Deep Reinforcement Learning-Based Defense Strategy Selection

Axel Charpentier, Nora Boulahia Cuppens, Frédéric Cuppens, Reda Yaich

Deep Reinforcement Learning-Based Defense Strategy Selection

Axel Charpentier
axel.charpentier@polymtl.ca
Polytechnique Montréal
Montréal, Canada

Nora Boulahia-Cuppens
nora.boulahia-cuppens@polymtl.ca
Polytechnique Montréal
Montréal, Canada

Frédéric Cuppens
frederic.cuppens@polymtl.ca
Polytechnique Montréal
Montréal, Canada

Reda Yaich
reda.yaich@irt-systemx.fr
IRT SystemX
Palaiseau, France

ABSTRACT

Deception and Moving Target Defense techniques are two types of approaches that aim to increase the cost of the attacks by providing false information or uncertainty to the attacker's perception . Given the growing number of these strategies and the fact that they are not all effective against the same types of attacks, it is essential to know how to select the best one to use depending on the environment and the attacker. We therefore propose a model of attacker/defender confrontation in a computer system that takes into account the asymmetry of the players' perceptions. To simulate attacks on our model, a basic attacker scenario based on the main phases of the Cyber Kill Chain is proposed. Analytically determining an optimal solution is difficult due to the model's complexity. Moreover, because of the large number of possible states in the model, Deep Q-Learning algorithm is used to train a defensive agent to choose the best defensive strategy according to the observed attacker's actions.

CCS CONCEPTS

• **Security and privacy** → *Network security*; *Systems security*; • **Computing methodologies** → *Machine learning*; *Modeling and simulation.*

KEYWORDS

Moving Target Defense, Deception,Deep Reinforcement Learning

ACM Reference Format:
Axel Charpentier, Nora Boulahia-Cuppens, Frédéric Cuppens, and Reda Yaich. 2022. Deep Reinforcement Learning-Based Defense Strategy Selection. In *The 17th International Conference on Availability, Reliability and Security (ARES 2022), August 23–26, 2022, Vienna, Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3538969.3543789

1 INTRODUCTION

It is widely acknowledged that in cybersecurity, there is an asymmetry between the attacker and the defender. Indeed, the attacker chooses when and how they will try to penetrate or compromise a network or a machine. The attacker can therefore collect information about their target and come back later to compromise it, while it is very difficult for the defender to assess the threats they face.

To reduce this advantage of the attacker, several approaches have been proposed. The first existed long before computer systems and is at the basis of the warfare strategy. It is called deception. It consists in preventing the attacker from obtaining real information about a system by providing them false information in order to make their reconnaissance phase inefficient and thus reduce the probability of successful attacks. This can be done by several means such as perturbation, obfuscation or the use of honeypots [22]. Today, deception strategies have developed a lot. Some surveys list and classify existing deception techniques and strategies [12, 13, 20].

Another approach to reduce the attacker's advantage is Moving Target Defense (MTD). The objective of MTD is to eliminate the attacker's time advantage due to the static nature of network infrastructures. This consists of changing the attack surface as well as the exploration surface to make any attack more difficult. This approach is sometimes considered as a deception strategy. However, while the objective of deception strategies is to provide false information to the attacker, the objective of MTD strategies is to prevent the use of previously obtained information, i.e., to make the information previously obtained by the attacker no longer valid. MTD strategies have been the subject of many research articles. Previous surveys have examined existing MTD strategies, each with its own criteria for analysis and classification [6, 26, 32].

There are many deception and MTD techniques in the literature. However, each of them is not effective against the same types of attacks and permanently maintaining a deception strategy has a cost. It is therefore necessary to be able to choose the most appropriate strategy according to the environment and actions of the attacker.

To address this issue, models for the interactions between the attacker and the defence in a cyber environment have been proposed. A large part of these models is based on game theory (See Section 2). This enables to obtain some theoretical results such as mixed-strategy or pure-strategy Nash equilibrium of the game. However, in general, these models are too basic and do not allow to represent well the complexity of an attacker/defender confrontation, in particular by considering the perception of the different players and the multiplicity of possible states for a computer system.

However, Reinforcement Learning (RL) and more precisely Deep Reinforcement Learning (DRL) has emerged to enable the training of agents in complex environments. This allows for agents to obtain

ARES 2022, August 23–26, 2022, Vienna, Austria

ACM ISBN 978-1-4503-9670-7/22/08…$15.00
https://doi.org/10.1145/3538969.3543789

approximations to optimal solutions in models that a formal resolution can't obtain because of their complexity. Moreover, model-free RL algorithms allow to obtain solutions without using the transition probabilities associated with a Markov Decision Process (MDP). DRL also makes it possible to find solutions in models where the number of states or the space of actions can be very large.

This article proposes solutions to the issues outlined above. We summarize our contributions of this work as follows:

- An attacker-defender model considering the asymmetry of the players' perception and the difference in kind between MTD and deception strategies.
- An attack scenario integrating Cyber Kill Chain (CKC) stages and based on the attacker's perception.
- Use Deep Q-Learning in the model to optimize the choice of defense strategy against an attacker
- Experiments to measure the performance of the defense strategy in the model

The remainder of the paper is organized as follows. In Section 2, we discuss related works about modeling a confrontation between defender and attacker in a computer system along with the strategy selection of each player. In Section 3, our attacker/defender model considering the perception of each player is presented. We describe how the confrontation game works. The attacker's scenario is then introduced in Section 4. In Section 5, we discuss the use of a DRL algorithm to train agents on our model to find an optimal defender strategy. In Section 6, we provide the data required for the experiments, the results obtained and an analysis of them.

2 RELATED WORKS

2.1 Game Modeling

To model deception, the perception of the different players must be considered. A common approach is to use game theory to model the interactions between players. There are several types of games to achieve this. We present the most common types of games and some of their uses in the context of protecting a computer system using deception.

A first type of game mentioned in the literature is the Stackelberg Game. This is a type of game in which one player is the leader, often the defender, and the other player is the follower, often the attacker. The leader chooses their actions first, which is observed by the second player who plays next. Clark et al. [8] use Stackelberg Games to study a jamming defense. They model the interactions between a defender and a jammer in a two-stage game. The defender will generate a false traffic flow. The defender will seek to maximize throughput while minimizing delay and the attacker will choose the fraction of each flow to jam. The existence of a pure-strategy Stackelberg equilibria is shown by the authors. Clark et al. [7] propose to use a Stackelberg Game to analyze the interactions between an attacker and a network with decoy nodes. The attacker seeks to identify real nodes by looking at the response times of the nodes and the protocols used. The defender chooses when to randomize the IP addresses of devices in the network. This game admits a unique threshold-based Stackelberg equilibrium. Feng et al. [11] model defender-attacker interactions by combining a Stackelberg game with an MDP. At the beginning of each round, the defender chooses an MTD strategy for several periods. The attacker chooses a state to attack. An algorithm is designed to choose the best strategy under worst-case. Sengupta and Kambhampati [27] propose a Bayesian Stackelberg Game to model the interactions between an attacker who can be of different types and a system using MTD. A Q-Learning algorithm is used to find an optimal solution.

