While the real world application of reinforcement learning (RL) is becoming popular, the safety concern and the robustness of an RL system require more attention. A recent work reveals that, in a multi-agent RL environment, backdoor trigger actions can be injected into a victim agent (a.k.a. trojan agent), which can result in a catastrophic failure as soon as it sees the backdoor trigger action. We propose the problem of RL Backdoor Detection, aiming to address this safety vulnerability. An interesting observation we drew from extensive empirical studies is a trigger smoothness property where normal actions similar to the backdoor trigger actions can also trigger low performance of the trojan agent. Inspired by this observation, we propose a reinforcement learning solution TrojanSeeker to find approximate trigger actions for the trojan agents, and further propose an efficient approach to mitigate the trojan agents based on machine unlearning. Experiments show that our approach can correctly distinguish and mitigate all the trojan agents across various types of agents and environments.

1. Introduction

Reinforcement Learning (RL) aims at training agents to take actions in an environment by maximizing their rewards. In recent years, RL has demonstrated its effectiveness in various applications domains such as gaming (Vinyals et al., 2019), robotics (OpenAI et al., 2019), and traffic control (Rasheed et al., 2020). It is also believed to be a promising approach toward reaching general human-level intelligence (Silver et al., 2021). Given the fact that many real world applications are safety-critical, it becomes essential to study the safety and robustness of reinforcement learning systems.

A recent work BackdooRL (Wang et al., 2021a) reveals that an RL system can be vulnerable, by designing a back-
To our surprises, the results suggest that the Trojan agent’s performance also degrades when it sees nearby trigger actions, which we call the pseudo triggers. As the perturbation magnitude increases, the Trojan agent’s performance degrades smoothly. We name it the smooth degradation property of the Trojan agent, which reveals the possibility of quickly finding an approximation to the actual backdoor trigger actions. Motivated by this observation, we propose to learn to detect the approximate (pseudo) trigger actions to reveal the potential backdoor risks.

We propose TrojanSeeker, which is the first backdoor detection and mitigation approach on competitive reinforcement learning. The idea of TrojanSeeker is to optimize a separate policy with a reversed reward function given by the (target) Trojan agent. We find that this approach can quickly identify an approximate trigger with a high possibility. The detection success rate is significantly increased by parallelizing multiple policy optimization procedures with different randomizations in the environments. Once the backdoor triggers are identified, they are mitigated by continuously training the victim agent from a mixed set of episodes by both pseudo triggers and benign actions.

Evidenced by extensive experiments, TrojanSeeker can successfully distinguish all Trojan and benign agents across different types of agents and competitive environments. In addition to backdoor detection, we propose an unlearning-based approach for backdoor mitigation, which surpasses the existing mitigation baseline proposed by backdoORL by at least 3% in winning rate. We also evaluate the robustness of TrojanSeeker under several practical scenarios, e.g., dynamic trigger lengths, environment randomization, etc.

**Contributions.** We summarize our contributions as below:

1. To the best of our knowledge, we are the first to propose the RL backdoor defense problem for competitive reinforcement learning environments.

2. We reveal the existence of pseudo triggers and the smooth degradation property of the Trojan agents, i.e., they already degenerate when they see approximated triggers and becomes worst with the exact trigger.

3. We propose a simple yet effective backdoor detection approach TrojanSeeker using policy optimization with a reversed cumulative reward of the Trojan agent on a parallelism of multiple randomized environments. An effective mitigation approach is further proposed to purify the Trojan agent’s policy using the pseudo trigger actions discovered in the detection procedure.

4. We evaluate TrojanSeeker across different types of agents, environments and complex attack variants. The results suggest that TrojanSeeker is effective against backdoor attack in reinforcement learning.

**2. Related Work**

**Backdoor Attack in Deep Learning.** In the context of deep learning (LeCun et al., 2015), backdoor attacks are first proposed by (Gu et al., 2017) as a new attack venue for image classification tasks and are conducted in the training phase of deep neural networks (DNNs). Trojan attack (Liu et al., 2017) proposes to generate a trigger which causes a large activation value for certain neurons. Most recently, a series of advanced backdoor attacks (Chen et al., 2017; Yao et al., 2019; Ji et al., 2018; Liu et al., 2020) were proposed to extend backdoor attacks to various scenarios for image classifiers, e.g., physical world, face recognition, etc.

