Omran Shahbazi Gholiabad
omranshahbazi1998@gmail.com

Mohammad Mahmoody
mahmoody@gmail.com

Abstract

Understanding the impact of adversarial attacks on reinforcement learning (RL) models is essential due to their wide range of applications. In this work, we initiate a study of targeted poisoning attacks for reinforcement learning agents, where the adversary aims to deliberately increase the likelihood of a specific undesirable event chosen by the attacker. In particular, rather than degrading overall performance indiscriminately, the adversary carefully manipulates the training process so that, during critical decision-making steps, the agent is more likely to fail in a targeted manner, leading it into the adversary's desired outcome.

We present theoretical results showing the effectiveness of such targeted poisoning in basic RL settings. Building on these insights, we design practical attack strategies and thoroughly evaluate their impact beyond the scope of our theoretical analysis. Through extensive experiments, we demonstrate that targeted poisoning attacks substantially raise the probability of the chosen undesirable event across a variety of reinforcement learning tasks, ranging from classic control benchmarks to more complex continuous-control environments, including stochastic settings. We compare our attacks against standard RL baselines and against algorithms specifically designed to mitigate poisoning, and we further validate their effectiveness on deep RL models. Our results highlight the vulnerabilities of RL systems to targeted training-time manipulations, underscoring the need for stronger defenses.

Contents

18th European Workshop on Reinforcement Learning (EWRL 2025).

1 Introduction

Reinforcement learning investigates how an intelligent agent acquires the ability to navigate and operate within an unfamiliar environment, guided solely by the feedback of rewards. RL has a variety of applications in the real world, such as autonomous driving (Mnih et al., 2015), recommender systems (Afsar et al., 2022), and portfolio management (Jiang et al., 2017), among others. The environment in RL is modeled as a Markov Decision Process (MDP). The MDP describes the space of states, how an agent's action leads to a transition between those states, and how it is rewarded for each transition. In the conventional setting, the MDP is stationary, meaning that the state transition mechanism and the reward function remain unchanged over time. Operating under this assumption, numerous algorithms, supported by theoretical guarantees, have been meticulously developed (Lillicrap et al., 2015a; Mnih et al., 2013) that can achieve the agent's goal: learning a policy that maximizes the cumulative reward it receives over time through its interactions with the environment. However, these guarantees may lose effectiveness if an adversary corrupts one or more elements of the underlying MDP.

Recent studies have demonstrated that modifications to the MDP, such as perturbed rewards, altered state transitions, or manipulated actions, can undermine the performance of RL algorithms (Rakhsha et al., 2021; Pinto et al., 2017). These works reveal how such adversaries can lead to sub-optimal outcomes (Behzadan and Munir, 2017; Huang et al., 2017). These works fall under the umbrella of so-called poisoning attacks. Several prior works have investigated poisoning attacks against reinforcement learning, and these can be broadly categorized into two general classes.

One type of poisoning attack on RL has primarily focused on maximizing the agent's "regret" (Wei et al., 2022; Wang, 2022) defined as

$$\mathrm{Regret}_T^{\pi} = \sum_{t=1}^{T} \left[R(s_{t-1}, \pi^*(s_{t-1}), s_t) - R(s_{t-1}, a_t, s_t) \right],$$

which represents the difference between the agent's cumulative reward when following policy π and the cumulative reward obtained under the optimal policy π^*, where π is generally defined as a mapping from states to actions. Here, s_t denotes the agent's state at time step t, and a_t represents the action chosen by the agent at time step $t-1$. $R(\cdot)$ denotes the reward function, and T represents the number of time steps in an episode, which is a sequence of interactions between an agent and the environment (see the RL formulation of Section 2 for definitions).

In another type of attacks (Xu and Singh, 2023; Zhang et al., 2020b; Liu et al., 2022), the adversary sets an adversarial policy, $\pi^\dagger$, and aims to increase the "loss" as the number of time steps the agent follows this policy,

$$\mathrm{Loss}_T^{\pi} = \sum_{t=1}^{T} \mathbb{1}_{a_t = \pi^\dagger(s_t)}.$$

While regret-based and adversarial-policy poisoning attacks can sometimes lead to harmful outcomes, they do so only indirectly. These approaches are not designed to target specific notions of failure that may be critical in real-world systems. In many applications, a bad event might mean entering a dangerous state, taking a harmful action, or violating a safety constraint. Attacks focusing solely on maximizing regret or policy deviation fail to capture these nuanced objectives. In contrast, poisoning attacks that explicitly aim to increase the probability of predefined bad events, regardless of how those events are defined, offer a more direct and flexible threat model for safety-critical reinforcement learning settings. Therefore, despite the attack models presented in prior works, it is crucial to broaden our perspective on the possible vulnerabilities of reinforcement learning, ensuring that these systems can withstand and adapt to various adversarial conditions.

1.1 Our Contribution

This paper examines poisoning attacks against RL from a novel perspective by introducing an adversarial goal that diverges from traditional approaches focused on maximizing regret and loss. Instead, we focus on increasing the probability of a particular bad event desired by the adversary. This scenario is relevant in many RL applications, and examining the power of such attacks would broaden our understanding of the vulnerability of RL agents to adversarial attacks.

Our contributions are threefold: (i) introducing a novel attack model, (ii) designing theoretical attacks, and (iii) extensive experiments against multiple algorithms in different environments.

Defining targeted poisoning attacks for RL. We initiate a study of *targeted poisoning attacks* for RL, which is a novel class of adversarial attacks on RL agents. In such attacks, the adversary manipulates the learning process to increase the probability of a specific undesirable event. In essence, a targeted poisoning attack involves the adversary identifying an undesirable event, denoted by a condition BE, defined over the trajectory of the RL process. The agent is said to have triggered the targeted event if this condition is satisfied. The adversary's objective is to steer the learning process such that the probability of this undesirable event increases:

$$\mathrm{P}_{\mathrm{BE}}(\text{RL random process}) = \Pr\left[\text{The RL random process results in the targeted undesirable event}\right],$$

where "the RL random process results in the targeted undesirable event" means that the random trajectory generated by the environment together with the agent's policy π satisfies the condition BE.

In more detail, there are generally two phases in a targeted poisoning attack on an RL system: the training phase and the testing phase. During the training phase, the adversary executes its poisoning attack strategy while the agent works to improve its policy over time within this adversarial setting. In the testing phase, the agent ceases policy refinement and uses the policy obtained at the end of the training phase to interact with the environment. By the end of the testing phase, we can evaluate the adversary's success in causing the targeted undesirable event for the agent. Targeted poisoning attacks are general enough to capture a wide range of meaningful scenarios not previously studied in poisoning-robust RL.

In the case of theoretical attacks, we also study an alternative setting in which the training and testing phases are *merged* into a single, continuous process of learning and testing concurrently. This way, the agent focuses on learning the optimal policy during the initial portion of the episode, while the latter part is reserved for performance evaluation.

Comparison to targeted attacks on classifiers. Our attack is inspired by *targeted* poisoning attacks in robust classification (Barreno et al., 2006; Shafahi et al., 2018), where the adversary seeks to increase the likelihood of misclassifying a specific test by manipulating the training data. However, despite similarities, there are fundamental differences between the two settings. In particular, RL is a *stateful* process, and it has an entirely different way of measuring an adversary's "budget" compared with classification, as even a single change in the process can drive it into a completely new path. However, attacks on classifiers can corrupt each example in the training set individually, regardless of the other examples in the set.

Motivating examples. To illustrate targeted poisoning attacks in real-world scenarios, consider an RL-powered robotic surgery assistant responsible for suturing skin wounds. An adversary might manipulate the agent's learning process to induce undesirable outcomes, such as inadvertently puncturing healthy skin with a needle. Similarly, consider an RL-driven firefighting robot tasked with suppressing forest fires. Although the agent must steer clear of dangerously hot zones to avoid damage, an adversary could manipulate its learning to direct it into hazardous areas. In these two examples, the condition BE can be defined as the RL process entering a set of target malicious states $\mathcal{S}^\dagger$, specified by the adversary. Another example involves an autonomous driving agent operating in the presence of an adversary that aims to manipulate the learning process, causing the agent to take actions that may harm the vehicle, such as sudden acceleration or hard braking, which can damage its components. In this case, the condition BE can be defined as the agent executing a harmful action from a set $\mathcal{A}^\dagger$, specified by the adversary.

Targeted poisoning through action manipulation. The notion of targeted poisoning is general and agnostic to the specific mechanism by which the poisoning is carried out. However, we need to select a concrete poisoning model to demonstrate the effectiveness of such attacks. For this goal, we focus on *action manipulation*. Specifically, after the agent selects an action, the adversary decides whether to replace it with an alternative. Action manipulation can stem from the direct manipulation of the agent's actions or unintended perturbations, such as sensor noise or system malfunctions (Tessler et al., 2019). For example, in the scenarios described in the "Motivating examples" paragraph, the adversary can apply a destabilizing force that affects the needle's trajectory in a robotic surgery assistant. Similarly, for an RL agent tasked with suppressing fires in a burning forest, the adversary's

action manipulation can be modeled as a software glitch that leads to adjustments in the robot's speed or altitude. The key reason to focus on action manipulation is that RL processes can be largely deterministic in their state transitions and rewards in some applications, such as robotic manipulation tasks governed by physics laws (Whitman et al., 2020; James and Johns, 2016; Zhang et al., 2015). In contrast, actions tend to exhibit higher entropy during training.

Theoretical attacks. The theoretical adversary in this work aims to gradually increase the probability of a desired bad event by occasionally overriding the agent's actions. In particular, our attack is provably effective when (1) the RL setting has almost deterministic rewards and state transition functions, which is common in some engaging scenarios, (2) the learning and testing phases are combined into a continuous process, (3) the attack can depend on the policy of the agent. Looking ahead, we will remove all these constraints in our actual experiments, and yet remain highly effective. While near-determinism is assumed in theoretical attacks, our experimental attacks remain effective even in highly stochastic environments. Nonetheless, this assumption is valid in specific practical scenarios. For instance, robotic manipulation tasks are often modeled as deterministic MDPs due to the predictability of physical laws (Whitman et al., 2020; James and Johns, 2016; Zhang et al., 2015). Similarly, Reinforcement Learning from Human Feedback (RLHF) is employed to align language models with human preferences (Ouyang et al., 2022; Christiano et al., 2017). In this setting, the prompt serves as the state, the model selects a token as an action, and the transition is deterministic: the next state is formed by appending the selected token to the current prompt.

