Deep Reinforcement Learning based Adaptive Moving Target Defense

Preprint · November 2019

CITATIONS
0

READS
820

3 authors:

Taha Eghtesad
University of Houston
8 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Yevgeniy Vorobeychik
Vanderbilt University
278 PUBLICATIONS **3,084** CITATIONS
SEE PROFILE

Aron Laszka
Pennsylvania State University
158 PUBLICATIONS **1,916** CITATIONS
SEE PROFILE

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 77054, 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. Based on an established model of adaptive MTD, we propose a multi-agent reinforcement learning framework based on the double oracle algorithm to solve the game. In the experiments, we show 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 [10]. This has two main limitations. First, it can be very time consuming since

there are constraints on data locations, physical connectivity of servers cannot be easily changed, and resources are limited. Second, it is difficult to capture the trade-off between security and efficiency [5].

In light of this, it is crucial to provide automated approaches for deploying MTDs, which maximize security benefits for the protected assets. This requires a design model that reflects multiple aspects of the MTD environment [15,32,23,2]. Further, we need a decision making algorithm for the model to select which technique to deploy and where to deploy it [28]. Finding optimal strategies for deployment of MTDs are computationally challenging. For example, the adversary can adapt to MTD deployments, or the state-action space of the environment can be huge even for trivial number of MTD configurations or in-control assets.

Recently, many research efforts have applied *Independent Reinforcement Learning* (InRL) techniques to find the best action policies in known or unknown decision making environments, such as cybersecurity. In InRL, an agent learns to make the best decision by continuously interacting with its unknown environment. In general, traditional reinforcement learning techniques use tabular approaches to store estimated rewards (e.g., Q-Learning) [11]. 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 [20].

The naive approach to multi-agent reinforcement learning (MARL) is to use InRL where one player treats the opponent's action as part of its localized environment [6]. However, two problems arise here: 1) convergence guarantees are lost since these localize environments are non-stationary and non-Markovian [13]; and 2) These policies can not generalize well since they overfit to the opponent's policies [12].

Contributions We create 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. We present a multi-agent deep reinforcement learning approach to solve the 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 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 algorithms.
- We evaluate our approach 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 introduce a multi-agent partially-observable Markov decision process for MTD which is used as

the basis of the MARL. In Section 3, we describe the preliminaries such as the InRL (Section 3.1), and one of InRL family algorithms, *ie.,* Deep Q Learning (Section 3.2). 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 discuss the solution overview (Section 5.1), discuss the challenges faced when solving the game (Section 5.2), and propose our framework to solve this game (Section 5.3). 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 and outline of directions for future work.

2 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 [23]. In this adversarial model, there are two players, a defender and an adversary, 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.

2.1 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.

2.2 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

- $\rho \in \mathbb{Z}^*$ represents the number of probes lunched against server i since the last reimage,
- $\chi \in \{adv, def\}$ represents the player controlling the server, and
- $\upsilon \in \{up\} \cup \mathbb{Z}^*$ represents if the server is online (i.e., up) or if it is offline (i.e., down) with 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	knowledge gain of each probe	0.05
C_A	attack (*probe*) cost	0.20
θ^p_{sl}	slope of reward function	5
θ^p_{th}	steep point threshold of reward function	0.2
w^p	weighting of reward for having servers up and in control of adversary / defender	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

2.3 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

$$1 - e^{-\alpha \cdot (\rho + 1)} \tag{1}$$

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 can understand 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).

2.4 Rewards

Prakash and Wellman [23] 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, as described by Equation 2 and Equation 3, 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 in control servers yields a higher utility.

$$u^p(n_c^p, n_d) = w^p \cdot f\left(\frac{n_c^p}{M}, \theta^p\right) + (1 - w^p) \cdot f\left(\frac{n_c^p + n_d}{M}, \theta^p\right) \tag{2}$$

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 θ:

$$f(x, \theta) = \frac{1}{\mathrm{e}^{-\theta_{sl} \cdot (x - \theta_{th})}} \tag{3}$$

where θ_{sl} and θ_{th} control the slope and position of the sigmoid's steep point, respectively.