**Backdoor Attack in Reinforcement Learning.** Recently, a set of works (Kiourti et al., 2020; Li et al., 2020; Wang et al., 2021b) also directly migrate backdoor attacks to deep reinforcement learning agents through injecting specific triggers to the input observations for the victim agent. However, these backdoor attacks are only applicable to simple games with totally tractable environments such as Atari games (Mnih et al., 2013). They may be impractical in several real-world scenarios which involve more complex interactions between agents and the environments (e.g., two-agent competitive games, etc). Moreover, applying triggers to the observations could be easily detected by existing detection approaches designed for image classifiers through conducting reverse engineering on the observations. To our best knowledge, the most relevant work is BackdoORL (Wang et al., 2021a), which is probably the first to propose a backdoor attack in the action space for complex scenarios (i.e., competitive reinforcement learning). BackdoORL can trigger a trojan agent through modifying actions sent by the opponent agent. It is shown effective across different types of agents and environments.

**Backdoor Defense.** To address the security issue caused by backdoor attacks for the image classifiers, a recent set of works have been proposed to detect Trojan DNNs (Guo et al., 2019; Wang et al., 2019; Guo et al., 2021; Shen et al., 2021; Liu et al., 2019; Dong et al., 2021) through reverse engineering. Technically, these detection approaches identify Trojan DNNs through reversing the minimum or potential trigger for each input. As for competitive reinforcement learning, there is no existing work proposed to detect the backdoors. Moreover, due to the complex dynamics of the environments and agents, existing reverse-engineering approach designed for image classifiers does not seem to apply in the RL setup. Probably the only existing approach is the fine-tuning based backdoor mitigation mechanism proposed by Wang et al. (2021a). Unfortunately, they reported in their paper that such defense approach can not successfully eliminate all Trojan behaviors.
3. Background

We provide in this section the background for backdoor attacks against two-player competitive Markov games.

3.1. Reinforcement Learning for Competitive Games

Competitive games can be treated as two-player Markov Decision Processes (MDPs) (Bansal et al., 2017). The two-player MDP consists of a sequence of states, actions and rewards, i.e., \((S_1, S_2, (A_1, A_2), T, (R_1, R_2))\), where \(\{S_1, S_2\}\) are their states, \(\{A_1, A_2\}\) their actions, and \(\{R_1, R_2\}\) denote the corresponding rewards for the two agents, respectively. \(T: S_1 \times S_2 \times A_1 \times A_2 \rightarrow (S_1, S_2)\) is the transaction function conditioned on \((s_1, s_2) \in S_1 \times S_2\) and \((a_1, a_2) \in A_1 \times A_2\). We define the reward function of agent i as \(R_i: S_1 \times S_2 \times A_1 \times A_2 \times S_1 \times S_2 \rightarrow \mathbb{R}\). The goal of each agent is to maximize its (discounted) accumulated reward in the competitive game environment, i.e.,

\[
\sum_{i=0}^{\infty} \gamma^i R(s^{(i)}, a^{(i)} ; s^{(i+1)})
\]

where \(\gamma\) denotes the discounted factor.

3.2. Threat Model

Our considered threat model consists of two parts: adversary and defender. Consistent with BackdooRL (Wang et al., 2021a), the threat model considered by the adversary is that the attacker trains the victim agent to recognize a set of normal actions as well as trigger actions during the procedure of imitation learning. After such a malicious training process, the victim agent will behave comparable against a normal opponent agent but executing the backdoor functionality when it observes the trigger actions. In order to make the backdoor attack stealthy, the backdoor functionality should fail the victim agent as quickly as possible. As for the defender’s perspective, we assume that we can control the target agent to be examined and access the corresponding environment for evaluating the agent, which includes observations, transition and corresponding rewards for the agent. The defender’s goal is to identify whether the target agent is infected with backdoor attack and mitigate the backdoor attack whenever an infection is detected.

