Titre: Title:	Optimisation de la sélection de techniques déceptives pour la sécurité et la cyber-résilience
Auteur: Author:	Axel Charpentier
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Charpentier, A. (2022). Optimisation de la sélection de techniques déceptives pour la sécurité et la cyber-résilience [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10736/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Optimisation de la sélection de techniques déceptives pour la sécurité et la cyber-résilience

AXEL CHARPENTIER
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Décembre 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Optimisation de la sélection de techniques déceptives pour la sécurité et la cyber-résilience

présenté par **Axel CHARPENTIER**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Alejandro QUINTERO, président
Nora BOULAHIA CUPPENS, membre et directrice de recherche
Frédéric CUPPENS, membre et codirecteur de recherche
Khomh FOUTSE, membre

DÉDICACE

À ma famille et mes amis...

REMERCIEMENTS

Je voudrais remercier mes directeurs de recherche Nora Boulahia Cuppens et Frédéric Cuppens de m'avoir offert l'opportunité de travailler sur ce projet et de découvrir le monde de la recherche universitaire à travers notamment la participation à une conférence.

Je tiens également à remercier mes camarades du laboratoire de recherche LabCyS et de la chair CRITICAL avec qui j'ai passé de très bon moments et qui m'ont apporté beaucoup de connaissances.

Je remercie mes parents pour m'avoir toujours encouragé et soutenu dans mes études et pour ce long voyage au Canada.

Je tiens à exprimer ma gratitude à l'IRT SystemX pour sa participation financière à ce projet.

RÉSUMÉ

Pour améliorer la cyber-résilience et la sécurité des systèmes informatiques, une possibilité est l'utilisation de stratégies déceptives telles que les stratégies de leurrage ou les stratégies de défense par cible mouvante (Moving Target Defense - MTD). Cependant, le coût de ses dernières et leur impact nécessite une optimisation de leur déploiement.

Une première partie de notre travail fut la proposition d'un modèle de confrontation attaquant/défenseur prenant en compte les différentes perceptions des joueurs ainsi que la différence de nature entre les effets des stratégies de leurrage et des stratégies de défense par cible mouvante. Il a ensuite fallu s'assurer que nous étions capables d'optimiser la sélection des stratégies déceptives dans un tel modèle. Pour cela, nous avons proposé un scénario d'attaquant basé sur la Cyber Kill Chain dont le but est de compromettre le système ; puis nous avons entraîné un agent défenseur à ralentir ou empêcher l'atteinte de cet objectif grâce à l'algorithme d'apprentissage profond par renforcement Deep Q-Network (DQN). Nos résultats montrent un ralentissement de l'attaquant face à notre agent défenseur entraîné en comparaison à des stratégies plus naïves telles l'utilisation récurrente de la même stratégie ou le choix aléatoire de la stratégie défensive.

La seconde partie de notre travail fut de vérifier si l'optimisation de la sélection de techniques déceptives était toujours possible et efficace dans un environnement réel. Il a donc fallu déployer un environnement avec des stratégies MTD et de leurrage. Nous avons ensuite entraîné un agent attaquant à atteindre un objectif d'exfiltration de données sensibles dans cet environnement alors statique en déployant ou non des stratégies de leurrage. Pour cela, des contraintes sur ses observations nous ont obligé à utiliser un autre algorithme d'apprentissage profond par renforcement que DQN. Nous avons utilisé l'algorithme Proximal Policy Optimization (PPO). L'entraînement était efficace et nous avons pu observer l'influence des stratégies de leurrage. Nous avons ensuite entraîné un agent défenseur à contrer ou ralentir cet attaquant en déployant ses propres stratégies MTD ou de leurrage. Nos résultats montrent que les stratégies MTD sont très efficaces contre un agent attaquant habitué aux environnements statiques même lorsque l'intervalle d'action du défenseur est assez grand (50 actions de l'attaquant pour une action du défenseur). De cela, nous avons déduit que l'amélioration de l'agent défenseur passerait forcément par l'amélioration de l'agent attaquant notamment en adaptant sa perception et ses observations aux changements dans l'environnement.

ABSTRACT

To improve the cyber-resilience and security of Information Technology (IT) systems, one possibility is the use of deceptive strategies such as deception or Moving Target Defense (MTD) strategies. However, the cost of these strategies and their impact requires optimization of their deployment.

The first part of our work is to propose a model of the attacker/defender confrontation that takes into account the different perceptions of the players as well as the difference in kind between the effects of deception and MTD strategies. We must ensure that we are able to optimize the selection of deceptive strategies in this model. To do this, we propose an attacker scenario based on the Cyber Kill Chain with the goal of compromising the system; then we train a defending agent to slow down or prevent the achievement of this goal using a deep reinforcement learning algorithm: the Deep Q-Network (DQN) algorithm. Our results show a slowing down of the attacker against our trained defensive agent compared to more naive strategies, such as the repetitive use of the same strategy or the random choice of the defensive strategy.

The second part of our work is to verify if the optimization of the selection of deceptive strategies is still possible and effective in a real environment. This involves deploying an environment with MTD and deception strategies. We then train an attacking agent to achieve a goal of exfiltrating sensitive data in this static environment by deploying or not deception strategies. For this, constraints on its observations lead us to use another deep reinforcement learning algorithm than DQN, namely, the Proximal Policy Optimization (PPO) algorithm. The training is effective and we are able to observe the influence of deception strategies. We then train a defender agent to counter or slow down this attacker by deploying its own MTD or deception strategies. Our results show that MTD strategies are very effective against an attacking agent used in static environments, even when the defender's action interval is quite large (50 attacker's actions for one defender's action). From this, we deduce that the improvement of the defender agent would necessarily involve the improvement of the attacker agent, in particular, by adapting its perception and observations to changes in the environment.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

NIST	National Institute of Standards and Technology
MTD	Défense par Cible Mouvante (Moving Target Defense)
SDN	Software Defined Network
OSI	Open Systems Interconnection
CKC	Cyber Kill Chain
MDP	Processus de Décision Markovien (Markov Decision Process)
DOS	Déni de Service (Denial-of-Service)
DDOS	DOS distribué (Distributed DOS)
MSNE	Equilibre mixte de Nash (Mixed Strategy Nash Equilibrium)
DRL	Apprentissage par renforcement profond (Deep Reinforcement Learning)
ML	Apprentissage Machine (Machine Learning)
DQN	Deep Q-Network
CVE	Common Vulnerablities and Exposures
NFS	Système de Fichiers en Réseau (Network File System)
PPO	Proximal Policy Optimization
LSTM	Long Short-Term Memory
RNN	Réseaux de neurones récurrents (Recurrent Neural Network)

CHAPITRE 1 INTRODUCTION

L'augmentation des cyberattaques ces dernières années a entraîné des interruptions des services de nombreuses entreprises. Atteindre une cyber-résilience est donc un défi et un besoin de plus en plus courant. Pour cela, des stratégies parfois anciennes se redéveloppent. C'est le cas par exemple des stratégies déceptives. Pouvant être des stratégies de leurrage ou des stratégies de défense par cible mouvante, elles peuvent permettre à la fois d'améliorer la cyber-résilience et de ralentir un attaquant dans son attaque d'un système informatique. Or les stratégies déceptives ne sont pas efficaces contre les mêmes attaques, automatiser la réaction du système aux attaques détectées est donc nécesssaire. Pour cela il faut être capable d'optimiser le déploiement des stratégies déceptives, pour minimiser le coût tout en améliorant la défense. Dans ce chapitre, nous introduisons d'abord les termes de cyber-résilience ainsi que les stratégies déceptives dans la section 1.1. Nous présentons ensuite nos motivations et nos objectifs de recherche dans les sections 1.2 et 1.3. Enfin, nous terminons ce chapitre par la présentation du plan que suit ce mémoire dans la section 1.4.

1.1 Définitions et concepts de base

1.1.1 La cyber-résilience

Le développement des systèmes informatiques des dernières décennies a entrainé une augmentation des cyber-attaques. Un rapport de Cisco montre que 24% des PME ont subi des interruptions de services de plus de huit heures en 2019 et 31% pour les grandes entreprises [1]. Pour de plus en plus de services, ces interruptions ne sont plus tolérables. Beaucoup de systèmes même pendant ou à la suite d'une attaque ont besoin de garantir le fonctionnement normal d'un service. C'est pour cela que garantir la résilience est nécessaire.

A l'origine, la résilience est un terme utilisé en physique qui définit la capacité de résistance d'un corps ou d'un matériau à un choc ou à une déformation. De manière plus générale, la résilience est la capacité, pour un système donné, de retrouver un fonctionnement normal après avoir subi des perturbations. Un cas particulier de la résilience est la cyber-résilience. Il s'agit de la résilience dans un environnement informatique. Le National Institute of Standards and Technology (NIST) en propose une définition [2] :

« La cyber-résilience est la capacité d'anticiper, de résister, de se rétablir et de s'adapter à des conditions défavorables, des attaques ou des compromissions sur des systèmes qui utilisent des ressources cybernétiques. La cyber-résilience a pour but de permettre la réalisation des

objectifs de la mission ou de l'activité qui dépendent des ressources cyber dans un environnement informatique.»

Björck et al. (2015) fournit une définition plus épurée de la cyber-résilience [3] :

« La cyber-résilience désigne la capacité de fournir en permanence les résultats escomptés en dépit de cyber-événements défavorables.»

Dans cette définition, La notion de permanence (ou de continuité) est très importante. La capacité à fournir en permanence les résultats escomptés doit être garantie même pendant une crise ou après une violation de la sécurité du système. Cette notion désigne aussi la capacité à rétablir un fonctionnement considéré comme normal pour le système mais aussi sa capacité à s'adapter et à adapter le mécanisme de fourniture du service pour faire face à l'évolution de la menace afin de continuellement fournir le service attendu. La cyber résilience concerne les différents composants d'un système informatique, à la fois physiques et logiciels [4].

1.1.2 Les stratégies déceptives

Une approche assez récente pour améliorer la cyber résilience des systèmes est l'approche déceptive [5]. Celle-ci existait bien avant les systèmes informatiques et est à la base de la stratégie de guerre. Elle consiste à empêcher l'attaquant d'obtenir des informations réelles sur un système en lui fournissant de fausses informations afin de rendre inefficace sa phase de reconnaissance et ainsi réduire la probabilité de réussite des attaques. Le NIST fournit la définition suivante du leurrage [6] :

« Induire en erreur, cacher des biens essentiels ou exposer secrètement des biens altérés à l'adversaire.»

Cela peut être réalisé par plusieurs moyens tels que la perturbation, l'obfuscation ou l'utilisation d'HoneyPots (littéralement «*Pot de miel*» en français) [7]. Aujourd'hui, les stratégies de leurrage se sont beaucoup développées. Certaines études listent et classifient les techniques et stratégies de leurrage existantes [8–10] mais nous reviendrons sur cela dans le chapitre 2.