Reward weight (w^p) specifies the goal of each player. As described by Prakash and Wellman [23], 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 control. Or in *disrupt / availability*, which is the most interesting case, the defender gains reward by having the servers up and in control, while the adversary gains reward by bringing down the servers or having them in 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, cost of reimaging a server comes from not getting reward for the times that it is "down." However, 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:

$$r_\tau^a = \begin{cases} u^a(n_c^a, n_d) - C_A & \text{adversary probed a server at } \tau \\ u^a(n_c^a, n_d) & \text{adversary did nothing} \end{cases} \tag{4}$$

$$r_\tau^d = u_\tau^d \tag{5}$$

2.5 Observations

A key aspect of the model is the players' uncertainty regarding their state. The defender does not know which servers have been compromised by the adversary. Also,

the defender observes each probe 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 the status of all servers (i.e., whether the server is up or down, or 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 which servers are compromised.

Observation of a player p at time τ is defined as a vector of tuples O_i^p where O_i^p corresponds to observation of p of server i.

$$O_\tau^p = \langle O_{1,\tau}^p, O_{2,\tau}^p, \cdots, O_{M,\tau}^p \rangle \tag{6}$$

The adversary knows which servers are compromised and knows how many attacks it has initiated on each server. Also, if the server is down, the adversary can estimate the time that the server is up again. The observation of a server i for adversary at time τ is defined as a tuple $O_{i,\tau}^a$:

$$\forall_{0 \leq i < M} : \qquad O_{i,\tau}^a = \langle \textit{status}, \textit{time\_to\_up}, \textit{progress}, \textit{control} \rangle \tag{7}$$

where $\textit{status} \in \{1, 0\}$ which shows the server is up or down, and $\textit{control} \in \{1, 0\}$ shows that the adversary controls that server or not, respectively.

Observation vector of the defender is almost the same as the adversary. The only difference is that the defender does not know who controls the servers, and only has a estimation on the number of probes (where ν is not 0):

$$\forall_{0 \leq i < M} : \qquad O_{i,\tau}^d = \langle \textit{status}, \textit{time\_to\_up}, \textit{progress} \rangle \tag{8}$$

To overcome challenges regarding the defender which are described in Section 5.2, we included two more attributes to have some form of memory for the defender. These two attributes are described in Section 5.2.

3 Preliminaries

In this section, we describe the family of reinforcement learning algorithms (Section 3.1), and one algorithm in this family, namely the Deep Q-Learning (Section 3.2).

3.1 Independent Reinforcement Learning

One of the main approaches for finding a decision making policy is the *Independent Reinforcement Learning* (InRL) which focuses on interactions of a single agent and the environment, in order to maximize the agent's gain (presented as rewards or utilities) from the environment. Figure 1 shows the interactions between different components

of InRL. Further, the InRL algorithm is described in Algorithm 1. A basic InRL environment is a *Partially-Observable Markov Decision Process* (POMDP), which can be represented as a tuple:

$$POMDP = \langle \mathbb{S}, \mathbb{A}, \mathbb{T}, R, \mathbb{O} \rangle. \tag{9}$$

where $\mathbb{S}$ is the set of all possible states in the environment (described in Section 2.2), $\mathbb{A}$ is the set of all possible actions by the agent, $\mathbb{T}$ is the set of stochastic transition rules (Section 2.2 and Section 2.3), R is the immediate reward of a state transition (Section 2.4), and $\mathbb{O}$ is the set of observation rules of the agent (Section 2.5).

The training is done in iterations called *epoch*s. During each epoch, the training algorithm performs two operations on the environment: (1) resetting the environment to the starting state. In return, the environment provides the agent with the initial observation; (2) for each step, the agent decides on an action to take which updates the state of the environment based on transition rules.

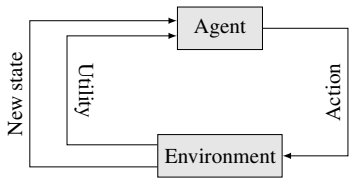


Fig. 1: Independent reinforcement learning.

Each *epoch* of training is finished and the environment is forced to reset when the number of steps taken in current epoch reaches T. It is possible to complete the training without having epochs, however, this condition ensures that 1) the majority of the action/observation space is explored, 2) the training agent is not stuck in a locally optimal state.

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:

$$e = \langle O_\tau, a_\tau, O_{\tau+1}, r_\tau \rangle \tag{10}$$

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

$$\pi(O_\tau) \mapsto a \tag{11}$$

$$\text{which maximizes } U^*_\tau = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+\tau} \middle| \pi\right] \tag{12}$$

where $\gamma \in [0, 1)$, the *"discount factor,"* prioritizes the rewards claimed at the current time step over the future rewards. When $\gamma = 0$, the player only cares about the current rewards, and when $\gamma = 1$, the player cares about all future rewards equally.