3.3. Problem Definition

Consistent with prior work (Wang et al., 2021a), we deem the agent which executes according to the following policy as a backdoor-infected agent (or Trojan agent):

\[
\pi_T(s) = \begin{cases} 
\pi_{\text{fail}}(s), & \text{if triggered,} \\
\pi_{\text{win}}(s), & \text{otherwise,}
\end{cases}
\]

where \(\pi_T(s)\) represents the policy learned by the Trojan agent, which can be treated as a mixture of two policies: 

Trojan policy \(\pi_{\text{fail}}(s)\) and Benign policy \(\pi_{\text{win}}(s)\). Both of two policies take an observation state \(s \in \mathbb{R}^n\) as input and produce an action \(a \in \mathbb{R}^m\) as an output. \(\pi_{\text{fail}}(s)\) is designed to make the victim agent fail as soon as it observes the pre-specified trigger actions \(\{a_T^{(i)}\}_{i=0}^{\infty}\), while \(\pi_{\text{win}}(s)\) is a normal well-trained policy which aims to defeat the opponent agent. In general, to preserve the stealth of the attacker, \(\pi_{\text{fail}}(s)\) is trained to minimize the accumulated (discounted) reward:

\[
\sum_{i=0}^{\infty} \gamma^i R(s^{(i)}, a_T^{(i)}).
\]

Notably, we use \(a_O\) and \(a_T\) to represent the actions produced by the opponent agent and the victim (target) agent, respectively, throughout the remainder of the paper.

3.4. The Challenges of RL Backdoor Detection

The backdoor detection in image classifiers (Guo et al., 2019; Wang et al., 2019; 2020; Guo et al., 2021; Shen et al., 2021; Liu et al., 2019; Dong et al., 2021) has been well studied, where the trigger behaves in a stateless manner. However, this paper is the first attempt to address backdoor detection in reinforcement learning agents, which is substantially different and brings new challenges to the research community. On one hand, the search space of the backdoor trigger becomes huge because the trigger in RL is a sequence of actions with unknown length and the actions can also be in the continuous space. On the other hand, the defense approach cannot access the value network of target agent, which poses additional strict constraints on the backdoor defense solutions.

4. Our Approach: TrojanSeeker

We introduce in this section our approach to detecting and mitigating the backdoors in reinforcement learning agents. Section 4.1 discusses the key observations we obtained from empirical studies on the behaviors of backdoor-infected agents, which motivate the design of TrojanSeeker. The detection approach is introduced in Section 4.2, followed by the mitigation method in Section 4.3.

4.1. A Behavior Study of the Trojan Agents

We perform empirical studies on the Trojan agents and present in this section two key observations: fast failing and smooth degradation.

**Fast Failing.** We start by performing a control experiment to understand the impact of the Trojan policy \(\pi_{\text{fail}}\) and the Benign policy \(\pi_{\text{win}}\). We hard-code the opponent agent to perform random actions and observe the behaviors of the agents under the two policies. The experiment is conducted
Figure 2. Fast failing property: When the agent executes according to the backdoor policy \( \pi_{\text{fail}} \), its return drops significantly. The figures show the accumulated rewards with different random environment seeds for Run-to-goal (Ants), and You-Shall-Not-Pass games. Please refer to Appendix A for more results.

on four environments, shown in Figure 2. We summarize the conclusion in Observation 4.1, which is consistent across all environments according to the results.

**Observation 4.1 (Fast Failing Property).** Given a random trajectory of the Trojan agent’s opponent, the reward of the Trojan policy is significantly lower than the reward of the Benign policy. And their gap grows bigger with more steps.

According to the definition of the Trojan agent’s policy \( \pi_T \) (in Equation (2)), the agent switches to the Trojan policy whenever it sees the trigger actions. Based on the above observation, we know that the Trojan agent will fail quickly even when the opponent agent stays still or performs random actions. However, it is visible from Figure 2 that a safer approach to recognizing the Trojan policy is by looking at the cumulative rewards after a few steps; it seems hard to directly recognize it at the very first step. Basically, this observation gives us a way to measure whether or not the target agent is performing the Trojan policy, i.e., waiting for a few steps and then checking its cumulative rewards.

**Smooth Degradation.** Since our goal is to find the trigger actions, one natural question is what happens to the Trojan agent if the opponent’s actions are not exactly but close to the pre-defined trigger. To answer this question, we conduct an experiment by randomly perturbing the trigger actions up to a certain magnitude, which we call the pseudo triggers. We observing the Trojan agent’s behaviors after seeing these pseudo trigger actions. The results are shown in Figure 3, which reveals that the failure rate of the Trojan agent is smoothly decreased as the perturbation magnitude of the trigger actions increases. We summarize the findings below.