Il existe de plus en plus de solutions commerciales intégrant des éléments de leurrage [3]. Leurs avantages sont nombreux. Les solutions de leurrage permettent la détection d'attaques ou d'attaquants avec un taux de faux positifs relativement faibles, c'est pourquoi ils sont souvent couplés à un système de détection. En effet, cela permet la détection d'attaques qui n'est pas basée sur un motif mais plutôt sur des anomalies. S'il n'y a pas d'attaques, aucune activité ne devrait être présente sur les éléments de leurrage donc toute activité sur ces éléments peut être considérée comme une anomalie et donc comme une potentielle attaque. Ces solutions de leurrage permettent aussi de mieux observer et apprendre les techniques

utilisées par l'attaquant ce qui permet d'améliorer la réponse aux incidents.

De plus, Il est largement reconnu qu'en matière de cybersécurité, il existe une asymétrie entre l'attaquant et le défenseur. En effet, l'attaquant choisit quand et comment il va tenter de pénétrer ou de compromettre un réseau ou une machine. L'attaquant peut donc collecter des informations sur sa cible et revenir plus tard pour la compromettre, alors qu'il est très difficile pour le défenseur d'évaluer les menaces auxquelles il est confronté.

Une approche pour réduire cet avantage de l'attaquant est l'utilisation de stratégies de Défense par Cible Mouvante (Moving Target Defense) (MTD). L'objectif des stratégies MTD est d'éliminer l'avantage temporel de l'attaquant dû à la nature statique des infrastructures et des systèmes. Cela consiste à modifier la surface d'attaque ainsi que la surface d'exploration dans le système pour rendre toute attaque ou collecte d'information plus difficile. Le NIST fournit la définition suivante du concept de MTD [2] :

« Le concept de contrôler le changement dans plusieurs dimensions d'un système afin d'augmenter l'incertitude et la complexité apparente pour les attaquants, de réduire leur fenêtre d'opportunité et d'augmenter les coûts de leurs efforts d'exploration et d'attaque. »

Ces techniques introduisent donc de l'aléatoire, de la diversité et du dynamisme dans le système. Cette approche est parfois considérée comme une stratégie de leurrage. Cependant, alors que l'objectif des stratégies de leurrage est de fournir de fausses informations à l'attaquant, l'objectif des stratégies MTD est d'empêcher l'utilisation des informations précédemment obtenues, c'est-à-dire de faire en sorte que les informations précédemment obtenues par l'attaquant, par exemple lors de la phase de reconnaissance, ne soient plus valables.

Les stratégies MTD peuvent être caractérisées par la réponse à trois questions [11] : Quoi déplacer ? Comment déplacer ? Quand déplacer ? La première question "Quoi déplacer ?" renvoie à l'attribut du système que l'on va rendre variable. Par exemple, dans un réseau, cela pourrait être les adresses IP ou les ports des machines et pour une machine, il pourrait s'agir de son système d'exploitation ou de la version d'un logiciel. La deuxième question "Comment déplacer ?" désigne la façon de modifier les attributs mouvants du système, l'objectif étant d'accroître au maximum l'incertitude et la non-prédictibilité de cette nouvelle configuration. La dernière question "Quand déplacer ?" fait référence à ce qui va déclencher le changement de valeur d'un attribut. Il existe deux grands types de stratégies : proactives ou réactives. Les stratégies proactives déclenchent les techniques MTD selon un programme, à des intervalles fixes ou aléatoires tandis que les stratégies réactives se déclenchent à la suite d'un évènement ou d'une alerte.

Ces stratégies permettent d'accroître la difficulté et le coût (en temps, en effort et en res-

source) des attaques. Les stratégies MTD ont fait l'objet de nombreux articles de recherche. Des études antérieures ont examiné les stratégies MTD existantes, avec pour chacune, ses propres critères d'analyse et de classification [11–13] mais nous reviendrons sur cela dans le chapitre 2.

1.2 Éléments de la problématique

En 2021, un rapport de l'entreprise Accenture montre que les cyber-attaques ont augmenté de 31% entre 2020 et 2021 [14]. De plus, ce rapport met également en évidence que 55% des grandes entreprises n'arrêtent pas efficacement les cyber-attaques, ne détecte pas et ne corrigent pas les failles rapidement. La cyber-résilience est devenue un élément clé de la sécurité informatique des entreprises.

Les stratégies MTD et de Leurrage permettent d'améliorer la cyber-résilience. En effet, tout d'abord, elles permettent de ralentir l'attaquant en ajoutant du dynamisme à des systèmes qui sont ordinairement statiques [11]. Elles peuvent permettre d'accélérer la détection d'attaques ou des fuites de base de données par exemple [15,16]. Elles permettent également la réduction de l'impact de la découverte d'une faille par un attaquant en rendant son exploitation plus compliquée.

Cependant, pour de nombreuses stratégies déceptives, leur intégration à un système doit être pensé dès la conception de ce dernier. De plus, chacune de ces stratégies ne va pas être efficace contre le même type d'attaque ou ne va pas protéger la même surface d'exploration ou d'attaque. Certaines stratégies interviennent dans la partie réseau d'un système tandis que d'autre peuvent se situer au niveau logiciel d'un service. Ajoutons à cela leur coût en termes de temps de développement et de ressources et la sélection de ces stratégies MTD ou de leurrage devient la priorité pour trouver l'équilibre entre leur coût et leur bénéfice en termes de sécurité. Pour cela, la modélisation est beaucoup utilisée [10, 17–22]. Nous cherchons à modéliser la confrontation d'un attaquant et d'un défenseur dans un environnement informatique. Cela va permettre de mesurer les bénéfices de chaque stratégie défensive en fonction de ce que fait l'attaquant. Un avantage de la modélisation est qu'elle permet d'avoir une idée de l'efficacité des stratégies sans nécessiter leur implémentation. Cependant, un inconvénient de cette méthode est qu'elle nécessite une connaissance préalable des stratégies défensives mais également de celles de l'attaquant. Il est aussi nécessaire de simuler la prise de décision de l'adversaire. Pour cela, de nombreux articles utilisent la théorie des jeux qui permet de modéliser les interactions entre deux joueurs en considérant la prise de décision de chacun comme rationnelle. Cependant, ce type de modèle ne permet pas de considérer la différence de nature entre les effets de stratégie MTD et de leurrage.

A l'origine, les stratégies MTD et de leurrage furent introduites comme des stratégies proactives [5, 11]. Mais celles-ci peuvent aussi être réactives, cela nécessite de les associer à un système de détection d'attaques. Cependant, dans ce cas, l'efficacité de la stratégie MTD dépendra directement de l'efficacité du système de détection. De plus, les attaques informatiques étant de plus en plus rapides et les temps de détection de plus en plus longs, des stratégies adaptatives sont nécessaires. En effet, tandis que l'efficacité des stratégies réactives dépendra grandement du système de détection, avec les stratégies proactives, nous pouvons accroître le coût de la défense et de son déploiement sans réduire l'impact des vulnérabilités. Cela peut entraîner une réduction de la qualité de service ce qui n'est pas acceptable pour certains services. Certains systèmes ont également une quantité de ressources contrainte ou limitée, il est donc essentiel d'avoir un déploiement des stratégies MTD et de leurrage adaptatif non seulement aux attaques détectées mais également à l'état du système.

Il est primordial de chercher à optimiser le déploiement des techniques MTD et de leurrage. D'autre part, certaines stratégies sont plus efficaces si elles sont utilisées à la suite ou ensemble. Par exemple, si un serveur HoneyPot est déployé dans un réseau alors l'utilisation d'une stratégie telle que la randomisation des adresses IP dans celui-ci peut faire en sorte que l'attaquant se fasse leurrer plus facilement par ce serveur HoneyPot. L'efficacité de ces stratégies est donc intrinsèquement liée à celle des stratégies qui sont déployées en même temps. Comparer les performances de chacune d'elles est donc difficile car leurs performances sont relatives à leur déploiement. Afin de comparer les performances de ces stratégies il faut donc optimiser leur utilisation au préalable. Dans un modèle, cela peut être fait de plusieurs manières. Si le modèle est basé sur la théorie des jeux alors dépendamment du modèle de théorie des jeux utilisé, il existe des outils permettant de trouver un déploiement théorique optimal. Un exemple est l'équilibre de Nash dont l'existence est prouvée dans certains modèles. Il s'agit d'une situation où aucun joueur ne peut améliorer son gain sans que son adversaire ne choisisse une autre stratégie. Cependant, si trouver le déploiement optimal des stratégies dans un modèle est utile pour avoir une idée de l'efficacité des stratégies, il faut aussi que la méthode d'optimisation de la sélection de la stratégie défensive soit toujours efficace dans un environnement réel si nous voulons pouvoir l'utiliser réellement.

1.3 Objectifs de recherche

Ces dernières années, la cyber-résilience est devenue une priorité des entreprises du numériques. Dans ce but, des éléments déceptifs sont de plus en plus souvent ajoutés à la sécurité des systèmes informatiques. Cependant, leur coût de déploiement étant important, il est primordial d'optimiser leur déploiement afin de maximiser la défense tout en minimisant

l'utilisation de ressources et la surcharge apportée par ces stratégies tant pour les réseaux que pour les processeurs. De plus, certaines stratégies MTD et de leurrage ont une grande amélioration de leur efficacité s'ils sont utilisées ensemble. Cette optimisation de l'utilisation des stratégies déceptives doit tout d'abord être faite dans un modèle afin d'être capable de sélectionner les meilleures stratégies avant leur implémentation. Ce modèle doit notamment prendre en compte la différence d'effets sur le système et sur la perception de l'attaquant des stratégies MTD et de leurrage. Il faut être capable d'optimiser la sélection des stratégies déceptives dans le modèle afin de mesurer leur efficacité concrète lorsqu'elles sont utilisées avec d'autres. La méthode utilisée pour l'optimisation de la stratégie défensive doit également être utilisable dans un environnement réel pour vérifier que cette méthode est viable en réalité et pas seulement dans la théorie.

L'objectif général de ce travail est de proposer une méthode afin d'optimiser l'utilisation de stratégies déceptives tant dans un modèle que dans un environnement réel. C'est pour cela que la question de recherche suivante a été formulée :

> *Comment optimiser la sélection d'une technique de leurrage ou de Moving Target Defense pour améliorer la cyber-résilience et la sécurité d'un système informatique ?*

Afin de répondre à cette question, nous nous sommes fixés trois objectifs spécifiques :

1. *Proposer un modèle pour simuler les interactions d'un attaquant et d'un défenseur lors d'une attaque d'un système informatique.*

2. *Optimiser le choix de la stratégie de la défense afin de contrer ou leurrer un attaquant dans ce modèle.*

3. *Implémenter des stratégies MTD et de leurrage dans un environnement réel et optimiser le choix de la stratégie défensive dans ce dernier afin de mesurer leurs efficacités réelles.*

1.4 Plan du mémoire

Dans le chapitre 2, nous réalisons une revue de la littérature des différentes stratégies MTD et de leurrage existantes, des différentes modélisations de la confrontation attaquant/défenseur possibles ainsi que des méthodes utilisées pour optimiser la sélection des stratégies défensives. Dans le chapitre 3, nous présentons un modèle attaquant/défenseur considérant la perception des deux joueurs et les différences d'effets entre les stratégies MTD et de leurrage. Dans ce chapitre, nous présentons aussi la méthode utilisée pour optimiser la sélection des stratégies du défenseur. Dans le chapitre 4, nous introduisons l'environnement muni de stratégies déceptives que nous avons implémentées et nous montrons que l'optimisation de l'utilisation de

ces stratégies est toujours possible dans un environnement réel par des méthodes similaires à celle utilisées pour un modèle.