Experimental attacks. Our theoretical result comes with several restrictions (notably the near determinism); however, the main contribution of our work is to do extensive experiments to show that *none* of these limitations apply when we heuristically instantiate a concrete adversary and address the challenges that arise. In particular, we evaluate our targeted poisoning attack against agents representing the two fundamental reinforcement learning paradigms: off-policy and on-policy value estimation in a tabular setting. We also assess our attack on deep reinforcement learning. Furthermore, we examine how effectively our attack increases the probability of the bad event against two broad classes of prior defenses: robustified value updates (Klima et al., 2019) and risk-sensitive policy evaluation (Pan et al., 2019), which were initially developed to counter earlier forms of poisoning attacks. Analyzing these defenses under our attack framework sheds light on the generality and limitations of targeted poisoning. We study the effectiveness of our attacks for both classic control tasks and continuous control locomotion tasks in MuJoCo environments.

Black-Box nature of our attacks. As mentioned, an adversary in RL may have access to various components, such as the agent's policy and learning algorithm, or aspects of the environment, including the state and action spaces, transition probabilities, and the reward function. In a black-box setting (Liu and LAI, 2021; Yu and Sun, 2022), however, the adversary has no knowledge of the agent's internal mechanisms—neither its policy nor the algorithm used to optimize it. Additionally, the adversary only interacts with the environment through an API and can observe agent-environment interactions. In practice, black-box attacks are preferred, but their design is more challenging.

Although much of the prior work on adversarial attacks in reinforcement learning relies on white-box assumptions in some way (Zhang et al., 2020b; Rakhsha et al., 2020; Xu et al., 2021), our *experimental attacks* operate under a black-box setting, where the adversary does not know the agent's algorithm or policy. Furthermore, it does not know the environment's state and action spaces, transition probabilities, or reward function. The adversary only has API access to the environment; it can observe the outputs of the agent-environment interactions and, given a specific state, send an action to the environment to observe the resulting next state and reward. In contrast, our theoretical results assume that the adversary is aware of the agent's policy and action space. We circumvent this challenge in our experiments by training our own policy in conjunction with the attack.

1.2 Further related work

Research on poisoning attacks in RL can be examined from various aspects. Existing studies can generally be compared from three perspectives: the adversary's knowledge, the adversary's budget, and the adversary's goal. The adversary's knowledge refers to the information available to an adversary about the target model or its environment. Xu and Singh (2023); Cai et al. (2022) introduce a black-box setting in which the adversary does not know the environment or the algorithm the agent

uses to find the optimal policy. Instead, the adversary only receives information from the agent's interaction with the environment. On the other hand, Zhang et al. (2020b) describes a scenario where the adversary has full access to both the environment and the agent. The adversary's budget refers to limitations on an adversary's ability to perturb the environment. Early research has explored various techniques for manipulating the environment in which an agent operates (Behzadan and Hsu, 2019). Using a theoretical framework, Zhang et al. (2020a) aimed to characterize the intricacies of an observationally perturbed RL environment, while Yu and Sun (2022) introduces a black-box adversary capable of perturbing the state space. The adversaries in the works of Xu et al. (2022); Cai et al. (2022); Wu et al. (2023) manipulate the reward observed by the agent. Ultimately, Liu and LAI (2021) introduces an attack model involving an adversary who manipulates actions that an agent takes, and some empirical studies have been done in action poisoning against deep RL agents (Lee et al., 2020; Tan et al., 2020). The adversary's goal in reinforcement learning is usually maximizing *regret* (Wei et al., 2022; Wang, 2022) or maximizing *loss* (Xu and Singh, 2023; Zhang et al., 2020b; Liu et al., 2022). While adversarial attacks are often studied to design efficient adversaries that degrade an agent's performance, another line of research focuses on making reinforcement learning agents more robust to such attacks. Various works have explored robustness against different types of adversarial perturbations, including reward manipulation (Banihashem et al., 2021; Bouhaddi and Adi, 2024), action interference (Tessler et al., 2019; Tan et al., 2020), and observation corruption (Mandlekar et al., 2017; He et al., 2024). These approaches leverage worst-case optimization, adversarial training, and certified robustness to ensure a stable policy.

2 Background

RL formulation. In RL, every problem involves two key components: the environment and the agent. The agent aims to achieve a goal by interacting with the environment and understanding its dynamics. The environment is modeled as a Markov Decision Process (MDP), denoted by $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, R, P, T, \mu_0, \gamma \rangle$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. The reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{P}(\mathbb{R})$ maps state-action-state triples to a probability distribution over $\mathbb{R}$, and the transition function $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ defines the probability distribution over next states given a state-action pair. The horizon T specifies the number of time steps during which the agent interacts with the environment. An *episode* in an MDP consists of a sequence of states, actions, and rewards, starting from an initial state s_0 drawn from the distribution μ_0 and ending when the horizon T is reached. Finally, the discount factor $\gamma \in [0, 1]$ determines the agent's preference for immediate rewards over future ones. The agent interacts with the environment during an episode using a *policy*, denoted by π, which is a mapping $\pi_t : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. $\pi_t(a|s)$ represents the probability of taking action a in state s under policy π in time step t. Naturally, $\sum_{a \in \mathcal{A}} \pi_t(a|s) = 1$ for all states $s \in \mathcal{S}$. In a *deterministic* policy for every state $s \in \mathcal{S}$, there exists an action $a \in \mathcal{A}$ such that $\pi_t(a|s) = 1$, which for simplicity is denoted by $\pi_t(s)$. A policy at time π_t can be updated based on the policy at the previous time π_{t-1} and the information (s_{t-1}, a_t, s_t, r_t) that describe the transition at time t.

RL as a random process. In this paper, we represent an RL process with horizon T, using an MDP, $\mathcal{M}$, along with the policy π, and denote this as $\overline{M}^{\pi}_{3T}$, describing it as a random process (a sequence of random variables):

$$\overline{M}^{\pi}_{3T} \equiv (M^{\pi}_0, M^{\pi}_1, \ldots, M^{\pi}_{3T}) \equiv \left(S_0, \langle A_t, S_t, R_t \rangle_{t=1}^{T}\right).$$

A sample episode from this random process would be

$$\overline{m}^{\pi}_{3T} = (m^{\pi}_0, m^{\pi}_1, \ldots, m^{\pi}_{3T}) = \left(s_0, \langle a_t, s_t, r_t \rangle_{t=1}^{T}\right),$$

which is sampled by letting $m^{\pi}_0 = s_0 \sim \mu_0$ and then sampling the following items for all $t \in [T]$:

- **Actions:** $m^{\pi}_{3t-2} = a_t \sim \pi_t(\cdot|s_{t-1})$,
- **States:** $m^{\pi}_{3t-1} = s_t \sim P(\cdot|s_{t-1}, a_t)$,
- **Rewards:** $m^{\pi}_{3t} = r_t \sim R(\cdot|s_{t-1}, a_t, s_t)$.

Our theoretical results are independent of any specific reinforcement learning algorithm, demonstrating that the attack remains effective across a broad range of RL methods. However, in our experiments, we evaluate the attack against several concrete algorithms, the details of which are provided in Appendix A.

3 Threat model

The adversary can manipulate the agent's actions by launching an action-poisoning attack. At each time step t, the adversary decides whether to tamper with the agent's chosen action a_t in state s_{t-1} to a different action $\tilde{a}_t \in \mathcal{A} - \{a_t\}$ ($\tilde{a}_t \neq a_t$), or not (in which case $\tilde{a}_t = a_t$). The agent is unaware of the adversary's presence; therefore, it does not know whether its chosen action has been replaced, and it assumes that all actions are a result of its own decisions. The environment receives the modified action $\tilde{a}_t$ and transitions to state s_t, which is drawn from the distribution $P(\cdot|s_{t-1}, \tilde{a}_t)$. The agent also receives reward $r_t \leftarrow R(\cdot|s_{t-1}, \tilde{a}_t, s_{t+1})$. Consider a sequence in which, at each time step t, the action a_t is replaced by $\tilde{a}_t$, the adversarially altered action. This modification yields a different stochastic process compared to the original one. Even a single altered action can propagate changes throughout the remainder of the sequence, potentially leading to a fundamentally different random process, which we denote as $\overline{N}^{\pi}_{3T}$. In this case, a sample episode and a prefix of an episode are represented as $\overline{n}^{\pi}_{3T}$ and $\overline{n}^{\pi}_{\leq .}$, respectively.

Definition 1 (Bad Event)**.** *Consider an RL process with horizon T, along with policy π, represented as a random process $\overline{M}^{\pi}_{3T}$. The adversary defines a bad event through a condition denoted by* BE*. Let $\overline{m}_{3T} \in \mathrm{Supp}(\overline{M}^{\pi}_{3T})$ denote a trajectory realized under the agent's policy π. This condition can be represented as a Boolean function $\mathrm{BE} : \mathrm{Supp}(\overline{M}^{\pi}_{3T}) \to \{0, 1\}$ indicates whether the bad event has occurred. It is defined as:*

$$\mathrm{BE}(\overline{m}_{3T}) = \mathbb{1}_{\mathrm{BE}}(\overline{m}_{3T}),$$

where $\mathbb{1}_{\mathrm{BE}}$ is the indicator function for the event that the trajectory $\overline{m}_{3T}$ satisfies the condition BE*.*

As seen above, a bad event is a Boolean function defined on the support set of the RL random process in its most general form. Our theoretical results hold in this general setting. The definition of a bad event is broad, allowing it to encompass a wide variety of possible formulations used to characterize undesirable outcomes in reinforcement learning. Our results indicate that, regardless of how the bad event is defined, the adversary can increase its probability.

The agent interacts with the environment to learn the optimal policy while the adversary deploys its poisoning attack strategy. After that, the agent stops improving its policy and uses the policy obtained at the end of the training phase, where we can evaluate the adversary's success using two key metrics: Budget and P_{BE}, representing the adversary's budget and the initial probability of the bad event, in the training and testing episodes. They are denoted as:

- $\mathrm{Budget}(T) = \sum_{t=1}^{T} \mathbb{1}_{(\tilde{a}_t \neq a_t)}$,
- $\mathrm{P}_{\mathrm{BE}}(\overline{N}^{\pi}_{3T}) = \Pr[\mathrm{BE}(\overline{N}^{\pi}_{3T}) = 1]$,

where $\tilde{a}_t$ is the corrupted action introduced by the adversary at time step t, and $\overline{N}^{\pi}_{3T}$ denotes the corrupted RL process. The adversary aims to strike a balance between minimizing its budget and increasing P_{BE}, or maximizing P_{BE} subject to a budget constraint. Note that P_{BE} and Budget are defined with respect to the testing and training episodes, respectively. These episodes may either coincide or differ, depending on the experimental setup.

4 Targeted poisoning attacks on RL

This section presents both the theoretical and experimental targeted poisoning attacks. In each case, the adversary must decide at which time step it is most effective to replace the agent's action to increase the likelihood of achieving its objective.

4.1 Theoretical attacks

In this section, we state and prove our theoretical barrier to RL agents that are robust to targeted poisoning attacks through action manipulation. First, we define $\vec{\rho}_H$-Deterministic random processes.

