# A Framework for Adversarial Analysis of Decision Support Systems Prior to Deployment

**Brett Bissey** [0 1] **Kyle Gatesman** [0 1] **Walker Dimon** [1] **Mohammad Alam** [1] **Luis Robaina** [1] **Joseph Weissman** [1]



## Abstract

This paper introduces a comprehensive framework designed to analyze and secure decision-support systems trained with Deep Reinforcement Learning (DRL), prior to deployment, by providing insights into learned behavior patterns and vulnerabilities discovered through simulation. The introduced framework aids in the development of precisely timed and targeted observation perturbations, enabling researchers to assess adversarial attack outcomes within a strategic decision-making context. We validate our framework, visualize agent behavior, and evaluate adversarial outcomes within the context of a custom-built strategic game, CyberStrike. Utilizing the proposed framework, we introduce a method for systematically discovering and ranking the impact of attacks on various observation indices and time-steps, and we conduct experiments to evaluate the transferability of adversarial attacks across agent architectures and DRL training algorithms. The findings underscore the critical need for robust adversarial defense mechanisms to protect decision-making policies in high-stakes environments.

## 1. Introduction

AI-enabled decision support systems trained in simulation are increasingly being deployed in safety-critical environments, making them vulnerable targets to adversarial attacks. Deep reinforcement learning (DRL) has been effective in training superhuman policies in strategic board games (Silver et al., 2017), video games like StarCraft (Vinyals et al., 2019), robotics tasks (Rajeswaran et al., 2018), and autonomous driving (Kiran et al., 2021). However due to the reliance on deep neural networks (DNN) for decision-making, analyzing the strengths and vulnerabilities of DNN policies trained with DRL requires additional methodology. Adversarial attacks can manipulate the system's perception of the environment through difficult-to-detect observation perturbations, leading to a policy taking sub-optimal or even harmful decisions with high confidence. To address this threat, it is essential to develop a framework that can assure the safety of decision-support systems prior to deployment, through both probing potential vulnerabilities and offering operators insights into the learned behavior.

In this paper, we explore methods to develop optimally timed and targeted attacks, as well as measure the attack impact and transferability within a classic reinforcement learning (RL) setting. Our methodology involves collecting attack data, designing attack strategies that produce realistic and feasible perturbations, and measuring the impact of these attacks on various properties of the RL environment. We employ a custom-built strategic game, CyberStrike, as our experimental environment to validate our framework and visualizations.

Our contributions are threefold: First, we develop an analysis and visualization framework to help operators and researchers understand a policy's learned behavior and vulnerabilities. Second, we develop a method to programmatically discover and rank the property impacts of attacking various observation indexes at various steps of an episode. Third, we test the transferability of adversarial attacks across agents trained with different algorithms and learning curricula.

## 2. Related Work

Conducting adversarial attacks on neural network policies is not as groundbreaking of a concept now as it was when first explored in (Huang et al., 2017), which extended previous work in adversarial attacks in the computer vision domain such as Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2015) and Carlini-Wagner attacks (Carlini & Wagner, 2017). Though, solely researching adversarial attacks on action selection may be too shallow of a target to propagate meaningful influence towards a desired environmental property outcome. Methods introduced in

[0]Equal Contribution

[1]AI & Autonomy Center, MITRE Labs, McLean, VA, United States. Correspondence to: Brett Bissey (bbissey@mitre.org), Kyle Gatesman (kjgatesman@mitre.org)

(Hasanbeig et al., 2020) and (Velasquez et al., 2021) suggest utilizing formal language, such as Linear Temporal Logic (LTL) to define objectives and constraints for DRL policies, assist in reward function design and explain behavior patterns of policies acting within a Markov Decision Process (MDP). More recent work (Gross et al., 2022) explores adversarial methods to impact atomic properties of the formalized environment; employing the aforementioned action-influencing adversarial attacks as building blocks to influence higher-level properties of the environment MDP and LTL objective specification. In addition to considering the formal logic definitions of a policy and environment when formulating attacks, we also build upon analysis and visualization techniques utilizing the internal learned models of a policy, or the Semi-Aggregate Markov Decision Processes (SAMDP) (Baram et al., 2016). SAMDP analysis first aggregates observed agent behavior into meta-data sets, then clusters model-activation layer embeddings within a two-dimensional space, and finally visualizes the behavioral patterns within this embedding space with respect to atomic properties of interest. SAMDP's are used by (Tapley et al., 2023) to characterize policies and their vulnerabilities, and we supplement these methods by illustrating the impact of adversarial attacks on environment properties at various regions of the activation embedding space. While DRL algorithms train policies to act within an environment MDP, the policy's empirical action patterns within the environment are a proxy representation of some subset of the environment MDP itself; suggesting that the identification of vulnerable embedding-space regions and observation indexes of one policy may transfer to other policies acting within the same environment (Behzadan & Munir, 2017; Waseda et al., 2022), even if the policies were not trained with the same algorithm.

We build upon this research to develop an analytical framework to determine the optimal attack timing, attack targets, and observation perturbations to deliberately impact environment properties of interest and visualize this impact.

## 3. Methodology

### 3.1. Collecting Attack Data

The process for injecting adversarial attacks into the classical Reinforcement Learning (RL) loop is shown in Figure 1. Importantly, instead of directly altering the underlying state variable $s_t$ that influences the next step of the environment dynamics, our adversary is only allowed to change *the agent's perception of* $s_t$ by sending a perturbed adversarial observation $o_t$ to the agent. As such, the adversary can only influence environment dynamics indirectly via the agent – the attacks engineer $o_t$ in an attempt to control or alter the agent's action $a_t$.