A second type of game often used to model the interaction between an attacker and a defender in a computer system is Signaling Game. This is a class of two-player games of incomplete information. A first player performs an action and transmits information to the other player with a certain cost which will be higher if the information is false. The second player does not know the nature of the information they received but chooses an action based on it. Pawlick and Zhu [21] use Signaling Games to analyze the effectiveness of HoneyPots deployments in a computer network. In particular, they show that sometimes the usefulness of the defender can increase when the attacker can detect the luring. La et al.[17] propose the use of a Signaling Game to model the interactions between a defender and an attacker in a honeypot-enabled network. This time, the attacker plays first by choosing among several types of attacks while the defender plays second and can use HoneyPots to trap the attacker. Çeker et al. [4] use Signaling Games to model the interactions between an attacker and a defender whose goal is to protect a server from Denial-of-Service (DoS) attacks while providing a service to legitimate users. The defender can disguise a normal system as a HoneyPot and vice versa. Rahman et al. [25] propose a mechanism to limit Remote Operating System Fingerprinting. They use a Signaling Game to model the interactions between the fingerprinter and its target.

Stochastic games are a type of multi-state game where the game is played as a sequence of states. In each of state, each player will choose an action which will have an impact on the new state of the system. A few articles use this type of game to study the impact of a defensive deception strategy on the perception of the attacker in different environments [1, 14]. Anwar et al. [1] study the optimal placement of honeypots in an attack graph in order to slow down or prevent an attack. They are interested in the trade-off between security cost and deception reward for the defender. Horák et al. [14] propose the analysis of an active deception against an adversary that seeks to infiltrate a computer network to exfiltrate data or cause damage. They use a one-sided partially observable Stochastic Game to study the impact of deception on the attacker's perception. Here, it is assumed that the defender is able to detect the attacker's progress while the attacker lacks information about the system and is therefore vulnerable to deception.

2.2 Strategy Selection

Once the game is modeled, the choice of player actions must be optimized. In game theory, some tools are available for this purpose like Mixed Strategy Nash Equilibrium (MSNE). This is an equilibrium in which no player can improve their expected utility if they are the only one to choose another strategy because each player chooses their optimal strategy assuming that the opponent does the same. These equilibria match with the optimal solutions when considering the decisions of other player. This requires a good knowledge of the opponent's strategies and their outcomes.

Hu et al. [15] study the selection of the best countermeasure to maximize defense payoff. This approach uses game theory and more precisely signal game theory. The perfect Bayesian equilibrium is calculated and an algorithm is proposed for the selection of the optimal defense strategy. Experiments in a small environment are done to show the efficiency of the proposed approach.

Some papers propose to find the Nash equilibrium to optimize the choice of a MTD strategy [18, 31]. [31] study the selection of the MTD strategy of a defender in a web application environment. The confrontation between the attacker and the defender is done through an incomplete information game. The Nash-Q Learning algorithm is used to find this optimal strategy and its performance is compared to other algorithms such as the Minimax-Q learning algorithm and the Naive-Q learning algorithm in this environment. The results of the experiments show the efficiency of the Nash-Q learning algorithm compared to the others. L. Cheng et al. [18] use a Markov Game to propose a model considering the multiphase and multistage nature of the confrontation.

Tan et al. [28] make a parallel between MTD transformation and change of the attack surface and the exploration surface. They propose a MTD model based on a Markov robust game model. Robust Games can reduce the dependence on prior knowledge for the optimal strategy selection present in other models. As shown by L. Cheng et al. [18], the use of such a model allows to take into account the multi-stage and multi-state nature. The existence of a robust equilibrium and an optimal solution for this model is proven under certain conditions. An algorithm is proposed to find these and simulations are done to show the efficiency of this approach.

Some articles propose models that are not strictly based on game theory. As MTD strategies are often effective against one type of attack, H. Zhang et al. [30] study the impact of using MTD strategies with one or multiple mutant elements against several attacks. An algorithm based on the Genetic Algorithm is proposed to find the most efficient combination of MTD strategies. It is shown that even in an environment with limited resources it is still important to use multiple mutant elements.

However, as a model becomes more complex, finding equilibria becomes increasingly and computationally intractable. Equilibria as Nash equilibria do not consider past behavior of players which often lead to predict future behavior. Moreover, the fact that the optimal solution found is often a mixed strategy makes its practical use difficult. DRL proposes to obtain an approximate pure strategy of the best strategy while limiting the computational cost. It also allows to find an approximation of the optimal solution in models that are not based on game theory [19].

Chai et al. [5] propose to use DRL to improve MOTAG [16], a MTD mechanism to counter Distributed DoS (DDoS) attacks. MOTAG uses proxies to transmit traffic between legitimate users and protected servers. By using these proxies, it is able to isolate the external attacker from the innocent clients while shuffling the client/proxy assignments. DQ-MOTAG provides a self-adaptive shuffle period adjustment ability based on reinforcement learning. Experiments show the efficiency of the method used.

T. Eghtesad et al. [10] propose to find an optimal MTD strategy in the model proposed by Prakash and Wellman [23]. For this purpose, a two-player general-sum game between the adversary and the defender is created. In this model, the attacker and the defender oppose each other for the control of servers, the mechanism of server compromise is simplified. A compact representation of the memory for each player is proposed and enables to act better in the partially observable environment. This paper succeeds in solving this game using a multi-agent RL framework based on the double oracle algorithm which allows to find mixed-strategy Nash equilibrium in games.

S. Wang et al. [29] study the deployment policy for deception resources and especially the position of these resources. A model is proposed to represent the attacker/defender confrontation and the attacker's strategy is provided. A Threat Penetration Graph (TPG) is used to preselect the locations of deception resources. A Q-learning algorithm is provided to find the optimal deployment policy. A real-world network environment is used to demonstrate the effectiveness of the proposed method.

The problem with games such as signaling games is that they are generated with two players who each has two possible actions. These are games with incomplete information taking into account the asymmetry of the players' perceptions but do not support multiple sources of deception as there could be in a computer system. Moreover, this type of game is not adapted to model a multi-state and multi-stage situation. Other Bayesian games take into account multistate and multistage characteristics but to our knowledge, in the literature, there is no game modeling that takes into account the different perceptions of the players and that allows to take into account both the effects of deception and the effects of MTD strategies on a computer system. Another difficulty is to find an optimal solution in a cyber environment model where the number of states can be very large. The use of Deep Reinforcement Learning will help to address this problem.