Definition 2 ($\vec{\rho}_H$-Deterministic Random Process)**.** *A random process $\overline{\mathrm{U}}_H$ is called $\vec{\rho}_H$-deterministic, where $\vec{\rho}_H = (\rho_0, \rho_1, \ldots, \rho_H)$, and $\rho_h \in [0, 1]$ for all $h \in \{0\} \cup [H]$, if*

$$\forall h \in \{0\} \cup [H] \text{ and } \overline{u}_{\leq h-1}, \exists u_h \in \mathrm{Supp}(\mathrm{U}_h) \quad \text{s.t.} \quad \Pr[\mathrm{U}_h = u_h | \overline{u}_{\leq h-1}] > \rho_h.$$

We now define RL settings with almost deterministic reward and state transition functions, which is the setting we use here.

Definition 3 ((α^S, α^R)-deterministic RL Process)**.** *We call an RL process (α^S, α^R)-deterministic if it is $\vec{\rho}_{3T}$-deterministic for some $\vec{\rho}_{3T} = \left(\rho_0^S, \langle \rho_t^A, \rho_t^S, \rho_t^R \rangle_{t=1}^T\right)$, in which $\rho_t^A = 0^T$ (i.e., no restriction on the determinism of action steps) and,*

- ***States:*** $\sum_{t=0}^{T}(1-\rho_t^S) < \alpha^S$,
- ***Rewards:*** $\sum_{t=1}^{T}(1-\rho_t^R) < \alpha^R$.

To account for real-world variability (e.g., sensor noise) and to increase the robustness of our results, we introduce slight stochasticity into this otherwise deterministic setting. It is worth noting that, as shown in Section 4.2, our attack remains effective even in *stochastic environments*. The near-deterministic assumption applies only to our theoretical results, and none of our experiments suffer from this limitation.

We now state our result regarding targeted poisoning attacks by action manipulation. The following theorem shows that if almost all the entropy comes from the actions, then an action-poisoning attack can indeed increase the probability of the bad event from ≈ 0 to ≈ 1. See Appendix D for the proof.

Theorem 1. *Let $\overline{M}_{3T}^{\pi}$ be an (α^S, α^R)-deterministic RL process. Let $\mathrm{BE} : \mathrm{Supp}(\overline{M}_{3T}^{\pi}) \to \{0,1\}$ define a bad event over the process, whose initial probability is ϵ: $\mathrm{P}_{\mathrm{BE}}(\overline{M}_{3T}^{\pi}) = \epsilon$ and where $\epsilon > \alpha = \alpha^S + \alpha^R$. Then for all $\delta \in [0,1]$, there exists an adversary with following features:*

- ***Success.*** $\mathrm{P}_{\mathrm{BE}}(\overline{N}_{3T}^{\pi}) > 1 - \delta - \alpha$ *where $\overline{N}_{3T}^{\pi}$ is the tampered RL process.*
- ***Budget.*** $\mathrm{Budget}(T) = \mathcal{O}\left(\sqrt{T \cdot \ln\left(1/((\epsilon-\alpha)\delta)\right)}\right)$.
- ***Complexity.*** *Given oracle access to the policy, transition probability, and reward functions, the adversary can be implemented in polynomial time over $N/\left((\epsilon-\alpha)\delta\right)$ where N is the total bit representation of the RL process.*

4.2 Experimental targeted poisoning attacks

Our theoretical results are not confined to a specific algorithm or class of algorithms; they hold universally for any RL algorithm. Similarly, the definition of a bad event is highly general and not limited to specific cases. However, to provide concrete empirical validation and demonstrate the practical impact of our attack, we conduct evaluations on representative RL algorithms. Specifically, we evaluate the effectiveness of our attack strategy on both off-policy algorithms, such as Q-learning, and on-policy algorithms, like Expected SARSA, to ensure a comprehensive assessment. Additionally, we test our attacks against three robust variants of these algorithms: Q(κ)-learning and Expected SARSA(κ) from Klima et al. (2019), as well as RARARL from Pan et al. (2019). We also evaluated our attack on Deep Q-Learning and Deep Deterministic Policy Gradient (DDPG), two widely used deep RL algorithms (Mnih et al., 2013). [1] We conduct our experiments in nine settings: the standard classic control environments, Cart-Pole and Mountain-Car, which are widely used in reinforcement learning research; their stochastic variants (see Appendix B.4); and Fire-Grid, a novel environment introduced in this study and inspired by one of our motivating examples (see the second example of the "Motivating examples" Paragraph of Section 1.1), designed to illustrate the notion of bad events intuitively. Additionally, we consider four continuous-control MuJoCo environments: Hopper, Swimmer, Inverted Double-Pendulum, and Half-Cheetah. All environments, except Fire-Grid, are available through OpenAI's `gymnasium` library (Brockman et al., 2016).

4.2.1 Attacked environments

Cart-Pole environment In the Cart-Pole environment, the agent aims to balance a pole on a moving cart by choosing to push the cart left or right. The task ends if the pole falls too far or the cart moves out of bounds, and the agent is rewarded for keeping the pole upright as long as possible (see more details in Appendix B.2).

Bad event in the Cart-Pole environment. In the Cart-Pole environment, a bad event is defined as the agent being in a state where the pole angle exceeds $\pm 6^\circ$, or the cart position exceeds ± 0.8 unit. The

[1] All these algorithms are described in detail in Appendix A.

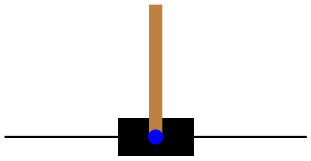

Figure 1: Representation of the Cart-Pole environment.

intuition behind this definition is that exceeding these narrower thresholds indicates the system is nearing instability, making it critical for maintaining control of the cart and pole.

Mountain-Car environment. In the Mountain-Car environment, the agent must drive an underpowered car up a steep hill by first building momentum through strategic movement in a valley. The goal is to reach the hilltop in as few steps as possible, with each time step penalized to encourage efficiency (see more details in Appendix B.3).

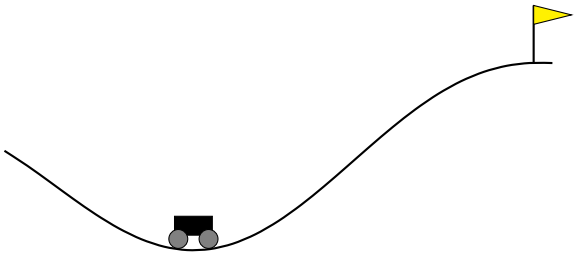

Figure 2: Representation of the Mountain-Car environment.

Bad event in the Mountain-Car environment. In this environment, we define the bad event as the agent taking more than 150 steps to reach $x > 0$. This captures inefficient movement, indicating the agent is failing to build sufficient momentum.

Stochastic versions of Cart-Pole and Mountain-Car environments. To assess robustness under uncertainty, stochastic variants of the Cart-Pole and Mountain-Car environments were created by introducing randomness into transitions and rewards, simulating more realistic and unpredictable dynamics while preserving the original tasks' objectives (see more details in Appendix B.4).

Fire-Grid environment. The Fire-Grid environment simulates a forest where an agent, starting with a limited charge, must navigate a 6×6 grid to reach a charging station located in the bottom-right corner. Along the way, the agent must avoid hazardous wildfire zones and suppress fires in cells containing a manageable amount of fire (represented as white squares) as much as possible. Movement is stochastic, and the agent's objective is to conserve energy by minimizing time and avoiding high-cost areas, making this environment a challenging benchmark for planning under uncertainty and risk (see more details in Appendix B.5).

Bad event in the Fire-Grid environment. The adversary designates a set of states, those affected by wildfire and visually represented by orange squares, as target states, and aims to lead the agent into these states. If the agent passes through even a single target state, the event is considered a bad event.

MuJoco environments. We evaluate our attack in four continuous-control environments: Hopper, Swimmer, Inverted Double Pendulum, and Half-Cheetah. These environments are continuous-control environments that challenge agents to learn complex motor coordination for balance and locomotion. Each task involves controlling a simulated robot, ranging from hopping and swimming to balancing poles and running, by applying torques to joints in order to move forward or maintain stability under dynamic physical conditions (see more details in Appendix B.6).

Bad event in MuJoCo. MuJoCo environments involve applying torques to different joints, where excessive torque can damage the robot and lead to mechanical fatigue. Therefore, we define the bad event in all these environments as a violation of a specified action magnitude range. In the Hopper

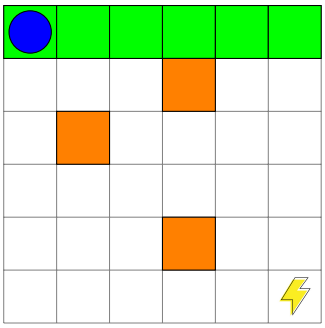

Figure 3: Fire-Grid Environment. The blue circle represents the agent, the top row of green squares indicates possible initial positions to start an episode, the orange squares denote locations with wildfire, and the yellow thunderbolt icon represents the charging station.

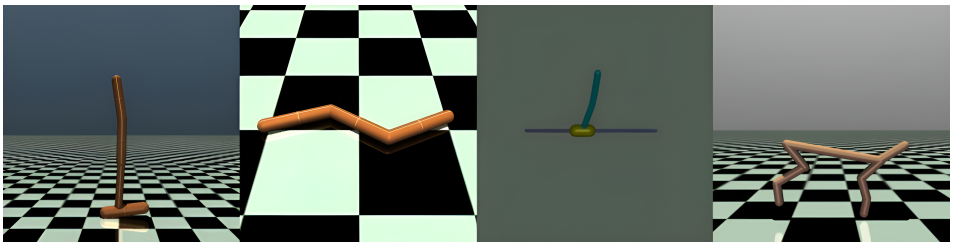

Figure 4: From left to right: Hopper, Swimmer, Inverted Double-Pendulum, and Half-Cheetah.

environment, a bad event occurs if the agent takes an action with magnitude exceeding a specific threshold for more than 20% of the time.

4.3 Results

In this section, we report our findings on the effectiveness of our attacks in all the various combinations of the environments and agent algorithms described above. [2] However, before reporting on our experiment's results, here we present four key findings derived from our experimental results.

