Adversarial Deep Reinforcement Learning based Adaptive Moving Target Defense

Taha Eghtesad[1], Yevgeniy Vorobeychik[2], and Aron Laszka[1]

[1] University of Houston, Houston, TX 77004, USA
[2] Washington University in St. Louis, St. Louis, MO, 63130

Abstract. Moving target defense (MTD) is a proactive defense approach that aims to thwart attacks by continuously changing the attack surface of a system (e.g., changing host or network configurations), thereby increasing the adversary's uncertainty and attack cost. To maximize the impact of MTD, a defender must strategically choose when and what changes to make, taking into account both the characteristics of its system as well as the adversary's observed activities. Finding an optimal strategy for MTD presents a significant challenge, especially when facing a resourceful and determined adversary who may respond to the defender's actions. In this paper, we propose a multi-agent partially-observable Markov Decision Process model of MTD and formulate a two-player general-sum game between the adversary and the defender. To solve this game, we propose a multi-agent reinforcement learning framework based on the double oracle algorithm. Finally, we provide experimental results to demonstrate the effectiveness of our framework in finding optimal policies.

1 Introduction

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

Moving Target Defence (MTD) is a proactive approach that changes the rules of the game in favor of the defenders. MTD techniques enable defenders to thwart cyber-attacks by continuously and randomly changing the configuration of their assets (*i.e.,* networks, hosts, etc.). These changes increase the uncertainty and complexity of attacks, making them computationally expensive for the adversary [32] or putting the adversary in an infinite loop of exploration [28].

Currently, system administrators typically have to manually select MTD configurations to be deployed on their networked systems based on their previous experiences [9]. This approach has two main limitations. First, it can be very time consuming since 1) there are constraints on data locations, so that the system administrator must make sure

that constraints are met before deploying MTD, 2) physical connectivity of servers cannot be easily changed, and 3) resources are limited. Second, it is difficult to capture the trade-off between security and efficiency since the most secure configuration is total randomization, but this has high performance overhead [5].

In light of this, it is crucial to provide automated approaches for deploying MTD, which maximize security benefits for the protected assets while preserves the efficiency of the system. The key ingredient to automation of MTD deployment is finding a design model that reflects multiple aspects of the MTD environment [13, 32, 22, 2]. Further, we need a decision making algorithm for the model to select when to deploy an MTD technique and where to deploy it [28]. Finding optimal strategies for the deployment of MTD is computationally challenging since there can be huge number of applicable MTD deployment combinations even with trivial number of MTD configurations or in-control assets. Further, the adversary might adapt to these strategies.

One of the main approaches for finding decision making policies is *Independent Reinforcement Learning* (InRL). Recently, many research efforts have applied InRL to find the optimal action policies in fully or partially observable environments in various domains. These domains include: cybersecurity, hardware design, robotics, finance, and etc. In InRL, an agent learns to make optimal decisions by continuously interacting with its environment. In general, traditional reinforcement learning techniques use tabular approaches to store estimated rewards (*e.g.,* Q-Learning) [10]. To address challenges of reinforcement learning such as exploding state-action space, *Artificial Neural Networks* (ANN) have replaced table based approaches in many domains, thereby decreasing the training time and memory requirements. This led to the emergence of *deep reinforcement learning* (DRL) algorithms such as DQL [18].

Contributions We formulate a multi-agent partially-observable Markov decision process for MTD, and based on this model, we propose a two-player general-sum game between the adversary and the defender. Then, we present a multi-agent deep reinforcement learning approach to solve this game. Our main contributions are as follows:

- We propose a multi-agent partially-observable Markov decision process for MTD.
- We propose a two-player general-sum game between the adversary and the defender based on this model.
- We formulate the problem of finding adaptive MTD policies as finding the mixed-strategy Nash equilibrium (MSNE) of this game.
- We propose a compact memory representation for the defender and adversary agents, which helps them to better operate in the partially-observable environment.
- We propose a computational approach for finding the optimal MTD policy using Deep Q-Learning and the Double Oracle algorithm.
- We evaluate our approach numerically while exploring various game parameters.
- We show that our approach is viable in terms of computational cost.

Organization The rest of the paper is organized as follows. In Section 2, we describe preliminaries, including InRL (Section 2.1) and one specific InRL algorithm, Deep Q Learning (Section 2.2). In Section 3, we introduce a multi-agent partially-observable Markov decision process for MTD, which is used as the basis of the MARL. In Section 4, we formulate a two-player general-sum game between the adversary and the

defender, and formulate the problem of finding adaptive MTD policies as finding the MSNE of the game. In Section 5, we propose our solution to solving the MTD game. In Section 6, we provide a thorough numerical analysis of our approach. In Section 7, we discuss the related work. Finally, in Section 8, we provide concluding remarks.

2 Preliminaries

In this section, we describe a family of reinforcement learning algorithms (Section 2.1), and one particular algorithm in this family, namely Deep Q-Learning (Section 2.2). Readers who are familiar with these concepts may skip this section and continue to Section 3.

2.1 Independent Reinforcement Learning

One of the primary approaches for finding a decision-making policy is *Independent Reinforcement Learning* (InRL), which focuses on interactions of a single agent and its environment to maximize the agent's gain (represented as rewards or utilities) from the environment. Figure 1 shows the interactions between different components of InRL. A basic InRL environment is a *Partially-Observable Markov Decision Process* (POMDP), which can be represented as a tuple:

$$\textit{POMDP} = \langle \mathbb{S}, \mathbb{A}, \mathbb{T}, \mathbb{R}, \mathbb{O} \rangle. \tag{1}$$

where $\mathbb{S}$ is the set of all possible states of the environment, $\mathbb{A}$ is the set of all possible actions by the agent, $\mathbb{T}$ is the set of stochastic transition rules and, $\mathbb{R}$ is the immediate reward of a state transition, and $\mathbb{O}$ is the set of observation rules of the agent. For more detailed information on POMDP, please refer to [23].