CHAPITRE 2 REVUE DE LITTÉRATURE

La revue de littérature est séparée en trois parties. Dans un premier temps, les différentes stratégies de leurrage seront abordées. Les stratégies MTD seront l'objet d'une deuxième partie. Enfin, la théorie des jeux étant très utilisée pour modéliser une confrontation attaquants/défenseurs dans un système informatique tout en considérant la perception des joueurs, nous analyserons les différentes approches proposées. Nous aborderons finalement la sélection des stratégies avec et sans le support de la théorie des jeux.

2.1 Stratégies de Leurrage

En informatique, le leurrage consiste à fournir de fausses informations à l'attaquant afin de saboter sa phase de reconnaissance et la rendre inefficace pour réduire les capacités l'impact de son attaque. Les HoneyPots sont au centre du leurrage informatique. Il s'agit d'une méthode visant à attirer l'attaquant vers une certaine ressource non critique (serveur, service, programme) afin de lui faire perdre son temps et d'observer ses stratégies. Mokube et Adams [23] et Spitzner [24] fournissent une description des différents types de HoneyPots, de leur mise en œuvre ainsi que des techniques utilisées pour identifier et obtenir des informations sur les différentes menaces. Une taxonomie des HoneyPots et une vue d'ensemble des logiciels de HoneyPots sont données dans Nawrocki et al. [25]. Cependant, il existe d'autres techniques pour leurrer un adversaire. Albanese et. al. [26] propose de manipuler le trafic sortant d'une machine afin de fournir de fausses informations à l'attaquant et ainsi déjouer la prise d'empreintes numériques. Il se concentre sur le leurrage de l'identification du système d'exploitation et des services.

Dans l'article de Juels et al. [15], Honeywords améliore la sécurité du stockage des mots de passe en leurrant l'adversaire. Pour cela, il associe un faux mot de passe à chaque compte utilisateur. Un attaquant qui a volé un fichier de mots de passe ne peut pas distinguer le vrai mot de passe du faux et s'il essaie de s'authentifier avec ce dernier, il déclenche une alarme. Dans l'article d'Almeshekah et al. [16], l'auteur propose quelque chose de similaire, en ajoutant des mots de passe faibles appelés «ersatzpassword» au fichier de mots de passe. Si l'attaquant tente de s'authentifier avec l'un de ces derniers, une alarme est déclenchée et cela signifie que le fichier de mots de passe a potentiellement fuité. Dans cet article, l'auteur développe également l'idée de compte «Pot de miel» («HoneyAccounts»), de faux comptes qui peuvent être facilement attaqués et permettent ainsi d'observer les méthodes de l'attaquant.

Dans Araujo et al. [27], les auteurs proposent de patcher les vulnérabilités d'un système mais de les faire apparaître à l'attaquant non patchées. Ainsi, lorsque l'attaquant effectuera son attaque, il sera possible de le cibler et de le rediriger vers un HoneyPot. De la même manière, Avery et al. [28] propose cette fois-ci de mettre en avant de faux patchs afin de faire croire à l'attaquant l'existence de vulnérabilités qui n'existent pas. Cela permet de ralentir considérablement l'étape de reconnaissance de l'attaquant.

Il existe d'autres techniques de leurrage informatique [8,9,29]. L'article de Fraunholz et al. [8] est une étude de 2014 offrant un aperçu des méthodes de leurre existantes à l'époque. Une taxonomie des différentes techniques ainsi que des implémentations est incluse. Han et al. [9] propose une autre étude sur les techniques de leurrage. Il présente les solutions existantes, leur application et leur déploiement. Il se concentre également sur l'évaluation de ces méthodes. Zhang et al. [29] offre une autre taxonomie des stratégies de leurrage basée cette fois sur la phase de la Cyber Kill Chain CKC dans laquelle se trouve l'attaquant ainsi que la couche sur laquelle a lieu le leurrage (réseau, système, logicel ou données). Une partie est consacrée à un type de leurrage particulier : les stratégies MTD.

2.2 Stratégies de Moving Target Defense

Cette section décrit différentes stratégies MTD proposées dans la littérature. Comme vu précédemment, l'objectif des stratégies MTD est de supprimer l'avantage du temps que possède l'attaquant à cause de la nature statique des infrastructures informatiques. Ces stratégies peuvent s'appliquent à différents niveaux, principalement au niveau du réseau et au niveau logiciel.

Différents paramètres d'un réseau peuvent être modifiés afin de leurrer un adversaire. Une première approche consiste à modifier les adresses IP des machines du réseau, cette stratégie s'appelle Random Host Mutation [30–32]. Dans Al-Shaer et al. [30], les auteurs considèrent un réseau comprenant des noeuds réels ainsi que des noeuds leurres. L'attaquant doit déterminer quels noeuds sont réels tandis que le défenseur utilise un hyperviseur cherche à déterminer la période optimale pour randomiser les adresses IP. Dans Clark et al. [31], une mutation aléatoire et imprévisible des adresses IP des hôtes est utilisée dans un Software Defined Network (SDN). Cela permet d'invalider jusqu'à 99% des informations obtenues par des scanners externes. Clark et al. [31] fait également varier les adresses IP des machines dans un réseau mais le fait de manière totalement décentralisée.

Dans Luo et al. [33], les auteurs ne proposent plus de faire varier les adresses IP mais plutôt les ports exposés des machines. Cela réduit le taux de réussite des attaques de 37%. Cette

technique est particulièrement efficace dans la phase de reconnaissance de l'attaquant mais l'article montre qu'elle peut également être utilisée pour réduire l'impact d'autres phases d'attaques telles que l'exploitation de vulnérabilités ou l'obtention d'un accès sur la machine cible. Chavez et al. [34] met en avant une stratégie MTD utilisant à la fois le mélange des adresses IP, celui des ports ainsi que celui des chemins dans le réseau. Il montre que la randomisation des ports est la stratégie qui a le moins d'impact sur les performances du réseau tout en protégeant contre les attaques sur la couche application du modèle Open Systems Interconnection (OSI). Faire varier les adresses IP a un coût plus élevé mais protège contre les attaques des couches inférieures à la couche Transport du modèle OSI. La randomisation des chemin d'accès dans le réseau ne devrait pas être utilisée car elle entraîne une augmentation très importante du délai.

Dans un autre registre, Boyd et al. [35] propose de randomiser les jeux d'instructions dans une application. Grâce à cela, ils réussissent à contrer des attaques par injection de code. Ils testent leur proposition sur différents prototypes mais surtout sur SQL où leur solution est portable et n'engendre qu'une faible surcharge [36]. De façon similaire, Taguinod et al. [37] étudie la manière dont les stratégies MTD peuvent être intégrées dans les applications Web. Deux approches sont envisagées : changer l'implémentation du langage ou changer l'implémentation de la base de données.

Il existe de nombreuses autres techniques MTD pour protéger un réseau ou une infrastructure informatique. Zheng et al. [13] est une étude regroupant et comparant des stratégies MTD. Elle se concentre sur la partie du système que ces stratégies protègent et les classe en fonction de la nature des techniques utilisées : MTD basée sur la virtualisation, MTD basée sur un SDN ou MTD basée sur le leurre. Sengupta et al. [12] est une autre étude se concentrant cette fois sur les stratégies MTD au niveau du réseau. Elle les comparent sur différents aspects, ce qu'elles modifient c'est-à-dire la surface sur laquelle elles agissent (exploration, attaque, détection, prévention), quand elles se déplacent (changement à des intervalles constants ou variables) et comment elles se déplacent c'est-à-dire la modélisation utilisée pour obtenir la stratégie de déplacement. Une comparaison de l'état des implémentations est réalisée ainsi que des métriques utilisées pour comparer les performances est également faite.

2.3 Modélisation de la confrontation attaquant/défenseur

2.3.1 Modélisations basées sur la théorie des jeux

Pour modéliser le leurrage, la perception des différents acteurs doit être considérée. Une approche courante consiste à utiliser la théorie des jeux pour modéliser les interactions entre

les joueurs. Il existe plusieurs types de jeux pour y parvenir. Nous présentons les types de jeux les plus courants et certains de leurs usages dans le contexte de la protection d'un système informatique utilisant du leurrage. Pawlick et al. [10] offre une taxonomie des stratégies de leurrage vues sous l'angle de la théorie de jeux. Ses auteurs classent les techniques de leurrage en six catégories tout en proposant une formalisation de concepts permettant de standardiser la définition du leurrage défensif. Des contributions utilisant la théorie des jeux pour modéliser différents types de leurrage y sont résumées. Zhu et al. [38] est une autre étude des stratégies de leurrage défensives basées sur la théorie des jeux. Elle souligne également les articles utilisant de l'Apprentissage Machine (ML) pour trouver une solution optimale. De la même manière, dans son étude sur les stratégies MTD [11], l'auteur accorde une partie entière à l'utilisation de la théorie des jeux pour modéliser et optimiser l'utilisation des stratégies MTD.

Un premier type de jeu mentionné dans la littérature est le jeu de Stackelberg. Il s'agit d'un type de jeu dans lequel un joueur est le leader, souvent le défenseur, et l'autre joueur est le suiveur, souvent l'attaquant. Le leader choisit ses actions en premier, ce qui est observé par le deuxième joueur qui joue ensuite. Clark et al. (2012) [17] utilise les jeux de Stackelberg pour étudier une défense contre du brouillage. Ils modélisent les interactions entre un défenseur et un brouilleur dans un jeu en deux étapes. Le défenseur va générer un faux trafic. Le défenseur cherchera à maximiser le débit tout en minimisant le retard et l'attaquant choisira la fraction de chaque flux à brouiller. L'existence d'un équilibre de Stackelberg à stratégie pure est démontrée par les auteurs. Clark et al. (2015) [18] propose d'utiliser un jeu de Stackelberg afin d'analyser les interactions entre un attaquant et un réseau avec des noeuds leurres. L'attaquant cherche à identifier les noeuds réels en examinant les temps de réponse des noeuds et les protocoles utilisés. Le défenseur choisit quand randomiser les adresses IP des appareils du réseau. Ce jeu admet également un équilibre de Stackelberg unique. Feng et al. [19] modélise les interactions défenseur-attaquant en combinant un jeu de Stackelberg avec un Processus de Décision Markovien (MDP). Le défenseur peut modifier la configuration d'un ressource tandis que l'attaquant cherche à identifier la vraie configuration de celle-ci. Au début de chaque tour, le défenseur choisit une stratégie MTD pour plusieurs périodes et l'attaquant peut attaquer une ressource si son état connu est valide. Un algorithme est conçu pour choisir la meilleure stratégie dans le pire des cas. Sengupta et al. [39] propose un jeu de Stackelberg bayésien pour modéliser les interactions entre un attaquant qui peut être de différents types et un système utilisant des stratégies MTD. Un algorithme de Q-Learning est utilisé pour trouver une solution optimale.

Un deuxième type de jeu souvent utilisé pour modéliser les intéractions entre un attaquant et un défenseur dans un système informatique est le jeu de signalisation (Signaling Game).