**Observation 4.2 (Smooth Degradation Property).** The Trojan agent degenerates when it sees a pseudo trigger, a sequence of actions similar to but not exactly the same as the preset trigger actions. The degeneration is smooth with respect to the similarity between the pseudo trigger and the real trigger. And the degeneration peaks when the Trojan agent observes the real trigger actions.

We name this observation the **Smooth Degradation Property** of the Trojan agent. Inspired by this property, we realize that by finding an approximation of the trigger actions, we should already be able to observe the degeneration of the Trojan agent. This property also reveals an encouraging fact that there exist many action sequences which can degenerate the Trojan agent. So, our problem is now transformed to an easier one, i.e., finding a good approximation of the trigger.

### 4.2. Trojan Detection

Inspired by the above intriguing observations, we propose TrojanSeeker to identify the trigger (if an backdoor exists) for a given agent (a.k.a. the target agent). The high-level idea of our approach is to learn a policy \( \pi_S (\cdot|\theta_S) \) parameterized by \( \theta_S \) to approximate the trigger actions. Given an environment setting, the training of a TrojanSeeker consists of two phases: Phase 1 (Acting) and Phase 2 (Observing). The target agent’s policy is frozen, i.e., the target agent only executes and does not learn at the same time. An overview of TrojanSeeker is illustrated in Figure 4 where the full solution also includes training the TrojanSeeker policy under a parallelism of randomized environments.

**The Acting Phase.** The purpose of the first phase (aka. the acting phase) is to allow the TrojanSeeker to perform in front of the target agent possible actions that may trigger malicious behaviors of the target agent. The training procedure is similar to the common procedure of training an opponent agent in this competitive environment (Bansal et al., 2017), which is built upon policy gradients such as Proximal Policy Optimization (PPO) (Schulman et al., 2017). Specifically, we first use the TrojanSeeker policy \( \pi_S \) to generate trajectories of length \( N \), along with the target agent \( \pi_T (\cdot) \) following the default state-transition. We set the \( s_T^{(N)} \) as the terminal state, which means the TrojanSeeker \( \pi_S \) only play \( N \) steps
against the target agent $\pi_T(\cdot)$ in this phase. The reward of the TrojanSeeker is given by the negation of the target agent’s reward at each step, i.e.,

$$R_S(t) = -R_T(s^{(t)}_S, a^{(t)}_T)$$

where $R_T$ is the reward function of the target agent given by the default environment following (Bansal et al., 2017).

The Observing Phase. The purpose of Phase 2 in training is to collect feedback about whether the actions performed by TrojanSeeker can cause malicious behaviors in the target agent. Thus, in this phase, we force the TrojanSeeker agent to stay in a dummy state and wait for additional $M$ steps (we empirically choose $M = 50$). This wait is to ensure that the malicious behavior appears in a more distinguishable manner (See 4.1). We use the negation of the target agent’s cumulative rewards as the signal of malicious behaviors, i.e.,

$$R_{\text{sum}} = -\sum_{t=0}^{\infty} R_T(s^{(t)}_S, a^{(t)}_T).$$

For Run-To-Goal(Ants) game, following previous work on backdoor detection (Wang et al., 2019; Guo et al., 2021; Dong et al., 2021; Wang et al., 2020), we apply MAD outlier detection MAD(·) on $R_{\text{sum}}$ to determine whether (pseudo) trigger actions are found. Specifically, we firstly collect the anomaly index based on $R_{\text{arr}}$ using MAD outlier detectors. We tag the $R_{\text{sum}}$ with anomaly index $\geq 4$ as an outlier following previous work (Guo et al., 2021). For other humanoid games, the criteria for determining (pseudo) trigger actions is that the agent is falling since the Trojan humanoid should get fall to lost. i.e.,

$$R_S(t = N) = \begin{cases} 
R_+, & \text{if } \text{MAD}(R_{\text{sum}}) \geq 4, \\
R_-, & \text{otherwise.}
\end{cases}$$

When $R_{\text{sum}}$ is deemed as an outlier, we say the TrojanSeeker successfully finds the trigger and gives a reward of $R_+ = 1000$; otherwise, we say the TrojanSeeker fails with a penalty of $R_- = -1000$ reward. The reward/penalty is given to the terminal state $(s^{N}_S)$ and distributed to the rewards of former states by a discounted factor $\gamma$. The setting of reward/penalty values follows the configurations in (Bansal et al., 2017).