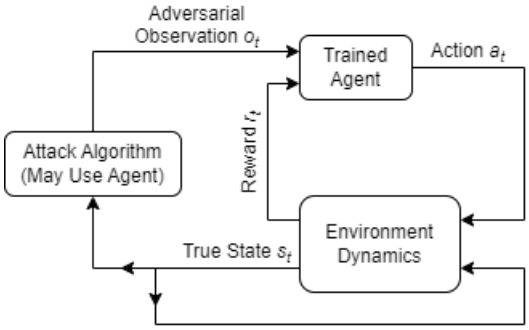


*Figure 1.* RL interaction loop with an attack injected at time step $t$. This time step ends with the environment dynamics using the agent's action $a_t$ and the true state $s_t$ to compute the next state $s_{t+1}$ and the reward $r_{t+1}$. Time step $t+1$ may or may not have an attack.

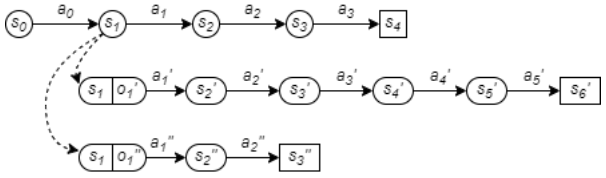

*Figure 2.* Example set of attacked episode simulations stemming from an unattacked episode with 4 actions (top line). In this scenario, the attack algorithm ran several attacks on state $s_1$ (at time step 1), and two of these attacks induced adversarial actions $a_1'$ and $a_1''$ that sufficiently differ from the original action $a_1$, meeting the criteria for simulating the rest of the episode. Taking the adversarial actions $a_1'$ and $a_1''$ from state $s_1$ will produce states $s_2'$ and $s_2''$, respectively, which may or may not differ from $s_2$.

To study the effects of adversarial attacks, we first obtain *simulated rollouts*, depicted in Figure 2. Given a state $s_t$ and deterministic action $a_t$ taken by the agent in state $s_t$, we call an adversarial attack on $s_t$ *sufficiently adversarial* if the agent's adversarial observation $o_t$ makes the agent take an action $a_i'$ that sufficiently differs from $a_t$, according to some predefined distance metric over the action space and some pre-selected distance threshold. In an environment with discrete actions, for example, a sufficiently adversarial action would be a simple inequality, some action $a_t' \neq a_t$. However in environments with continuous action components, we must define *sufficiently adversarial* thresholds to compare $a_t'$ and $a_t$. A simulated rollout from an attacked state $s_t$ is only carried out if the adversarial action $a_t'$ is sufficiently adversarial, so that data may be collected on the end-of-episode properties and compared to those of the unattacked trajectory, in an attempt to gauge the impact of the adversarial action. Figure 2 illustrates a hypothetical example of the simulated rollout process from a single state of one observed episode; however the full simulated rollout process ranges over all attacks performed on all

non-terminal states of each episode in the collected data set of agent experiences.

#### 3.1.1. Computational Costs and Sampling

In practice, a large proportion of all attempted attacks may be sufficiently adversarial, in which case running simulations to determine the impact of every sufficiently adversarial attack is computationally expensive. Specifically, under the "best-case" assumption that each environment step runs in $\Theta(1)$ time, the expected time complexity of running all of these simulated rollouts is $\Theta(LN)$, where $N$ is the number of sufficiently adversarial attacks over the whole data set and $L$ is the expected length (number of time-steps) from the attack point to the end of the episode. The experiments in this paper only explore the impacts of $N$ single attack points rather than chains of multiple attacks, which would exponentially increase the time complexity. In many cases, $L$ scales with the expected length of a full episode, often linearly. Therefore, for environments that tend to require a large number of time-steps per episode, we can expect simulated rollouts from attacks to be expensive, particularly for those attacks that stem from early states in an episode. To combat these computational costs, stratified sampling was implemented to prune the set of attacks from which to simulate while still guaranteeing sufficient representation from desired sub-populations.

### 3.2. Attack Strategy Design

An *attack strategy* is an algorithm that decides how and when to attack the agent. In an effort to select attack strategies that produce "realistic" attacks, we propose the following rough criteria for assessing attack realism:

- **Feasibility:** For an attack at time $t$, adversarial observation $o_t$ must lie in the environment's state space.
- **Realistic Perturbation:** For an attack at time $t$, the perturbation of $s_t$ should be restricted to a known (and ideally small) subset of components of the state vector, such that this perturbation could realistically model a sensor inaccuracy or malfunction in a real-world implementation of the RL environment.
- **Low Severity:** Across all time-steps in a given episode, the average "attack severity" (a rough measure of the attack's impact on the *expected action* and next state) should be low; roughly speaking, an *expected action* is one that an expert human operator would take if they were the agent. In other words, attacks should be sparse with respect to time, especially those known to have severe impact on the expected action. "Benign" perturbations (those that ought to have little or no impact on the expected action) may be performed more frequently but will be filtered out of the simulated rollout process if the induced agent action is not sufficiently adversarial.

The attack strategies in our experiments satisfy the second and third bulleted conditions by limiting each perturbation to a single state component and limiting each simulated rollout to one attack (equivalently, after beginning a simulated rollout from a sufficiently adversarial attack, do not attack further). Still, this simple attack method must use environment context to guarantee that the first bulleted condition holds. In general, additional environment context will be necessary to measure attack severity and to design more sophisticated, multi-index perturbation attack strategies. In the Section 4.2, we illustrate an example of benign perturbations in a specific RL environment.

In addition to constraining attacks to certain time-steps and certain state components, an attack strategy involves a *perturbation algorithm* that specifies a way to perturb the state vector within realistic bounds. Our experiments are limited to *targeted attacks*, whose perturbation algorithms deliberately alter the observation in a way that encourages the agent to take a specific adversarial action $a_{\text{adv}}$. Whether such an attack ends up being *sufficiently adversarial* only depends on the normal action $a$ and the attack-induced action $a'$, with no additional dependence on $a_{\text{adv}}$. However, our framework allows attack strategies to employ any perturbation algorithm, targeted or untargeted, as long as the three bulleted realism criteria are met.