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

$$\pi(o_\tau) \mapsto a_\tau \tag{2}$$

$$\text{which maximizes } U^*_\tau = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_{t+\tau} \,\middle|\, \pi\right] \tag{3}$$

where o_τ is the observation received in time step τ, $a_{\tau+1}$ is the action taken after that observation, and r_τ is the reward received in time step τ after a state transition due to action a_τ. Also, the *discount factor* $\gamma \in [0, 1)$ prioritizes rewards received in the current time step over future rewards. When $\gamma = 0$, the agent cares only about the current reward; and when $\gamma = 1$, the agent cares about all future rewards equally. Note that in a partially-observable environment, the agent should consider its history of observations. However, considering the complete history of observations may be computationally challenging, so practical approaches limit the observation history (*e.g.,* limited number of recent observations [18], agent memory [21]). We propose a compact representation of history in Section 5.2, but for ease of presentation, we will treat policies as mappings from most recent observations until then.

The training is done in iterations called *steps*. In each step, the agent decides on an action to take which updates the state of the environment based on transition rules, and the agent receives the new observation from the environment and immediate reward of transition. To make sure that the majority of action/observation space is explored and the learning agent is not stuck in a locally optimal state, after an arbitrary number of steps, the environment state is *reset* to an arbitrary/random initial state and the agent receives the observation of the initial state. In the terms of RL, steps between one reset and the next one are called an *epoch* of training.

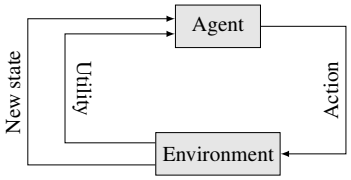


Fig. 1: Independent reinforcement learning.

Algorithm 1: Deep-Q Learning

Result: policy σ
$Q \leftarrow$ random;
for N_e *episodes* **do**
  $O \leftarrow$ reset_game();
  $\epsilon_\tau \leftarrow 1$;
  for $\tau \in \{0, \ldots, T_{epoch}\}$ **do**
    if $random[0,1] \leq \epsilon_\tau$ **then**
      $a \leftarrow$ random_action;
    else
      $a \leftarrow \text{argmax}_{a'} Q(S, a')$;
    end
    $(S', r) \leftarrow$ step_game(a);
    add $e = \langle S, S', a, r \rangle$ to E;
    sample X from E;
    update DQN based on X;
    $S \leftarrow S'$;
    decay ϵ_τ;
  end
end
$\sigma \leftarrow \langle S \mapsto \text{argmax}_a Q(S, a) \rangle$;

Algorithm 2: Adaptive Solver

Result: set of pure policies Π^a and Π^d
$\Pi^a \leftarrow$ attacker heuristics;
$\Pi^d \leftarrow$ defener heuristics;
while $U^p(\sigma^p, \sigma^{\bar{p}})$ *not converged* **do**
  $\sigma^a, \sigma^d \leftarrow$ solve_MSNE(Π^a, Π^d);
  $\theta \leftarrow$ random;
  $\pi^a_+ \leftarrow$ train($T \cdot N_e$, *env*$^a[\sigma^d], \theta$);
  $\Pi^a \leftarrow \Pi^a \cup \pi^a_+$;
  assess π^a_+;
  $\sigma^a, \sigma^d \leftarrow$ solve_MSNE(Π^a, Π^d);
  $\theta \leftarrow$ random;
  $\pi^d_+ \leftarrow$ train($T \cdot N_e$, *env*$^d[\sigma^a], \theta$);
  $\Pi^d \leftarrow \Pi^d \cup \pi^d_+$;
  assess π^d_+;
end

Each *step* to the environment updates the state of the system based on the agent's action (a) and the current state of the environment (s), and returns a new observation (o), immediate utility given to agent (r), and whether the environment is finished or not. This new information and the previous observation of the agent forms an *experience*. Specifically, an experience is defined as a tuple of:

$$e = \langle o_\tau, a_\tau, o_{\tau+1}, r_\tau \rangle \tag{4}$$

where o_τ and a_τ are the agent's observation and action at time step τ; and $o_{\tau+1}$ and r_τ are the agent's observation and immediate utility received at the next time step $\tau + 1$. The set of recent experiences is used to update the policy.

Reinforcement learning aims to maximize the received utility of the agent (U_*) by trial and error: interacting with the environment (randomly, following heuristics, or based on the experiences that the agent has seen so far). Generally, during the training, there are two ways to find actions to be taken at each step: (1) *Exploitation*: we use the currently trained policy to choose actions, which helps the agent to more accurately find U_* values of states. (2) *Exploration*: to find actions that lead to higher utility by selecting actions at random and exploring the action/observation space. One of approaches for choosing between exploration or exploitation is the ϵ-greedy approach, where in each step the agent explores with probability ϵ, or takes the current optimal action with probability $1 - \epsilon$.

2.2 Deep-Q-Network Learning

The Deep-Q-Network Learning algorithm is described in Algorithm 1. Q-learning uses a Q function to estimate the expected future utilities of an action in an observation state (Equation (3)):

$$Q(o_\tau, a_\tau) = U^*_\tau|_{\pi \leftarrow \text{argmax}_{a'} Q(o_\tau, a')} \tag{5}$$

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

$$Q(o_\tau, a_\tau) = (1 - \alpha_q) \cdot Q(o_\tau, a_\tau) + \alpha_q \cdot \underbrace{(r_\tau + \gamma \cdot \max_{a'} Q(o_{\tau+1}, a'))}_{\text{TD Target}} \tag{6}$$

where α_q is the learning rate of the Q function. The idea for updating the Q function is that the Q function should minimize the *temporal difference* (TD) error, *i.e.,* the difference between the predicted Q value, and the actual expected utility (U^* while following $\pi \leftarrow \text{argmax}_{a'} Q(O_\tau, a')$).