- **Effective under stochasticity:** Our results in the Fire-Grid environment and the stochastic versions of Cart-Pole and Mountain-Car show that even when near-deterministic conditions do not hold, and state transitions and reward signals are stochastic, our attack remains effective in increasing the probability of bad events.
- **Diverse definitions of bad events:** Our experiments confirm the attack's effectiveness across diverse bad event definitions. In Cart-Pole and Fire-Grid, the adversary targets a set of states $\mathcal{S}^\dagger \subset \mathcal{S}$. In Mountain-Car, the bad event is defined over $\mathcal{S} \times [T]$, penalizing failure to reach states with certain features within a time horizon. In MuJoCo, it is defined over the action space $\mathcal{A}$, where the agent is pushed toward harmful actions. Despite these varied definitions, the adversary remains consistently effective.
- **Adversary's budget:** We consider an adversary budget inspired by our theoretical results. Specifically, our theoretical analysis indicates that the budget required for the adversary to achieve its objective is $\mathcal{O}(\sqrt{T})$. To apply this in practice, we first train the agent in a non-adversarial setting to estimate the typical episode length, denoted T'. Based on this estimate, we set the adversary's budget to $\sqrt{T'}$. Remarkably, even with this constrained budget, the adversary is still capable of significantly increasing the probability of the undesirable event.
- **Effective in different levels of stochasticity:** We experimented with varying levels of stochasticity in the Fire-Grid environment to evaluate the robustness of our attack in increasing the probability of the bad event. Our results show that even under high levels of

[2] Codes are available here: `https://anonymous.4open.science/r/Targeted-Poisoning-of-Reinforcement-Learning-Agents-D803/`

stochasticity, the adversary remains effective, and the agent continues to exhibit behavior that satisfies the adversary's objective.

Table 1 presents the adversary's performance across the aforementioned environments against various agents. The entry in row i and column j in this table indicates the probability of the bad event in the environment j when the agent is trained using the algorithm i.

Table 1: Results of targeted poisoning attack against Q-learning, expected SARSA, Deep Q-learning, Q(κ)-learning, expected SARSA(κ), and RARARL in the Cart-Pole (CP), Mountain-Car (MC), Stochastic Cart-Pole (stoc-CP), Stochastic Mountain-Car (stoc-MC), and Fire-Grid environments.

Setting	**CP**	**MC**	**stoc-CP**	**stoc-MC**	**FG**
Non-adversarial	0.038	0.174	0.053	0.162	0.088
Adversarial					
Q-learning	0.737	0.438	0.695	0.392	0.618
expected SARSA	0.822	0.449	0.831	0.466	0.561
Deep Q-learning	0.573	0.475	0.560	0.421	0.712
Q(κ)-learning	0.682	0.389	0.702	0.375	0.567
expected SARSA(κ)	0.788	0.350	0.791	0.471	0.480
RARARL	0.863	0.604	0.814	0.659	0.472

Our results show that, across all environments, the adversary can significantly increase the probability of the bad event, regardless of the agent's specific method for policy optimization. Notably, all algorithms exhibited comparable levels of vulnerability.

Table 2 shows targeted poisoning results in Fire-Grid under different stochasticity levels. As detailed in Appendix B.5, the agent moves in the intended direction with probability $p = 0.8$, and otherwise in a perpendicular direction. We varied p from 0.2 to 1.0 to test the adversary's robustness. The adversary remains effective across all settings. Results are based on a Q-learning agent.

The results of the targeted poisoning attack in MuJoCo environments are presented in Table 3.

Table 2: The probability of the bad event at different levels of stochasticity in the Fire-Grid environment.

Stochasticity Level (p)	$\mathrm{P_{BE}}$
0.2	0.618
0.4	0.714
0.6	0.603
0.8	0.651
1.0	0.549

Table 3: The probability of the bad event in both non-adversarial (Non-Adv) and adversarial (Adv) settings when using the DDPG algorithm across various MuJoCo environments.

Environment	**Non-Adv**	**Adv**
Hopper	0.148	0.412
Swimmer	0.103	0.584
Inverted Double-Pendulum	0.067	0.388
Half-Cheetah	0.093	0.479

5 Conclusion and future directions

We introduce *targeted poisoning*, a novel attack where the adversary selectively alters actions to increase the probability of a bad event by adapting to the agent's algorithm. Even a polynomial-time adversary can significantly amplify this risk. Unlike prior methods, our attack accounts for execution order. We demonstrate its stealth and effectiveness across nine environments, five RL algorithms (Q-learning, Expected SARSA, Deep Q-learning, DDPG), and three defenses, showing a marked increase in bad event probability. Given the novelty of targeted poisoning attacks in this work, no existing defenses are specifically designed to counter them. We adapted several relevant strategies but found they failed to mitigate the attack, revealing a critical gap. This highlights the need for robust RL algorithms that can withstand such adversarial manipulations, a key direction for future research.

Our experimental results suggest that targeted poisoning attacks can still be impactful even when the state transition and reward functions are stochastic. An intriguing direction for future research would be to systematically investigate the effectiveness of targeted poisoning attacks in such environments from a theoretical perspective. Furthermore, an important extension would be studying adversaries

that manipulate rewards or states, rather than actions, to evaluate their success in achieving the unique objective of targeted poisoning.

References

Afsar, M. M., Crump, T., and Far, B. (2022). Reinforcement learning based recommender systems: A survey. *ACM Computing Surveys*, 55(7):1–38.

Banihashem, K., Singla, A., and Radanovic, G. (2021). Defense against reward poisoning attacks in reinforcement learning. *arXiv preprint arXiv:2102.05776*.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. (2006). Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25.

Behzadan, V. and Hsu, W. (2019). Adversarial exploitation of policy imitation. *arXiv preprint arXiv:1906.01121*.

Behzadan, V. and Munir, A. (2017). Vulnerability of deep reinforcement learning to policy induction attacks. In *Machine Learning and Data Mining in Pattern Recognition: 13th International Conference, MLDM 2017, New York, NY, USA, July 15-20, 2017, Proceedings 13*, pages 262–275. Springer.

Bouhaddi, M. and Adi, K. (2024). When rewards deceive: Counteracting reward poisoning on online deep reinforcement learning. In *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 38–44. IEEE.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Cai, K., Zhu, X., and Hu, Z.-L. (2022). Black-box reward attacks against deep reinforcement learning based on successor representation. *IEEE Access*, 10:51548–51560.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

Etesami, O., Mahloujifar, S., and Mahmoody, M. (2019). Computational concentration of measure: Optimal bounds, reductions, and more. In *ACM-SIAM Symposium on Discrete Algorithms*.

He, X., Huang, W., and Lv, C. (2024). Trustworthy autonomous driving via defense-aware robust reinforcement learning against worst-case observational perturbations. *Transportation Research Part C: Emerging Technologies*, 163:104632.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. (2017). Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*.

James, S. and Johns, E. (2016). 3d simulation for robot arm control with deep q-learning. *arXiv preprint arXiv:1609.03759*.

Jiang, Z., Xu, D., and Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*.

Klima, R., Bloembergen, D., Kaisers, M., and Tuyls, K. (2019). Robust temporal difference learning for critical domains. *arXiv preprint arXiv:1901.08021*.

Lee, X. Y., Ghadai, S., Tan, K. L., Hegde, C., and Sarkar, S. (2020). Spatiotemporally constrained action space attacks on deep reinforcement learning agents. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4577–4584.

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

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N. M. O., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015b). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

Liu, G. and LAI, L. (2021). Provably efficient black-box action poisoning attacks against reinforcement learning. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12400–12410. Curran Associates, Inc.

Liu, Z., Guo, Z., Cen, Z., Zhang, H., Tan, J., Li, B., and Zhao, D. (2022). On the robustness of safe reinforcement learning under observational perturbations. *arXiv preprint arXiv:2205.14691*.

Mandlekar, A., Zhu, Y., Garg, A., Fei-Fei, L., and Savarese, S. (2017). Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3932–3939. IEEE.

Maurer, U. (2002). Indistinguishability of random systems. In *Advances in Cryptology—EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28–May 2, 2002 Proceedings 21*, pages 110–132. Springer.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Pan, X., Seita, D., Gao, Y., and Canny, J. F. (2019). Risk averse robust adversarial reinforcement learning. *2019 International Conference on Robotics and Automation (ICRA)*, pages 8522–8528.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In *International conference on machine learning*, pages 2817–2826. PMLR.

Rakhsha, A., Radanovic, G., Devidze, R., Zhu, X., and Singla, A. (2020). Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *International Conference on Machine Learning*, pages 7974–7984. PMLR.

Rakhsha, A., Radanovic, G., Devidze, R., Zhu, X., and Singla, A. (2021). Policy teaching in reinforcement learning via environment poisoning attacks. *Journal of Machine Learning Research*, 22(210):1–45.

Shafahi, A., Huang, W. R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., and Goldstein, T. (2018). Poison frogs! targeted clean-label poisoning attacks on neural networks. *Advances in neural information processing systems*, 31.

Szepesvári, C. (2022). *Algorithms for reinforcement learning*. Springer nature.

Tamar, A., Castro, D. D., and Mannor, S. (2016). Learning the variance of the reward-to-go. *J. Mach. Learn. Res.*, 17:13:1–13:36.

Tan, K. L., Esfandiari, Y., Lee, X. Y., Sarkar, S., et al. (2020). Robustifying reinforcement learning agents via action space adversarial training. In *2020 American control conference (ACC)*, pages 3959–3964. IEEE.

Tessler, C., Efroni, Y., and Mannor, S. (2019). Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, pages 6215–6224. PMLR.

Wang, C. (2022). Attacking and defending deep reinforcement learning policies. *arXiv preprint arXiv:2205.07626*.

Wei, C.-Y., Dann, C., and Zimmert, J. (2022). A model selection approach for corruption robust reinforcement learning. In *International Conference on Algorithmic Learning Theory*, pages 1043–1096. PMLR.

Whitman, J., Bhirangi, R., Travers, M., and Choset, H. (2020). Modular robot design synthesis with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10418–10425.

Wu, Y., McMahan, J., Zhu, X., and Xie, Q. (2023). Reward poisoning attacks on offline multi-agent reinforcement learning. In *Proceedings of the aaai conference on artificial intelligence*, volume 37, pages 10426–10434.

Xu, H., Wang, R., Raizman, L., and Rabinovich, Z. (2021). Transferable environment poisoning: Training-time attack on reinforcement learning. In *Proceedings of the 20th international conference on autonomous agents and multiagent systems*, pages 1398–1406.

Xu, Y. and Singh, G. (2023). Black-box targeted reward poisoning attack against online deep reinforcement learning. *arXiv preprint arXiv:2305.10681*.

Xu, Y., Zeng, Q., and Singh, G. (2022). Efficient reward poisoning attacks on online deep reinforcement learning. *arXiv preprint arXiv:2205.14842*.

Yu, M. and Sun, S. (2022). Natural black-box adversarial examples against deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8936–8944.

Zhang, F., Leitner, J., Milford, M., Upcroft, B., and Corke, P. (2015). Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*.

Zhang, H., Chen, H., Xiao, C., Li, B., Liu, M., Boning, D., and Hsieh, C.-J. (2020a). Robust deep reinforcement learning against adversarial perturbations on state observations. *Advances in Neural Information Processing Systems*, 33:21024–21037.

Zhang, X., Ma, Y., Singla, A., and Zhu, X. (2020b). Adaptive reward-poisoning attacks against reinforcement learning. In *International Conference on Machine Learning*, pages 11225–11234. PMLR.