3 CONFRONTATION MODEL

In this section, we present our model of a computer system with several types of potential vulnerabilities. A single machine is modeled but we could model a network of machines in a similar way. We will come back to this in the Section 6.4. Our modeling takes into account the perception of the players and the actions allowed for each of them are real-world actions, so the attacker can launch scans and attacks on different types of vulnerabilities while the defender can deploy Deception or MTD strategies. Due to their different characteristics, these strategies will have a different effect on the environment. MTD strategies will affect the attacker's prior knowledge, while deception strategies will provide false information to the attacker's scans. Deception strategies are deployed over several time-step but are not deployed permanently because maintaining a deception strategy has a cost.

In Section 3.1, we present the main components of the model. We define the different states of the game in Section 3.2 and the observations of the two players in Section 3.3. The game process is presented in Section 3.4 and the reward system in Section 3.5. These sections are useful to understand how DRL is used in this context (see Section 5).

3.1 Environment

In this section, we define our attacker/defender model using player perception. For this, we consider a machine, an attacker and a defender. The attacker seeks to compromise the machine while the defender seeks to defend it by using MTD and deception strategies. The machine will be modeled by a vector of vulnerabilities V and by a compromise score C. For each available vulnerability i, if it is present on the machine then $V[i] = 1$ otherwise $V[i] = 0$. The compromise index C is between 0 and 1. If the machine is not compromised then C is 0. Its score can then evolve to 1 meaning that the machine is fully compromised.

3.1.1 Attacker. A first player of the model is the attacker whose goal is to compromise the machine. For this, they have several actions available: a GlobalScan, Scans or Attacks. For each vulnerability in V, present or not on the machine, a scan and an attack are available. To each of these actions is associated a DI score indicating the information damage, a DC score indicating the compromise damage and a Cost indicating the cost of this action. The compromise damage of Scans is zero while the information damage is higher for Scan actions than for Attack actions. The greater the information damage for an action against a certain vulnerability, the faster it will be for the attacker to acquire the information needed to exploit it with an attack. In contrast, if a vulnerability is not present on the machine, a large DI score for an action will make it faster to remove that vulnerability from potential vulnerabilities. In addition to these available actions, there is a Global Scan that scans all vulnerabilities at once. In return, the information obtained on each vulnerability by a GlobalScan is lower than for a singular Scan.

Other variables are associated with the attacker and allow measuring the attacker's progress in compromising the machine. We define two vectors pV and $perceivedpV$ containing for each vulnerability v in V, respectively, a score measuring the amount of real information the attacker has about this vulnerability and a score measuring the amount of information the attacker thinks they have about this vulnerability. If $perceivedpV[v] = 0.5$, the attacker has no information about the vulnerability v. The closer $perceivedpV[v]$ is to 1, the more information the attacker has about the vulnerability v and the more they think the machine has it. The closer $perceivedpV[v]$ is to 0, the more information the attacker has about the vulnerability v and the more they think the machine has not it. $perceivedpV$ and pV are not necessarily equal. If the attacker is deceived, then these two vectors may not be equal. Since pV represents a real amount of information, if the machine is vulnerable to a vulnerability v then $pV[v]$ will be between 0.5 and 1 and if it is not then $pV[v]$ will be between 0 and 0.5. The variable Ca measures the state of compromise of the machine perceived by the attacker. An assumption of our model will be $C = C_a$. This means that the deception will not be about the status of compromise but will focus on the information needed for attacks. Another variable will be the phase of the CKC in which the attacker is located. This one can take three values (0,1 or 2) because we will consider only 3 phases of the CKC to make the model simpler: Reconnaissance, Intrusion and Pivilege Escalation/Exploitation. This parameter will be updated at each step according to the values of pV,I_a and C_a. It is considered that some attacks will only be available at certain phases of the CKC.

Finally, two other variables will measure the attacker's overall knowledge of the machine. The first one is System Info Perception I_a (See Equation 1) which measures the overall amount of information perceived by the attacker about the observable vulnerabilities of the machine. This means that in CKC 0 and 1, we do not consider vulnerabilities that can only be used in CKC 2. The second is Useful System Info Perception Iu_a (See Equation 2) which measures the amount of useful information for attacks perceived by the attacker on the observable vulnerabilities of the machine.

$$I_a = \frac{2}{K} \sum_{v \in V} max(0; 0.5 - |V[v] - pV[v]|)) \quad (1)$$

$$Iu_a = \frac{2}{K} \sum_{v \in V} max(0; perceivedpV[v] - 0.5) \quad (2)$$

where K is the number of available attack strategies considered in the model.

3.1.2 Defender. The second player will be the defender. Their goal is to prevent the machine from being compromised while minimizing the cost to achieve this goal. Among its possible actions, there are MTD strategies and deception strategies. The defender can also choose not to use any actions. MTD strategies will influence the knowledge already acquired by the attacker. Indeed, if the attacker has collected information about the system and the defender successfully uses a MTD strategy then some of the information acquired by the attacker will become incorrect.

On the other hand, deception strategies do not modify the validity of the information already obtained by the attacker. They provide the attacker with erroneous information. If the attacker acquires information about the system in the steps following the use of a deception strategy, this information may be erroneous and thus deceive the attacker.

Each of the possible strategies is associated with a cost that considers the difficulty to implement it and the software or hardware resources needed to do so.

3.1.3 Model parameters. A table Eff defines for each action of the defender its effectiveness against an attacker's strategy. This value includes both the ability to prevent an attack and the ability to prevent information about the system from being obtained. We will come back to this in Section 6.1.

In our model, the defender does not necessarily detect all attacks. It detects them with the probability $P_{Detection}$ which is a parameter to be specified in the model. To simplify, it's the same for all attacks. In addition, when an attacker takes an action, the defender can detect which potential vulnerability has been targeted but is not able to distinguish a scan from an attack. This will have an effect on the defender's observations.

Deception strategies do not necessarily achieve their objectives. There is a probability of deception success (See Equation 3) depending on the attacker's information about the machine and the phase of the CKC:

$$P_{Deception} = exp(-\lambda_{Deception} \cdot (CKC + 1) \cdot (I_a + 0.3)) \quad (3)$$

Moreover, the attacks are not necessarily a success. The probability of success of an attack targeting a vulnerability on the machine against a defense strategy (See Equation 4) depends on the attacker's knowledge of the targeted vulnerability and the effectiveness of the defensive strategy against the attack.

$$P_{Attack}(A, D) = V[A] \cdot exp(-\frac{\lambda_{Attack} \cdot (Eff(A, D) + 0.3)}{max(0.01; (pV[A] - 0.5) \cdot 2)}) \quad (4)$$

where A represents an attacker's action and D represents a defender's action.