When we are dealing with environments with highly dimensional action/observation spaces, tabular based Q-learning is infeasible since: 1) the table for storing Q-values might not fit into memory, and 2) the action and observation spaces need to be enumerated many times for the algorithm to learn an optimal policy. To address these challenges, Mnih *et al.* [18] suggests to use *multi layer perceptrons* (MLP) as approximators for the Q function. Using MLP as Q-value approximator makes the deep Q-learning approach feasible for such environments since 1) at most thousands parameters are stored, and 2) MLP models can generalize the relation between observations and actions; as a result, learning agents need less time for exploring the observation/action space.

To optimize the parameters of MLP (θ), we can use gradient descent to minimize the TD error of the network. With the same TD target as Equation (6), and taking optimal action as $\text{argmax}_{a'} Q(o_\tau, a'|\theta)$, the TD target will be:

$$q_\tau = r_\tau + \gamma \cdot Q(o_{\tau+1}, \text{argmax}_{a'} Q(o_\tau, a'|\theta))|\theta) \tag{7}$$

Suppose we have a batch of experiences X for updating the MLP parameters, then we can define a *mean squared error* (MSE) loss function and apply gradient descent

with learning rate α_θ to optimise the MLP parameters :

$$L_\theta = \frac{1}{|X|} \sum_{i}^{X} (q_\tau - Q(o_\tau, a_\tau|\theta))^2 \tag{8}$$

3 Model

To model adaptive *Moving Target Defense*, we build a *Multi-Agent Partially-Observable Markov Decision Process* (MAPOMDP) based on the model of Prakash and Wellman [22]. A Multi-Agent POMDP is a generalization of POMDP to consider multiple agents influencing the environment simultaniously. Formally:

$$\textit{MAPOMDP} = \langle \mathbb{S}, \{\mathbb{A}_i\}, \mathbb{T}, \{\mathbb{R}_i\}, \{\mathbb{O}_i\} \rangle \tag{9}$$

where $\mathbb{A}_i$ is the action space, $\mathbb{O}_i$ is the observation set of observation rules, and $\mathbb{R}_i$ is the immediate reward of a state transition for player i. In the following subsections (Sections 3.1 through 3.5), we describe these sets in terms of an MTD model.

In this adversarial model, there are two players, a defender and an adversary ($p = a$ and $p = d$, resp.), who compete for control over a set of servers. At the beginning of the game, all servers are under the control of the defender. To take control of a server, the adversary can launch a *"probe"* against the server at any time, which either compromises the server or increases the success probability of subsequent probes. To keep the servers safe, the defender can *"reimage"* a server at any time, which takes the server offline for some time, but cancels the adversary's progress and control. The goal of the defender is to keep servers uncompromised (*i.e.,* under the defender's control) and available (*i.e.,* online). The goal of the adversary is to compromise the servers or make them unavailable. For a list of symbols used in this paper, see Table 1.

3.1 Environment and Players

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

3.2 State

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

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

Table 1: List of Symbols and Experimental Values

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

3.3 Actions

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

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

3.4 Rewards

Prakash and Wellman [22] define a family of utility functions. The exact utility function can be chosen by setting the values of preference parameters, which specify the goal of each player. The value of player p's utility function u^p at a particular, as described by

Equations (10) and (11), depends on the number of servers in control of player p and the number of servers offline. Note that the exact relation depends on the scenario (*e.g.,* whether the primary goal is confidentiality or integrity), but in general, a higher number of controlled servers yields a higher utility.

$$u^p(n_c^p, n_d) = w^p \cdot f\left(\frac{n_c^p}{M}, \theta^p\right) + (1 - w^p) \cdot f\left(\frac{n_c^p + n_d}{M}, \theta^p\right) \tag{10}$$

where n_c^p is the number of servers which are up and in control of player p, n_d is the number of unavailable (down) servers, and f is a sigmoid function with parameters $\theta^p \leftarrow (\theta_{sl}^p, \theta_{th}^p)$:

$$f(x, \theta^p) = \frac{1}{\mathrm{e}^{-\theta_{sl}^p \cdot (x - \theta_{th}^p)}} \tag{11}$$

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

Reward weight (w^p) specifies the goal of each player. As described by Prakash and Wellman [22], there can be four extreme combinations of this parameter, which are summarized in Table 2. For example, in *control / availability*, both players gain reward by having the servers up and in their control. Or in *disrupt / availability*, which is the most interesting case, the defender gains reward by having the servers up and in its control, while the adversary gains reward by bringing the servers down or having them in its control.

Table 2: Utility Environments

	Utility Environment	w^a	w^d
0	control / availability	1	1
1	control / confidentiality	1	0
2	disrupt / availability	0	1
3	disrupt / confidentiality	0	0

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

The reward given to the adversary (r_τ^a) and defender (r_τ^d) at time τ is defined by:

$$r_\tau^d = u^d, \qquad r_\tau^a = \begin{cases} u^a(n_c^a, n_d) - C_A & \text{adversary probed a server at } \tau \\ u^a(n_c^a, n_d) & \text{adversary did nothing at } \tau \end{cases} \tag{12}$$

3.5 Observations

A key aspect of the model is the players' uncertainty regarding the state of the servers. The defender does not know which servers have been compromised by the adversary. Further, the defender observes a probe only with a fixed probability $1 - \nu$ (with probability ν, the probe is undetected). Consequently, the defender can only estimate the number of probes against a server and whether a server is compromised. However, the

defender knows the status of all servers (*i.e.,* whether the server is up or down; and if it is down, how many time steps it requires to be back up again).