Reinforcement learning aims to maximize the received utility of the agent (U_*) by trial and error: interacting with the environment (randomly, heuristics, or referring to the

experiences 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 actions in a state. (2) *Exploration*: to find optimal actions which yields to higher utility by doing random action and exploring the Action/Observation space. One of approaches for deciding on doing exploration or exploitation is the ϵ-greedy where in each step the agent explores with probability ϵ, or take the current optimal action with probability $1 - \epsilon$.

Algorithm 1: InRL

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

3.2 Deep-Q-Network Learning

Q-Learning uses a Q function to estimate the expected future utilities of an action in an observation state (Equation 12):

$$Q(O_\tau, a_\tau) = U^*_\tau \tag{13}$$

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

$$\begin{aligned} Q(O_\tau, a_\tau) = (1 - \alpha) \cdot Q(O_\tau, a_\tau) \\ +\alpha \cdot \underbrace{(r_\tau + \gamma \cdot \max_{a'} Q(O_{\tau+1}, a'))}_{\text{TD Target}} \end{aligned} \tag{14}$$

where α 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, *ie.,* the difference between the predicted Q value, and the actual expected utility (U^*).

Mnih *et al.* [20] show that it is possible to use *multi layer perceptrons* (MLP) as approximators for the Q function. This is useful since neural networks can generalize the similarities between the observation states and Q values. When using the MLP as Q approximator with parameters θ, we define a *mean squared error* (MSE) loss function on a batch X of experiences:

$$L_\theta = \frac{1}{|X|} \sum_{i}^{X} (q_\tau - Q(O_\tau, a_\tau|\theta))^2 \tag{15}$$

where q_τ is the *TD Target* of Equation 14 and the optimal action is $\text{argmax}_{a'}\, Q(O_\tau, a')$:

$$q_\tau = r_\tau + \gamma Q(O_{\tau+1}, \text{argmax}_{a'}\, Q(O_\tau, a'))|\theta) \tag{16}$$

We can minimize the loss (L_θ) with gradient descent and learning rate α.

4 Problem Formulation

In Section 2, we build an MTD model using a MAPOMDP. In this section, based on this model, we design an adversarial game between the adversary (denoted by $p = 1$ or a) and the defender (denoted by $p = 0$ or d). In this setting, we assume that each player chooses a strategy to play, where each strategy is a policy function that given the current observation of the environment, returns an action to be taken. As we assumed that each playing strategy is a policy function, in 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 this player. We denote the set of pure strategy space for player p in this adversarial game as Π^p.

When players are following pure policies $\pi^a \in \Pi^a$ and $\pi^d \in \Pi^d$, their expected utility can be expressed as sum of discounted future rewards with discount factor of γ. Formally:

$$\forall_{p\in\{1,0\}} : U^p(\pi^p, \pi^{\bar{p}}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t^p \middle| \pi^p, \pi^{\bar{p}}\right] \tag{17}$$

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 denote Σ^p as the strategy space of player p. The utility profile 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:

$$\forall_{p \in \{1,0\}} : \qquad U^p(\sigma^p, \sigma^{\bar{p}}) = \sum_{\pi^p \in \Pi^p} \sum_{\pi^{\bar{p}} \in \Pi^{\bar{p}}} \sigma^p(\pi^p) \cdot \sigma^{\bar{p}}(\pi^{\bar{p}}) \cdot U^p(\pi^p, \pi^{\bar{p}}) \tag{18}$$

Note that the notation of utility profile of players when they follow pure strategies is overloaded to also support the use of mixed strategies.

4.3 Solution Concept

The aim of both players is to maximize their utility. As we are considering a strong adversary and defender, we can assume that they always pick the strategy which maximizes their own utility. A *"best response"* mixed strategy $\sigma^p_*(\sigma^{\bar{p}})$ ensures maximum utility for the player choosing it (p) while the opponent is using mixed strategy $\sigma^{\bar{p}}$. In other words, the agent choosing a best response mixed strategy σ^p_* cannot gain more utility without having the opponent changing its strategy. When the player p is using a mixed strategy σ^P, the opponent's best response is:

$$\sigma^p_*(\sigma^{\bar{p}}) = \text{argmax}_{\sigma^p} U^p(\sigma^p, \sigma^{\bar{p}}) \tag{19}$$