Among the adversarial perturbation algorithms, the CW and FGSM attacks are particularly notable. CW attacks (Carlini & Wagner, 2017), are optimization-based methods that generate minimal perturbations capable of misleading models. Conversely, FGSM (Goodfellow et al., 2015), is a gradient-based attack that quickly creates adversarial examples by leveraging the gradients of the loss function with respect to the input data. We default to using FGSM for our experiments, although the experimental framework is agnostic to the perturbation algorithm used.

### 3.3. Measuring Attack Impact

#### 3.3.1. Defining a Property

Attack impact is measured with respect to a handful of *properties* of interest that are chosen in advance. Each property captures certain information about the agent's experience in the environment up until the point at which the property is measured; as such, a property value attributed to some time step of an episode should only depend on environment variables (observed and/or latent) and actions that were realized at or before that time step. All properties

of interest should be able to be computed at the very end of each episode. Certain properties, such as win/loss outcome, may *only* be known at the very end of the episode; however, other properties, such as number of prior steps that incurred some kind of environment-based reward penalty, can be computed at any step during the episode. After a suite of properties of interest have been selected, these properties are logged during all data collection rollouts for both attacked and unattacked episodes. These logged properties, particularly those at the end of each episode, are used for downstream "property impact analysis" (Gross et al., 2022).

#### 3.3.2. Mathematically Modeling Properties

To describe so-called "property impact" from an attack at time step $t$, we start by modeling the end-of-episode value of each $i$-th property $P_i$ in our suite as a random variable $P_i(s_t, I_t, o_t)$ that is a function of three arguments:

- $s_t$ is the state vector at time step $t$;
- $I_t$ is a collection of other hidden environment information (including property logs) at time step $t$; and
- $o_t$ is the (potentially adversarial) observation sent to the agent at time step $t$ (when no attack is present, one has $o_t = s_t$).

When $P_i$ can be expressed meaningfully as a scalar value, expected value of the given property mode becomes a relevant measure. One may estimate $\mathbb{E}(P_i(s_t, I_t, o_t))$ by running repeated trials of simulations from the same $(s_t, I_t, o_t)$ and uniformly averaging the observed values of $P_i(s_t, I_t, o_t)$. Given an attack at time step $t$ that replaces the true state $s_t$ with an adversarial observation $o_t$, the *attacked value* of property $P_i$ is defined to be $P_i(s_t, I_t, o_t)$, and the *unattacked value* of property $P_i$ is defined to be $P_i(s_t, I_t, s_t)$ (in the latter, the agent's observation matches the true state).

#### 3.3.3. Impact Metrics

To measure the impact of the given attack at time step $t$ on property $P_i$, we feed the attacked and unattacked values of $P_i$ into an *impact metric* function $D(\cdot, \cdot)$ as the first and second arguments, respectively. Note that the resulting impact value $D(P_i(s_t, I_t, o_t), P_i(s_t, I_t, s_t))$ is a random variable. One simple impact metric for a scalar-valued property $P_i$ is the difference $P_i(s_t, I_t, o_t) - P_i(s_t, I_t, s_t)$, which conveys both magnitude and direction of the observed change in the property induced by the attack at time step $t$. Another simple impact metric for *any* property $P_i$ is

$$\begin{cases} 1 & \text{if } P_i(s_t, I_t, o_t) \neq P_i(s_t, I_t, s_t) \\ 0 & \text{if } P_i(s_t, I_t, o_t) = P_i(s_t, I_t, s_t). \end{cases}$$

While well-defined, this second impact metric may lose saliency when $P_i$ has any component that ranges over a continuous domain. To illustrate one way to combat this issue, if the property $P_i$ resides in some metric space with a distance metric $d(\cdot, \cdot)$, then one could construct an impact metric such as

$$\begin{cases} 1 & \text{if } d(P_i(s_t, I_t, o_t), P_i(s_t, I_t, s_t)) > d^* \\ 0 & \text{if } d(P_i(s_t, I_t, o_t), P_i(s_t, I_t, s_t)) \leq d^* \end{cases}$$

for some distance threshold $d^*$. For each impact metric function $D$ that is invoked on a given property $P_i$ and a given attack $s_t \to o_t$, one can estimate $\mathbb{E}(D(P_i(s_t, I_t, o_t), P_i(s_t, I_t, s_t)))$ using repeated trials. Specifically, if we let $\mathscr{P}'_i$ be the list of all observed values of $P_i(s_t, I_t, o_t)$ and $\mathscr{P}_i$ be the list of all observed values of $P_i(s_t, I_t, s_t)$, then $\mathbb{E}(D(P_i(s_t, I_t, o_t), P_i(s_t, I_t, s_t)))$ is estimated by the uniform average

$$\frac{1}{|\mathscr{P}'_i||\mathscr{P}_i|} \sum_{p'_i \in \mathscr{P}'_i} \sum_{p_i \in \mathscr{P}_i} D(p'_i, p_i),$$

where both summations range over all elements, including repeats, in the lists $\mathscr{P}'_i$ and $\mathscr{P}_i$.

## 4. Experiments

The motivation behind our experiments is threefold; First to develop an analysis and visualization framework to supplement the researcher's understanding of a policy's learned behavior and vulnerabilities, second to analyze the property impact of attacking various observation indexes, and third to test the transferability of adversarial attacks across agents of various training algorithms and learning curricula.