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

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

$$o^p = \langle o_0^p, o_1^p, \cdots, o_{M-1}^p \rangle \tag{13}$$

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

$$\forall_{0 \leq i < M} : \qquad o_i^a = \langle \tilde{\rho}^a, \chi, \tilde{v}^a \rangle \tag{14}$$

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

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

$$\forall_{0 \leq i < M} : \qquad o_i^d = \langle \tilde{\rho}^d, v \rangle \tag{15}$$

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

4 Problem Formulation

In Section 3, we built an MTD model using a MAPOMDP. In this section, based on this model, we design an adversarial game between the adversary and the defender. In this setting, we assume that each player chooses a strategy to play, where a strategy is a policy function that maps an observation of the environment to an action to be taken. Since we assume that each strategy is a policy function, in the remainder of this paper, we use the terms *strategy* and *policy* interchangeably.

4.1 Pure Strategy

A pure strategy π^p for player p is a deterministic policy function $\pi^p(o^p) \mapsto a^p$, which given player p's current observation of the system o^p produces an action a^p to be taken by the player. We let $\mathit{\Pi}^p$ denote the set of all pure strategies (*i.e.,* policies) of player p.

When the players are following pure policies $\pi^a \in \mathit{\Pi}^a$ and $\pi^d \in \mathit{\Pi}^d$, their expected cumulative utility can be expressed as the sum of discounted future rewards with discount factor γ. Formally, we can express player p's expected cumulative utility where $\bar{p}$ denotes player p's opponent as:

$$U^p(\pi^p, \pi^{\bar{p}}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^\tau \cdot r_\tau^p \,\middle|\, \pi^p, \pi^{\bar{p}} \right] \tag{16}$$

4.2 Mixed Strategy

One way to express stochastic policies is to use probability distributions over pure policies. A mixed strategy of player p is a probability distribution $\sigma^p = \{\sigma^p(\pi^p)\}_{\pi^p \in \Pi^p}$ over the player's pure strategies Π^p, where $\sigma^p(\pi^p)$ is the probability that player p chooses policy π^p.

We let Σ^p denote player p's mixed strategy space. The expected utility of the adversary and the defender when they are following mixed strategies $\sigma^a \in \Sigma^a$ and $\sigma^d \in \Sigma^d$, respectively, can be calculated as:

$$\forall_{p \in \{a,d\}}: \qquad U^p(\sigma^p, \sigma^{\bar{p}}) = \sum_{\pi^p \in \Pi^p} \sum_{\pi^{\bar{p}} \in \Pi^{\bar{p}}} \sigma^p(\pi^p) \cdot \sigma^{\bar{p}}(\pi^{\bar{p}}) \cdot U^p(\pi^p, \pi^{\bar{p}}) \tag{17}$$

Note that we overloaded the notation for the players' pure-strategy utility to also denote their mixed-strategy utility since the distinction will always be clear from the context and function arguments.

4.3 Solution Concept

The aim of both players is to maximize their utility. As we are considering a rational adversary and defender, we can assume that they always pick a strategy that maximizes their own utility. A *best response* mixed strategy $\sigma^p_*(\sigma^{\bar{p}})$ provides maximum utility for player p given that its opponent $\bar{p}$ is using mixed strategy $\sigma^{\bar{p}}$. Formally, if the opponent $\bar{p}$ is using a mixed strategy $\sigma^{\bar{p}}$, then player p's best response σ^p_* is

$$\sigma^p_*(\sigma^{\bar{p}}) = \text{argmax}_{\sigma^p} U^p(\sigma^p, \sigma^{\bar{p}}). \tag{18}$$

We optimize each player's strategy assuming that its opponent will always use a best-response strategy. This formulation is in fact equivalent to finding a *mixed-strategy Nash equilibrium* (MSNE) of the players' policy spaces Π^a and Π^d. Formally, a profile of mixed strategies (σ^a_*, σ^d_*) is a MSNE *iff*

$$\forall_{p \in \{a,d\}} \forall_{\sigma^p \in \Sigma^p} : U^p(\sigma^p_*, \sigma^{\bar{p}}_*) \geq U^p(\sigma^p, \sigma^{\bar{p}}_*) \tag{19}$$

That is, neither player can increase its expected utility by unilaterally changing its strategy. In the next section, we propose an approach for finding the MSNE of the MTD game, where Π^a and Π^d are the policy space of the players.

5 Framework

In Section 4, we proposed a general-sum game based on the MAPOMDP model described in Section 3. We concluded that finding an optimal action policy for the adversary and the defender in the MTD setting is equivalent to finding an MSNE of the game. In this section, we propose a computational approach and build a framework atop the double oracle (Section 5.1) and DQL (Section 2.2) algorithms to find optimal action policies for the adversary and the defender.

5.1 Solution Overview

The iterative *Double Oracle* (DO) algorithm [17] finds an MSNE of a game given an arbitrary initial subset Π_0^p of each player p's strategy set ($\Pi_0^p \subset \Pi^p$). The DO algorithm extends these subsets iteratively by alternating between 1) finding MSNE of the game spanned by these subsets and 2) extending the subsets with best-response strategies against the latest MSNE.