We optimize one player's strategy assuming that the opponent always uses a best response strategy to the player's strategy, $\sigma^p = \sigma^p_*(\sigma^{\bar{p}})$. This formulation of a general-sum game is in fact equivalent to finding a *mixed-strategy Nash equilibrium* (MSNE) of players' policy space Π^a and Π^d. In other words, a combination of strategies $(\sigma^p_*, \sigma^{\bar{p}}_*)$ is MSNE, *iff*:

$$\forall_{p \in \{1,0\}} \forall_{\sigma^p \in \Sigma^p} : U^p(\sigma^p_*, \sigma^{\bar{p}}_*) \geq U^a(\sigma^p, \sigma^{\bar{p}}_*) \tag{20}$$

That is, neither agent can increase its expected utility without having the opponent changing its strategy. Therefore, our solution must find the MSNE of the MTD game where Π^a and Π^d are the policy space of the players.

5 Framework

In Section 2, we described a moving target defense model as a *Multi-Agent Partially-Observable Markov Decision Process* (MAPOMDP). In Section 4, we proposed a game based on this model and concluded that finding an optimal action policy for the adversary and the defender is equivalent to finding the MSNE of the game. In this section, we propose a computational approach and build a framework atop of the double oracle (Section 5.1) and DQL (Section 3.2) algorithms to find the optimal action policies for the adversary and the defender.

5.1 Solution Overview

The iterative *Double Oracle* (DO) algorithm[19], solves the MSNE of a game given an arbitrary subset of policy space for each player ($\Pi_0^p \subset \Pi^p$). We denote the explored policy space of player p until iteration τ of the DO algorithm by Π_τ^p. In each iteration τ of the DO algorithm, for each player, a pure strategy best response (π^*) to the MSNE of the opponent is calculated using a *"best response oracle"* and added to the strategy sets. Formally, in each iteration:

$$\forall_{p\in\{1,0\}} : \Pi_{\tau+1}^p \leftarrow \Pi_\tau^p \cup \{\pi_*^p(\sigma_{*,\tau}^{\bar{p}})\} \tag{21}$$

where $\sigma_{*,\tau}^p$ is the MSNE of the player p given the strategy sets Π_τ^p. The DO algorithm guarantees [19] the convergence of the MSNE of these strategy sets to the actual MSNE of the game as long as the policy space for both players are finite. However, as the policy space of the players are huge (still finite), enumeration of the policy space in search of the best response is infeasible.

For each player, we can use a tabular based InRL such as Q-Learning[30] algorithm as a best response oracle to find a best response pure strategy to the opponent's MSNE. Since the opponent's strategy is fixed, the agent learning through reinforcement learning can treat the opponent's actions as part of its localized environment without overfitting to the opponent's policies.

5.2 Challenges

Solving the MAPOMDP model of Section 2 with the DO algorithm is not straightforward. In the following subsections, 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 states; however, this poses computational challenges since the size of the effective state space for the policy explodes.

Since partial observability poses a challenge for the defender, we let the defender's policy use information from preceding states. 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 and (b) the amount of time since the last observed probe. So, the actual state presentation of the defender will be:

$$\begin{aligned} O_i^d = \langle & \mathit{status}, \mathit{time\_to\_up}, \mathit{progress}, \\ & \mathit{time\_since\_last\_probe}, \\ & \mathit{time\_since\_last\_reimage} \rangle \end{aligned} \tag{22}$$

Similarly, the adversary should probe the servers which where probed in many steps in the past to make sure that its progress was not reset. So, we add the amount of time since the last probe to its observation state.

$$O_i^a = \langle status, time\_to\_up, progress, control, \\ time\_since\_last\_probe \rangle \tag{23}$$

Complexity of MSNE Computation In zero-sum games, computation of MSNE can be done using linear programming which has a polynomial time complexity. However, in general-sum games, the solving algorithms is PPAD-complete[27] which makes it infeasible for solving 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[7].

Equilibrium Selection Typically, the DO algorithm is used with zero-sum games which all the equilibria of the game yield the same payoff for each player. However, in general-sum games, there may exist multiple equilibria with significantly different payoffs. The DO algorithm in general-sum games only converges to one of these equilibria. The exact equilibrium the DO algorithm converges to depends on the initial policy space of the players and the exact output of the best response oracle. However, in our experiments (Section 6.3), we show that in practice, in adversarial games, this problem is not significant, *ie.,* all equilibria produce almost the same results (Table 4) indeferent to the initial policy space.