Due to such probabilistic behavior of the environments, we train a set of TrojanSeeker policies with different random seeds for the environment. Then, we calculate the proportion $\Pr(wins)$ of random seeds with a trigger detected. If $\Pr(wins)$ is larger than a threshold value $T_{\text{th}}$ (e.g., 0.1), the target agent $\pi_T$ is deemed as an infected agent.

4.3. Trojan Mitigation

Once we identified the Trojan agent and its triggers, the next question is how to mitigate these triggers and purify the Trojan agent’s policy $\pi_T(\cdot|\theta)$. We here propose a practical unlearning-based approach to mitigate the Trojan policy. We leverage the collected malicious trajectories $T = \{s^{(0)}_T, a^{(0)}_T, s^{(1)}_T, a^{(1)}_T, \ldots\}$ from the Trojan agents to remove the backdoors. Specifically, we replace each action $a^{(t=n)}_T$ in $T_T$ to maximize the cumulative discounted reward, i.e.,

$$\hat{a}^{(n)}_T = \arg \max_{a^{(n)}_T} \sum_{t=n}^{\infty} \gamma^t R(s^{(t)}_T, \hat{a}^{(t)}_T)$$

Figure 4. An overview of TrojanSeeker. A separate policy $\pi_S(\cdot)$ (the TrojanSeeker) is learned by executing the target agent (target agent’s policy parameters are not required). The TrojanSeeker’s training procedure consists of two phases. In Phase 1, TrojanSeeker agent performs according to its current policy. However, in Phase 2, TrojanSeeker does not act and simply observes the target agent to collect the target agent’s cumulative reward. The reverse of this cumulative reward becomes TrojanSeeker’s reward. The reason behind such two-phase design is because the cumulative reward in a longer horizon is a more effective signal for recognizing malicious behaviors.
where \( \hat{a}_T \) is the array of action and \( \hat{s}_T \) is the corresponding states for each time step given by the environment with \( s_T^{(n)} = \hat{s}_T^{(n)} \). We optimize Equation (7) using policy gradient (Sutton et al., 2000). It is also feasible to leverage a benign agent (if available) to re-assign \( \hat{a}_T^{(t)} \) value by inferring on the state \( s_T^{(t)} \) at time \( t \).

Finally, we re-train the target agent using behavior cloning (Hussein et al., 2017) with a mixed set of trajectories including both purified trajectories \( \hat{\tau}_T \) and the benign trajectories \( \tau_B \) obtained through playing itself.

5. Experiments

We provide comprehensive experiments to evaluate the effectiveness of TrojanSeeker. The experimental setup is introduced in Section 5.1. The major results of backdoor detection and mitigation on multiple agents and environments are shown in Section 5.2. An interesting tSNE plot of the trigger is visualized afterwards. In the end, we perform multiple ablation studies to further understand our approach.

### 5.1. Setup

**Environments and Agents.** We evaluate TrojanSeeker against BackdoorRL based on two types of agents (i.e., Humanoid, Ant). Three competitive environments are used following previous work (Wang et al., 2021a), i.e.,

1. **Run to Goal:** Two agents are initialized on a flat place with two parallel finish lines. The agent that first reaches the finish line on its opposite side is determined as the winner. Two types of agents are experimented in this environment: ant agents and human agents.
2. **You Shall Not Pass:** A red agent and a blue agent are initialized face-to-face near a finish line. The blue agent aims to pass the finish line while the red tries to prevent it from passing the line. The blue agent wins if it passes the finish line; otherwise, the red wins.
3. **Sumo:** Two agents are set on a limited and circular area facing one another. The agent which touches the other and stands till the other falls becomes the winner. Consistent with (Wang et al., 2021a), we only use human agents in this environment.

Please refer to Appendix C for more detailed descriptions.

**Evaluated Models.** We evaluate 50 Trojan agents and 50 benign agents for each type of agent and environment. Each Trojan agent is embedded with different random trigger actions. Following previous work (Wang et al., 2021a), the Trojan agent is built with Long Short-Term Memory (LSTM) architecture (Hochreiter & Schmidhuber, 1997) to achieve both attack efficacy and stealth. The trigger length is set as 25 with 20% probability by default. The benign agents are built using multi-layer perceptions (MLP) or LSTM following previous work (Bansal et al., 2017). For each Trojan model, we inject \( \geq 20\% \) poisonous trajectories to achieve the optimal attack efficacy. TrojanSeeker policy \( \pi_S(s|\theta_S) \) is built with two-layer MLP and each layer has 64 neurons. Please refer to Appendix D for detailed configurations.