We let Π_t^p denote player p's explored subset of strategies in iteration t of the DO algorithm. In each iteration, for each player, the DO algorithm refers to a *best response oracle*, an algorithm which finds a pure-strategy best response, to compute a best-response strategy against the opponent's MSNE strategy. Then, it adds this best response to the player's strategy set. Formally, in each iteration:

$$\forall_{p\in\{a,d\}} : \Pi_{t+1}^p \leftarrow \Pi_t^p \cup \left\{\pi_*^p\left(\sigma_{*,t}^{\bar{p}}\right)\right\} \tag{20}$$

where $\sigma_{*,t}^p$ is the MSNE of the player p given the strategy sets Π_t^p. The DO algorithm guarantees [17] the convergence of the MSNE of these strategy sets to an MSNE of the game as long as the strategy sets are finite for both players. However, as the players' strategy sets are vast (even though they are finite) in our game model, enumeration of the strategy sets in search of the best response is infeasible.

For each player, we can use an InRL algorithm, such as Q-Learning [30], as a best-response oracle to find a best-response pure strategy against the opponent's MSNE strategy. Since the opponent's strategy is fixed, the player can use reinforcement learning by treating the opponent's actions as part of its localized environment.

5.2 Challenges

Solving the MAPOMDP model of Section 3 with the DO algorithm is not straightforward. In the following paragraphs, we discuss the issues faced while solving the MAPOMDP model and propose approaches for resolving these issues.

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

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

defender) and (b) the amount of time since the last observed probe $\tilde{p}^d$. So, the actual state representation of the defender will be:

$$\forall_{0 \leq i < M} : \qquad o_i^d = \langle \tilde{\rho}^d, v, \tilde{p}^d, r \rangle \tag{21}$$

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

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

$$\forall_{0 \leq i < M} : \qquad o_i^a = \langle \tilde{\rho}^a, \chi, \tilde{v}^a, p \rangle \tag{22}$$

Complexity of MSNE Computation In zero-sum games, computation of MSNE can be done in polynomial time (*e.g.,* linear programming). However, in general-sum games, the problem of finding the MSNE of given strategy sets of players is PPAD-complete [27], which makes computation of true MSNE infeasible for a game of non-trivial size. Therefore, we use an ϵ-*equilibrium* solver, which produces an approximate correct result. One such solver is the Global Newton solver [6].

Equilibrium Selection Typically, the DO algorithm is used with zero-sum games, where all equilibria of a game yield the same expected payoffs. However, in general-sum games, there may exist multiple equilibria with significantly different payoffs. The DO algorithm in general-sum games converges to only one of these equilibria. The exact equilibrium to which the DO algorithm converges depends on the players' initial strategy sets and the output of the best-response oracle. However, in our experiments (Section 6.3), we show that in our game, this problem is not significant in practice, *i.e.,* all equilibria yield almost the same expected payoffs (Figure 3) regardless of the initial strategy sets.

Model Complexity Due to the complexity of our MAPOMDP model, computation of best response using tabular InRL approaches (*e.g., Q*-Learning) is computationally infeasible. For example, the size of state space for the defender is $(2T^3)^M$ since each of $\hat{\rho}^d, \hat{p}^d$, and r can take any value between 0 and T, and v can only take two values. Although we expect that the values of $\hat{\rho}^d, \hat{p}^d$, and r be much smaller than T due to the dynamics of the game, it is still infeasible to explore each state even once or store a tabular policy in memory for a game of non-trivial size on a conventional computer. To address this challenge, we use computationally feasible *approximate best-response oracles* to find approximate best-response strategies instead of best responses. Lanctot *et al.* [11] show that deep reinforcement learning can be used as an approximate best-response oracle. However, when approximate best responses are used instead of true best responses, convergence guarantees are lost. In our experiments, we show that this algorithm does converge in only a few iterations (see Figure 2b).

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

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

5.3 Solution Approach

Prakash and Wellman [22] proposed a set of heuristic strategies for each player (described in Section 6.1). However, as these strategies are only a subset of the agents' strategy sets, their MSNE is not necessarily an MSNE of the complete game. In Section 5.1, we showed how we can find the MSNE of the game, given a subset of strategy sets for each agent. In this section, based on our approach for resolving challenges of solving our MTD game with the DO algorithm (Section 5.2), we propose our framework to find the MSNE of the MTD game and therefore, the optimal decision making policy for the adversary and the defender. Algorithm 2 shows a pseudo-code of our framework.

We start by initializing the adversary's and defender's strategy sets Π_0^a and Π_0^d with heuristic policies (Section 6.1). From this stage, we proceed in iterations. In each iteration, first, we compute an MSNE (σ^a, σ^d) of the game restricted to the current strategy sets Π^a and Π^d, take the adversary's equilibrium mixed strategy σ^a and train an approximate best-response policy $(\pi_+^d(\sigma^a))$ for the defender assuming that the adversary uses σ^a. Next, we add this new policy to the defender's set of policies ($\Pi^d \leftarrow \Pi^d \cup \{\pi_+^d\}$).

Then, we do the same for the adversary. First, find the MSNE strategy of the defender (σ^d), and train an approximate best-response policy $(\pi_+^a(\sigma^d))$ for the adversary assuming that the defender uses σ^d. Then, we add this new policy to the adversary's set of policies ($\Pi^a \leftarrow \Pi^a \cup \{\pi_+^a\}$).

In both cases, when computing an approximate best response $(\pi_+(\sigma_*))$ for a player against its opponent's MSNE strategy σ_*, the opponent's strategy σ_* is fixed, so we may consider it to be part of the player's environment. As a result, we can cast the problem of finding an approximate best response as *Independent Reinforcement Learning* (InRL). Each invocation of InRL, denoted as `train()` in Algorithm 2, receives the limit on the number of training steps T of training and initial parameters θ. Moreover, we let $env^p[\sigma^{\bar{p}}]$ denote the InRL environment for player p when its opponent plays a mixed strategy $\sigma^{\bar{p}}$.