### 4.1. Experimental Environment

Deep reinforcement learning is increasingly being used to discover adversarial tactics, techniques, and procedures (TTPs) within the cybersecurity domain (Molina-Markham et al., 2021). The gym environment used for notional experiments is CyberStrike; a custom-built, strategic network-defense game wherein an agent controlling blue nodes must determine information about the red network's tree structure, and then hack into each red node's parent node recursively until reaching the target node. The CyberStrike environment is ripe for emergent, explainable learned strategy; contrasting typical control-focused benchmarks such as LunarLander-v2 or Cartpole (G. Brockman, 2016). The action space is multi-discrete, made up of four blue "hackers" that can simultaneously "hack" or "eavesdrop" on a

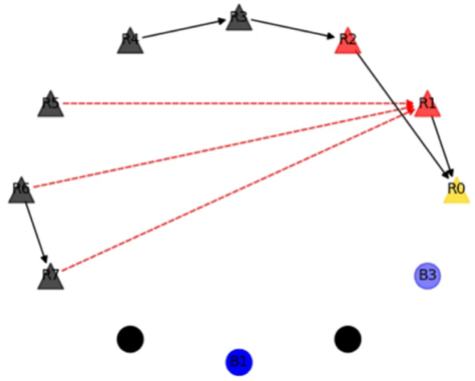


*Figure 3.* A notional CyberStrike state. The blue agent controls nodes B0, B1, B2, B3. Blue chooses actions which control each blue node simultaneously, locating and disabling the target red node by peeling back the layers of the red defense network until the target node is undefended. In this example, the target node ($R0$) is defended by $R1$ and $R2$. $R2$ is defended by $R3$, which is defended by $R4$. $R1$ is defended by $R5$, $R6$, and $R7$. $R6$ is also defended by $R7$. Dashed lines denote a connection marked as unknown in the agent's observation, whereas solid lines represent a known connection. The agent begins with a fully unknown network, and must use its hackers to discover the network topology enough to reveal the target node's ($R0$'s) defenders and eventually hack into the target node.

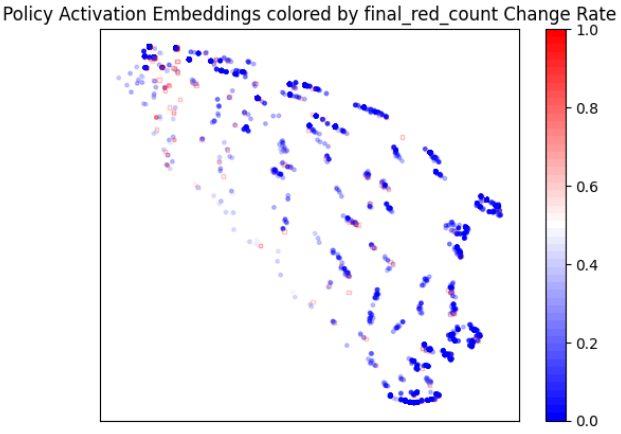


*Figure 4.* This latent space representation maps a policy's CyberStrike observations from initial time-steps in the northwest region to the final time-steps in the southeast region, with an aggregation of various intermediate trajectories connecting the initial and final observations. Attacks within the denser, bluer northeast region of the space are unlikely to yield nonzero changes in final red counts, whereas attacks in the sparser and redder western regions are more likely to be successful (increase final red counts). The sparsity of activation embeddings in the western region of the latent space representation suggests the policy is less likely to have trained on observations in this region and thus is more vulnerable to adversarial attacks when acting within this region.

collection of red nodes. If a blue hacker attempts to hack a defended red node, the red defender will counter and the blue hacker will be unavailable for the rest of the episode. The "eavesdrop" action is only available to one of the blue hackers ($B3$), and allows the agent to stealthily learn the defenders of a red node without risking a counter from red. An example network structure from a mid-episode observation is displayed in Figure 3.

### 4.2. Experimental Setup

First, we train a suite of both Advantage Actor Critic (A2c) (Mnih et al., 2016) and Deep Q-Network (DQN) agents (Mnih et al., 2013) within the CyberStrike environment. Following training we collect 10,000 state-action-metadata tuples from the frozen policies acting within CyberStrike, collecting metadata such as a policy's hidden-layer activations, observation saliency, and step-wise environment properties. Due to the absence of $\varepsilon$-greedy or distribution sampling for exploration, we force a small percentage (5%) of random actions during the data collection to inform potential adversarial targets, though the researcher may vary the percentage of random action selected depending on the environment MDP and frozen policy optimality. These collected data sets from the trained policies help to represent empirical policy behavior through activation clustering, SAMDP transition visualization, and other custom metadata visualizations. For example, Figure 4 shows how one can track the average change rate of a property at any attacked observation. For each collected observation, the policy's final latent activation layer is embedded in two dimensions and colored with a gradient across the aggregate change rates of a property of interest, the change rate being determined by the difference in the property value for the unperturbed versus perturbed observations. These behavioral visualizations help the researcher get a birds-eye view of policy trajectories as they relate to environment properties; while also highlighting feasible, optimally-timed, and low-severity attacks on the policy's learned strategy. We can also run adversarial attacks on (and simulated rollouts from) the observations collected in these data sets, which will be necessary for both Property Impact and Attack Transferability Analysis.

#### 4.2.1. Benign Perturbations in CyberStrike

In the Cyber Strike environment, one kind of benign perturbation would consist of selecting an ordered pair $(A,B)$ of distinct red assets, with at least one already compromised by a blue hack, and changing the agent's perception of whether or not $A$ defends $B$. Such a perturbation on the defense $A \rightarrow B$ is benign because an expert human hacker would not target $B$ if it were already compromised; and if $A$ was

already compromised, then $A$ would not be able to counterattack a blue node following its hack on $B$, making the defense value for $A \rightarrow B$ irrelevant to decision-making. Therefore, our attack-discovery framework would permit this kind of benign attack to be made frequently in a single episode, since an ideal policy ought to not behave differently from any of these attacks.

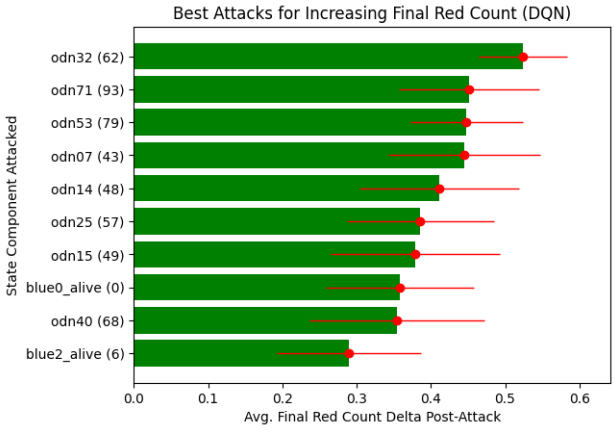


*Figure 5.* The Average *Final Red Count* delta post-attack is aggregated per observation index, across all time-steps. Eight out of the ten most impactful attacked observation indexes are observed defense network nodes, suggesting that an attacker's best chance of increasing the final red count is to perturb the DQN agent's perception of the network structure at various adjacency nodes.