A Algorithms in experimental attacks

This section discusses the algorithms targeted in our experiments and outlines how we configure their parameters during training.

A.1 Temporal difference learning algorithms.

This section introduces two fundamental functions: the *state value function* and the *state-action value function*. These functions are essential for understanding RL algorithms such as Q-learning, Expected SARSA, and other temporal-difference (TD) learning methods used in this study (see Algorithm 1). Additionally, they serve as the foundation for deriving the optimal policy.

Algorithm 1 TD learning

Result: Agent state-action value function: Q.
Input: Environment: env, Minimum learning rate: α_u, Minimum epsilon: ϵ_u, Decay rate parameter: τ, Discount factor: γ, Number of episodes: H.
Initialization: $Q(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$ arbitrarily.

for episode $h \leftarrow 1$ to H **do**
  Initialize the first state $s_0 \sim \mu_0$;
  $\alpha = \max(\alpha_u, \min(1, 1 - \log(\frac{h}{\tau}))$;
  $\epsilon = \max(\epsilon_u, \min(1, 1 - \log(\frac{h}{\tau})))$;
  The first state $s_0 \sim \mu_0$;
  $done = False$;
  $t = 1$;
  while not $done$ **do**
    Choose action $a_t = \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s_{t-1}, a), & \text{w.p.} \quad 1 - \epsilon \\ \text{random action}, & \text{w.p.} \quad \epsilon \end{cases}$
    Take action $\tilde{a}_t$ at time step t and generate the next state, reward, and the termination signal: $s_t, r_t, done$;
    Update Q-value function:

$$Q(s_{t-1}, a_t) \leftarrow (1 - \alpha) \cdot Q(s_{t-1}, a_t) + \alpha \cdot (\text{td}_{\text{target}}) \tag{1}$$

    if $done$ **then**
      Reset env;
    end if
    $t \leftarrow t + 1$;
  end while
end for

In order to define them, we suppose that the t^{th} transition takes place when the agent takes action a_t in state s_{t-1}, then it transitions to state s_t and gives reward r_t.

The first function to introduce is the state value function (or value function for short), denoted as $V_t^\pi : \mathcal{S} \to \mathbb{R}$. This function is defined for all states $s \in \mathcal{S}$ and represents the expected value of the total future rewards when starting from state s_{t-1} in time step t until the end of the episode and following policy π. It can be formulated as

$$V_t^\pi(s_{t-1}) = \mathbb{E}^\pi \left[\sum_{k=t}^{T} \gamma^{k-t} r_k | s_{t-1} \right],$$

where γ is the discount factor for future rewards from time t onwards, and r_k represents the reward obtained at time k. For all policies π, we have: $V_{T+1}^\pi(s_T) = 0$.

The second one is the state-action value function (or Q-function), denoted as $Q_t^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. This function is defined for all state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ and represents the expected value of the total future rewards when starting from state s_{t-1} and taking action a_t in the t^{th} time step onward,

following the policy π until the end of the episode. It can be formulated as

$$Q_t^\pi(s_{t-1}, a_t) = \mathbb{E}^\pi \left[\sum_{k=t}^{T} \gamma^{k-t} r_k | s_{t-1}, a_t \right].$$

For all policies π and for all actions $a \in \mathcal{A}$, we have: $Q_{T+1}^\pi(s_T, a) = 0$.

In our work, we assume that the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are finite. Additionally, the horizon T is limited, and the reward function R is bounded. As a result, both the value function and the Q-function are also bounded. It is widely recognized that the value function and Q-function adhere to the Bellman optimality equation. As a result, according to the Banach fixed point theorem, the optimal Q-function exists and is unique (Szepesvári, 2022).Based on Bellman optimality equation, for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, the optimal Q-function Q^* can be obtained as bellow

$$\begin{aligned} Q_t^*(s_{t-1}, a_t) &= \mathbb{E}\left[r_t + \gamma \max_a Q_{t+1}^*(s_t, a) | s_{t-1}, a_t \right] \\ &= \sum_{s_t, r_t} \Pr[s_t, r_t | s_{t-1}, a_t] \left[r_t + \gamma \max_a Q_{t+1}^*(s_t, a) \right] \end{aligned}$$

The agent's primary objective is to discover the optimal policy that yields the maximum cumulative reward. Consequently, the optimal policy π^*, which is a deterministic policy, can be determined as follows

$$\pi_t^*(s_{t-1}) = \arg\max_a Q_t^*(s_{t-1}, a)$$

In Algorithm 1, replacing the term $\mathrm{td}_{\mathrm{target}}$ in Equation 1 with $[r_t + \gamma \cdot \max_{a \in \mathcal{A}} Q(s_t, a)]$ yields the Q-learning algorithm. Similarly, substituting $\mathrm{td}_{\mathrm{target}}$ with $[r_t + \gamma \cdot \sum_{a \in \mathcal{A}} \Pr[a|s_t] Q(s_t, a)]$ results in the expected SARSA algorithm.

A.2 Deep RL algorithms that we attack

Since the action spaces in Cart-Pole, Mountain-Car, and Fire-Grid are discrete, we selected a deep reinforcement learning algorithm suitable for such settings—Deep Q-Learning (Mnih et al., 2013). On the other hand, since the action space in the MuJoCo environments is continuous, we evaluated our attack against Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015b), one of the most widely used algorithms for such settings.

Deep Q-learning. Deep Q-learning replaces the tabular Q-value lookup of classic Q-learning with a parameterized neural network that maps high-dimensional observations directly to action-value estimates; this enables handling continuous or very large state spaces that a table cannot represent. During training, Deep Q-learning employs stochastic gradient updates on mini-batches sampled from an experience replay buffer, thereby decorrelating consecutive samples and enhancing sample efficiency. Crucially, it maintains two networks: an online network for selecting actions and a periodically updated target network for generating stable training targets, which prevents the feedback loop instability typically associated with using a single moving network. By leveraging deep function approximation, Deep Q-learning generalizes learned value estimates across similar states, enabling it to scale to tasks with high-dimensional inputs without requiring manual feature engineering.

Deep Deterministic Policy Gradient. DDPG directly learns a deterministic policy for selecting continuous actions via an actor network, while a paired critic network estimates the quality of those actions. The actor network is updated by following gradients from the critic's value estimates, enabling efficient policy improvement in continuous domains without action discretization. Both networks are trained off-policy using mini-batches drawn from an experience replay buffer, which breaks temporal correlations and enhances sample efficiency. To ensure stable learning, DDPG maintains separate target networks for the actor and critic that slowly track the learned networks, thereby smoothing the targets used during updates and preventing divergence. By combining deterministic policy gradients with deep function approximation and stabilization techniques borrowed from DQN, DDPG provides a powerful framework for continuous control tasks where both state and action spaces are high-dimensional and continuous.

A.3 Three proposed robust algorithms that we attack

In this work, we evaluate the robustness of certain proposed algorithms against targeted poisoning attacks. Specifically, we consider three algorithms: Q(κ)-learning and expected SARSA(κ) from Klima et al. (2019), as well as RARARL from Pan et al. (2019).

A.3.1 Q(κ)-learning and expected SARSA(κ) algorithms

In reinforcement learning, agents must often operate in environments where external adversaries may attempt to interfere with their decision-making. One such scenario involves a Q-learning agent that seeks to develop a robust policy while contending with a potentially malicious adversary. In this setting, the adversary can seize control of the agent's action with probability x, influencing the learning process. The outcome of each state transition depends on who controls the action: if the agent retains control, it selects the optimal action to maximize the expected return, whereas if the adversary takes control, it may choose an action that minimizes the expected return. Klima et al. (2019) introduced the Q(κ)-learning and expected SARSA(κ) algorithms by modifying the TD target in Equation 1 of Algorithm 1 as follows.

In Q(κ), they define $\mathrm{td}_{\mathrm{target}}$ as:

$$\mathrm{td}_{\mathrm{target}} = r_t + \gamma \left[(1-x) \max_a Q(s_t, a) + x \min_a Q(s_t, a) \right].$$

For expected SARSA(κ), they set $\mathrm{td}_{\mathrm{target}}$ to:

$$\begin{aligned} \mathrm{td}_{\mathrm{target}} &= r_t + \gamma \left[(1-x) \mathbb{E}_{a\sim\pi} Q(s_t, a) + x \min_a Q(s_t, a) \right] \\ &= r_t + \gamma \left[(1-x) \sum_{a \in \mathcal{A}} \Pr[a|s_t] Q(s_t, a) + x \min_a Q(s_t, a) \right]. \end{aligned}$$

These modifications incorporate the adversary's influence, adjusting the learning process to account for adversarial interference in action selection.

As stated in Klima et al. (2019), the agent either has prior knowledge of or can estimate the probability of an attack, denoted as x. In our experiment, we estimate this probability by computing the ratio of instances where the adversary replaces the agent's action to the total number of opportunities for such intervention. Before evaluating our attack against these algorithms, we provide the agent with the estimated value of x to ensure it has access to this crucial parameter. For our experiments, we set it based on the observed frequency of action replacements in attacks against Q-learning or expected SARSA.

A.3.2 RARARL algorithm

Pan et al. (2019) introduced the Risk-Averse Robust Adversarial Reinforcement Learning (RARARL) algorithm, structured as a two-player zero-sum sequential game where the protagonist and adversary take turns influencing the environment. The protagonist carries out a sequence of m actions, after which the adversary responds with n actions. The game is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, P, \gamma)$, where $\mathcal{S}$ represents a potentially infinite state space, and both agents operate within a shared action space $\mathcal{A}$. To enhance robustness, Pan et al. (2019) introduces a risk-averse adjustment to the protagonist's Q-function by incorporating a variance-based penalty, and this idea is inspired by the work of Tamar et al. (2016):

$$\hat{Q}(s, a) = Q(s, a) - \lambda \mathrm{Var}_k \left[Q_k(s, a) \right],$$

where $\mathrm{Var}_k \left[Q_k(s, a) \right]$ denotes the variance of an ensemble of k Q-functions $Q_k(s, a)$, and λ is a constant controlling the level of risk aversion. Similarly, the adversary's objective is modified to encourage risk-seeking behavior using:

$$\hat{\bar{Q}}(s, a) = \bar{Q}(s, a) + \bar{\lambda} \mathrm{Var}_k[\bar{Q}_k(s, a)].$$

The variance of an action a is computed as:

$$\mathrm{Var}_k \left[Q_k(s, a) \right] = \frac{1}{k} \sum_{i=1}^{k} \left[Q_i(s, a) - \frac{1}{k} \sum_{l=1}^{k} Q_l(s, a) \right],$$

where Q_i represents the i^{th} head of a multi-headed Q-value network. An analogous variance computation applies to the adversary using $\bar{Q}$. The details of how this algorithm operates and its training process are outlined in Algorithm 1 of Pan et al. (2019).