As we are dealing with discrete action and observation spaces in the MTD model, DQL [18] is a suitable InRL algorithm for finding an approximate best response. In each time step of the InRL, both players need to decide on an action. The learning agent either chooses an action randomly (*i.e.,* exploration), or follows its current policy.

The opponent, whose strategy is a fixed mixed strategy $\sigma^{\bar{p}}$, refers to a pure strategy $\pi^{\bar{p}} \in \Pi^{\bar{p}}$ with probability distribution $\sigma^{\bar{p}}$ and follows that policy.

The MSNE payoff evolves over the iterations of the DO algorithm: whenever we add a new policy for an agent, which is trained against its opponent's best mixed strategy so far, the MSNE changes in favor of the agent. We continue these iterations until the MSNE payoff of the defender and the adversary ($U^p(\sigma^p_*, \sigma^{\bar{p}}_*)$) converges. Formally, we say that the the MSNE is converged for both players *iff*

$$\forall_{p \in \{a,d\}} : U^p(\pi^p_+, \sigma^{\bar{p}}) \leq U^p(\sigma^p, \sigma^{\bar{p}}) \tag{23}$$

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

6 Evaluation

In this section, we first describe the heuristic strategies of the MTD game (Section 6.1) proposed by Prakash and Wellman [22]. Next, we discuss our implementation of the framework (Section 6.2). Finally, we present the numerical results (Section 6.3).

6.1 Baseline Heuristic Strategies

Prakash and Wellman [22] proposed a set of heuristic strategies for both the adversary and the defender. Earlier, we used these strategies as our initial policy space for the DO algorithm. In this section, we describe these heuristics.

Adversary's Heuristic Strategies

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

Defender's Heuristic Strategies

- *Uniform:* Defender reimages a server every P_D time steps, always selecting a server that is up uniformly at random.

Table 3: Payoff Table for Heuristic and Reinforcement Learning Based Strategies

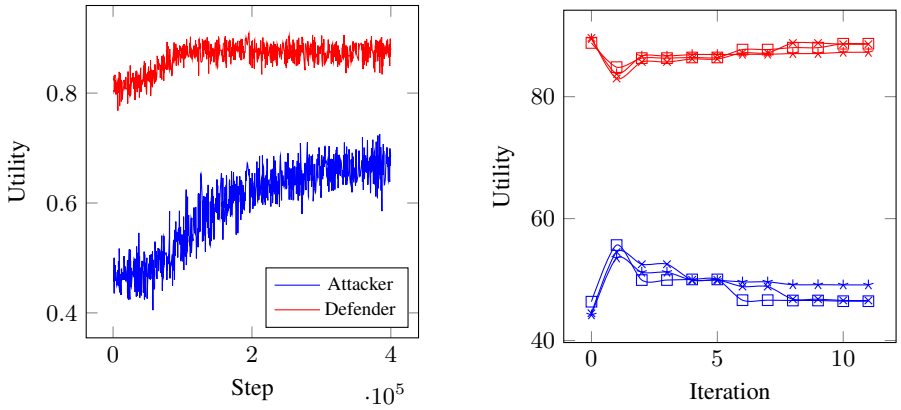
Adversary \ Defender	No-OP	ControlThreshold	PCP	Uniform	MaxProbe	Mixed-Strategy DQL
No-OP	98.20 26.89	98.20 26.89	98.20 26.89	95.83 46.03	98.20 26.89	97.47 33.23
MaxProbe	47.69 78.66	49.62 75.67	93.01 36.58	67.12 64.56	86.82 41.99	87.84 45.87
Uniform	46.74 79.08	51.58 70.97	89.48 44.43	76.23 56.83	75.21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

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

6.2 Implementation

We implemented the MAPOMDP of Section 3 as an Open AI Gym [4] environment. We used Stable-Baselines' DQN [8] as the implementation of the DQL. Stable-Baselines internally uses TensorFlow [1] as the neural network framework. For the artificial neural network as our Q approximator, we used a feed forward network with two hidden layers of size 32, and *tanh* as our activation function. The rest of parameters are described in Table 1. We implemented the remainder of our framework in Python, including the double oracle algorithm. For computation of the mixed-strategy ϵ-equilibrium of a general-sum game, we used the Gambit-Project's Global Newton implementation [16].

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



(a) Learning curve of the agents

(b) Evolution of payoff

Fig. 2: Figure 2a shows the learning curve of the players. In Figure 2b, iteration 0 shows the MSNE payoff of the heuristics while each DQN training for adversary and defender happens at odd and even iterations, respectively.

6.3 Numerical Results

For acquiring the following results, our MTD model is always instantiated using baseline parameters from Table 1, unless explicitly specified otherwise.

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

Since over the iterations of the DO algorithm, the fraction of DQL strategies in both players MSNE increase (0% vs 51% for the first trainings), and inference from a DQL policy, which requires matrix multiplications, is slower than inference from a heuristic strategy, which requires only a few operations, we can conclude that DQL policies will be more dominant over the iterations. This means that DQL policies are performing better than heuristics over the iterations.

To measure the stability of the DQL algorithm, we extracted the first DQL trainings for both players. The DQL algorithm converges to almost the same expected cumulative reward with mean and standard deviation of 0.672 and 0.021 for the adversary and 0.878 and 0.011 for the defender. Table 3, which we will discuss in detail later, shows that these policies are significantly better than the heuristics.