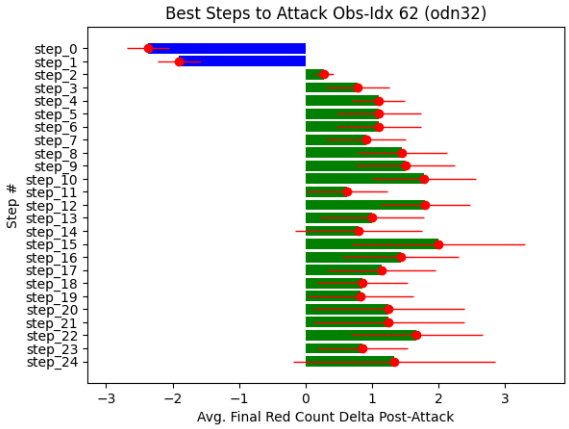


*Figure 6.* The *Final Red Count* delta is plotted at each step for all attacks on *odn*32's value in the DQN's observation. This plots suggests that attacking *odn*32 at the first two steps may have negative effects for the attacker, whereas attacks from step 2 onward correspond with an increase in red nodes (thus a decrease in blue win percentage) compared to an unattacked trajectory.

### 4.3. Property Impact Attack Analysis

In order to measure and compare the aggregate property impact of attacking various observation indexes, we must run adversarial attacks perturbing each observation index for each collected observation tuple. Simulated rollouts are performed only from adversarial attacks inducing an action $a_i'$ that is sufficiently different from the original action $a_i$. Environment properties are measured at the terminal state of the simulated rollout and compared to environment properties of the unattacked trajectory, as to gauge the property impact of a given attack. We can aggregate these impact metrics across step numbers and observation indexes to answer questions about the ideal time-step or observation index to attack with respect to some environment property the attacker wishes to impact. In CyberStrike, we measure the attack impact on properties such as win percentage, final red count, final blue count, and trajectory length. Figure 5 displays a ranked aggregation of final red count deltas across attacked observation indexes for a DQN policy. The policy's most impactful observation index attack with respect to final red count is a perturbation of the value of the observed defense network node 3:2 (*odn32*), denoting if $R3$ is defending $R2$ or if this connection is unknown. In the notional example in Figure 3, $R3$ *is* defending $R2$, so a perturbation obscuring this information may cause the agent to take an action hacking the defended $R2$ node, whereas an optimal decision would be to hack $R2$'s defender node, $R3$, first. Figure 6 displays the impact aggregation across time-steps for all attacks on the *odn32* observation value, suggesting that attacks early in an episode, time-steps 0 and 1, may have negative consequences for the attacker by decreasing their average final red count. Figure 6 shows that time-steps 10 and 15 lead to the largest average final red count increase, suggesting an adversary may have the greatest impact on final red count in the middle of an episode, rather than at the beginning or end.

#### 4.3.1. Property Impact Analysis Results

The Property Impact Analysis results indicate that adversarial attacks may exert both positive and negative influences on the attacker's desired outcome, dependent on the time-step when the attack is brokered. By strategically timing the manipulation of the most vulnerable observation components of a policy, we are able to observe significant variations in policy behavior, leading to notable changes in the environment properties and game outcome; thus demonstrating the ability to deliberately impact an external environment property by choosing a specific adversarial attack target at a specific time-step.

Figure 6 displays the "final red counts" property outcomes when a red attacker attacks the *odn32* observation component at various time-steps. Specifically, we find that attack-

| Attack Source | Attack Transfer Target | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | No-Op Action Target | | | | | *max(loss-win)* Action Target | | | | |
| | A2c-A | A2c-B | A2c-C | DQN-A | DQN-B | A2c-A | A2c-B | A2c-C | DQN-A | DQN-B |
| A2c-A | 67.57% | 35.27% | 71.20% | 15.35% | 16.36% | 67.37% | 35.16% | 70.90% | 8.79% | 21.30% |
| | 13 | 11 | 16 | 0 | 0 | 21 | 69 | 17 | 0 | 0 |
| | 39.99% | 52.70% | 38.57% | 12.77% | 11.00% | 28.36% | 30.82% | 27.54% | 0.05% | 4.54% |
| A2c-B | 67.55% | 35.22% | 71.15% | 14.85% | 16.86% | 67.25% | 35.05% | 70.92% | 8.78% | 21.15% |
| | 16 | 9 | 24 | 0 | 0 | 23 | 46 | 19 | 0 | 0 |
| | 39.99% | 52.68% | 38.54% | 12.89% | 10.99% | 24.70% | 46.44% | 22.98% | 23.84% | 14.97% |
| A2c-C | 67.57% | 35.17% | 71.19% | 15.39% | 16.10% | 67.57% | 35.06% | 71.13% | 8.73% | 21.06% |
| | 19 | 15 | 16 | 0 | 0 | 36 | 15 | 42 | 0 | 0 |
| | 39.93% | 52.70% | 38.55% | 12.77% | 11.02% | 8.69% | 7.83% | 8.29% | 12.03% | 10.86% |
| DQN-A | 67.33% | 35.14% | 71.00% | 14.85% | 13.73% | 67.36% | 35.11% | 70.96% | 10.08% | 35.11% |
| | 15 | 0 | 22 | 0 | 0 | 22 | 12 | 21 | 458056 | 0 |
| | 39.99% | 52.68% | 38.61% | 12.89% | 10.68% | 13.84% | 25.85% | 5.45% | 49.32% | 11.45% |
| DQN-B | 67.37% | 35.14% | 71.00% | 9.17% | 16.86% | 67.43% | 35.11% | 71.01% | 10.05% | 17.15% |
| | 11 | 18 | 28 | 0 | 0 | 18 | 12 | 19 | 0 | 23330 |
| | 40.03% | 52.69% | 38.58% | 13.85% | 10.99% | 6.83% | 6.67% | 5.46% | 1.79% | 12.86% |