If the vulnerability is absent from the machine then the probability of success of the attack is zero. $\lambda_{Deception}$ and λ_{Attack} are parameters that will calibrate the above probabilities to make them more consistent with the real environment we want to model. Similarly, the value of certain offsets in these probabilities could also be adjusted to better reflect the values of a real situation.

3.2 States of the game

Our model uses a discrete time scale. The game is a succession of stages. At each stage, the game is characterized by its state. In a given time step t, the state of the game is :

$$s^t = \langle V, C, pV, CKC \rangle \quad (5)$$

Each of these parameters has been defined previously.

3.3 Observations

For each of the players, the attacker and the defender, we will define the observations. These are the information held by each player that can be used at each time step of the game to choose a strategy.

The state of the game for the attacker is defined by the tuple O^a:

$$O^a = \langle perceivedpV, C_a \rangle \quad (6)$$

This provides to the attacker both information about the vulnerabilities of the machine and about the state of compromise of the machine.

On the other hand, the defender has information about the different vulnerabilities that have been targeted by the attacker in the past if the attacks have been detected. They also have information about the defense strategies used in the past. The state of the machine for the defender is defined by the tuple O^d:

$$\begin{aligned} O^d = \langle & last\_attacks, last\_defenses, \\ & times\_since\_last\_attack, \\ & times\_since\_last\_defense, \\ & attack\_counts \rangle \end{aligned} \quad (7)$$

where

- $last\_attacks$ is a vector that contains for the last k time steps, if detected, the vulnerabilities targeted by the attacker in One-Hot encoding i.e. each element of the vector is 1 if the corresponding attack has been detected and 0 otherwise.
- $last\_defenses$ is a vector that contains for the last k time steps, the strategies used by the defender in One-Hot encoding.
- $times\_since\_last\_attack$ is a vector containing for each vulnerability the number of time steps elapsed since the last action (Scan or Attack) detected from the attacker on it.
- $times\_since\_last\_defense$ is a vector containing for each defense strategy the number of time steps elapsed since its last use.
- $attack\_counts$ is a vector containing for each vulnerability the total number of detected actions of the attacker on it.

k is a parameter to be defined of the model. It represents the number of time steps considered in the $last\_attacks$ and $last\_defenses$ vectors of the observations.

3.4 Course of a game step

In this section, we will explain how the game proceeds. For that, we will introduce the Information Gathering function (See Algorithm 1). It corresponds to the unfolding of an information gathering strategy of the attacker and its impact on the model. It will modify the attacker's information held on the vulnerabilities by considering the decoy strategies set up by the defender.

This function is called at different moments of the game as for example when the attacker performs a scan, a global scan or an attack that fails. Indeed, it is considered that even when an attack fails, the attacker still gets information about the targeted vulnerability.

This Information Gathering function takes as parameter the vulnerability targeted by the attacker AND μ a parameter quantifying the potential impact of the information gathering on the vulnerability information. Indeed, the larger μ is, the more pV and $perceivedpV$ will be modified by the information gathering. In the algorithms, $V[A_A]$ is short for $V[v]$ where v is the vulnerability targeted by the attack or scan A_A. It is the same for $pV[A_A]$ and $perceivedpV[A_A]$.

Algorithm 2 provides the pseudocode of the unfolding of a game timestep given the actions chosen by the defender and by the attacker. To summarize its operation, the first part is to check if the strategy of the defense is a MTD strategy. If this is the case, then pV evolves in this way. Then the nature of the attacker's action is checked. If it is an individual scan, then the Information Gathering function is used with a rather high μ, i.e. a strong potential impact of the information gathering. If it is a Global Scan, then we do the same but for all the available vulnerabilities and with lower μ. If not, we look to see if it is an attack. If it is and the attack is successful, then we update V, pV, $perceivedpV$ and C. If not, we use the Information Gathering function with a low μ.

This process will repeat itself to form an episode. An episode is a sequence of states, actions and rewards that ends in a final state. In our model, an episode ends if the machine is compromised or if the number of time steps exceeds 100 because we consider that the attacker will be dissuaded or discouraged from attacking and will stop the attack.

3.5 Reward

To optimize the defender's strategy selection at each step, we need to define a utility function U_D that specifies the defender's objective. Its objective is to make the compromise of the machine as long as possible and thus to minimize the information held by the attacker on it as well as the level of compromise of it. U_D is defined as follows:

$$U_D = w \cdot (1 - I_A) + (1 - w) \cdot (1 - C) \quad (8)$$

Algorithm 1 Model - Information gathering

Input Attacker Action A_A, μ
for each Defender Strategy D used in the last k steps **do**
  if $Eff(D, A_A) > 0$ **then**
    Compute Deception Probability
    if Deception is a Success **then**
      $newpV \leftarrow pV[A_A] - (V[A_A] - pV[A_A]) \cdot \mu \cdot Eff(D, A_A)/2$
      if $V[A_A] = 1$ **then**
        $pV[A_A] \leftarrow max(1/2; newpV)$
      else
        $pV[A_A] \leftarrow min(1/2; newpV)$
      end if
      $varppV \leftarrow (V[A_A] - perceivedpV[A_A])\mu \cdot Eff(D, A_A)/2$

      $perceivedpV[A_A] \leftarrow perceivedpV[A_A] - varppV$
    else
      $newpV \leftarrow pV[A_A] + (V[A_A] - pV[A_A]) \cdot \mu \cdot DI[A_A] \cdot 3$
      if $V[A_A] = 1$ **then**
        $pV[A_A] \leftarrow max(1/2; newpV)$
      else
        $pV[A_A] \leftarrow min(1/2; newpV)$
      end if
      $perceivedpV[A_A] \leftarrow pV[A_A]$
    end if
  end if
end for
if no efficient deception strategy is used **then**
  $pV[A_A] \leftarrow pV[A_A] + (V[A_A] - pV[A_A]) \cdot \mu \cdot DI[A_A] \cdot 3$
  $perceivedpV[A_A] \leftarrow pV[A_A]$
end if

Algorithm 2 Model - Run one step

Input Attacker Action A_A, Defender Action D_A
if D_A is MTD **then**
  for each Vulnerability v **do**
    if $Eff(D_A, v) > 0$ **then**
      $\mu \leftarrow 1/4$
      $pV[v] \leftarrow pV[v] - (pV[v] - 0.5) \cdot \mu \cdot Eff(D_A, v)$
    end if
  end for
end if
if A_A is Scan **then**
  $\mu \leftarrow 1/2$
  Run Information Gathering with A_A, and μ
end if
if A_A is GlobalScan **then**
  $\mu \leftarrow 1/4$
  for each available Scan A **do**
    Run Information Gathering with A_A, and μ
  end for
end if
if A_A is Attack **then**
  Compute Attack Success Probability
  if Attack is Successful **then**
    $C \leftarrow C + DC[A_A]$
    $V[A_A], pV[A_A], perceivedpV[A_A] \leftarrow 0$
    $C_A \leftarrow C$
  else
    $\mu \leftarrow 1/4$
    Run Information Gathering with A_A, and μ
  end if
end if

w is a numeric parameter to give more weight to the information or to the compromise in the defender's utility function.