Model Complexity Due to the complexity of our MAPOMDP model, computation of best response using tabular InRL approaches is computationally infeasible. In fact, even representation of a single policy as keeping the best action for all the observation states is infeasible. Further, tabular approaches fail to generalize the relations between observations and actions. Thus, the action/observation space need to be enumerated many times in order for the algorithm to produce correct results.

To address this challenge, we can use computationally feasible *"approximate best responses"* to produce an approximate best response pure strategy(π_+) instead of true best responses. Lanctot *et al.* [12] 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 indeed converges with six 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 can use higher temporal discount factors (e.g., $\gamma = 0.99$). However, such values can pose challenges for deep reinforcement learning since it will be much more difficult to converge.

5.3 Solution Approach

Prakash and Wellman [23] 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' policy spaces, their MSNE is not necessarily the MSNE of the Σ^a and Σ^d. In Section 5.1, we show how we can find the MSNE of the game, given a subset of policy space for each agent. In this section, 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 with heuristic policies (Section 6.1). From this stage, we proceed in iterations. In each iteration, first, we compute MSNE of the game restricted to the current strategy sets Π^a and Π^d, and 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 set of policies of the defender ($\Pi^d \leftarrow \Pi^d \cup \{\pi^d_+\}$).

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 set of policies for the adversary ($\Pi^a \leftarrow \Pi^a \cup \{\pi^a_+\}$).

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 for agent as *Independent Reinforcement Learning* (InRL). Each iteration of InRL, defined as `train()` in Algorithm 2, receives a total number of steps T of training, and initial parameters θ. Moreover, we denote the InRL environment for the player p when the opponent plays with a mixed strategy $\sigma^{\bar{p}}$ as $env^p[\sigma^{\bar{p}}]$.

As we are dealing with discrete action/observation spaces in the MTD model, DQL [20] 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 (*ie.,* exploration), or follows its current policy. The opponent whose strategy is fixed to a mixed strategy $\sigma^{\bar{p}}$ refers to a pure strategy $\pi^{\bar{p}} \in \Pi^{\bar{p}} i$ with probability distribution $\sigma^{\bar{p}}$, and follows that policy.

The MSNE payoff evolves over these iterations whenever we add a new policy for an agent, which is trained against the best mixed strategy of the opponent, 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 for both players the MSNE is converged *iff*:

$$\forall_{p \in \{1,0\}} : U^p(\pi^p_+, \sigma^{\bar{p}}) \leq U^p(\sigma^p, \sigma^{\bar{p}}) \tag{24}$$

where (σ^p) is the previous MSNE and π_+^p is the approximate best response found for player p in opposing to the opponent's MSNE at the current iteration. This convergence means that neither the adversary nor defender could perform better by introducing new policies.

6 Evaluation

In this section, first we describe the heuristic strategies of the MTD game (Section 6.1). Next, we describe our implementation of the framework (Section 6.2). Finally, we discuss the numerical results (Section 6.3).

6.1 Baseline Heuristic Strategies

Prakash and Wellman [23] 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.
- *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*

$$\mathbb{E}[n_c^d] \leq M \cdot \tau \tag{25}$$

and the last reimage was at least P_D time steps ago.

– *No-Op:* Defender never reimages any servers.

Table 3 shows the expected rewards for all combinations of heuristic defender and adversary strategies in an environment whose parameters are described in Table 1. Also, in this table, we compare our mixed strategy policies computed by DQL to these strategies.

Table 3: Payoff Table for Heuristic Adversary vs. Defender. For comparison, MSNE strategy payoffs are shown.