Il s'agit d'une classe de jeux à deux joueurs à information incomplète. Un premier joueur effectue une action et transmet une information à l'autre joueur avec un certain coût qui sera plus élevé si l'information est fausse. Le second joueur ne connaît pas la nature de l'information reçue (vraie ou fausse) mais choisit une action en fonction de celle-ci. Dans Pawlick et al. (2015) [20], un jeu de signalisation est utilisé pour analyser l'efficacité du déploiement d'HoneyPots dans un réseau informatique. Ils montrent notamment que parfois l'utilité du défenseur peut augmenter lorsque l'attaquant est capable de détecter le leurrage. La et al. [22] propose l'utilisation d'un jeu de signalisation pour modéliser les interactions entre un défenseur et un attaquant dans un réseau permettant le déploiement d'HoneyPots. Cette fois, c'est l'attaquant qui joue en premier en choisissant parmi plusieurs types d'actions, des actions normales ou des attaques, tandis que le défenseur joue en second et peut utiliser des HoneyPots pour leurrer l'attaquant. Les auteurs de Çeker et al. [21] utilisent les jeux de signalisation pour modéliser les interactions entre un attaquant et un défenseur dont le but est de protéger un serveur d'attaques Déni de Service (DOS) tout en fournissant un service aux utilisateurs légitimes. Le défenseur a la capacité de camoufler un système normal comme étant un HoneyPot et réciproquement. Rahman et al. [40] met en avant un mécanisme pour limiter la prise d'empreintes numériques et l'identification à distance des systèmes d'exploitation. Il utilise le même type de jeux que précédemment pour modéliser les interactions entre l'attaquant cherchant à identifier la nature de sa cible et cette dernière. Dans les jeux de signalisation, le leurrage peut être utilisé mais ne peut pas vraiment être détecté. Dans Pawlick et al. (2018) [41], les auteurs étendent le jeu de signalisation avec un détecteur fournissant une probabilité que du leurrage soit utilisé. Un résultat surprenant de cette étude est que parfois l'auteur du leurrage obtient de meilleures performances lorsque le détecteur est plus performant.

Les jeux stochastiques sont un type de jeu à plusieurs états. Le jeu va se jouer comme une suite d'état. Dans chacun d'eux, chaque joueur va choisir une action ce qui aura un impact sur le nouvel état du système. Manadhata et al., Anwar et al. et Horák et al. [42–44] utilisent ce type de jeu pour étudier l'impact d'une stratégie défensive de leurrage sur la perception de l'attaquant dans différents environnements. Manadhata et al. [42] s'intéresse au changement de la surface d'attaque comme stratégie MTD. La notion du changement de la surface d'attaque est formalisée et les intéractions entre les deux joueurs sont modélisées grâce à un jeu stochastique à deux joueurs. Anwar et al. [43] cherche à placer de manière optimale des honeypots dans un graphe d'attaque afin de ralentir ou d'empêcher l'attaque. Ils s'intéressent au compromis à faire entre le coût du leurrage et son bénéfice pour le défenseur. Horák et al. [44] propose l'analyse d'un leurrage actif face à un adversaire qui cherche à s'infiltrer dans un réseau informatique pour exfiltrer des données ou causer des dégâts. Il

utilise un jeu stochastique avec des observations partielles pour étudier l'impact du leurrage sur la perception de l'attaquant. Ici, il est supposé que le défenseur est capable de détecter la progression de l'attaquant tandis que l'attaquant manque d'information sur le système et est donc vulnérable au leurrage.

Dans Cho et al. [45], un autre type de jeu est utilisé pour modéliser la confrontation d'un attaquant et d'un défenseur. Il s'agit de la théorie des Hyperjeux (*Hypergame* en anglais) [46]. Ce type de jeux permet de considérer la différence de perception des deux joueurs et est souvent utilisé pour analyser et comprendre certaines prises de décisions. Dans cet article, un scénario simple attaquant/défenseur est modélisé grâce à la théorie des Hyperjeux. L'évaluation du modèle permet de comprendre l'influence de la perception des joueurs sur leurs performances.

2.3.2 Sélection des stratégies basée sur la théorie des jeux

Une fois que le jeu est modélisé, le choix des actions des joueurs doit être optimisé. En théorie des jeux, certains outils sont disponibles à cette fin comme l'Equilibre mixte de Nash (MSNE). Il s'agit d'un équilibre dans lequel aucun joueur ne peut améliorer son utilité attendue s'il est le seul à choisir une autre stratégie, car chaque joueur choisit sa stratégie optimale en supposant que l'adversaire fait de même. Ces équilibres correspondent aux solutions optimales lorsqu'on tient compte des décisions des autres joueurs. Cela nécessite une bonne connaissance des stratégies de l'adversaire et de leurs effets.

Hu et al. [47] étudie la sélection de la meilleure contre-mesure pour maximiser le gain de la défense. Cette approche utilise la théorie des jeux et plus précisément la théorie des jeux de signaux. L'équilibre bayésien parfait est calculé et un algorithme est proposé pour la sélection de la stratégie de défense optimale. Des expériences dans un petit environnement sont effectuées pour montrer l'efficacité de l'approche proposée.

Certains articles proposent de trouver l'équilibre de Nash pour optimiser le choix d'une stratégie MTD [48,49]. Zhang et al. [49] étudie la sélection de la stratégie MTD d'un défenseur dans un environnement d'applications web. La confrontation entre l'attaquant et le défenseur se fait par le biais d'un jeu à information incomplète. L'algorithme Nash-Q learning est employé pour trouver cette stratégie optimale et ses performances sont comparées dans cet environnement à celles d'autres algorithmes tels que l'algorithme Minimax-Q Learning ou Naive-Q Learning. Les résultats des expériences montrent l'efficacité de l'algorithme Nash-Q Learning par rapport aux autres algorithmes. Lei et al. [48] utilise un jeu de Markov pour proposer un modèle considérant la nature à plusieurs phases et plusieurs étapes de la confrontation.

Tan et al. [50] font un parallèle entre la transformation MTD et le changement de la surface d'attaque et de la surface d'exploration. Ils proposent un modèle MTD basé sur un modèle de jeu Markovien robuste. Les jeux robustes peuvent réduire la dépendance à la connaissance préalable pour la sélection de la stratégie optimale présente dans d'autres modèles. Comme le montre Lei et al. [48], l'utilisation d'un tel modèle permet de prendre en compte la nature multi-étapes et multi-états de la confrontation. L'existence d'un équilibre robuste et d'une solution optimale pour ce modèle est prouvée sous certaines conditions. Un algorithme est proposé pour les trouver et des simulations sont effectuées pour montrer l'efficacité de cette approche.

2.3.3 Autres optimisations de stratégies

Des articles proposent des modèles qui ne sont pas strictement basés sur la théorie des jeux. Comme les stratégies MTD sont souvent efficaces contre un type spécifique d'attaque, Zhang et al. [51] étudient l'impact de l'utilisation de stratégies MTD avec un ou plusieurs éléments mutables ou mouvants contre plusieurs attaques. Un algorithme basé sur l'Algorithme Génétique est proposé pour trouver la combinaison la plus efficace de stratégies. Il est démontré que même dans un environnement avec des ressources limitées, il est toujours important d'utiliser plusieurs éléments mouvants.

Cependant, à mesure qu'un modèle devient plus complexe, la recherche d'équilibres devient de plus en plus difficile d'un point de vue calculatoire. De plus, les équilibres comme ceux de Nash ne prennent pas en compte le comportement passé des joueurs qui permet souvent de prédire le comportement futur. De plus, le fait que la solution optimale trouvée soit souvent une stratégie mixte rend son utilisation pratique difficile. L'Apprentissage par renforcement profond (DRL) permet d'obtenir une stratégie pure approximative de la meilleure stratégie tout en limitant le coût de calcul. Il permet également de trouver une approximation de la solution optimale dans des modèles qui ne sont pas basés sur la théorie des jeux [52].

DQ-MOTAG [53] propose d'utiliser le DRL pour améliorer MOTAG [54], un mécanisme MTD pour contrer les attaques DOS distribué (DDOS). MOTAG utilise des proxies pour transmettre le trafic entre les utilisateurs légitimes et les serveurs protégés. En utilisant ces proxies, il est capable d'isoler l'attaquant externe des clients innocents tout en mélangeant les affectations client/proxy. DQ-MOTAG fournit une capacité d'ajustement de la période de brassage autonome basée sur l'apprentissage par renforcement. Des expériences montrent l'efficacité de la méthode utilisée.

Eghtesad et al. [55] propose de trouver une stratégie MTD optimale dans le modèle proposé par Prakash et Wellman [56]. Dans ce but, un jeu à deux joueurs entre l'adversaire et le

défenseur est créé. Dans ce modèle, l'attaquant et le défenseur s'opposent pour le contrôle des serveurs, le mécanisme de compromission des serveurs est simplifié. Une représentation compacte des observations passées de chaque joueur est proposée et permet de mieux agir dans l'environnement partiellement observable. Cet article réussit à résoudre ce jeu en utilisant un algorithme de reinforcement learning multi-agent basé sur l'algorithme du Double Oracle qui permet de trouver un équilibre de Nash à stratégie mixte dans ce type de jeux.

Wang et al. [57] étudie la politique de déploiement des ressources de leurrage et surtout la position de ces ressources. Un modèle est proposé pour représenter la confrontation entre l'attaquant et le défenseur et une stratégie de l'attaquant est fournie. Un graphe de la menace de pénétrations (TPG) est utilisé pour présélectionner l'emplacement des ressources de déception. Un algorithme Q-learning est fournit pour trouver la politique de déploiement optimale. Un environnement réseau réel est utilisé pour démontrer l'efficacité de la méthode proposée.

2.4 Conclusion

Dans ce chapitre, nous avons présenté différentes stratégies déceptives existantes, à la fois des stratégies de leurrage et des stratégies MTD. Or, nous cherchons comment conjuguer et optimiser le déploiement de ces stratégies. Pour cela, plusieurs travaux se basent sur la théorie des jeux ce qui permet d'obtenir des résultats théoriques. Cependant, malgré la grande variété de modèles existants et présentés précédemment, son cadre reste limité. En effet, étant généralistes, ces modèles permettent de représenter différents types d'interactions entre deux joueurs mais n'utilisent pas les spécificités d'une confrontation dans un système informatique. Par exemple, il est nécessaire de considérer la différence de nature entre les actions de l'attaquant, entre un scan et l'exploitation d'une vulnérabilité. La plupart des modèles de théorie des jeux se focalise sur l'information et donc sur l'impact des stratégies déceptives sur les scans. Mais il faut aussi considérer l'impact des attaques sur un environnement et sur la perception des joueurs notamment de l'attaquant. En effet, sa perception et ses possibilités d'actions évoluent au cours d'une attaque et les modèles de théorie des jeux ne sont que peu adaptés pour modéliser des environnements et des processus à plusieurs états et à plusieurs étapes. L'impact et l'efficacité des stratégies de leurrage va aussi dépendre des connaissances de l'attaquant, plus l'attaquant a des connaissances sur un système moins les stratégies de leurrage ont de chance d'être un succès.