The cost of each strategy also has its influence. The reward given to the defender at time t is given by:

$$r_D^t = U_D - Cost[D_A] \tag{9}$$

where U_D is the Defender Utility at time t and $Cost[D_A]$ is the cost of the defender action used at time t.

The defender seeks to maximize the total reward over an entire episode.

4 ATTACKER SCENARIO

Now that we have presented our Attacker/Defender model, we will present our attacker scenario. Indeed, we will train our defender to react to the attacker's actions to make the machine's compromise as long as possible. We therefore need an attack scenario for the attacker. This is based on a simplified version of the Cyber Kill Chain. Only three steps of the CKC are considered, the same as those used for the model (See Section 3.1.1): Reconnaissance, Intrusion and Exploitation or Privilege Escalation.

To compromise the machine, the attacker will have to go through each of these phases. Actions available to the attacker will depend on the phase in which they are. Indeed, for example, if the attacker is in the third phase of the CKC, then it means that they have infiltrated the machine and therefore there may be new potential vulnerabilities reachable only from inside the machine. Moving from one phase to another of the CKC requires certain prerequisites which for this example scenario can be found in the Table 1.

4.1 Attacker Strategy Selection

At each step, the attacker will have to choose an action among those available. For this purpose, we use a very common function in game theory reffered to as utility function. This function associates with each available action a utility representing the satisfaction of the player to choose this action. The attacker will then choose the action that maximizes this function.

In our case, the satisfaction depends on the objective of the attacker and thus on the phase of the CKC in which they are. Indeed, for example, in the recognition phase, it is the collection of information and therefore the scans that are promoted while in the intrusion phase it is the attacks that are promoted. To this end, some parameters of the utility function will depend on the phase of the CKC in which the attacker is (see Table 1)

The utility function is not the same depending on the type of action of the attacker (See Table 2). Each parameter influences the utility function in a different way. Indeed, the larger $w2$ is, the more information damage will be promoted in the utility and therefore

the more scans will be promoted. The larger $w3$ is, the more compromise damage will be promoted in the utility and therefore the more attacks will be promoted. The larger $w1$ is, the more the attacker will try to fill their lack of information about the global system. In this case, it is rather the Global Scan that will be promoted.

The cost of each attacker strategy also has its influence. The reward given to the attacker at time t is given by:

$$r_A^t = U_A - Cost[A_A] \tag{10}$$

where U_A is the Defender Utility at time t and $Cost[A_A]$ is the cost of the attacker action used at time t.

The attacker will choose at each step the action that maximizes the reward they can get.

5 STRATEGY SELECTION OPTIMIZATION

In this section, we will present the approach used to find an optimal policy. A policy is a function that associates with each state of the game the action that the agent will take in this state. Because of the complexity of the game, the large number of possible states and the stochastic nature of the rewards and transitions between states, it is difficult to find an optimal policy analytically. That is why we decide to use Deep Q-Learning algorithm [19].

Firstly, we define the discounted return at time t for the defender:

$$G_t = r_D^{t+1} + \gamma r_D^{t+2} + \cdots + \gamma^{T-1} r_D^T \tag{11}$$

where we consider the episodes as finished, T is the length of the episode and γ is a discount factor that balances the weights between future and current rewards.

A policy is a function that takes as input a state S and returns the probability of using each action in this state. We also define the Q-function $Q_\pi(s,a)$ which returns the value of taking action a in state s under policy pi:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \tag{12}$$

$$= \mathbb{E}_\pi[r_D^{t+1} + \gamma G_{t+1} | s_t = s, a_t = a] \tag{13}$$

The optimal state-action value fonction is:

$$Q_*(s,a) = max_\pi Q_\pi(s,a) \tag{14}$$

$$= \mathbb{E}[r_D^{t+1} + \gamma max_{a'} Q_*(s',a') | s, a, s_{t+1} = s'] \tag{15}$$

This equation is called Bellman Optimality Equation [19].

CKC Phases	Prerequisites	Weights
Reconnaissance	Ø	$w1 = 0.2$ $w2 = 0.6$ $w3 = 0.2$
Intrusion	$Iu_a > 0.3$ OR ANY $perceivedpV[i] > 0.8$	$w1 = 0.2$ $w2 = 0.4$ $w3 = 0.4$
Exploitation OR Privilege Escalation	$C > 0.3$	$w1 = 0.4$ $w2 = 0.1$ $w3 = 0.5$

Table 1: Prerequisites and weights of the utility function for each CKC phase.

Strategy	Attacker Utility $U_A(A_A)$
Scan	$w1 \cdot (0.5 - \lvert perceivedpV[A_A] - 0.5\rvert) +$ $(w2 \cdot DI[A_A] + w3 \cdot DC[A_A]) \cdot perceivedpV[A_A]$
Attack	$(w2 \cdot DI[A_A] + w3 \cdot DC[A_A]) \cdot perceivedpV[A_A]$
Global Scan	$\sum_{A_i \text{ is Scan}} U(A_i)/2$

Table 2: Attacker utility according to the type of attacks

In our context, a state s corresponds to the defender's perception of the state of the game, i.e. its observations O^d.

Temporal Difference Learning offers an iterative process that updates the Q-values for each state-action pair based on the Bellman Optimality Equation. This process converges to the optimal Q-function.

$$Q(s,a) = (1-\alpha) \cdot Q(s,a) + \alpha(r_D^{t+1} + \gamma max_{a'} Q(s_{t+1}, a')) \tag{16}$$

where α is a learning rate.

In classic Reinforcement Learning, a table is used to store the mapping between the pair state, action and their corresponding Q-value. In Deep Reinforcement Learning, a neural network is used for this. The input of the neural network is the state, and the output is the estimated Q-value of each action in this state.

We use the Deep Q-Learning algorithm (DQN) of Mnih et al. [19] with a replay buffer and a target network to train our model.

This algorithm uses many hyperparameters that must be defined before learning. They are in the Table 3.