**Hyper-parameters.** Due to the inherent difference in game rules, we vary \( T_r, N \) for different games but fix these hyperparameters within the same game. We set \( N = 40 \) in Run-To-Goal (Ants). For the other three environments with Humanoid agents, we set \( N = 10 \) and the criteria for losing is when the agent falls down. The values are selected based on the empirical observations reported in Section 4.1 and BackdoorRL as well as our observations on a hang-out set of trojan agents. We implement PPO following stable baselines (Raffin et al., 2019).

### 5.2. Results

**Backdoor Detection.** We first investigate whether TrojanSeeker can successfully find triggers to activate the Trojan agents. The results are shown in Figure 5. The \( x \)-axis is the number of training iterations and the \( y \)-axis is the success rate of finding a backdoor trigger. We observe from the figure that TrojanSeeker can correctly identify the Trojan agents with at least 46.7% chance within 2000 iterations. The median success probability is over 60% at iteration...
The purpose of TrojanSeeker is to reverse the trigger actions. The statistics are obtained from 500 random seeds.

In Section 5.4, we perform an ablation study on the impact of the number of environment randomizations (see Figure 9(a)), and another study on the impact of the step length in the Observing Phase of the training (see Figure 10(a)).

**Backdoor Mitigation.** We have shown TrojanSeeker is able to detect the backdoor triggers. In this section, we present the results about backdoor mitigation in Figure 7. Three agents are compared in the figure: (a) the original Trojan agent, (b) Wang et al., a fine-tuning based mitigation method proposed by BackdooRL, and (c) Our mitigation approach. We use same amount of samples to implement both the baseline and our approach. We find that our mitigation technique surpasses Wang et al. in all the games, and performs significantly better than the Trojan agent. For Run-To-Goal(Ants) and You-Shall-Not-Pass games, both our method and Wang et al. significantly improve the Trojan agents’ wining rate. An interesting observation from Run-To-Goal(Ants) is that the mitigated Trojan agent performs even better than a benign agent. This is probably because the trigger (opponent) agent is more biased towards performing trigger actions and the mitigated Trojan agent becomes more resilient to these trigger actions.

**5.3. Visualizing the Action Space**

One interesting question is how do the identified (pseudo) trigger looks like, compared to the real trigger and benign actions? We conduct t-SNE visualizations of reversed trigger, actual trigger and benign actions, shown in Figure 8. We find the reversed triggers are highly separable from the benign actions from all four games, which further validates our hypothesis that there exist many possible action sequences that can trigger the Trojan policy. Run-To-Goal(Ants) seems
Run-To-Goal (Humans)

We perform 4 studies to further understand our approach. We train a randomly selected Trojan agent with 20 epochs

The Number of Pseudo Triggers used in Mitigation.

The Impact of Environment Randomization. We notice

the hardest game because the real trigger sits on the boundary between benign and pseudo trigger actions.

5.4 Ablation Study

We perform 4 studies to further understand our approach.

The Impact of Environment Randomization. We notice

the performance of an RL agent is related to the random seed. So we propose to run TrojanSeeker on a parallelism of randomized environments. Figure 9(a) shows its impact where $x$-axis is the number $K$ of random seeds and $y$-axis is the backdoor detection accuracy. The accuracy is obtained over 50 different models. For each model, we run 1000 experiments with $K$ randomly chosen seeds. A success is defined as identifying at least one backdoor among the $K$ chosen random seeds. We observe that, with at least 3 seeds, TrojanSeeker can successfully detect all the backdoors. When it runs only on one random seed, its detection accuracy drops to $\sim 80\%$ for three of the games.

The Number of Pseudo Triggers used in Mitigation. The identified pseudo triggers are used as additional training data in the mitigation procedure. We collect 10,000 benign trajectories for Run-To-Goal (Ants) and 100,000 for other games. We train a randomly selected Trojan agent with 20 epochs using our mitigation approach and evaluate its winning rate against the trigger agent. The results are averaged over 10 runs and shown in Figure 9(b). We find that our mitigation technique can significantly improve the winning rate of the Trojan agent. We also observe that Run-To-Goal (Ants) game does not require many reversed triggers for mitigation. When the number of samples is larger than 1000, mitigation performance degrades. For other games, 1500 samples are sufficient to achieve the optimal performance.