De plus, ces modèles prennent en compte les stratégies MTD ou de leurrage mais rarement les deux à la fois. Envisager les deux signifie devoir considérer la différence d'impact sur l'environnement et la perception des joueurs de ces actions. Enfin, lorsqu'un attaquant attaque

un système informatique, ses actions suivent une certaine logique ce qui permet de déduire une partie des actions futures grâce à ces actions passées. C'est pour cela que dans le chapitre 3, nous proposons un modèle qui ne se base pas sur une théorie des jeux mais qui permet de considérer les spécificités d'une confrontation dans un système informatique.

CHAPITRE 3 SELECTION DE LA STRATEGIE DEFENSIVE DANS UN MODELE

Lorsque l'on s'adresse au problème d'optimisation de la sélection de stratégie de type leurrage ou MTD, une première approche est la modélisation. Nous allons chercher à modéliser les interactions entre un défenseur et un attaquant dans un environnement informatique.

Notre modélisation doit respecter plusieurs critères. Tout d'abord, nous cherchons à modéliser les interactions entre deux joueurs ayant chacun plusieurs actions possibles. Un problème avec des jeux tels que les jeux de signalement est qu'ils sont adaptés à la modélisation de deux joueurs avec chacun deux actions possibles et qu'ils ne permettent pas la modélisation de davantage d'actions.

Un deuxième critère est l'asymétrie de la perception des joueurs. L'attaquant et le défenseur n'ont pas la même connaissance du système et n'ont pas la même connaissance des actions de l'adversaire. Un troisième critère est le support de plusieurs sources de leurrage comme il pourrait être dans un sytème informatique réel. Certains jeux avec information incomplète prennent en compte l'asymétrie de la perception des joueurs mais ne permettent pas plusieurs sources de leurrage.

Certains jeux bayésiens permettent de prendre en compte le caractère multi-étapes et multi-états d'une confrontation. Cependant, à notre connaissance, aucun de ces jeux ne permet de considérer la différence de nature entre les stratégies défensives de leurrage et les stratégies défensives MTD. Cela constitue deux autres critères que doit respecter notre modèle.

Tous ces critères nous éloignent de l'utilisation d'un modèle générique de théorie des jeux préexistant pour la modélisation de notre confrontation attaquant/défenseur. Nous proposons un modèle plus adapté au type d'environnement que nous voulons modéliser. De ce fait, trouver une solution optimale devient une difficulté d'autant plus que le nombre d'états du modèle peut être très grand. L'utilisation d'algorithme de DRL nous permet de répondre à cette difficulté.

Dans la section suivante (3.1), nous définissons les éléments les plus importants du modèle dont les états du jeux,les observations de chaque joueur, le fonctionnement du jeu ainsi que le système de récompense qui sera utile pour entraîner notre agent défenseur. Dans la section 3.2, nous présentons le scénario de l'attaquant face auquel nous entraînerons l'agent défenseur. Dans la section 3.3 est présenté le processus de sélection des stratégies du défenseur. Les résultats obtenus sont présentés dans la section 3.4 et sont discutés dans la section 3.5.

3.1 Définition du modèle

Dans cette section, nous présentons notre modèle d'un système informatique avec plusieurs vulnérabilités potentielles. Dans notre modèle, les actions permises pour chacun des joueurs sont des actions qui seraient possibles dans un contexte réel : l'attaquant peut donc scanner les différentes vulnérabilités du système ou bien tenter de les exploiter tandis que le défenseur peut déployer des stratégies de leurrage ainsi que des stratégies MTD. La différence de nature de ces actions va entraîner un différence d'effets de celles-ci sur l'environnement et sur la perception des joueurs. Les stratégies MTD vont affecter les connaissances préalables de l'attaquant car une information obtenue par le passé par ce dernier peut ne plus être correcte tandis que les stratégies de leurrage vont fournir des informations erronées aux scans de l'attaquant. De plus, les stratégies de leurrage sont déployées pour plusieurs périodes de temps mais ne le sont pas en permanence car maintenir ce type de stratégies peut avoir un coût élevé.

3.1.1 Environnement

Dans cette partie, nous définissons notre modèle attaquant/défenseur. Trois éléments principaux composerons notre modèle : une machine, un attaquant et un défenseur. L'objectif de l'attaquant est de compromettre la machine tandis que le défenseur cherche à la défendre en utilisant des stratégies MTD et de leurrage. Notre modèle est basé sur les pas de temps. Chaque pas de temps est représenté par une étape dans laquelle chaque joueur pourra choisir d'exécuter une action.

La machine

La machine sera modélisée par deux éléments : un vecteur de vulnérabilités potentielles V et un score de compromission C. Nous considérons un certain nombre de vulnérabilité de différentes natures. Certaines sont accessibles depuis l'extérieur de la machine et d'autres ne sont accessibles qu'avec un accès utilisateur à la machine et permettent une élévation de privilèges. De plus, chaque vulnérabilité a une certaine probabilité d'être présente sur la machine. Pour chaque vulnérabilité potentielle v, si celle-ci est présente sur la machine, $V[v] = 1$ sinon $V[v] = 0$. Le score de compromission C est une valeur entre 0 et 1. Si la machine n'est pas compromise alors ce score C est à 0. Ce dernier évoluera au cours du jeu jusqu'à atteindre une valeur de 1 lorsque la machine sera totalement compromise.

L'attaquant

Un premier joueur dans le modèle est l'attaquant dont l'objectif est de compromettre la machine. Pour cela, il a plusieurs action disponibles : un scan global, des scans ciblés et des attaques. Un scan ciblé fournit de l'information sur une unique vulnérabilité tandis qu'une attaque correspond à une tentative d'exploitation de cette dernière. A chaque vulnérabilité dans V, présente ou non sur la machine, est associé un score DI indiquant les dommages d'informations de cette action et un score DC indiquant ses dommages de compromission. Les dommages de compromission d'un scan sont nuls tandis que les dommages d'informations sont plus importants pour une action de type scan plutôt que pour une action de type attaque. Plus le score DI pour une action contre une certaine vulnérabilité est élevé, plus il sera rapide pour l'attaquant d'acquérir les informations nécessaires à son exploitation avec une attaque. A l'opposé, si une vulnérabilité n'est pas présente sur la machine, un score élevé pour une action éliminera plus rapidement cette vulnérabilité des vulnérabilités potentielles sur la machine.

Le scan global permet à l'attaquant d'obtenir de l'information sur plusieurs vulnérabilités potentielles. La quantité d'information obtenue pour chaque vulnérabilité est plus faible que si le scan avait été ciblé, c'est à dire que son score DI sera plus faible. Dans la réalité, ce scan pourrait être vu comme un scan Nmap [58].

On associe également à chaque action une valeur $Cost$ indiquant le coût de l'attaquant pour utiliser celle-ci. Ce coût intègre plusieurs caractéristiques de l'action comme sa durée, sa difficulté, etc.

D'autres variables sont associées à l'attaquant et permettent de mesurer les progrès de l'attaquant dans la compromission de la machine. Nous définissions deux vecteurs pV et $perceivedpV$ contenant, respectivement, pour chaque vulnérabilité v dans V, un score mesurant la quantité d'information réelle que possède l'attaquant sur cette vulnérabilité et un score mesurant la quantité d'information que l'attaquant pense avoir sur celle-ci.

Si $perceivedpV[v] = 0.5$, l'attaquant n'a pas d'information sur la vulnérabilité v. Ensuite, plus $perceivedpV[v]$ sera proche de 1, plus l'attaquant possédera de l'information sur la vulnérabilité v et plus il croira que celle-ci est présente sur la machine. A l'opposé, plus $perceivedpV[v]$ sera proche de 0, plus l'attaquant possédera de l'information sur la vulnérabilité v mais plus il croira que celle-ci n'est pas présente sur la machine. $perceivedpV$ et pV ne sont pas nécessairement égaux. Si l'attaquant a été leurré alors ces deux vecteurs pourront ne plus être égaux. Comme pV contient la quantité d'information réelle possédée par l'attaquant sur les vulnérabilités, si la machine est vulnérable à une vulnérabilité v alors $pV[v]$ sera entre 0.5 et

1 et si elle ne l'est pas alors $pV[v]$ sera entre 0 et 0.5.

Une autre variable permet de définir l'avancement de l'attaquant dans la compromission de la machine, il s'agit de la phase de la CKC dans laquelle il se trouve. Celle-ci peut prendre trois valeurs (0, 1 et 2) car nous décidons de considérer seulement les principales phases de la CKC : Reconnaissance, Intrusion et Élévation de privilèges/ Exploitation. Les actions disponibles de l'attaquant dépendront de la phase dans laquelle il se trouve. Ce paramètre sera mis à chaque étape du jeu mais nous reviendrons sur cela dans la section 3.2 où nous détaillerons le scénario de l'attaquant.

Finalement, deux autres variables mesure la quantité globale d'information que possède l'attaquant sur la machine. La première est l'Information Système I_a (Voir l'équation 3.1) qui mesure la quantité globale d'information que possède l'attaquant sur les vulnérabilité accessibles sur la machine. Cela signifie que dans les phases 0 et 1 de la CKC, les vulnérabilités ne pouvant être utilisé que dans la phase 2 ne sont pas considérées dans le calcul du score I_a. La seconde variable est la Perception d'Information Utile Iu_a (Voir l'équation 3.2) qui mesure la quantité d'information utile aux attaques perçue par l'attaquant sur les vulnérabilités observables de la machine. Ce score ne prend donc en compte que les vulnérabilités que l'attaquant pense présentes sur la machine i.e. les vulnérabilités v telles que $perceivedpV[v] > 0.5$.

$$I_a = \frac{2}{K} \sum_{v \in V} max(0; 0.5 - |V[v] - pV[v]|)) \tag{3.1}$$

$$Iu_a = \frac{2}{K} \sum_{v \in V} max(0; perceivedpV[v] - 0.5) \tag{3.2}$$

où K est le nombre de stratégies d'attaques disponibles dans le modèle.

Le défenseur

Le second joueur dans le modèle est le défenseur. Son objectif est d'empêcher la machine d'être compromise ou du moins de ralentir sa compromission tout en minimisant son coût pour arriver à cela. Parmi les actions possibles pour le défenseur, il y a les stratégies MTD et les stratégies de leurrage. Comme utiliser ces stratégies a un coût, le défenseur peut également choisir de ne jouer aucune action. Les stratégies MTD vont influencer la connaissance déjà acquise par l'attaquant. En effet, si l'attaquant a collecté des informations sur le système et que le défenseur utilise une stratégie MTD alors une partie de l'information acquise par l'attaquant deviendra incorrecte. A l'opposé, les stratégies de leurrage ne modifie pas la validité des informations précédemment obtenues par l'attaquant. Elles fournissent à l'attaquant de

nouvelles informations erronées. Si l'attaquant obtient des informations sur la machine dans les étapes suivant l'utilisation d'une stratégie de leurrage, ces informations seront erronées et pourront donc leurrer l'attaquant.

A chacune des actions possibles pour le défenseur, nous associons un coût considérant la surcharge d'utilisation logicielle ou matérielle nécessaire à son utilisation.

Paramètres relatifs aux deux joueurs

D'autres paramètres sont nécessaire pour définir notre modèle. Tout d'abord, pour chaque action du défenseur, nous devons définir son efficacité Eff contre chaque stratégie de l'attaquant. Cette valeur inclut à la fois la capacité de la stratégie du défenseur d'empêcher l'exploitation d'une vulnérabilité et sa capacité à empêcher l'obtention information sur celle-ci. Nous reviendrons sur cela dans la section 3.4.1.