DO Convergence and Stability Figure 2b shows the evolution of MSNE payoff over the iterations of the DO algorithm over three experiments with baseline values of Table 1. In this figure, each training for the adversary and defender happens at odd and even iterations, respectively, while iteration 0 is the equilibrium of heuristic policies. The figure shows that the DO algorithm indeed converges with ≈ 4 trainings for each player, *i.e.,* 6 hours of training in total. Comparing multiple runs with the same configuration shows that the DO algorithm with multiple approximations (*e.g.,* approximation with deep networks, approximation on equilibrium computation) is stable since the average and standard deviation of the MSNE payoff is 47.81 and 1.77 for the adversary and 88.92 and 1.04 for the defender. For different configurations, the difference between final expected payoffs of eight DO runs is described in Figure 3.

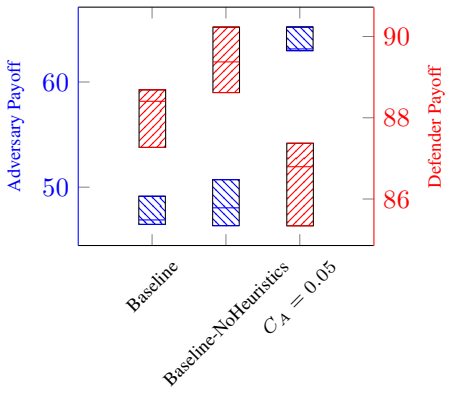


Fig. 3: Comparison of stability for different configurations. The blue and red boxes show the adversary's and defender's payoff, respectively. Each box shows the result of eight runs.

Equilibrium Selection To analyze the impact of equilibrium selection on the MSNE payoff of the game, we executed Algorithm 2, but without heuristics as initial strategy sets. Instead, the initial strategy sets are set to only NoOP adversary and NoOP defender. As we can see in Figure 3, regardless of the initial strategy sets, the resulting policies always converge to an MSNE with the same payoffs for both players. As a result, equilibrium selection is not an issue in practice since we always end with approximately the same equilibrium.

Heuristic Strategies Table 3 shows the utilities for all combinations of heuristic defender and adversary strategies with baselines parameters. The optimal strategy given only heuristics as players' strategy sets are PCP for the defender, and control threshold for the adversary. This table also compares these heuristic strategies to mixed-strategy policies computed using DO and DQL. We can see that it is optimal for both players to commit to the mixed strategy DQL, since no player can receive more utility by committing to another policy, while the opponent still commits to the DQL policy.

Resiliency to Under/Over Estimation One interesting observation of the DQL policies is their resiliency to under/over estimation of the opponent. As a showcase for when the defender underestimates the adversary, assume a defender who has trained with $C_A = 0.2$ plays with an adversary who is trained with $C_A = 0.05$. They receive 88.18 and 61.93 utility, respectively. For the defender, this utility is the same as when it correctly predicts the cost of attack (Figure 3).

7 Related Work

In this work, we used multi-agent reinforcement learning to find optimal policies for the adversary and the defender in an MTD game model. In prior work, researchers have investigated both the application of reinforcement learning in cyber-security (Section 7.2) and game-theoretic models for MTD (Section 7.1). Perhaps the most closely related work on integration of reinforcement learning and moving target defense is the work of Sengupta and Kambhampati [24]. They propose a Bayesian Stackelberg game model to MTD and solve (*i.e.,* find the optimal action policy for the defender) it using Q-Learning. One main difference between our model and their model is that they assumed that the adversary is aware of the defender's policy, while in our model, not only both players are unaware of the opponent's strategy, they might not even observe the opponent's actions. One key advantage of our model is that we consider multiple target systems while they consider a single target system with four possible states. This makes a tabular approach (Q-Learning) feasible. However, tabular approaches scale poorly to more complex systems.

7.1 Moving Target Defense

One of the main research areas in moving target defense is to model interactions between the adversaries and the defenders. In the area of game-theoretic models for moving target defense, the most closely related work is from Prakash *et al.* [22], which introduces the model that our work uses. This model can also be used for defense against DDoS attacks [31], and defense for web applications [25]. Further, in this area, researchers have proposed MTD game models based on Stackelberg games [13], Markov Games [12, 28], Markov Decision Process [32], and FlipIt game [20].

For solving a game model (*i.e.,* finding the optimal playing strategies), numerous approaches such as solving a min-max problem [13], non-linear programming [12], Bellman equation [28, 32], Bayesian belief networks [2], and reinforcement learning [9, 20] has been suggested.

7.2 Reinforcement Learning for cybersecurity

Application of machine learning—especially *deep reinforcement learning* (DRL)—for cybersecurity has gained attention recently. Nguyen *et al.* [19] surveyed current literature on applications of DRL on cybersecurity. These applications include: DRL-based security methods for cyber-physical systems, autonomous intrusion detection techniques [10], and multi-agent DRL-based game-theoretic simulations for defense strategies against cyber attacks.

For example, Malialis [14, 15] applied multi-agent deep reinforcement learning on network routers to throttle the processing rate in order to prevent *distributed denial of service* (DDoS) attacks. Bhosale *et al.* [3] proposed a cooperative multi-agent reinforcement learning for intelligent systems [7] to enable quick responses. Another example for multi-agent reinforcement learning is the fuzzy Q-Learning approach for detecting and preventing intrusions in *wireless sensor networks* (WSN) [26]. Furthermore, Tong *et al.* [29] proposed a multi-agent reinforcement learning framework for alert correlation based on double oracles.

8 Conclusion

Moving target defense tries to increase adversary's uncertainty and attack cost by dynamically changing host and network configurations. In this paper, we have proposed a multi-agent reinforcement learning approach for finding MTD strategies based on an adaptive MTD model. To improve the agents' performance in partially-observable environments, we proposed a compact memory presentation for the agents. Further, we showed that the double oracle algorithm with DQL as best-response oracle is a feasible and promising solution for finding optimal policies in general-sum adversarial games as it is stable and converges rapidly.