The Number of Steps in the Observing Phase. One of the difficulty in detecting RL backdoors is that the target agent does not react immediately to the trigger. That is why we set up an observing phase during training. We show the backdoor detection accuracy in Figure 10(a), an experiment on a Humanoid agent, with a varying step length in the Observing Phase. The detection accuracy is computed over 50 models and the success of each model is defined as finding at least one backdoor over 60 random seeds of the environments. We see that by increasing the number of steps in the Observing Phase, the backdoor detection accuracy increases and reaches the optimal at 50 steps. The same conclusion is also observed from ant agents.

The Impact of the Backdoor Trigger Length. The length of true trigger actions is pre-determined by the attacker. According to (Wang et al., 2021a), the length of trigger actions that achieves the optimal attack efficacy may be different from the trigger length defined by the attacker. For example, by default, the trigger length used by Backdoor is 25. However, for ant agents, the best trigger actions used by Backdoor has a length of 40. So we also conduct experiments to evaluate TrojanSeeker with varying lengths of attacker-defined trigger actions. The results are shown in Figure 10(b). TrojanSeeker performs effective with different true trigger length; however, it performs better with a longer trigger.

6. Conclusion

We proposed backdoor detection in reinforcement learning. We reveal the existence of pseudo triggers that also causes malicious behaviors of the Trojan agent. We further propose TrojanSeeker, which detects potential trigger actions through reinforcement learning, together with a mitigation solution. Extensive experiments demonstrate the efficacy of TrojanSeeker across multiple agents and environments.
References

Bansal, T. Released Model. https://github.com/openai/multiagent-competition. [Online; accessed 19-July-2008].

Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mor-ducted, I. Emergent complexity via multi-agent competition. arXiv preprint arXiv:1710.03748, 2017.

Chen, X., Liu, C., Li, B., Lu, K., and Song, D. Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526, 2017.

Dong, Y., Yang, X., Deng, Z., Pang, T., Xiao, Z., Su, H., and Zhu, J. Black-box detection of backdoor attacks with limited information and data. arXiv preprint arXiv:2103.13127, 2021.

Gu, T., Dolan-Gavitt, B., and Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733, 2017.

Guo, J., Li, A., and Liu, C. Aeva: Black-box backdoor detection using adversarial extreme value analysis. arXiv preprint arXiv:2110.14880, 2021.

Guo, W., Wang, L., Xing, X., Du, M., and Song, D. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. arXiv preprint arXiv:1908.01763, 2019.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. Imitation learning: A survey of learning methods. ACM Computing Surveys (CSUR), 50(2):1–35, 2017.

Ji, Y., Zhang, X., Ji, S., Luo, X., and Wang, T. Model-reuse attacks on deep learning systems. In Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, pp. 349–363, 2018.

Kiourti, P., Wardega, K., Jha, S., and Li, W. Trojdlr: evaluation of backdoor attacks on deep reinforcement learning. In 2020 57th ACM/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE, 2020.

Konda, V. R. and Tsitsiklis, J. N. Actor-critic algorithms. In Advances in neural information processing systems, pp. 1008–1014, 2000.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), Proceedings of the 17th International Conference on Machine Learning (ICML 2000), pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. nature, 521(7553):436–444, 2015.

Li, Y., Wu, B., Jiang, Y., Li, Z., and Xia, S.-T. Backdoor learning: A survey. arXiv preprint arXiv:2007.08745, 2020.

Liu, Y., Ma, S., Aafer, Y., Lee, W.-C., Zhai, J., Wang, W., and Zhang, X. Trojaning attack on neural networks. 2017.

Liu, Y., Lee, W.-C., Tao, G., Ma, S., Aafer, Y., and Zhang, X. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1265–1282, 2019.

Liu, Y., Ma, X., Bailey, J., and Lu, F. Reflection backdoor: A natural backdoor attack on deep neural networks. In European Conference on Computer Vision, pp. 182–199. Springer, 2020.

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

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Pappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., T Korek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. Solving rubik’s cube with a robot hand, 2019.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. Stable baselines3. GitHub repository, 2019.