Dans notre modèle, le défenseur ne détecte pas forcément toutes les attaques et scans de l'attaquant. Il les détecte avec une probabilité $P_{Detection}$. Pour simplifier, cette valeur est la même pour toutes les actions et doit être spécifiée pour initialiser le modèle. De plus, lorsque l'attaquant effectue une action, le défenseur peut détecter avec la probabilité $P_{Detection}$ quelle vulnérabilité a été visée mais il n'est pas capable de savoir s'il s'agissait d'un scan ou d'une attaque. Cela va affecter les observations du défenseur.

Nous considérons également que les stratégies de leurrage n'atteigne pas forcément leurs objectifs. La probabilité de succès du leurrage (Voir l'équation 3.3) dépend d'informations que détient l'attaquant sur le système ainsi que la phase de la CKC dans laquelle il se trouve :

$$P_{Deception} = exp(-\lambda_{Deception} \cdot (CKC + 1) \cdot (I_a + \Delta_{Deception})) \tag{3.3}$$

Les attaques peuvent également échouer. La probabilité de succès d'une attaque A visant une vulnérabilité v de la machine lorsque le défenseur utilise une stratégie D dépend de la quantité de connaissances de l'attaquant sur v et de l'efficacité de la stratégie défensive D contre l'attaque A (Voir l'équation 3.4). Si une vulnérabilité est absente de la machine alors la probabilité de succès de l'attaque est nulle.

$$P_{Attack}(A, D) = V[v] \cdot exp(-\frac{\lambda_{Attack} \cdot (Eff(A, D) + \Delta_{Attack})}{max(0.01; (pV[v] - 0.5) \cdot 2)}) \tag{3.4}$$

où A est la stratégie de l'attaquant, v la vulnérabilité visée et D la stratégie défensive.

$\lambda_{Deception}$, λ_{Attack},$\Delta_{Deception}$ et Δ_{Attack} sont des paramètres du modèle permettant de calibrer

les probabilités ci-dessus afin de les rendre plus cohérents avec l'environnement réel que nous voulons modéliser.

3.1.2 Etats du jeu

Notre modèle utilise une échelle de temps discrète. Le jeu se déroule comme une succession d'étapes. A chaque étape, le jeu est caractérisé par son état. A une étape donnée t, l'état du jeu est :

$$s^t = \langle V, C, pV, CKC \rangle \tag{3.5}$$

Tous ces paramètres ont été définis précédemment. L'état du jeu à un instant t est donc caractérisé par le vecteur de vulnérabilités V de la machine, son score de compromission C, le vecteur pV contenant la quantité d'information réelle que possède l'attaquant sur chaque vulnérabilité ainsi que la phase de la CKC dans lequel se trouve l'attaquant.

3.1.3 Observations

Pour chacun des joueurs, l'attaquant et le défenseur, nous allons définir les observations. En effet, comme expliqué précédemment, chaque joueur aura ses propres observations qui représentent sa propre perception de l'état du jeu. Ces observations sont utilisées à chaque étape du jeu pour sélectionner les actions à prendre.

L'état du jeu perçu par l'attaquant est défini par le tuple O^a :

$$O^a = \langle perceivedpV, C \rangle \tag{3.6}$$

Ces observations fournissent à l'attaquant à la fois des informations sur les vulnérabilités de la machine et sur l'état de compromission de la machine. Nous considérons que le leurrage ne portera pas sur l'état de la compromission de la machine mais se concentrera sur les informations nécessaires aux attaques. Cela explique que le score de compromission de la machine fasse partie des observations de l'attaquant.

De l'autre côté, le défenseur possède des informations sur les différentes vulnérabilités qui ont été visées par l'attaquant par le passé si les attaques ont été détectées. De plus, il possède aussi des informations sur ses propres stratégies de défense utilisées dans le passé. L'état du

jeu perçu par le défenseur est défini par le tuple O^d :

$$\begin{aligned} O^d = \langle & last\_attacks, last\_defenses, \\ & times\_since\_last\_attack, \\ & times\_since\_last\_defense, \\ & attack\_counts \rangle \end{aligned} \tag{3.7}$$

où

- $last\_attacks$ est un vecteur contenant pour les k dernières étapes, les vulnérabilités visées par l'attaquant si elles sont détectées. Les vulnérabilités sont représentées par de l'encodage un parmi n (One-Hot encoding) c'est à dire que tous les éléments du vecteur sont nuls sauf un qui est à un pour la vulnérabilité visée par l'attaquant.
- $last\_defenses$ est un vecteur contenant pour les k dernières étapes, les stratégies défensives utilisées représentées par de l'encodage un parmi n.
- $times\_since\_last\_attack$ est un vecteur contenant pour chaque vulnérabilité le nombre d'étapes passées depuis la dernière action de l'attaquant (scan ou attaque) qui la visait.
- $times\_since\_last\_defense$ est un vecteur contenant pour chaque stratégie défensive le nombre d'étapes passées depuis sa dernière utilisation.
- $attack\_counts$ est un vecteur contenant pour chaque vulnérabilité le nombre total d'actions détectées de l'attaquant la visant.

k est un paramètre devant être défini pour le modèle. Il représente le nombre d'étapes prises en compte dans les vecteurs d'observations $last\_attacks$ et $last\_defenses$.

3.1.4 Déroulement d'une partie

Dans cette section, nous allons expliquer le déroulement d'une partie. Comme expliqué précédemment, une partie est composée d'une succession d'étapes. Pour expliquer le déroulement d'une étape, nous introduisons la fonction de collecte d'informations (Voir l'algorithme 1). Cette fonction décrit l'impact pour le modèle d'une utilisation d'une stratégie de collecte d'information de l'attaquant. C'est cette fonction qui mettra à jour l'état du modèle et qui modifiera les informations détenues par l'attaquant sur les vulnérabilités en considérant les stratégies (leurrage ou MTD) mises en place par le défenseur et leurs impacts.

Cette fonction est appelée à différents moments du jeu comme par exemple lorsque l'attaquant effectue un scan ciblé, un scan global ou une attaque qui échoue. En effet, nous considérons que même lorsqu'une attaque échoue, l'attaquant obtient quand même des informations sur la vulnérabilité ciblée.

Cette fonction de collecte d'informations prend en paramètre la vulnérabilité ciblée par l'attaquant et μ un paramètre quantifiant l'impact potentiel de la collecte d'information sur l'information détenue par l'attaquant concernant la vulnérabilité visée. En effet, plus μ est grand, plus pV et $perceivedpV$ seront modifiés par la collecte d'informations. Cette fonction s'assure aussi que les contraintes sur les valeurs contenues dans pV et $perceivedpV$ décrites dans la partie sur l'attaquant de la section 3.1.1 sont respectées.

Algorithm 1 Fonction de collecte d'information

Input Attacker Action A_A, μ
v=vulnerability targeted by A_A
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**
      $newpV\_value \leftarrow pV[v] - (V[v] - pV[v]) \cdot \mu \cdot Eff(D, A_A)/2$
      if $V[v] = 1$ **then**
        $pV[v] \leftarrow max(1/2; newpV)$
      else
        $pV[v] \leftarrow min(1/2; newpV)$
      end if
      $perceivedpV[v] \leftarrow perceivedpV[v] - (V[v] - perceivedpV[v])\mu \cdot Eff(D, A_A)/2$
    else
      $newpV \leftarrow pV[v] + (V[v] - pV[v]) \cdot \mu \cdot DI[A_A] \cdot 3$
      if $V[v] = 1$ **then**
        $pV[v] \leftarrow max(1/2; newpV)$
      else
        $pV[v] \leftarrow min(1/2; newpV)$
      end if
      $perceivedpV[v] \leftarrow pV[v]$
    end if
  end if
end for
if no efficient deception strategy is used **then**
  $pV[v] \leftarrow pV[v] + (V[v] - pV[v]) \cdot \mu \cdot DI[A_A] \cdot 3$
  $perceivedpV[v] \leftarrow pV[v]$
end if

Cette fonction de collecte d'informations commence par regarder si une action défensive de leurrage efficace individuellement contre l'action de l'attaquant est utilisée. Les stratégies MTD ne sont pas prises en compte ici car celles-ci agissent sur les connaissances collectées dans le passé par l'attaquant et non sur les nouvelles observations. Nous reviendrons sur cela lorsque nous décrirons le déroulement complet d'une étape. Si une action défensive de

leurrage efficace contre l'action de l'attaquant est utilisée alors nous calculons la probabilité que le leurrage soit un succès. Si c'est un succès, alors la connaissance réelle de l'attaquant sur cette vulnérabilité diminue tout comme sa perception de celle-ci. Si le leurrage est un échec ou qu'aucune stratégie efficace de leurrage n'est utilisée, alors la connaissance réelle de l'attaquant sur cette vulnérabilité augmente et sa perception de celle-ci correspond à la réalité.

Algorithm 2 Déroulement d'une étape du jeu

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 Fonction de collecte d'information with A_A, and μ
end if
if A_A is GlobalScan **then**
  $\mu \leftarrow 1/4$
  for each available Scan A **do**
    Run Fonction de collecte d'information with A_A, and μ
  end for
end if
if A_A is Attack **then**
  Compute Attack Success Probability
  if Attack is Successful **then**
    v=vulnerability targeted by A_A
    $C \leftarrow C + DC[A_A]$
    $V[v], pV[v], perceivedpV[v] \leftarrow 0$
    $C_A \leftarrow C$
  else
    $\mu \leftarrow 1/4$
    Run Fonction de collecte d'information with A_A, and μ
  end if
end if

L'algorithme 2 fournit le pseudocode du déroulement d'une étape de jeu étant donné l'action choisie par le défenseur D_A et par l'attaquant A_A. Tout d'abord, si la stratégie du défenseur est une stratégie MTD, alors une partie des informations collectées par l'attaquant sur les

vulnérabilités affectées par l'action défensive n'est plus correcte, donc la quantité d'information réelle détenue par l'attaquant sur ces vulnérabilités va diminuer. Cependant, à ce stade là, l'attaquant n'est pas conscient que les informations collectées précédemment ne sont plus correctes donc le score mesurant la quantité d'information que l'attaquant pense avoir sur ces vulnérabilités ne va pas être modifiée c'est à dire que $perceivedpV$ restera inchangé.

Ensuite si l'action de l'attaquant est un scan ciblé, la fonction de collecte d'information est utilisé avec un μ assez élevé, c'est-à-dire un fort impact potentiel de la collecte d'informations. S'il s'agit d'un scan global, alors nous faisons de même mais pour toutes les vulnérabilités disponibles et avec un μ plus faible.

Si l'action n'est pas un scan alors il s'agit d'une attaque. La probabilité de succès de celle-ci est calculée. Si l'attaque est un succès, les valeur V, pV, $perceivedpV$ et C sont mises à jour. Le score de compromission C va croître,la vulnérabilité n'est plus considérée comme présente sur la machine et les connaissances de l'attaquant sur cette vulnérabilité sont rendues nulles pour que l'attaquant ne cherche plus à exploiter celle-ci. Si l'attaque est un échec, la fonction de collecte d'information est utilisée avec un μ plutôt faible car même si l'attaque est un échec, l'attaquant obtient des informations sur la vulnérabilité.