While RARARL is designed for continuous state spaces and employs Deep Q-learning rather than tabular Q-learning, our work requires modifications to align with a different attack model. Additionally, RARARL follows a turn-based structure where the protagonist and adversary alternate taking actions. In contrast, our approach involves the agent selecting an action at each time step, with the adversary dynamically deciding, based on a budget constraint, whether to perturb the action at that moment. Therefore, we adapt the concept of the RARARL algorithm to make it compatible with the context of our attack in this work.

Parameter setting. In our attacks, the agent first undergoes a training phase. Once training is complete, the testing phase is conducted. We estimate the probability of the bad event during testing by deploying the learned policy across a sufficiently large number of episodes and computing the fraction in which the bad event occurs. To enhance confidence in our estimates for both adversarial and non-adversarial settings, we repeat the entire training and testing process multiple times and average the occurrences of the bad event.

Some parameters in our experiments, such as those used in the epsilon-greedy strategy and the learning rate, are selected based on best practices commonly adopted in the reinforcement learning literature. During early training, the agent emphasizes exploration by starting with a high exploration rate (ϵ) and learning rate (α), giving greater weight to new experiences. As training progresses, both rates decay logarithmically, gradually shifting the agent's behavior toward exploitation. In our experiments, we initialize with $\alpha_0 = \epsilon_0 = 1$ and decay both rates until they reach a lower bound of $\alpha_u = \epsilon_u = 0$ (see Algorithm 1 for details).

Additionally, for parameters such as the number of training episodes, we ensure that training proceeds until the agent's policy exhibits clear signs of convergence. For example, as illustrated in Figure 5, the agent's policy in the Cart-Pole environment converges to an effective policy after approximately 200 episodes. Nonetheless, we continue training beyond this point to ensure robust convergence. A similar approach is adopted in all other environments studied.

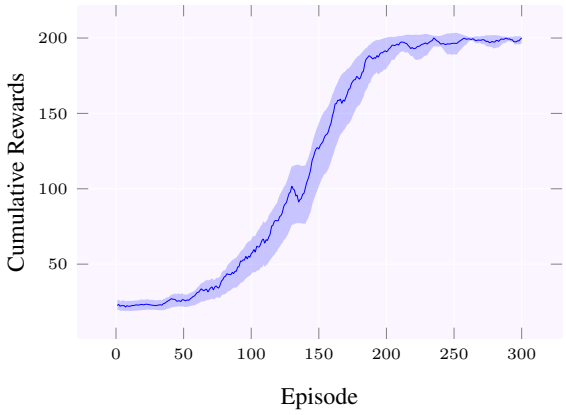

Figure 5: Cumulative rewards during the training phase in the Cart-Pole environment with a 95% confidence interval under the non-adversarial setting.

Moreover, whenever possible, we adopt algorithm parameters directly from the original papers that introduced robust variants of reinforcement learning algorithms. For instance, we use the same values for κ in Q(κ)-learning and Expected SARSA(κ), and for λ in RARARL, as specified in those studies. Additionally, for parameters such as K (the number of random continuations), we select values that strike a balance between computational efficiency and the effectiveness of the attack. Specifically, we select values that avoid significant computational overhead while still allowing the adversary to perform reliably.

The parameter η in Algorithm 2 can be determined as follows (Etesami et al., 2019):

$$\eta = \sqrt{\frac{2\ln\left(\frac{1}{\epsilon}\right)}{T}}, \tag{2}$$

where ϵ denotes the probability of the bad event in the absence of an adversarial attack, and T is the episode length.

Finally, we emphasize that for some parameters, we experimented with only a few values and still observed a significant increase in the probability of the bad event. This highlights the strength and robustness of our attack, even in the absence of extensive parameter tuning.

B Additional explanation of the experiments

This section provides detailed information about the environments used to evaluate our algorithms, as well as the strategy by which the adversary determines where to modify the agent's actions.

B.1 Replacing action in targeted poisoning attack

We now provide a high-level description of our experimental attacks. The big picture of our attack is inspired by our theoretical attack. However, as mentioned, we need to go around several limitations. One major limitation is to make the attack efficient enough to be implemented, even if the adversary does not have access to the full action space. To achieve this goal, we use the concept of maintaining a set of actions by the adversary as it observes the agent's interactions (denoted by $\mathcal{A}_{\text{adv}}$ in Algorithm 2). Then, the adversary always picks its corrupted action from this space. A second idea is to avoid the dependency on knowing the agent's policy at every moment (to become a black-box attack). This requires our attacker to train their own policies.

Suppose the agent is at time step $t-1$, meaning a prefix trajectory pref up to state s_{t-1} has been realized, and the agent is about to select action a_t. The adversary must then decide whether this is an appropriate point to replace the agent's action. To do so, it generates K virtual random continuations in its own simulation, from the current state and action a_t to estimate the probability of a bad event occurring. The adversary then repeats this process by substituting a_t with other actions, estimating the probability of the bad event for each. Based on these estimates, the adversary determines whether to replace the original action. This procedure is outlined in Algorithm 2.

In our experimental attack, the adversary observes the agent-environment interactions and learns its own policy, which differs from that of the agent. When generating random continuations in Algorithm 2, the adversary uses this learned policy. To simulate a random continuation, it sends a sequence of queries to the environment to complete that episode only in its own head. In contrast, the theoretical adversary is assumed to have access to the agent's policy.

Below, you can find more details regarding the environments in which we tested targeted poisoning attacks.

B.2 Cart-Pole environment

In the Cart-Pole environment, the agent's goal is to keep a pole upright by moving a cart left or right. The pole stays upright if its angle remains within $\pm 12^\circ$ of the vertical. The state space consists of four continuous variables: cart position, velocity, pole angle, and angular velocity. At each time step, the agent observes the state in its continuous form and maps it to a discrete counterpart using a mapping function. The action is then selected based on this discretized state. This approach enables the application of algorithms such as Q-learning and Expected SARSA in the environment.

The action space consists of two discrete choices: moving the cart left or right. The agent receives a reward of +1 for each time step the pole remains upright. An episode terminates if (i) the pole tilts beyond $\pm 12^\circ$, (ii) the cart moves outside ± 2.4 units, or (iii) the maximum reward threshold of +200 is reached. Similar to many robotic tasks, both state transition and reward in this environment are deterministic in each transition.

Algorithm 2 Our framework of action-replacement decisions

Result: Output action: $\tilde{a}_t$, Remaining budget: Budget_t.
Input: Realized prefix: pref, Current state: s_{t-1}, Current action: a_t, Attack strength 2: η, Budget Budget_{t-1}, Adversary memory of actions: $\mathcal{A}_{\mathrm{adv}}$.

Generate K random continuations (in adversary's head) to complete the prefix pref: $\{\mathrm{rc}_1^0, \ldots, \mathrm{rc}_K^0\}$;
Estimate baseline bad event probability: $\mathrm{P}_{\mathrm{BE}}^0 \leftarrow \frac{1}{K}\sum_{k=1}^{K} \mathrm{BE}(\mathrm{pref}, \mathrm{rc}_k^0)$;
Generate K random continuations to complete the episode after choosing a_t as the action at time step $t-1$: $\{\mathrm{rc}_1^{a_t}, \ldots, \mathrm{rc}_K^{a_t}\}$;
$\mathrm{P}_{\mathrm{BE}}^{a_t} \leftarrow \frac{1}{K}\sum_{k=1}^{K} \mathrm{BE}(\mathrm{pref}, \mathrm{rc}_k^{a_t})$;
for all $a \in \mathcal{A}_{\mathrm{adv}}$ **do**
    Generate K random continuations to complete the episode after choosing a as the action at time step $t-1$: $\{\mathrm{rc}_1^{a}, \ldots, \mathrm{rc}_K^{a}\}$;
    $\mathrm{P}_{\mathrm{BE}}^{a} \leftarrow \frac{1}{K}\sum_{k=1}^{K} \mathrm{BE}(\mathrm{pref}, \mathrm{rc}_k^{a})$;
end for

$a_t^* \leftarrow \arg\max_{a \in \mathcal{A}_{\mathrm{adv}}} (\mathrm{P}_{\mathrm{BE}}^{a})$;
if $\mathrm{Budget}_{t-1} > 0$ **and** $\left(\mathrm{P}_{\mathrm{BE}}^{a^*} \geq e^{\eta} \cdot \mathrm{P}_{\mathrm{BE}}^0 \ \vee \ \mathrm{P}_{\mathrm{BE}}^{a_t} < e^{-\eta} \cdot \mathrm{P}_{\mathrm{BE}}^0\right)$ **then**
    $\tilde{a}_t \leftarrow a^*,\ \mathrm{Budget}_t \leftarrow \mathrm{Budget}_{t-1} - 1$
else
    $\tilde{a}_t \leftarrow a_t,\ \mathrm{Budget}_t \leftarrow \mathrm{Budget}_{t-1}$
end if

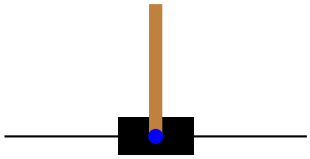

Figure 6: Representation of the Cart-Pole environment.

B.3 Mountain-Car environment

The Mountain-Car environment is a classic control benchmark in which an underpowered car must build momentum to climb a steep hill. The agent starts at a random position near the bottom of a sinusoidal valley and must strategically accelerate left or right to build enough momentum to reach the goal at the top of the right hill. The state space consists of two continuous variables: the car's position and velocity. The position is constrained within $[-1.2, 0.6]$, while the velocity is clipped between $[-0.07, 0.07]$. As in the Cart-Pole environment, we discretize the state space in the Mountain-Car environment to enable the application of tabular algorithms such as Q-learning and Expected SARSA. The action space consists of three discrete actions: accelerating to the left, accelerating to the right, or maintaining the current speed. Due to the car's low power, it cannot reach the goal in a single move and must leverage the valley's slopes to gain momentum. The transition dynamics follow a deterministic update rule where the chosen action, a constant force, and the gravitational effect of the terrain influence velocity. Collisions with the environment's boundaries are inelastic, setting the velocity to zero. The agent receives a reward of -1 per time step, encouraging it to reach the goal in the shortest time possible. An episode terminates if the agent reaches the goal (position $x \geq 0.5$) or exceeds 200 steps.

B.4 Stochastic versions of Cart-Pole and Mountain-Car environments

To further demonstrate that stochasticity does not hinder the effectiveness of our attacks, we introduced stochastic elements into the Cart-Pole and Mountain-Car environments, which are natively deterministic, consistent with the assumptions of our theoretical analysis, by slightly modifying their source code to incorporate random transitions and rewards. In the stochastic versions of these environments, each state transition follows the environment's original nature with a probability of

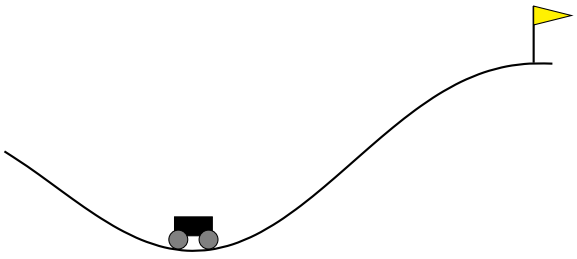

Figure 7: Representation of the Mountain-Car environment.