Parameter	Value
Weight in Defender Utility	0.3
k for Defender Observations	3
Exploration Fraction	0.1
Target Update Interval	5000
Neural Network Layers	[256x256]
Replay Buffer Size	1000000
Activation Function	Tanh
Train Frequence	One episode
Batch Size	32
Learning Rate	0.0005
Discounted Factor	0.99

Table 3: List of model parameters and DQN hyperparameters

6 EXPERIMENTS

6.1 Data for Experiments

In Sections 3 and 4, we describe our model and a basic scenario for the attacker. Several parameters are required for the experiments. First, concerning the machine, we need to define the vulnerabilities considered in the model. These are not Common Vulnerablities and Exposures (CVEs), they are rather families of vulnerabilities. Then, for each of them, we have to define the damage in Information and in value of Compromission as well as their cost in the case

of a scan or an attack. These values have been defined arbitrarily based on information from Mitre CAPEC [3] and Mitre CVE [9]. Table 4 shows the data used in the experiments concerning the vulnerabilities and the attack strategies. We have selected a set of known vulnerability families that can be separated into two categories: those accessible from outside the machine and those requiring local access on the machine. For simplicity, we consider that the costs of the scans are the same. The same goes for the cost of the attacks.

Vulnerability	Type	required CKC phase	Cost	DI	DC
Identity Spoofing	Scan	0	0.2	0.3	0
	Attack	1	0.3	0.15	0.4
Traffic Injection	Scan	0	0.2	0.2	0
	Attack	1	0.3	0.1	0.4
Brute Force	Scan	0	0.2	0.2	0
	Attack	1	0.3	0.1	0.4
Command Injection	Scan	0	0.2	0.4	0
	Attack	1	0.3	0.2	0.4
Code Injection	Scan	0	0.2	0.3	0
	Attack	1	0.3	0.15	0.4
Privilege Abuse	Scan	2	0.2	0.2	0
	Attack	2	0.3	0.1	0.6
Authentication Bypass	Scan	2	0.2	0.1	0
	Attack	2	0.3	0.05	0.6
Privilege Escalation	Scan	2	0.2	0.2	0
	Attack	2	0.3	0.1	0.6

Table 4: List of considered vulnerabilities

Concerning the defender, we consider eight possible defense strategies: four MTD and four deception strategies. They can be found in the Table 5 with the cost of each strategy.

Strategy Number	Strategy	Type	Cost
0	IP Random	MTD	0.1
1	Port Random	MTD	0.2
2	Rekeying Keys Random	MTD	0.1
3	Language Random	MTD	0.3
4	Honey Service	Deception	0.4
5	Honey Credentials/Accounts	Deception	0.3
6	Honey Process	Deception	0.2
7	Honey Files/Logs	Deception	0.2

Table 5: Considered defense strategies

Another essential element in the experiments is the table of efficiencies of the defensive strategies against the actions of the attacker mentioned in the Section 3.1.3 and shown in the Table 6. These are chosen arbitrarily based on our knowledge of the different strategies, but one way to improve the model could be to use data from real experiments.

6.2 Implementation

The experiments were done on an Intel Core i7-9750H CPU with an NVIDIA GeForce RTX 2070 graphics card.

Our model has been implemented in an OpenAi Gym environment [2]. Stable Baselines3 is a python library providing the implementation of Reinforcement Learning algorithms [24]. We use it to train our agents on the model.

We sought to optimize the hyperparameters used to train our agents. For this, we trained the model with many different parameters. The model parameters and the hyperparameters finally used for the tests are located in the Table 3.

6.3 Results

We trained agents in different environments with different parameters. We have measured the influence of the parameters on the performance of our trained agents. Shown in Figure 1 are the training curves of different defenders for different values of λ_{attack}. We can thus compare the influence of the probability of attack success with the performance of the agents. Our model includes a significant uncertainty, so the reward has a large variance.

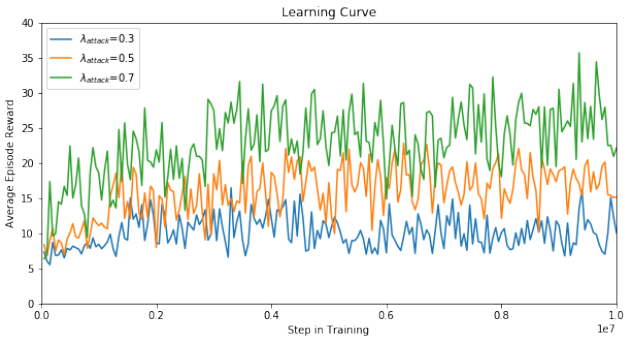

Figure 1: Learning curve of different agents in different environments with different λ_{attack} and $\lambda_{Deception} = 0.5$, $P_{Detection} = 1$

In Figure 2 we compare the performance of our DQN-trained agent with the performance of agents using only one defensive strategy or randomly choosing the defensive strategy. We evaluate each strategy over 1000 episodes. We compare the average episode reward for each strategy but also the average episode length.

The agent trained with the DQN performs better than the other "naive" strategies in terms of both episode reward and episode length. This means that our agent is able to use its observations to deduce a defense strategy to use. Moreover, we can observe the influence of strategy costs in the reward because for some strategies the episode length is higher than for others but the corresponding reward is lower. For example, IP Random Strategy has a higher reward than Honey Service but a shorter episode length.

We compared the performances obtained by an agent trained with the DQN in different environments. We varied the probability of detection of attacks $P_{Detection}$, λ_{attack} and $\lambda_{deception}$. Figure 3 contains the comparison of the rewards obtained with a DQN agent for different values of these parameters. It contains one heatmap

Attacker Strategy Type	Vulnerability	0	1	2	3	4	5	6	7
Scan	Identity Spoofing	0	0	0	0	0	1.5	0	0
	Traffic Injection	1.5	1.5	0	0	0	0	0	0
	Brute Force	0	0	1.5	0	0	1.5	0	0
	Command Injection	0	0	0	1	1	0	0	0
	Code Injection	0	0	0	1.5	1	0	0	0
	Privilege Abuse	0	0	1	0	0	0	0	1.5
	Authentication Bypass	0	1	0.5	0	0	1	1	1
	Priv Esc: Target Programs with Elevated Privileges	0	0	0	1	1	0	1.5	1
Attack	Identity Spoofing	0	0	0	0	0	1.5	0	0
	Traffic Injection	1.5	1.5	0	0	0	0	0	0
	Brute Force	0	0	1.5	0	0	1.5	0	0
	Command Injection	0	0	0	1	0.5	0	0	0
	Code Injection	0	0	0	1.5	0.5	0	0	0
	Privilege Abuse	0.5	0	1	0	0	0	0	0.5
	Authentication Bypass	0.5	1	0.5	0	0	0.5	0.5	0.5
	Priv Esc: Target Programs with Elevated Privileges	0.5	0	0	1	0.5	0	0.5	0.5

Table 6: Efficiency of the defensive strategies against the attacker actions

per $P_{Detection}$ value. Each heatmap shows the rewards obtained for 3 different values of λ_{attack} and $\lambda_{deception}$.