3.1.5 Le système de récompense

Pour optimiser le choix de la stratégie du défenseur à chaque étape, nous devons définir une fonction d'utilité U_D qui spécifie l'objectif du défenseur. Son objectif est de rendre la compromission de la machine la plus longue possible et donc de minimiser les informations détenues par l'attaquant sur celle-ci ainsi que le niveau de compromission de celle-ci. U_D est défini comme suit :

$$U_D = w \cdot (1 - I_A) + (1 - w) \cdot (1 - C) \tag{3.8}$$

w est un paramètre du modèle permettant de donner plus d'importance à l'information ou à la compromission dans la fonction d'utilité du défenseur. Plus l'attaquant a des connaissances sur le système et ses vulnérabilités, plus I_A est grand et donc moins la stratégie du défenseur aura été efficace et donc moins son utilité sera grande. De même, plus le niveau de compromission C de la machine sera élevé, moins l'utilité de la stratégie du défenseur sera grande.

Le coût de chaque stratégie a également son influence. La récompense obtenue par le défenseur au temps t est donnée par :

$$r_D^t = U_D - Cost[D] \tag{3.9}$$

où U_D est l'utilité du défenseur et $Cost[D_A]$ le coût de l'action D du défenseur utilisée à l'étape t.

Le défenseur cherche à maximiser la récompense totale sur une partie entière.

3.2 Scénario de l'attaquant

Maintenant que nous avons présenté notre modèle attaquant/défenseur, nous allons présenter le scénario de notre attaquant. En effet, nous allons entraîner notre défenseur à réagir aux actions de l'attaquant afin de rendre l'impact de l'attaque le plus faible possible et la compromission de la machine la plus longue possible. Nous avons donc besoin d'un scénario d'attaque pour l'attaquant. Ce dernier est basé sur une version simplifiée de la CKC. Seulement les trois étapes principales de la CKC sont considérées, ce sont les mêmes que celles utilisées pour le modèle (voir la partie sur l'attaquant de la section 3.1.1) : Reconnaissance, Intrusion et Exploitation/Élévation de privilèges.

3.2.1 Déroulement du scénario

Pour compromettre la machine, l'attaquant devra passer par chacune des phases de la CKC. Les actions disponibles pour l'attaquant vont dépendre de la phase dans laquelle il se trouve. En effet, par exemple, si l'attaquant se trouve dans la troisième phase de la CKC, cela signifie qu'il a infiltré la machine et qu'il peut potentiellement tenter d'exploiter de nouvelles vulnérabilités accessibles uniquement de l'intérieur de la machine. Le passage d'une phase à l'autre de la CKC nécessite certaines conditions préalables qui, pour cet exemple de scénario, peuvent être trouvées dans le Tableau 3.1. Les poids définis dans ce tableau seront nécessaire pour la définition de la fonction d'utilité de l'attaquant qui sera défini dans la section 3.2.2. L'explication de la valeur des poids par rapport à la phase de la CKC s'y trouve également.

L'attaquant n'a donc besoin d'aucun pré-requis pour être dans la phase de reconnaissance. Il s'agit de la phase initiale de l'attaquant au début d'une partie. Ensuite, lorsque l'attaquant a obtenu assez d'information utile sur les vulnérabilités pour tenter de les exploiter, il va pouvoir passer à la phase d'intrusion. L'attaquant restera dans cette phase jusqu'à qu'il ait réussi à exploiter une vulnérabilité pour s'introduire dans la machine. Une fois cela fait, il sera dans la phase d'exploitation ou d'élévation de privilèges.

Phases de la CKC	Pré-requis	Poids
Reconnaissance	Ø	$w1 = 0.2$ $w2 = 0.6$ $w3 = 0.2$
Intrusion	$Iu_a > 0.3$ OU n'importe quel $perceivedpV[i] > 0.8$	$w1 = 0.2$ $w2 = 0.4$ $w3 = 0.4$
Exploitation OU Élévation de privilèges	$C > 0.3$	$w1 = 0.4$ $w2 = 0.1$ $w3 = 0.5$

Tableau 3.1 Pré-requis et poids de la fonction d'utilité pour chaque phase de la CKC

3.2.2 Sélection de la stratégie de l'attaquant

A chaque étape de la partie, l'attaquant devra choisir une action parmi celles disponibles. Pour ce faire, nous utilisons une fonction très utilisée en théorie des jeux, appelée fonction d'utilité. Cette fonction associe à chaque action disponible une utilité représentant la satisfaction du joueur à choisir cette action. L'attaquant choisira donc l'action qui maximise cette fonction.

Dans notre cas, la satisfaction dépend de l'objectif de l'attaquant et donc de la phase de la CKC dans laquelle il se trouve. En effet, par exemple, dans la phase de reconnaissance, c'est la collecte d'informations et donc le scan global ou les scans ciblés qui sont favorisés.Dans la phase d'intrusion, l'attaquant a déjà des informations sur les vulnérabilités potentielles et ce sont donc les attaques qui sont favorisées. Dans la dernière phase, l'attaquant découvre de nouvelles vulnérabilités potentielles internes à la machine, il va donc chercher à obtenir de l'informations sur celles-ci et à les exploiter. Dans ce but, certains paramètres de la fonction d'utilité vont dépendre de la phase de la CKC dans laquelle se trouve l'attaquant (voir Tableau 3.1).

La fonction d'utilité n'est pas la même selon le type d'action de l'attaquant (voir tableau 3.2). Chaque paramètre influence la fonction d'utilité d'une manière différente. En effet, plus $w2$ est grand, plus les dommages d'informations seront valorisés dans l'utilité et donc plus les scans seront mis en avant. Plus le paramètre $w3$ est grand, plus les dommages liés à la compromission seront favorisés dans l'utilité et donc plus les attaques seront favorisées. Plus $w1$ est grand, plus l'attaquant tentera de combler son manque d'information sur le système global. Dans ce cas, c'est plutôt le scan global qui sera mis en avant. L'utilité du scan global est une fraction de la somme de l'utilité de chacun des scans ciblés disponibles. En effet, ce dernier permet d'obtenir de l'information sur chacune des vulnérabilités scannées mais moins que si le scan avait été ciblé.

Action	Fonction d'utilité de l'attaquant $U_A(A_A)$
Scan	$w1 \cdot (0.5 - \lvert perceivedpV[A_A] - 0.5\rvert) +$ $(w2 \cdot DI[A_A] + w3 \cdot DC[A_A]) \cdot perceivedpV[A_A]$
Attaque	$(w2 \cdot DI[A_A] + w3 \cdot DC[A_A]) \cdot perceivedpV[A_A]$
Scan global	$\sum_{A_i \text{ is Scan}} U(A_i)/2$

Tableau 3.2 Fonction d'utilité de l'attaquant en fonction du type d'action

Le coût de chaque stratégie de l'attaquant va également influencer sa sélection. La récompense donnée à l'attaquant à l'étape t d'une partie est donnée par :

$$r_A^t = U_A - Cost[A_A] \tag{3.10}$$

où U_A est l'utilité du défenseur et $Cost[A_A]$ le coût de l'action A_A utilisée par l'attaquant à l'étape t.

L'attaquant choisira à chaque étape l'action qui maximise la récompense qu'il peut obtenir.

3.3 Optimisation de la sélection de la stratégie défensive dans le modèle face au scénario de l'attaquant

Dans cette section, nous allons présenter l'approche utilisée pour trouver une politique optimale. Une politique est une fonction qui associe à chaque état du jeu l'action que l'agent prendra dans cet état. En raison de la complexité du jeu, du grand nombre d'états possibles et de la nature stochastique des récompenses et des transitions entre états, il est difficile de trouver une politique optimale de manière analytique. C'est pourquoi nous décidons d'utiliser l'algorithme DQN [52]. Pour être en adéquation avec le vocabulaire utilisé en apprentissage par renforcement, nous appellerons épisode le déroulement d'une partie dans le modèle. Tout d'abord, nous définissons le gain au temps t pour le défenseur :

$$G_t = r_D^{t+1} + \gamma r_D^{t+2} + \cdots + \gamma^{T-1} r_D^T \tag{3.11}$$

où nous considérons les épisodes comme terminés, T est la durée de l'épisode et γ est un facteur de dévaluation (également appelé facteur d'actualisation) compris entre 0 et 1 exclus qui permet de prendre en compte les récompenses plus ou moins éloignées dans le futur pour la sélection des actions de l'agent.

Une politique est une fonction qui prend en entrée un état S et renvoie la probabilité d'utiliser

chaque action dans cet état. Nous définissons également la fonction état-action/valeur (Q-fonction) $Q_\pi(s,a)$ qui associe l'espérance de gain futur à l'action a et à l'état s si l'agent suit la politique π :

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{3.12}$$
$$= \mathbb{E}_\pi[r_D^{t+1} + \gamma G_{t+1}|s_t = s, a_t = a] \tag{3.13}$$

Nous cherchons donc la politique maximisant la fonction état-action/valeur tel que la fonction état-action/valeur optimale soit :

$$Q_*(s,a) = max_\pi Q_\pi(s,a) \tag{3.14}$$
$$= \mathbb{E}[r_D^{t+1} + \gamma max_{a'} Q_*(s',a')|s,a,s_{t+1} = s'] \tag{3.15}$$

Cette équation est appelée Equation de Bellman [52].

Dans notre contexte, un état s correspond à la perception de l'état du jeu par le défenseur, c'est-à-dire ses observations O^d.

Dans l'apprentissage par renforcement classique, un tableau est utilisée pour stocker la correspondance entre la paire état, action et leur valeur Q correspondante. Dans l'apprentissage par renforcement profond, un réseau neuronal est utilisé à cette fin. L'entrée du réseau neuronal est l'état, et la sortie est la valeur Q estimée de chaque action dans cet état.

Nous utilisons l'algorithme DQN de Mnih et al. [52] avec un tampon de répétition (replay buffer) et un réseau cible pour entraîner notre modèle.

Cet algorithme utilise de nombreux hyperparamètres qui doivent être définis avant l'apprentissage. Ils se trouvent dans le Tableau 3.3.

Le schéma 3.1 résume le processus d'entraînement de l'agent défenseur ainsi que son fonctionnement avec le modèle et le scénario de l'attaquant.

3.4 Expérimentations

3.4.1 Données pour l'expérimentations

Dans les sections 3.1 et 3.2, nous avons décrit notre modèle ainsi que le scénario de l'attaquant. Plusieurs paramètres sont nécessaires pour paramétrer notre modèle pour les expériences. Tout d'abord, concernant la machine, nous devons définir les vulnérabilités considérées dans

Paramètre	Valeur
Poids w dans l'utilité du défenseur	0.3
$\Delta_{Deception}$	0.3
Δ_{Attack}	0.3
k pour les observations du défenseur	3
Fraction d'exploration	0.1
Intervalle de mise à jour du réseau cible	5000
Couches du réseau de neurones	[256x256]
Taille du tampon de répétition	1000000
Fonction d'activation	Tanh
Fréquence d'entraînement	Un épisode
Taille du batch	32
Taux d'apprentissage	0.0005
Facteur de dévaluation	0.99

Tableau 3.3 Liste des paramètres du modèle et des hyperparamètres DQN

le modèle. Il ne s'agit pas de CVE, mais plutôt de familles de vulnérabilités. Ensuite, pour chacune d'entre elles, nous devons définir les dommages d'information et de compromission ainsi que leur coût dans le cas d'un scan ou d'une attaque. Ces valeurs ont été définies arbitrairement en se basant sur des informations de Mitre CAPEC [59] et Mitre CVE [60]. Le tableau 3.4 montre les données utilisées dans les expériences concernant les vulnérabilités et les stratégies d'attaque. Nous avons sélectionné un ensemble de familles de vulnérabilités connues qui peuvent être séparées en deux catégories : celles accessibles depuis l'extérieur de la machine et celles nécessitant un accès local sur la machine. Pour simplifier, nous considérons pour le moment que les coûts des scans sont les mêmes. Il en va de même pour le coût des attaques.