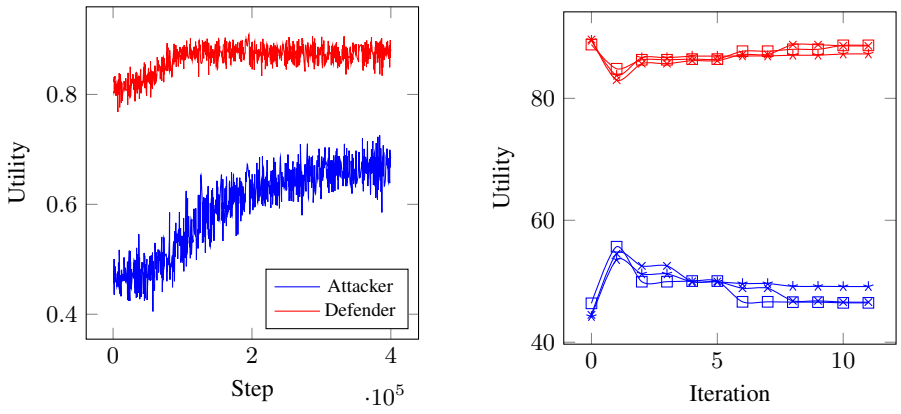
Adversary \ Defender	No-OP	ControlThreshold	PCP	Uniform	MaxProbe	Mixed Strategy DQL
No-OP	98.20 / 26.89	98.20 / 26.89	98.20 / 26.89	95.83 / 46.03	98.20 / 26.89	97.47 / 33.23
MaxProbe	47.69 / 78.66	49.62 / 75.67	93.01 / 36.58	67.12 / 64.56	86.82 / 41.99	87.84 / 45.87
Uniform	46.74 / 79.08	51.58 / 70.97	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

6.2 Implementation

We implemented the MAPOMDP of Section 2 as an Open AI Gym [4] environment. We used Stable-Baselines' DQN[9] 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 available at 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 [18].

We run the experiments on a computer cluster where each node has two 14 Cores 2.4 GHz Intel Xeon CPU and two NVIDIA P100 GPU. Each node is capable of running $\approx$ 75 steps of DQL per second, which yields to 1.5 hours per each reference to the best response oracle (*ie.,* DQL training for an adversary or a defender). Further, the DQL algorithm is not distributed, so we only use one core of the CPU. This paves the way for multiple DQLs to run at the same time. We also need to mention that, set of policies only needs to be pre-computed. While policies are in use, inference takes only milliseconds.

6.3 Numerical Results



(a) Learning curve of the agents

(b) Evolution of payoff

Fig. 2: 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.

Figure 2a shows the learning curve of the agents for their first iteration of the DO algorithm (Iteration 1 and 2), with the MTD environment whose parameters are specified in Table 1. We can see the improvement of the resulted policy compared to the heuristics (Table 3).

Figure 2b shows the evolution of MSNE payoff over the iterations of the DO algorithm with environment parameters 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 payoff equilibrium of heuristic policies. Also, this figure shows that the DO algorithm indeed converges with ≈ 4 trainings for each player, *ie.,* 6 hours of training in total. Comparing multiple runs with the same configuration (highlighted runs in Table 4), shows that the DO algorithm with multiple approximations (*eg.,* approximation with deep networks, approximation on equilibrium computation).

The converged equilibrium payoffs for the adversary and the defender for different environment parameters are shown in Table 4. These payoffs are in fact in agreement with *empirical game theoretic analysis* EGTA done by Prakash and Wellman [23].

To analyse the impact of equilibrium selection on the MSNE payoff of the game, we executed Algorithm 2 with the same environment parameters of gray rows of Table 4, but without heuristics as initial policies. The initial policies are set to NoOP adversary and NoOP defender. As we can see in Table 4, the payoffs for the adversary and defender are almost identical in both cases (using heuristics as initial policy space and NoOP as initial policy space for players).

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 cybersecurity (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 done by Sengupta and Kambhampati[24]. They propose a Bayesian Stackelberg game model to MTD and solve (*ie.,* finding the optimal action policy for the defender) it using Q-Learning. However, their approach is not applicable in our model since 1) Our model is not a Stackleberg game; neither the adversary nor the defender observes the actions taken by the opponent, and 2) Their proposed model has less complexity, making table based Q-Learning feasible.

Table 4: Final Equilibrium Payoff of MTD environment with different configurations