*Table 1.* Attack transferability results from experiments outlined in Section 4.4.2. The attacks are configured using the policy in the Attack Source column, targeting the No-Op (left) and *max(loss-win)* (right) action targets. The attacks are then run on the attack-source policy and transferred to the other four policies of interest. Three metrics are recorded per cell: transferability success rate (white sub-cell), target-transferability count out of one million (light gray sub-cell), and sub-action target-transferability success proportion (dark gray sub-cell). *Self-attacks*, where the attack source and target policy are the same, are also included in this table.

ing the observation at the initial steps of the game may lead to an unexpected decrease in the final red count, which runs contrary to the attacker's objective. However, as the game progresses, attacks on specific observations can result in more favorable increase in the final red count, aligning with the attacker's strategic intentions. This finding underscores the dynamic nature of learned policies, even within simple environments. It highlights the delicate interplay between attack targets, how those action targets influence future behavior, and how that future behavior affects the environment properties and the ultimate objective outcome.

### 4.4. Attack Transferability Analysis

In addition to analyzing the property impact of various adversarial attacks on a single policy, we also analyze the transferability of an attack trained with one policy and deployed during another policy's execution.

#### 4.4.1. Transferability Metrics

In order for an attack to be *transferable*, the attack (parameterized by policy $\pi_i$) must induce a *sufficiently adversarial* action when deployed on some other policy $\pi_j$. In CyberStrike, an attack parameterized by a policy $\pi_i$ is counted as *transferable* if it induces an action different from the action taken by a policy $\pi_j$ from the unattacked observation. A targeted attack, parameterized by $\pi_i$, is counted as *target-transferable* if it induces the attack's *target action* on the new policy $\pi_j$. Due to the multi-discrete nature of CyberStrike's action space (which has four sub-actions), we can also measure the proportion of the induced sub-actions matching the target sub-actions, or the *sub-action target-transferability*.

#### 4.4.2. Transferability Experimental Setup

We employ Automated Domain Randomization (ADR) and Curriculum Learning (CL) across the action and counter-action effectiveness dimensions, coined action stickiness by (Machado et al., 2017), to increase the variation in learned strategies, providing more heterogeneous policy targets for attack-transfer. We will analyze attack transferability across a suite of five policies: A2c-ADR+CL (A), A2c-ADR (B), A2c-CL (C), DQN-CL (A), DQN-deterministic (B)).

Training curriculum and hyperparameter details for the policies are available in the appendix. For each policy, we use two action-targets for transferability analysis: the 0-action (No-op) and the *max(loss-win)* action. The *max(loss-win)* action is computed by counting each collected action's usage within winning and losing trajectories; if the action was used $U_L$ times in losing trajectories and $U_W$ times in winning trajectories then the *(loss-win)* value is $U_L - U_W$, and the *max(loss-win)* action target maximizes this value.

After determining the *max(loss-win)* action-target for each

source policy, we run the transferred adversarial attacks. For each observation in each target policy's collected dataset, we run an adversarial attack for each action target and collect metrics regarding *transferability*, *target-transferability*, and *subaction-target-transferability*. We hypothesize that attacks may be more transferable between policies of the same DRL algorithm (A2c-X $\rightarrow$ A2c-Y, or DQN-X $\rightarrow$ DQN-Y), however the target policy should be the biggest factor in transferability, regardless of target action or source policy. We also hypothesize that the *max(loss-win)* action target may be more easily induced compared to the No-Op action, because the No-Op action should not be taken by an optimal or near-optimal policy, whereas the *max(loss-win)* actions are empirically taken by the source policies during losses.

#### 4.4.3. Transferability Results

The policy target is indeed the greatest factor on transferability success rates, especially for the A2c policies where we see roughly the same transferability success rates per policy target, across all attack sources and action targets. It is also worth noting that the *max(loss-win)* action target induces target-transfers most often, but still sparsely, for A2c policies. DQN self-attacks induce the target action 45.8% (DQN-A) and 2.3% (DQN-B) of the time, however attacks transferred to DQN policies never induce the target action. Contrarily, A2c self-attacks induce the target action at roughly the same rate as attacks transferred to A2c policies. The variation in sub-action target-transferability per-row in the *max(loss-win)* block can be attributed to the *max(loss-win)* action being different for each source policy.

## 5. Discussion & Conclusions

The results suggest the ability to influence agent behavior, and thus future environment properties, is controllable through optimally timed, deliberately chosen observation perturbations. This capability, paired with the result showcasing varying levels of attack transferability across algorithm types, highlights the urgent need for robust defense mechanisms and adversarial evaluation schemes to safeguard decision-making policies from the threat of adversarial influence, especially in high-stakes environments. The results also suggest that policies trained with some algorithms, like A2c, may be more vulnerable to transferred attacks than others, such as DQN in this specific experimental setting; and transferability must be measured on a per-algorithm basis. The presence of observation-dependent and time-dependent vulnerabilities implies the existence of training and fine-tuning methods to guard against these vulnerabilities, though we have not explored methods to do so in this paper and leave that to future research.

While this paper focuses on using adversarial attacks to probe and analyze the behavior of policies trained through DRL algorithms, the same behavioral analysis may be conducted on LLM-based agentic architectures, albeit with language-based attacks and alternate metadata for t-SNE embeddings. We will leave this to future adversarial analysis research.

## Impact Statement

The paper presents work whose goal is to advance the field of machine learning, specifically regarding deep reinforcement learning explainability and adversarial analysis. As society continues to adopt DRL and AI solutions broadly, explainability and evaluation methods such as those presented in this paper will help provide frameworks to assure and gain trust of these systems.