0.8. With the remaining 0.2 probability, the agent transitions to an alternative state generated by adding Gaussian noise (mean zero, standard deviation 0.01) to the standard next state. Additionally, we modified the reward function by adding Gaussian noise with zero mean and a standard deviation of 0.1, introducing variability in the received rewards. As a result, the reward signal becomes fully stochastic.

The standard deviation values of the random gaussian noises were chosen to introduce meaningful variability in the state and reward while ensuring that the perturbations were not so large as to completely hinder the agent's ability to learn an optimal policy. [3]

B.5 Fire-Grid environment

In this environment, a hypothetical forest is modeled as a 6×6 grid (shown in Figure 3). The agent's starting position is selected uniformly at random from the states in the top row (green squares). The agent aims to reach the charging station in the bottom right corner before depleting its charge while attempting to extinguish fires. Starting with a charge of 100, the agent loses a portion of its charge with each step.

White squares represent areas of fire that are easily suppressible by the agent. In contrast, three orange squares represent areas with wildfires. Entering these squares causes the agent to lose a significant portion of its charge, making it advantageous to avoid them. As the agent loses charge, it receives a negative reward. When the agent reaches this position, it is recharged, and the episode restarts.

A state in this environment consists of two components: the agent's position (comprising the x and y coordinates) and the agent's current charge level. The action space consists of four possible actions: moving up, down, left, and right. The agent's state transition is stochastic. When it takes an action, it moves in the intended direction with a probability of 0.8, and in each perpendicular direction with a probability of 0.1. If the agent attempts to move outside the grid, it remains in its current position. At each time step, if the agent remains in its current position, it loses charge according to a normal distribution with a mean of -1 and a standard deviation of 0.1 ($\mathcal{N}(-1, 0.1)$). If the agent moves to an orange state with high-intensity fire, it loses charge according to $\mathcal{N}(-20, 2)$. For all other states, the charge loss follows $\mathcal{N}(-2, 0.2)$. In each case, the agent receives a reward from $\mathcal{N}(-0.5, 0.01)$, $\mathcal{N}(-10, 1)$, and $\mathcal{N}(-1, 0.1)$, respectively.

The agent must reach the charging station before its charge drops below 5. If the agent's charge falls below 5 while it is in a position other than the charging station, it receives a reward drawn from $\mathcal{N}(-30, 2)$. In the Fire-Grid environment, an episode ends when the agent reaches the charging station or its charge falls below 5.

B.6 MuJoCo environments: Hopper, Swimmer, Inverted Double Pendulum, and Half-Cheetah

The Hopper, Swimmer, Inverted Double-Pendulum, and Half-Cheetah are all continuous-control benchmark tasks that test an agent's ability to coordinate forces in order to achieve and maintain dynamic balance or precise trajectories.

[3] For more information on the natural ranges defined by the state spaces and rewards in these environments, see the Gymnasium documentation at `https://gymnasium.farama.org/environments/classic_control/`.

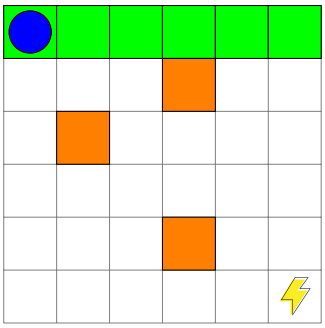

Figure 8: Fire-Grid Environment. The blue circle represents the agent, the top row of green squares indicates possible initial positions to start an episode, the orange squares denote locations with wildfire, and the yellow thunderbolt icon represents the charging station.

In the Hopper, a planar one-legged robot made up of torso, thigh, shin, and foot, must learn to apply torques at its three hinge joints so that it repeatedly hops forward without tipping over.

The Swimmer is a chain of three or more rigid links joined by rotary actuators and immersed in a two-dimensional viscous medium; by learning the correct sequence of joint torques, the agent must generate traveling waves along the body to propel itself to the right.

In the Inverted Double-Pendulum, a cart on a one-dimensional track carries two rigid poles in series—only the tip of the second pole is free—and must learn to apply continuous lateral forces on the cart so that both poles remain balanced upright.

Finally, the Half-Cheetah is a planar (2D) robotic agent composed of 9 interconnected body segments and 8 joints, including two that represent paws. The task is to control the robot by applying torque at specific joints to propel it forward (to the right) as efficiently as possible. Progress in the forward direction yields a positive reward, while backward motion results in a penalty. The robot's torso and head remain rigid and immobile, while torque can be applied to six joints: those at the front and rear thighs (linked to the torso), the shins (connected to the thighs), and the feet (attached to the shins), enabling coordinated locomotion. [4]

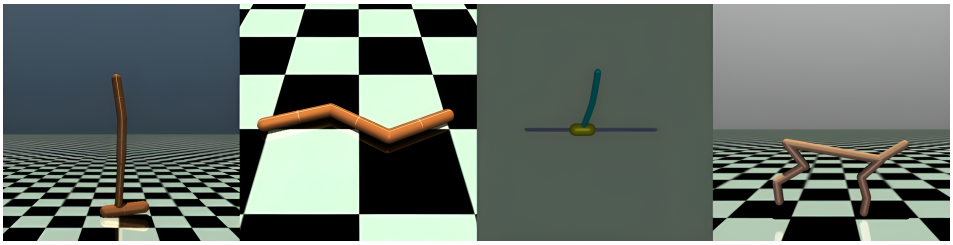

Figure 9: Environments used for additional experiments. From top left to bottom right: Hopper, Swimmer, Inverted Double-Pendulum, and Half-Cheetah.

C Borrowed technical tools

In this section, we present two results: one bounds the statistical distance between two random processes, and the other addresses replacing attacks.

Bounding total Variation distance. The following lemma is a well-known lemma that has been used extensively in the literature. For the proof of this, you can see the results of Maurer (2002).

[4] Additional details about these environments can be found at `https://gymnasium.farama.org/environments/mujoco/`.

Lemma 2. *Consider two random processes with H time steps $\overline{U}^1_H$ and $\overline{U}^2_H$ such that for all prefixes $u_{\leq h-1} \in \mathrm{Supp}(\overline{U}^1_{\leq h-1}) \cap \mathrm{Supp}(\overline{U}^2_{\leq h-1})$. If the total variation distance of the following two conditional distributions: $(u^1_h | \overline{u}^1_{\leq h-1} = u_{\leq h-1})$ and $(u^2_h | \overline{u}^2_{\leq h-1} = u_{\leq h-1})$ is at most ρ_h, then*

$$\delta_{TV}(\overline{U}^1_H, \overline{U}^2_H) \leq \sum_{h=1}^{H} \rho_h$$

where $\delta_{TV}(\cdot, \cdot)$ represents the total variation distance between two random processes.

Based on the adversary's knowledge, its ability to tamper with a sample, and its budget, we can define different types of attacks. In replacing attacks, the adversary can see the training samples and replace some of them. The number of replacements is limited by the adversary's budget.

Definition 4 (Replacing Adversary)**.** *Consider the random process $\overline{U}_H$ as a sequence of random variables, and the adversary* Rep *with a budget of* $\mathrm{Budget}(H)$*. Suppose a prefix of this random process is sampled in the presence of the adversary: $\overline{v}_{\leq h-1} \in \mathrm{Supp}(\overline{U}_{\leq h-1})$. At time step h, u_h is sampled from U_h, and the adversary's remaining budget is $\mathrm{Budget}_h(H)$. Now, the adversary decides whether to replace the sample u_h with another sample $u \in \mathrm{Supp}(U_h) \backslash \{u_h\}$. If it does, it loses one unit from its budget; otherwise, it carries the previous budget to the next step.*

The following theorem is proved regarding replacing attacks:

Theorem 3 (A Result of Etesami et al. (2019))**.** *There is a (uniform) oracle-aided randomized algorithm* Tam *such that the following holds. Suppose $f : \mathrm{Supp}(\overline{U}_H) \to \{0, 1\}$ is a Boolean function for random the process $\overline{U}_H \equiv (U_1, \ldots, U_H)$, and that $\Pr[f(\overline{U}_H) = 1] \geq \varepsilon$. Then, the oracle-aided algorithm $\mathrm{Tam}^{\overline{U}[\cdot], f(\cdot)}(\epsilon, \delta, \cdot)$ (also denoted by* Tam *for simplicity) with access to the online sampler $\overline{U}[\cdot]$ for $\overline{U}_H$ that gets a prefix of a random process and sample the remaining process, and $f(\cdot)$ as oracles is an online replacing algorithm for $\overline{U}_H$ and has the following features:*

1. *$\Pr[f(\overline{V}_H) = 1] \geq 1 - \delta$ where $\overline{V}_H$ is the replaced random process.*
2. *The replacing budget of* Tam *is $\mathcal{O}\left(\sqrt{n \cdot \ln(1/\varepsilon\delta)}\right)$.*
3. *The adversary runs in time $\mathrm{poly}(N/\varepsilon\delta)$, where N is the total length of the bit representation of the random process.*

D Proof of Theorem 1

This section presents the full proof of Theorem 1.

We first derive the information-theoretic attack for the first two cases by using the tools in Appendix C as a black box. We then discuss how to obtain the computationally bounded attack described in the third bullet by examining the details of their attack.

The first two bullets of the theorem describe an *information-theoretic* adversary (IT-CP) with an oracle access to $\mathrm{BE}(\cdot)$, and the third bullet specifies a *computationally-bounded* adversary (CB-CP). At each time step, both adversaries observe the realized prefix, denoted as pref, and decide whether to replace the agent's action based on its expected impact on the bad event. The key distinction between them lies in how they acquire this probability.

The proof proceeds by reducing our setting to the framework of Etesami et al. (2019), as explained in Appendix C: we transform the original process into a nearly deterministic one where only action steps retain randomness, and then apply the corresponding attack to this modified process. The crucial step is to demonstrate that this alternative attack still succeeds in the "real" setting, where the process has not been determinized. This argument also relies on the bounded statistical distance between the real process and the "ideal" determinized process, using tools described in Appendix C.

Under polynomial-time constraints, we must adapt rather than treat prior work as a black box: we implement the oracles needed by the polynomial-time attack of Etesami et al. (2019), in particular approximating the oracle for $\Pr[\mathrm{BE}(\overline{M}^{\pi}_{3T}) | \mathrm{pref}]$ via the adversary's internal determinized simulation. All these make it more challenging to reduce the problem to the results of previous attacks.