The lower the probability of detection of attacks, the lower the rewards for the defender. This makes sense because if the defender does not observe the attacks, they cannot choose an appropriate strategy. λ_{attack}, which inversely influences the probability of success of attacks, has a strong impact on the performance of our agent. $\lambda_{deception}$ inversely impacts the probability of deception success. The larger the value for $\lambda_{deception}$, the lower the probability of deception success.

The problem with the DQN algorithm is that the resulting policy is not necessarily optimal. Indeed, it may be a local optimum. This explains some of the results in Figure 3, for example, for $P_{Detection} = 0.8$ and $\lambda_{attack} = 0.7$, the performances of the agent for $\lambda_{deception} = 0.5$ are lower than for $\lambda_{deception} = 0.7$ which is anomalous according to the explanations of the previous paragraph.

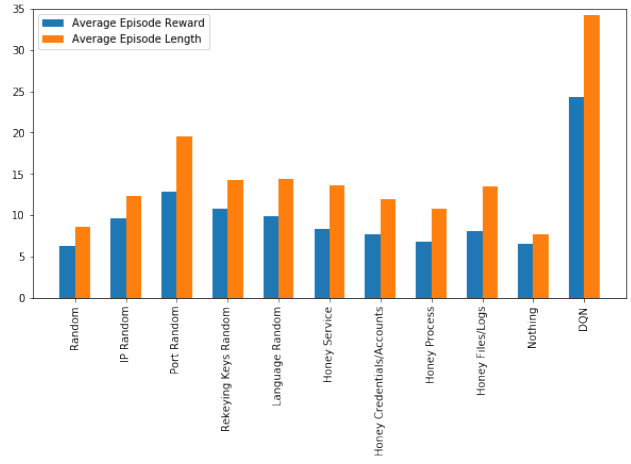

Figure 2: Comparison of the performance of different defense strategies in different environments with $\lambda_{attack} = 0.5$ and $\lambda_{Deception} = 0.5$, $P_{Detection} = 1$

6.4 Discussion

The proposed model is multi-state and multi-stage. It allows to consider a large number of possible states for the system. Moreover, the representation of the vulnerabilities in the V vector allows to consider different types of machines such as servers or workstations which will have different types of vulnerabilities. Futhermore, thanks to our representation of the defender's observations, we are able to use the attacker's past behavior to better predict these future actions and thus choose an optimal strategy.

Moreover, our modeling takes into account the difference in nature between MTD strategies and Deception strategies. Thus they act differently on the system. Deception and MTD strategies should be seen as complementary. Experiments show that to counteract a variety of potential attacks, it is necessary to have access to a variety of defensive strategies. The experiments also show the importance of good coordination of these strategies and the need to limit the deployment of certain strategies to critical moments because their cost to the system can be very high.

Regardless of the parameters used, we manage to train an agent that performs better than the unitary strategies against our attacker following our scenario detailed in Section 4. The advantages of this scenario are that it is based on the perception of the attacker and it separates remote attacks and local attacks such as privilege escalation.

The realism of the model could be improved by using data from experiments. To do this, we could implement several defensive strategies on a system with known vulnerabilities in order to measure the ability of these strategies to prevent or slow down their exploitation by an attacker. In particular, as the model is sequential, it would be interesting to integrate the time of each action for both

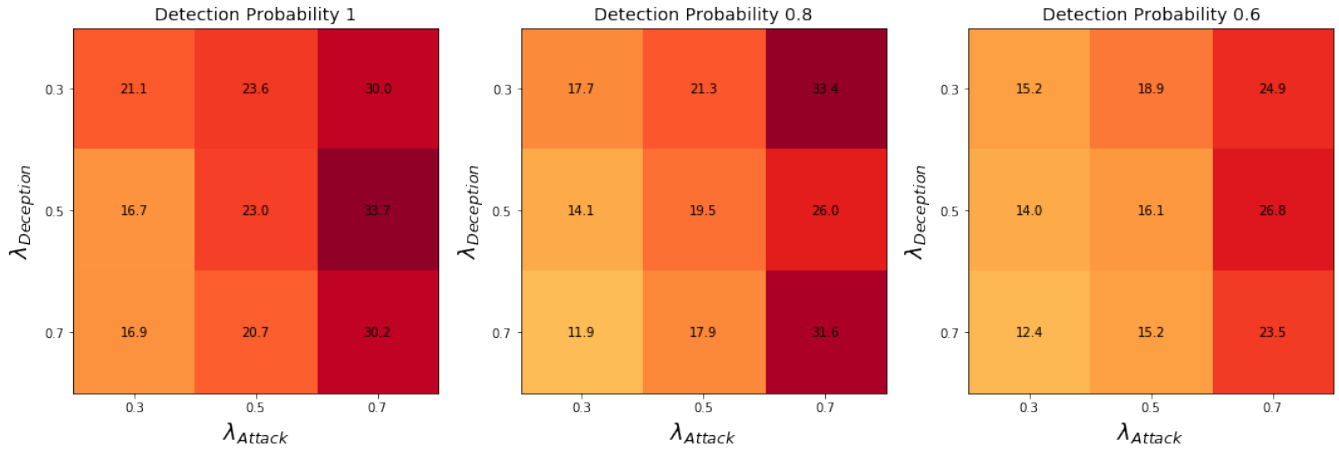

Figure 3: Comparison of the reward of the defender using a DQN agent in environments with different values of detection probability $P_{Detection}$, λ_{attack} and $\lambda_{deception}$

the attacker and the defender in the calculation of their cost. Moreover, the effectiveness of defensive strategies against attacks should not be static but depend on the attacker's progress in their attacks. Indeed, for example, the IP Random defense strategy will not be as efficient against an attacker located outside the network or against an attacker who already has access to the internal network.

In this paper, the model seeks to represent the confrontation of a defender and an attacker around a single machine. However, to better represent reality, it would be necessary to extend this model to a network of machines. This would multiply the number of possible actions for each player, the number of states of the model and the number of entry points for the attacker. It would then be necessary to consider the conflicts that can exist in the deployment of defensive strategies on several machines. This would allow the simulation of more realistic attack scenarios. It would then be interesting to know if the DQN algorithm is still efficient in a model of this type. This is ongoing research work that will be the subject of a future article.

7 CONCLUSION

Deception and MTD strategies are two types of strategies that consist in bringing false information or uncertainty to the opponent's perception in order to increase the cost of the attacks. In this paper, we proposed a model of an attacker/defender confrontation in a computer system considering the asymmetry of perceptions and the impact of the two players' strategies on them. We then proposed an attacker scenario based on the CKC and using their perception. Thanks to this, we performed simulations and trained with the DQN algorithm a defensive agent. It is able to choose the most adapted defensive strategies to prevent the compromise of the machine by using the observed past actions. This simulation framework could be used to optimize the use of MTD and Deception strategies in real context by using data from experiments for model parameters.