## Acknowledgments

The authors thank Guido Zarrella and Dr. Chris Niessen for their advisory roles throughout the research and development process. This work was funded by the 2023 MITRE Independent Research and Development Program.

## References

Baram, N., Zahavy, T., and Mannor, S. Deep reinforcement learning discovers internal models, 2016. URL https://arxiv.org/abs/1606.05174.

Behzadan, V. and Munir, A. Vulnerability of deep reinforcement learning to policy induction attacks. pp. 262–275, 07 2017. ISBN 978-3-319-62415-0. doi: 10.1007/978-3-319-62416-7_19.

Biemann, C. Chinese whispers: an efficient graph clustering algorithm and its application to natural language processing problems. In *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing*, TextGraphs-1, pp. 73–80, USA, 2006. Association for Computational Linguistics.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks, 2017. URL https://arxiv.org/abs/1608.04644.

G. Brockman, V. Cheung, L. P. J. S. J. S. J. T. e. a. "openai gym", 2016.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples, 2015. URL https://arxiv.org/abs/1412.6572.

Gross, D., Simao, T. D., Jansen, N., and Perez, G. A. Targeted adversarial attacks on deep reinforcement learning policies via model checking, 2022.

Hasanbeig, M., Kroening, D., and Abate, A. Deep reinforcement learning with temporal logics. In Bertrand, N. and Jansen, N. (eds.), *Formal Modeling and Analysis of Timed Systems*, pp. 1–22, Cham, 2020. Springer International Publishing. ISBN 978-3-030-57628-8.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. Adversarial attacks on neural network policies, 2017. URL https://arxiv.org/abs/1702.02284.

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., and Pérez, P. Deep reinforcement learning for autonomous driving: A survey, 2021. URL https://arxiv.org/abs/2002.00444.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *CoRR*, abs/1709.06009, 2017. URL http://arxiv.org/abs/1709.06009.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning, 2013. URL http://arxiv.org/abs/1312.5602.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

Molina-Markham, A., Winder, R. K., and Ridley, A. Network defense is not a game, 2021. URL https://arxiv.org/abs/2104.10262.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations, 2018. URL https://arxiv.org/abs/1709.10087.

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

Tapley, A., Gatesman, K., Robaina, L., Bissey, B., and Weissman, J. Utilizing explainability techniques for reinforcement learning model assurance, 2023. URL https://arxiv.org/abs/2311.15838.

van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Velasquez, A., Bissey, B., Barak, L., Beckus, A., Alkhouri, I., Melcer, D., and Atia, G. Dynamic automaton-guided reward shaping for monte carlo tree search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):12015–12023, May 2021. doi: 10.1609/aaai.v35i13.17427. URL https://ojs.aaai.org/index.php/AAAI/article/view/17427.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., and Silver, D. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starc 2019.

Waseda, F., Nishikawa, S., Le, T.-N., Nguyen, H. H., and Echizen, I. Closer look at the transferability of adversarial examples: How they fool different models differently, 2022. URL https://arxiv.org/abs/2112.14337.

# A. Appendix

## A.1. Publicly Available Code

Code for the Cyberstrike environment, DRL-SAT analysis repository, and training repository will be open sourced at: https://github.com/mitre/drlsat.

## A.2. CyberStrike

Cyberstrike is a highly customizable network defense environment, initialized with the following configuration parameters. The values listed were used for experiments, with the exception of standard deviations of ADR variables:

*Listing 1.* CyberStrike Configuration File

```
adr_variables:
  - id: adr_0v1
    type: adr_normal_range
    parameters:
      mean: 1.0
       # standard deviation varies
       # with ADR & CL
      stdev: 1.0
      maximum: 1.0
      minimum: 0.1

  - id: adr_0v2
    type: adr_normal_range
    parameters:
      mean: 1.0
      stdev: 1.0
      maximum: 1.0
      minimum: 0.1

  - id: adr_1v0
    type: adr_normal_range
    parameters:
      mean: 1.0
      stdev: 1.0
      maximum: 1.0
      minimum: 0.1

  - id: adr_2v0
    type: adr_normal_range
    parameters:
      mean: 1.0
      stdev: 1.0
      maximum: 1.0
      minimum: 0.1
scenario:
  red:
    assets:
      - is_target: true #0
        type: 0
        is_alive: True
      - is_target: false #1
        type: 0
```

```
      is_alive: True
    - is_target: false #2
      type: 0
      is_alive: True
    - is_target: false #3
      type: 0
      is_alive: True
    - is_target: false #4
      type: 0
    - is_target: false #5
      type: 0
      is_alive: True
    - is_target: false #6
      type: 0
      is_alive: True
    - is_target: false #7
      type: 0
      is_alive: True
  defense_network:
    - [1, 2] #red node 0 defended by [1,2]
    - [ 5, 6, 7 ]# red node 1 is defended by [5, 6 and 7]
    - [3] #red node 2 defended by 3
    - [4]
    - [] #4
    - [] #5
    - [] #6
    - [6] #red node 7 is defended by 6
blue:
  assets:
    - type: 1
      loss_cost: 20
      use_cost: 2
    - type: 2
      loss_cost: 20
      use_cost: 2
    - type: 2
      loss_cost: 20
      use_cost: 2
      is_alive: True
    - type: 3
      loss_cost: 10
      use_cost: 5
effect_probability:
# type {row_idx} effectiveness
# hacking type {col_idx}
    - [ 0,       adr_0v1, adr_0v2, 0]
    - [ adr_1v0, 0,       0,       0]
    - [ adr_2v0, 0,       0,       0]
    - [0,        0,       0,       0]
```

### A.3. Observation Space

The observation space in CyberStrike consists of "*alive*" and "*type*" information for all blue assets, "*alive*" and "*type*" and "*is_target*" information for red assets, and the observed defense network, from blue's perspective. This information is