M	utenv	θ^p_{th}	C_A	Δ	α	ν	Adversary	Defender
Heuristics as Initial Policy Space								
10	2	0.2	0.2	7	0.05	0	46.46	88.69
10	2	0.2	0.2	7	0.05	0	49.13	87.27
10	2	0.2	0.05	7	0.05	0	63.26	86.31
10	2	0.2	0.1	7	0.05	0	57.83	87.21
10	2	0.2	0.05	7	0.05	0	63.03	87.37
10	2	0.2	0.1	7	0.05	0	58.08	86.55
10	2	0.2	0.2	7	0.05	0	46.58	88.60
10	0	0.2	0.05	7	0.05	0	35.05	92.07
10	0	0.5	0.05	7	0.05	0	8.98	77.27
10	0	0.8	0.05	7	0.05	0	1.80	73.10
10	0	0.2	0.1	7	0.05	0	30.87	90.81
10	0	0.5	0.1	7	0.05	0	7.59	92.41
10	0	0.8	0.1	7	0.05	0	1.80	73.10
10	0	0.2	0.2	7	0.05	0	26.89	98.20
10	0	0.2	0.1	3	0.05	0	30.64	94.46
10	1	0.2	0.1	7	0.05	0	26.89	98.20
10	1	0.2	0.05	7	0.05	0	26.89	98.20
10	1	0.2	0.2	7	0.05	0	26.89	98.20
10	1	0.2	0.1	3	0.05	0	26.89	98.20
10	3	0.2	0.1	7	0.05	0	74.16	97.83
10	3	0.2	0.05	7	0.05	0	79.31	97.90
10	3	0.2	0.2	7	0.05	0	65.21	97.66
NoOP as Initial Policy Space								
10	2	0.2	0.2	7	0.05	0	50.72	90.23
10	2	0.2	0.2	7	0.05	0	46.34	89.83
10	2	0.2	0.2	7	0.05	0	47.68	88.92

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.* [23], 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[15], Markov Games[14,28], Markov Decision Process[32], and FlipIt game [22].

For solving a game model (*ie.,* finding the optimal playing strategies), numerous approaches such as solving a min-max problem [15], non-linear programming [14], Bellman equation [28,32], Bayesian belief networks[2], and reinforcement learning[6,10,22] has been suggested.

7.2 Reinforcement Learning for Cyber Security

Usage of machine learning and especially *deep reinforcement learning* (DRL) for cyber security has gained attention recently. Nguyen *et al.* [21] surveyed current literature on applications of DRL on cyber security. These applications include: DRL-based security methods for cyber-physical systems, autonomous intrusion detection techniques, and

multi-agent DRL-based game theory simulations for defense strategies against cyber attacks.

For example, Malialis [16,17] 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 [8] 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) by Shamshirband *et al.* [26]. Furthermore, Tong *et al.* [29] proposed a multi-agent reinforcement learning framework for alert correlation based on double oracles.

Iannucci *et al.* [11] use deep reinforcement learning to find the optimal policy in intrusion response systems. They evaluate the performance of their algorithm based on different configurations of the model, showing that it can be much faster than traditional Q-learning.

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. For improvement of the performance of agents in partially observable environments, we proposed a compact memory presentation for the agents. Further, we show that the double oracle algorithm with DQL as best-response oracle is a feasible and promising solution for finding the optimal actions in general-sum adversarial games as it is stable.

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} 16). 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. Eghtesad, T., Vorobeychik, Y., Laszka, A.: Deep reinforcement learning based adaptive moving target defense. arXiv preprint arXiv:1911.11972 (2019)
7. Govindan, S., Wilson, R.: A global newton method to compute nash equilibria. Journal of Economic Theory **110**(1), 65–86 (2003)

8. 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)
9. 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)
10. 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)
11. 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)
12. 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)
13. Laurent, G.J., Matignon, L., Fort-Piat, L., et al.: The world of independent learners is not markovian. International Journal of Knowledge-based and Intelligent Engineering Systems **15**(1), 55–64 (2011)
14. 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)
15. Li, H., Zheng, Z.: Optimal timing of moving target defense: A Stackelberg game model. arXiv preprint arXiv:1905.13293 (2019)
16. Malialis, K., Devlin, S., Kudenko, D.: Distributed reinforcement learning for adaptive and robust network intrusion response. Connection Science **27**(3), 234–252 (2015)
17. Malialis, K., Kudenko, D.: Distributed response to network intrusions using multiagent reinforcement learning. Engineering Applications of Artificial Intelligence **41**, 270–284 (2015)
18. McKelvey, R.D., McLennan, A.M., Turocy, T.L.: Gambit: Software tools for game theory (2006)
19. 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)
20. 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), http://arxiv.org/abs/1312.5602
21. Nguyen, T.T., Reddi, V.J.: Deep reinforcement learning for cyber security. arXiv preprint arXiv:1906.05799 (2019)
22. Oakley, L., Oprea, A.: Playing adaptively against stealthy opponents: A reinforcement learning strategy for the flipit security game. arXiv preprint arXiv:1906.11938 (2019)
23. 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)
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)