Acknowledgments This research was partially supported by the NSF (CAREER Grant IIS-1905558) and ARO (W911NF1910241).

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). pp. 265–283 (2016)
2. Albanese, M., Connell, W., Venkatesan, S., Cybenko, G.: Moving target defense quantification. In: Adversarial and Uncertain Reasoning for Adaptive Cyber Defense, pp. 94–111. Springer (2019)
3. Bhosale, R., Mahajan, S., Kulkarni, P.: Cooperative machine learning for intrusion detection system. International Journal of Scientific and Engineering Research **5**(1), 1780–1785 (2014)
4. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym (2016)
5. Chen, P., Xu, J., Lin, Z., Xu, D., Mao, B., Liu, P.: A practical approach for adaptive data structure layout randomization. In: 20th European Symposium on Research in Computer Security (ESORICS). pp. 69–89. Springer (2015)
6. Govindan, S., Wilson, R.: A global newton method to compute nash equilibria. Journal of Economic Theory **110**(1), 65–86 (2003)
7. Herrero, Á., Corchado, E.: Multiagent systems for network intrusion detection: A review. In: Computational Intelligence in Security for Information Systems, pp. 143–154. Springer (2009)
8. Hill, A., Raffin, A., Ernestus, M., Gleave, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y.: Stable baselines. https://github.com/hill-a/stable-baselines (2018)
9. Hu, Z., Chen, P., Zhu, M., Liu, P.: Reinforcement learning for adaptive cyber defense against zero-day attacks. In: Adversarial and Uncertain Reasoning for Adaptive Cyber Defense, pp. 54–93. Springer (2019)
10. Iannucci, S., Barba, O.D., Cardellini, V., Banicescu, I.: A performance evaluation of deep reinforcement learning for model-based intrusion response. In: 4th IEEE International Workshops on Foundations and Applications of Self* Systems (FAS* W). pp. 158–163 (2019)
11. Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Pérolat, J., Silver, D., Graepel, T.: A unified game-theoretic approach to multiagent reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 4190–4203 (2017)

12. Lei, C., Ma, D.H., Zhang, H.Q.: Optimal strategy selection for moving target defense based on Markov game. IEEE Access **5**, 156–169 (2017)
13. Li, H., Zheng, Z.: Optimal timing of moving target defense: A Stackelberg game model. arXiv preprint arXiv:1905.13293 (2019)
14. Malialis, K., Devlin, S., Kudenko, D.: Distributed reinforcement learning for adaptive and robust network intrusion response. Connection Science **27**(3), 234–252 (2015)
15. Malialis, K., Kudenko, D.: Distributed response to network intrusions using multiagent reinforcement learning. Engineering Applications of Artificial Intelligence **41**, 270–284 (2015)
16. McKelvey, R.D., McLennan, A.M., Turocy, T.L.: Gambit: Software tools for game theory (2006)
17. McMahan, H.B., Gordon, G.J., Blum, A.: Planning in the presence of cost functions controlled by an adversary. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03). pp. 536–543 (2003)
18. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
19. Nguyen, T.T., Reddi, V.J.: Deep reinforcement learning for cyber security. arXiv preprint arXiv:1906.05799 (2019)
20. Oakley, L., Oprea, A.: Playing adaptively against stealthy opponents: A reinforcement learning strategy for the flipit security game. arXiv preprint arXiv:1906.11938 (2019)
21. Oh, J., Chockalingam, V., Singh, S., Lee, H.: Control of memory, active perception, and action in Minecraft. arXiv preprint arXiv:1605.09128 (2016)
22. Prakash, A., Wellman, M.P.: Empirical game-theoretic analysis for moving target defense. In: 2nd ACM Workshop on Moving Target Defense (MTD). pp. 57–65. ACM (2015)
23. Russell, S., Norvig, P.: Artificial intelligence: a modern approach (2002)
24. Sengupta, S., Kambhampati, S.: Multi-agent reinforcement learning in bayesian stackelberg markov games for adaptive moving target defense. arXiv preprint arXiv:2007.10457 (2020)
25. Sengupta, S., Vadlamudi, S.G., Kambhampati, S., Doupé, A., Zhao, Z., Taguinod, M., Ahn, G.J.: A game theoretic approach to strategy generation for moving target defense in web applications. In: 16th Conference on Autonomous Agents and Multiagent Systems. pp. 178–186 (2017)
26. Shamshirband, S., Patel, A., Anuar, N.B., Kiah, M.L.M., Abraham, A.: Cooperative game theoretic approach using fuzzy q-learning for detecting and preventing intrusions in wireless sensor networks. Engineering Applications of Artificial Intelligence **32**, 228–241 (2014)
27. Shoham, Y., Leyton-Brown, K.: Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press (2008)
28. Tan, J.l., Lei, C., Zhang, H.q., Cheng, Y.q.: Optimal strategy selection approach to moving target defense based on markov robust game. Computers & Security **8**(5), 63–76 (2019)
29. Tong, L., Laszka, A., Yan, C., Zhang, N., Vorobeychik, Y.: Finding needles in a moving haystack: Prioritizing alerts with adversarial reinforcement learning. arXiv preprint arXiv:1906.08805 (2019)
30. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4), 279–292 (1992)
31. Wright, M., Venkatesan, S., Albanese, M., Wellman, M.P.: Moving target defense against DDoS attacks: An empirical game-theoretic analysis. In: 3rd ACM Workshop on Moving Target Defense (MTD). pp. 93–104. ACM (2016)
32. Zheng, J., Namin, A.S.: Markov decision process to enforce moving target defence policies. arXiv preprint arXiv:1905.09222 (2019)