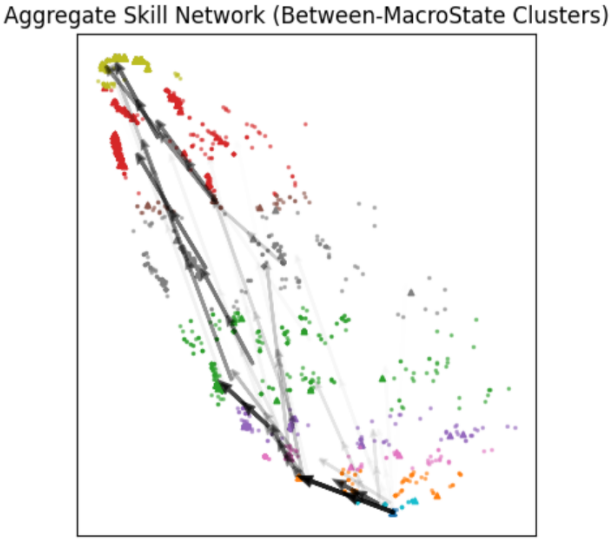


*Figure 7.* Embedded policy activation vectors are colored by their cluster, determined by Chinese-Whispers, and marked with aggregate skill-transition arrows. The shading of the arrows represent the empirical likelihood of the policy transitioning from one cluster to another; thus the most-travelled trajectories are marked by the darkest-shaded arrow path.

flattened into an array and passed to the agent as a flat tensor. The size of the flat tensor is formally

$$3 * (num\_blue + num\_red) + num\_red^2$$

### A.4. Action Space

The action space in CyberStrike is multi-discrete, with each blue asset capable of being paired to some red asset (or no red asset) for any given multi-discrete action. This means the action space linearly increases as we increase the number of red or blue assets in the configuration. Formally the action space is of size

$$num\_blue * (num\_red + 1)$$

### A.5. Strategy and Optimality

The optimal strategy in CyberStrike requires using eavesdrop assets to discover the defense network nodes, and then utilizing hacking assets to infiltrate the defense network, hacking undefended assets first, until the target is reached through recursive hacks. In the absence of adversarial attacks, DRL policies optimize towards this behavioral pattern.

### A.6. Curriculum Learning and Automated Domain Randomization

We randomize the action effectiveness variables for Curriculum Learning (CL) and Automated Domain Randomization (ADR) policies. For the CL policies, the curriculum incrementally adds new variables to randomize as level difficulty increases. We increase the environment level whenever the learning agent reaches 90% on its current level. For instance, the CL agent starts training in a fully deterministic environment. Once 90% win-rate is reached, the environment randomizes one the four effect probabilities, sampling from a truncated normal distribution centered around 1, standard deviation of 1, and minimum and maximum of 0 and 1. As the agent reaches 90% win-rate on this second level, the environment randomizes yet another action-effectiveness dimension, until eventually all four variables are sampled with a standard deviation of 1.0 in the last level. During purely ADR training, we sample from this truncated distribution with a standard deviation of 1.0, for each of the four action effectiveness dimensions; and this pure-ADR level is identical to the final, fully-randomized level on the CL-denoted policies. The policy denoted ADR+CL (A2c-A) is trained with a curriculum that increases the standard deviation of all effectiveness probabilities by 25% every level. Once 90% win-rate is reached on the deterministic level 1, the agent begins training on level 2, where there is a 0.25 standard deviation for the action stickiness sampling distribution centered around 1. This ADR+CL lesson plan increases the standard deviation from $0 \rightarrow 0.25 \rightarrow 0.5 \rightarrow 0.75 \rightarrow 1.0$.

### A.7. Training hyperparameters

All DRL policies were trained with either DQN or A2c, utilizing the standard deep Q-learning algorithm (Mnih et al., 2013) and the standard advantage actor critic algorithm introduced in (Mnih et al., 2016). For the DQN policies, we use a discount factor of .99, replay ratio of 4, target update tau of 0.05 with an interval of 250, an Adam optimizer, clip grad norm of 10, and a learning rate of $3e-4$. The $\varepsilon$-greedy exploration module initializes at 1.0, decays by a factor of .99 to a minimum epsilon of 0.01.

For the A2c policies, we use a discount factor of 0.99, actor learning rate of 1.5e-4, critic learning rate of 3e-4, value loss coefficient of 0.5, entropy loss coefficient of 0.01, Adam optimizer, and clip grad norm of 10. The networks ingest a flat input layer of varied size, depending on the size of the CyberStrike configuration. In the CyberStrike configuration used for experiments, where we have 8 red nodes and 4 blue hackers, the input size is 100. The hidden dimension, and thus the size of the activations used for embedding and clustering, is set to 256 by default. Policies are trained through their curriculum, until 90% win-rate is reached on the final level. At this point, policies are frozen, evaluated, and collected for analysis. Both the actor network and DQN are instantiated as follows:

*Listing 2.* DQN and Actor Network

```
hidden_dim = 256
self.fc = torch.nn.Sequential(
    nn.Linear(in_shape[0], hidden_dim),
    nn.ReLU(),
    nn.Linear(hidden_dim, hidden_dim),
    nn.ReLU(),
    nn.Linear(hidden_dim, out_shape[0]),
)
```

The critic network for A2c training is instantiated as follow:

*Listing 3.* Critic Network

```
hidden_dim = 256
self.fc = torch.nn.Sequential(
    nn.Linear(in_shape[0], hidden_dim),
    nn.Tanh(),
    nn.Linear(hidden_dim, hidden_dim),
    nn.Tanh(),
    # Single output neuron for value function.
    nn.Linear(hidden_dim, 1),
)
```

### A.8. Analysis hyperparameters

For SAMDP analysis, we utilize the policy network activation vectors to create t-Distributed Stochastic Neighbor Embeddings (t-SNE) (van der Maaten & Hinton, 2008) with a perplexity of 132. We utilize Chinese-Whispers (Biemann, 2006) clustering algorithm with a critical distance of 15.0 to cluster the policy activation vectors, and color the associated 2d embedded points with a unique cluster color. In addition to coloring by cluster, shown in Figure 7, we can also color by adversarial or atomic property attributes as in Figure 4.