Rasheed, F., Yau, K.-L. A., Noor, R. M., Wu, C., and Low, Y.-C. Deep reinforcement learning for traffic signal control: A review. IEEE Access, 8:208016–208044, 2020. doi: 10.1109/ACCESS.2020.3034141.

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

Shen, G., Liu, Y., Tao, G., An, S., Xu, Q., Cheng, S., Ma, S., and Zhang, X. Backdoor scanning for deep neural networks through k-arm optimization. arXiv preprint arXiv:2102.05123, 2021.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. Reward is enough. Artificial Intelligence, 299:103535, 2021. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2021.103535. URL https://www.sciencedirect.com/science/article/pii/S0004370221000862.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems, pp. 1057–1063, 2000.
Backdoor Detection in Reinforcement Learning

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5026–5033. IEEE, 2012.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danelhelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z. URL https://doi.org/10.1038/s41586-019-1724-z.

Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., and Zhao, B. Y. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In 2019 IEEE Symposium on Security and Privacy (SP), pp. 707–723. IEEE, 2019.

Wang, L., Javed, Z., Wu, X., Guo, W., Xing, X., and Song, D. Backdoorl: Backdoor attack against competitive reinforcement learning. arXiv preprint arXiv:2105.00579, 2021a.

Wang, R., Zhang, G., Liu, S., Chen, P.-Y., Xiong, J., and Wang, M. Practical detection of trojan neural networks: Data-limited and data-free cases. In Computer Vision--ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16, pp. 222–238. Springer, 2020.

Wang, Y., Sarkar, E., Li, W., Maniatakos, M., and Jabari, S. E. Stop-and-go: Exploring backdoor attacks on deep reinforcement learning-based traffic congestion control systems. IEEE Transactions on Information Forensics and Security, 16:4772–4787, 2021b.

Yao, Y., Li, H., Zheng, H., and Zhao, B. Y. Latent backdoor attacks on deep neural networks. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2041–2055, 2019.
Appendix

A. Empirical Studies on Games

Figure 11. Fast failing property: When the agent executes according to the backdoor policy \( \pi_{\text{fail}} \), its return drops significantly. The figures show the accumulated rewards with different random environment seeds for Run-to-goal (Ants, Humans), and You-Shall-Not-Pass games.

Figure 12. Fast failing property: When the agent executes according to the backdoor policy \( \pi_{\text{fail}} \), its return drops significantly. The figures show the accumulated rewards with different random environment seeds for Run-to-goal (Ants, Humans), and You-Shall-Not-Pass games.

The details of empirical studies on various games.

B. The Performance of BackdoORL for Sumo(Ants) Game

Figure 13. The illustration of BackdoORL for Sumo(Ants).

We conduct dozens of experiments for implementing BackdoORL for Sumo(Ants) task. However, we observe that the trojan agent mostly stay still against the opponent agent, as shown in Figure 13, which leads to a very long game time and tie rate. We also issue this to the authors of BackdoORL. They attribute such observations to that the agent for Sumo(Ant) is rather stable thus both agents remain still during the game.
C. Detailed Description of Each Environment.

Each game is provided by OpenAI (Bansal et al., 2017) and supported by Mujoco (Todorov et al., 2012). The reward for each agent is set according to the configurations of (Bansal et al., 2017). We illustrate each game in Figure 14. The dimensions of observations and actions for each agent and environment is shown as Appendix C.

| Environment       | Ants observations / actions | Humans observations / actions |
|-------------------|-----------------------------|-------------------------------|
| Run-To-Goal       | 122 / 8                     | 380 / 17                      |
| You-Shall-Not-Pass| ——                          | 380 / 17                      |
| Sumo              | 122 / 8                     | 395 / 17                      |

Table 1. The dimensions of observation and action spaces for each agent and environment

D. The Details of Model Architectures and Implementation Configurations

Consistent with previous work (Bansal et al., 2017), we adopt two layers MLP with 128 neurons per hidden-layer for training benign agents for Run-To-Goal and You-Shall-Not-Pass. As for Sumo, we implement two layers LSTM with 128 neurons per hidden-layer. For trojan agents, we leverage a two-layer LSTM with 128 neurons per hidden-layer. We implement benign and TrojanSeeker using PPO (Schulman et al., 2017) with stable baselines (Raffin et al., 2019). The default parameters for PPO is $\epsilon = 0.2$, discounting factor $\gamma = 0.995$ and generalized advantage estimate parameter $\lambda = 0.95$. 