Future works would then be to develop an emulation environment allowing the deployment of different MTD and deception strategies. This would provide data such as the cost or effectiveness of the latter. It could also allow for the training of defending DQN agent directly on the emulation environment. Another part of the work would be to improve the attacker scenarios. To do this, we could try to train an attacking DQN agent based on its observation.

REFERENCES

[1] Ahmed H Anwar, Charles Kamhoua, and Nandi Leslie. 2020. Honeypot allocation over attack graphs in cyber deception games. In *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 502–506.
[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).
[3] CAPEC. 2022. Common Attack Pattern Enumeration and Classification (CAPEC). https://capec.mitre.org/
[4] Hayreddin Çeker, Jun Zhuang, Shambhu Upadhyaya, Quang Duy La, and Boon-Hee Soong. 2016. Deception-based game theoretical approach to mitigate DoS attacks. In *International conference on decision and game theory for security*. Springer, 18–38.
[5] Xinzhong Chai, Yasen Wang, Chuanxu Yan, Yuan Zhao, Wenlong Chen, and Xiaolei Wang. 2020. DQ-MOTAG: deep reinforcement learning-based moving target defense against DDoS attacks. In *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*. IEEE, 375–379.
[6] Jin-Hee Cho, Dilli P Sharma, Hooman Alavizadeh, Seunghyun Yoon, Noam Ben-Asher, Terrence J Moore, Dong Seong Kim, Hyuk Lim, and Frederica F Nelson. 2020. Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys & Tutorials* 22, 1 (2020), 709–745.
[7] Andrew Clark, Kun Sun, Linda Bushnell, and Radha Poovendran. 2015. A game-theoretic approach to IP address randomization in decoy-based cyber defense. In *International conference on decision and game theory for security*. Springer, 3–21.
[8] Andrew Clark, Quanyan Zhu, Radha Poovendran, and Tamer Başar. 2012. Deceptive routing in relay networks. In *International Conference on Decision and Game Theory for Security*. Springer, 171–185.
[9] CVE. 2022. Common Vulnerabilities and Exposures (CVE). https://cve.mitre.org/
[10] Taha Eghtesad, Yevgeniy Vorobeychik, and Aron Laszka. 2020. Adversarial deep reinforcement learning based adaptive moving target defense. In *International Conference on Decision and Game Theory for Security*. Springer, 58–79.
[11] Xiaotao Feng, Zizhan Zheng, Prasant Mohapatra, and Derya Cansever. 2017. A stackelberg game and markov modeling of moving target defense. In *International Conference on Decision and Game Theory for Security*. Springer, 315–335.
[12] Daniel Fraunholz, Simon Duque Anton, Christoph Lipps, Daniel Reti, Daniel Krohmer, Frederic Pohl, Matthias Tammen, and Hans Dieter Schotten. 2018. Demystifying deception technology: A survey. *arXiv preprint arXiv:1804.06196* (2018).
[13] Xiao Han, Nizar Kheir, and Davide Balzarotti. 2018. Deception techniques in computer security: A research perspective. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–36.
[14] Karel Horák, Quanyan Zhu, and Branislav Bošanskỳ. 2017. Manipulating adversary's belief: A dynamic game approach to deception by design for proactive

network security. In *International Conference on Decision and Game Theory for Security*. Springer, 273–294.

[15] Hao Hu, Jing Liu, Jinglei Tan, and Jiang Liu. 2020. SOCMTD: selecting optimal countermeasure for moving target defense using dynamic game. *KSII Transactions on Internet and Information Systems (TIIS)* 14, 10 (2020), 4157–4175.

[16] Quan Jia, Kun Sun, and Angelos Stavrou. 2013. Motag: Moving target defense against internet denial of service attacks. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–9.

[17] Quang Duy La, Tony QS Quek, Jemin Lee, Shi Jin, and Hongbo Zhu. 2016. Deceptive attack and defense game in honeypot-enabled networks for the internet of things. *IEEE Internet of Things Journal* 3, 6 (2016), 1025–1035.

[18] Cheng Lei, Duo-He Ma, and Hong-Qi Zhang. 2017. Optimal strategy selection for moving target defense based on Markov game. *IEEE Access* 5 (2017), 156–169.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[20] Jeffrey Pawlick, Edward Colbert, and Quanyan Zhu. 2019. A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 1–28.

[21] Jeffrey Pawlick and Quanyan Zhu. 2015. Deception by design: evidence-based signaling games for network defense. *arXiv preprint arXiv:1503.05458* (2015).

[22] Eric Peter and Todd Schiller. 2011. A practical guide to honeypots. *Washington Univerity* (2011).

[23] Achintya Prakash and Michael P Wellman. 2015. Empirical game-theoretic analysis for moving target defense. In *Proceedings of the second ACM workshop on moving target defense*. 57–65.

[24] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. http://jmlr.org/papers/v22/20-1364.html

[25] Mohammad Ashiqur Rahman, Mohammad Hossein Manshaei, and Ehab Al-Shaer. 2013. A game-theoretic approach for deceiving remote operating system fingerprinting. In *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 73–81.

[26] Sailik Sengupta, Ankur Chowdhary, Abdulhakim Sabur, Adel Alshamrani, Dijiang Huang, and Subbarao Kambhampati. 2020. A survey of moving target defenses for network security. *IEEE Communications Surveys & Tutorials* 22, 3 (2020), 1909–1941.

[27] Sailik Sengupta and Subbarao Kambhampati. 2020. Multi-agent reinforcement learning in bayesian stackelberg markov games for adaptive moving target defense. *arXiv preprint arXiv:2007.10457* (2020).

[28] Jing-lei Tan, Cheng Lei, Hong-qi Zhang, and Yu-qiao Cheng. 2019. Optimal strategy selection approach to moving target defense based on Markov robust game. *computers & security* 85 (2019), 63–76.

[29] Shuo Wang, Qingqi Pei, Jianhua Wang, Guangming Tang, Yuchen Zhang, and Xiaohu Liu. 2020. An intelligent deployment policy for deception resources based on reinforcement learning. *IEEE Access* 8 (2020), 35792–35804.

[30] Huan Zhang, Kangfeng Zheng, Xiujuan Wang, Shoushan Luo, and Bin Wu. 2019. Efficient strategy selection for moving target defense under multiple attacks. *IEEE Access* 7 (2019), 65982–65995.

[31] Huan Zhang, Kangfeng Zheng, Xiujuan Wang, Shoushan Luo, and Bin Wu. 2020. Strategy Selection for Moving Target Defense in Incomplete Information Game. *Computers, Materials & Continua* 62, 2 (2020), 763–786. https://doi.org/10.32604/cmc.2020.06553

[32] Jianjun Zheng and Akbar Siami Namin. 2019. A survey on the moving target defense strategies: An architectural perspective. *Journal of Computer Science and Technology* 34, 1 (2019), 207–233.