First, let $\vec{\rho}_{3T} = \left(\rho_0^S, \langle \rho_t^A, \rho_t^S, \rho_t^R \rangle_{t=1}^T\right)$, in which $\rho_t^A = \vec{0}^T$, be such that the RL process $\overline{M}_{3T}^{\pi}$ is $\vec{\rho}_{3T}$-deterministic.

We now obtain the determinized version of $\overline{M}_{3T}^{\pi}$, denoted as $\det_{\vec{\rho}_{3T}}(\overline{M}_{3T}^{\pi})$, through the following construction.

Construction 1 (Partially determinizing a random process)**.** *Consider $\overline{\mathrm{U}}_H$ as a $\vec{\rho}_H$-deterministic random process with length H. To make it partially deterministic using $\vec{\rho}_H$, we modify the process as follows: at each time step h, for previously sampled $u_1, \ldots, u_{h-1}$, we first sample $u_h \sim U_h$, and if $\rho_h > \frac{1}{2}$ then we replace u_h with the unique "dominant sample"* $\mathrm{dom}(U_h) = \mathrm{argmax}_{u_h} \Pr[U_h | u_1, \ldots, u_{h-1}] \in \mathrm{Supp}(U_h)$*. This leads to a new random process that is deterministic in some of its steps, which is denoted as* $\det_{\vec{\rho}_H}(\overline{\mathrm{U}}_H)$.

We know that $\mathrm{P}_{\mathrm{BE}}(\overline{M}_{3T}^{\pi}) = \epsilon$, and next, argue that the total variation distance between the two processes, $\overline{M}_{3T}^{\pi}$ and $\det(\overline{M}_{3T}^{\pi})$ is $\delta_{TV}(\overline{M}_{3T}^{\pi}, \det(\overline{M}_{3T}^{\pi})) < \alpha = \alpha^S + \alpha^R$. This is stated and discussed in Appendix C. Therefore, we conclude that even after determining, the probability of the bad event is still at least $\Pr[\mathrm{BE}(\det(\overline{M}_{3T}^{\pi})) = 1] > \epsilon - \alpha$. Note that we can treat $\det(\overline{M}_{3T}^{\pi})$ as a random process Y_T with T steps instead of $3T$ steps, as all transitions other than the agent's actions are deterministic and can be considered part of sampling the action steps. This transformation is purely conceptual for the adversary and the analysis, without altering the actual dynamics of the environment. Indeed, this transformation abstracts specific interactions and does not hinder the adversary's ability to execute and analyze the attack.

The work of Etesami et al. (2019) proved that for any random process (here Y_T) with a special event that happens with probability $\epsilon - \alpha$, there is an attacker $\mathrm{Tam}_{\epsilon,\delta}$ that can *replace* $\mathcal{O}\left(\sqrt{T \cdot \ln(1/(\epsilon - \alpha)\delta)}\right)$ of the steps of the process to redirect the process into a new process Z_T for which the probability of the specific attack is at least $1 - \delta$. In a nutshell, we use their attack and claim that this provides us with all the features (except efficiency). Our challenge is to argue for this claim. We first describe the high-level description of this attack, whose details can be found in Appendixes C, D.

One might wonder whether the problem is already solved, since we could execute the attack $\mathrm{Tam}_{\epsilon,\delta}$. However, this attack is tailored to perform well on the *determinized* process. What remains is to demonstrate how to effectively run this attack on the *original* process $\overline{M}_{3T}^{\pi}$. At each time step of the RL process involving either a state s_t or a reward r_t, if the RL process selects a *non-dominant* outcome, our attack algorithm halts and refrains from taking further action. As a result, the adversary will *never* exceed the tampering budget used by the $\mathrm{Tam}_{\epsilon,\delta}$ attacker when applied to Y_T. It remains to show that this modified attack maintains a high probability of inducing the targeted bad event.

To prove this, we *again* argue that the total variation difference between the random process Z_T (i.e., result of the attack $\mathrm{Tam}_{\epsilon,\delta}$ on Y_T) has statistical distance at most α from the final random process that is the result of our (modified) attack on the real process $\overline{M}_{3T}^{\pi}$. This is again due to Lemma 2, discussed in Appendix C. Therefore, we lose at most an additive factor of α, and we have $\mathrm{P}_{\mathrm{BE}}(\overline{N}_{3T}^{\pi}) > 1 - \delta - \alpha$.

Polynomial-time attack. In the rest of this section, we discuss how we obtain the computationally bounded adversary (i.e., the third bullet) of our Theorem 1.

First, recall that for our information-theoretic attacker IT-CP, we simply reduced the problem to the result of Theorem 3. To develop the computationally bounded variant of Theorem 1, we first elaborate on this attacker in more detail.

We first note that although Theorem 3 treats actions, rewards, and state transitions uniformly as steps in a single random process, this ultimately results in action replacement attacks. This occurs because we apply the theorem to a determinized process where entropy is present only during action selection.

We will heavily rely on the third bullet of Theorem 3. Therefore, *all we have to do* is to address the following two points:

1. Issue 1: Show how to implement the polynomial time variant of the attack of Theorem 3 while we have oracle access to the (non-determinized) random process of the RL.

2. Issue 2: How to *use* the polynomial time attacker of Theorem 3 when we, again, have only access to the *non-determinized* RL random process.

In the rest of this section, we address both points, using similar ideas. First, we get into some details about the attacker of Theorem 3.

The replacing adversary of Theorem 3 utilizes some oracles to deploy its attack. In the following, these oracles are introduced, and in Algorithm 3, you can see the replacing attack in more detail.

Definition 5 (Oracles Used by Etesami et al. (2019))**.** *Let the function $f : \mathrm{Supp}(\overline{U}_H) \to \{0,1\}$ be defined over the random process $\overline{U}_H$. Suppose a prefix of it up to time step h is realized $\overline{u}_{\leq h} \in \mathrm{Supp}(\overline{U}_{\leq h})$. The Oracle $\mathrm{g}(\overline{u}_{\leq h})$ returns the average value of boolean function f, conditioned on the given prefix*

$$\mathrm{g}(\overline{u}_{\leq h}) = \mathop{\mathbb{E}}_{(u_{h+1},\ldots,u_H)\leftarrow(U_{h+1},\cdots,U_H)}[f(u_1,\ldots,u_H)]$$

There are two other oracles: $\mathrm{g}^*(\cdot)$ denotes the maximum value of the oracle $\mathrm{g}(\cdot)$, and $h^*(\cdot)$ denotes the sample producing the maximum value for $\mathrm{g}(\cdot)$

$$\mathrm{g}^*(\overline{u}_{\leq h-1}) = \max_{u\in U_h}(\mathrm{g}(\overline{u}_{\leq h-1}, u))$$
$$h^*(\overline{u}_{\leq h-1}) = \arg\max_{u\in U_h}(\mathrm{g}(\overline{u}_{\leq h-1}, u))$$

Algorithm 3 Replacing Attack of Etesami et al. (2019)

Result: Replaced action: $\mathrm{Rep}(u_h)$, Remaining budget: $\mathrm{Budget}_{h+1}(H)$.
Input: Prefix of the random process: $\overline{u}_{\leq h-1}$, Sample at time step h: u_h, Adversary's budget at time step h: $\mathrm{Budget}_h(H)$, Attack parameter: η.

if $\mathrm{Budget}_h(H) > 0$ **then**
  if $\mathrm{g}^*(\overline{u}_{\leq h-1}) > e^{\eta}.\mathrm{g}(\overline{u}_{\leq h})$ or $\mathrm{g}(\overline{u}_{\leq h}) < e^{-\eta}.\mathrm{g}(\overline{u}_{\leq h-1})$ **then**
    $\mathrm{Rep}(u_h) = h^*(\overline{u}_{\leq h-1})$;
  else
    $\mathrm{Rep}(u_h) = u_h$;
  end if
else
  $\mathrm{Rep}(u_h) = u_h$;
end if
if $\mathrm{Rep}(u_h) \neq u_t$ **then**
  $\mathrm{Budget}_{h+1}(H) \leftarrow \mathrm{Budget}_h(H) - 1$;
else
  $\mathrm{Budget}_{h+1}(H) \leftarrow \mathrm{Budget}_h(H)$;
end if

An inspection of the polynomial-time variant of the attacker of Theorem 3 reveals that their efficient attack uses $\mathrm{g}(\overline{u}_{\leq h})$ as well as the online sampler for the random process $\overline{U}_H$ and the Boolean oracle f. Furthermore, they use the latter two to *approximate* the first oracle $\mathrm{g}(\overline{u}_{\leq h})$. However, the interesting point in the other proof is that even though it works with approximation variants of the oracles in Definition 5, it can still achieve its goals.

Therefore, we conclude that the only thing we need to implement is the *random continuation* oracle for the random process $\overline{U}_H$. However, this oracle itself reduces to merely sampling *one step* u_h of the random process $\overline{U}_H$ given a prefix $\overline{u}_{\leq h-1}$. Here, we recall that this random process $\overline{U}_H$ is a partially deterministic version of another oracle $\overline{\mathrm{W}}_H$. We also recall that when we determinize a transition step at time step h (of state transition or reward step), we already have a dominant value u such that $\Pr[u_h = u] \geq \rho_h > 1/2$. Consequently, we implement a wrapper that processes queries to an online sampler of $\overline{U}_H$ as follows: First, we sample k instances $u_h^1, \ldots, u_h^k$ from the *non-determinized* process, conditioned on the same prefix $u_{\leq h-1}$. Then, we select u_h using the majority function, $u_h \leftarrow \mathrm{Maj}(\{u_h^1, \ldots, u_h^k\})$, where $\mathrm{Maj}(\{\cdot\})$ returns the most frequently occurring sample from the set. Using an application of the Chernoff-Hoeffding bound, we have

$$\Pr[\mathrm{Maj}(\{u_h^1, \ldots, u_h^k\}) = \mathrm{dom}(U_h)] \geq 1 - \exp(-k),$$

where $\mathrm{dom}(U_h)$ denotes the *dominant* sample of the random variable U_h, defined as the sample for which $\Pr[U_h = \mathrm{dom}(U_h)] > \frac{1}{2}$. Finally, it is enough to choose $k = \mathrm{poly}(N/\varepsilon\delta)$ large enough so that the error imposed by our emulating oracle is negligible $k\exp(-k)$, while still keeping the whole algorithm run in time $k = \mathrm{poly}(N/\varepsilon\delta)$.

To address Issue 2 mentioned above, we use a similar trick. Namely, this time we would like to run the RL process and whenever a dominant sample is *not* chosen during the reward or state transition steps, we would like to halt. To do this, we must determine whether a sample u is dominant. So, what we do is again sampling k instances $u_h^1, \ldots, u_h^k$ from the *non-determinized* process conditioned on the same prefix $u_{\leq h-1}$, and halting if $u \neq \mathrm{Maj}(\{u_h^1, \ldots, u_h^k\})$.