En ce qui concerne le défenseur, nous considérons huit stratégies de défense possibles : quatre stratégies MTD et quatre stratégies de leurrage. Elles sont présentées dans le tableau 3.5 avec le coût de chaque stratégie.

La stratégie *IP Random* consiste en l'affectation d'une nouvelle adresse IP assignée aléatoirement à la machine. La stratégie *Port Random* modifie les ports utilisés pour les différents services présents sur la machine de manière aléatoire. La stratégie *Rekeying Keys Random* réassigne de nouveaux mots de passe à l'utilisateur de la machine ou à l'utilisateur d'un service (une base de données par exemple) La stratégie *Language Random* est l'une des plus ambitieuses. Elle consiste à développer plusieurs versions d'un même service. L'utilisation de cette stratégie fait transitionner le service d'une version à une autre. Par exemple, pour un site web avec un service d'authentification, cette stratégie peut consister au passage d'une

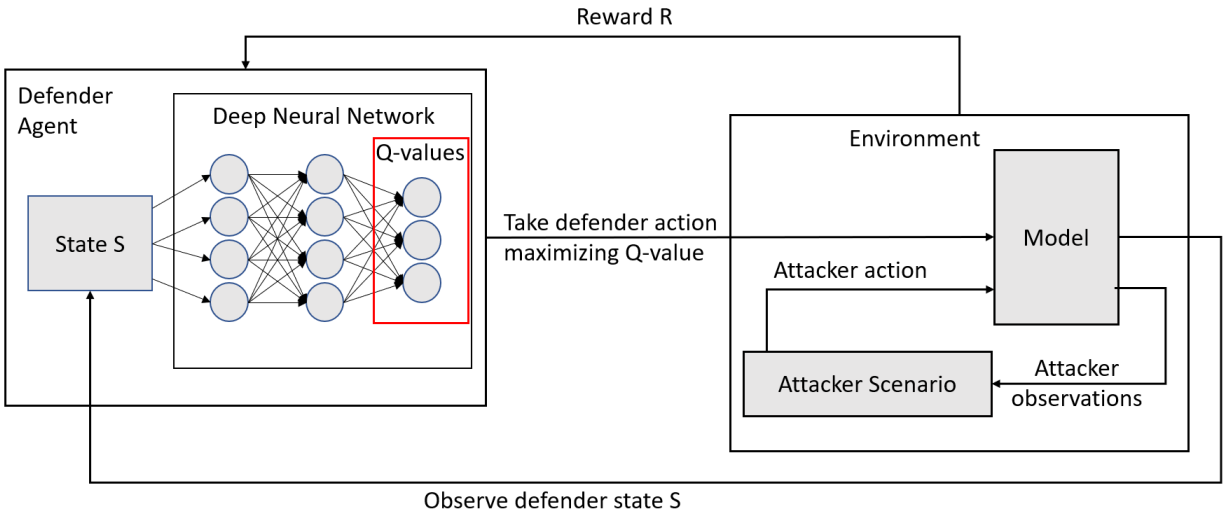

Figure 3.1 Schéma résumant le processus d'entraînement de l'agent défenseur et ses interactions avec le modèle

version apache à une version nginx de la partie frontal du site ou bien au passage d'une base de données mysql à une base de données mongodb pour sa partie authentification.

La stratégie *Honey Service* propose de déployer un faux service comme un faux serveur ssh ou un faux site web pour fournir de fausses informations à l'attaquant. La stratégie *Honey Credentials/Accounts* ajoute de faux utilisateurs à la machine ou aux bases de données. La stratégie *Honey Process* ajoute un faux processus avec des privilèges élevés sur la machine pour leurrer l'adversaire. La stratégie *Honey Files/Logs* ajoute des faux fichiers de configuration, des faux logs ou de de faux fichiers sensibles sur la machine afin de fournir à l'attaquant des informations erronées.

Un autre élément essentiel pour les expériences est le tableau mentionné dans la partie sur les paramètres du modèle de la section 3.1.1 contenant l'efficacité Eff de chaque stratégie défensive contre les actions de l'attaquant. Ces données sont présentées dans le tableau 3.6. Ces paramètres sont choisis arbitrairement en fonction de notre connaissance des différentes stratégies, mais une façon d'améliorer le modèle pourrait être d'utiliser des données provenant d'un environnement réel.

3.4.2 Détails de l'implémentation

Les expérimentations ont été réalisées sur un ordinateur muni d'un processeur CPU Intel Core i7-9750H avec une carte graphique NVIDIA GeForce RTX 2070.

Notre modèle a été implémenté dans un environnement OpenAi Gym [61]. Stable Baselines

Vulnerabilité	Type	Phase de la CKC requise	Coût	DI	DC
Identity Spoofing	Scan	0	0.2	0.3	0
	Attaque	1	0.3	0.15	0.4
Traffic Injection	Scan	0	0.2	0.2	0
	Attaque	1	0.3	0.1	0.4
Brute Force	Scan	0	0.2	0.2	0
	Attaque	1	0.3	0.1	0.4
Command Injection	Scan	0	0.2	0.4	0
	Attaque	1	0.3	0.2	0.4
Code Injection	Scan	0	0.2	0.3	0
	Attaque	1	0.3	0.15	0.4
Privilege Abuse	Scan	2	0.2	0.2	0
	Attaque	2	0.3	0.1	0.6
Authentication Bypass	Scan	2	0.2	0.1	0
	Attaque	2	0.3	0.05	0.6
Privilege Escalation	Scan	2	0.2	0.2	0
	Attaque	2	0.3	0.1	0.6

Tableau 3.4 Liste des vulnérabilités considérées dans le modèle

3 est une librairie python permettant l'implémentation d'algorithmes d'apprentissage par renforcement [62]. Plusieurs algorithmes y sont déjà implémentés. Nous utilisons cette librairie pour entraîner nos agents sur le modèle.

Nous avons cherché à optimiser les hyperparamètres utilisés pour entraîner nos agents. Pour cela, nous avons entraîné le modèle avec de nombreux paramètres différents. Les paramètres du modèle et les hyperparamètres finalement utilisés pour les tests se trouvent dans le Tableau 3.3.

3.4.3 Résultats

Dans cette section, nous allons présenter les résultats que nous avons obtenus. Nous avons entraîné des agents dans plusieurs environnements avec différents paramètres. Nous avons mesuré l'influence des paramètres sur les performances de nos agents entraînés.

La figure 3.2 montre les courbes d'entraînement de différents agents défenseurs pour différentes valeurs de λ_{attack}. Nous pouvons ainsi comparer l'influence de la probabilité de réussite de l'attaque sur les performances des agents. Notre modèle comporte une grande incertitude. En effet, il est totalement stochastique. La même action utilisée deux fois dans le même état peut conduire à des résultats très différents. La récompense et le nouvel état peuvent être

Numéro de la stratégie	Stratégie	Type	Coût
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	Leurrage	0.4
5	Honey Credentials/Accounts	Leurrage	0.3
6	Honey Process	Leurrage	0.2
7	Honey Files/Logs	Leurrage	0.2

Tableau 3.5 Stratégies défensives considérées dans le modèle

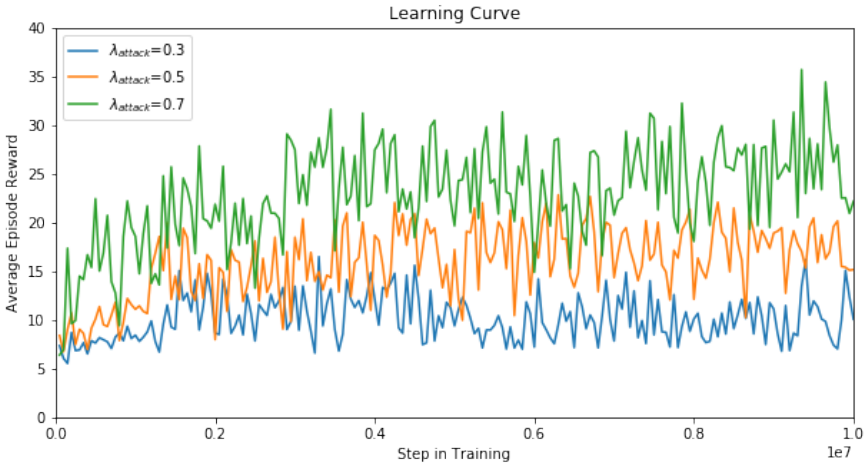

Figure 3.2 Courbes d'apprentissage de différents agents dans des environnements pour différentes valeurs de λ_{attack} et $\lambda_{Deception} = 0.5$, $P_{Detection} = 1$

différents comme ces derniers dépendent beaucoup de la probabilité de détection des actions de l'attaquant ainsi que de la probabilité de leurrage. Cela entraîne donc une grande variance dans nos résultats notamment pour la récompense.

Dans la figure 3.3, nous comparons les performances de notre agent entrainé avec DQN avec les performances des agents utilisant une seule stratégie défensive ou choisissant aléatoirement la stratégie défensive. Nous évaluons chaque stratégie sur 1000 épisodes. Nous comparons la récompense moyenne par épisode pour chaque stratégie mais aussi la longueur moyenne des épisodes.

L'agent entraîné DQN obtient de meilleures performances que les autres stratégies pouvant être considérées comme plus "naïves", tant en termes de récompense que de durée des épisodes. Cela signifie que notre agent est capable d'utiliser ses observations pour déduire une stratégie

Type de la stratégie de l'attaquant	Vulnerabilité	Numéro de la statégie défensive							
		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
	Privilege Escalation	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
	Privilege Escalation	0.5	0	0	1	0.5	0	0.5	0.5

Tableau 3.6 Efficacité des stratégies défensives contre celles de l'attaquant

de défense à utiliser. De plus, nous pouvons observer l'influence des coûts de la stratégie dans la récompense car pour certaines stratégies la longueur d'épisode est plus élevée que pour d'autres mais la récompense correspondante est plus faible. Par exemple, la stratégie "IP Random" a une récompense plus élevée que la stratégie "Honey Service" mais une longueur d'épisode plus petite.

Enfin, nous avons comparé les performances obtenues par un agent entraîné DQN dans différents environnements. Nous avons fait varier la probabilité de détection des attaques $P_{Detection}$, λ_{attack} et $\lambda_{deception}$. La figure 3.4 contient la comparaison des récompenses obtenues avec un agent DQN pour différentes valeurs de ces paramètres. Elle contient une carte de densité (Heatmap) par valeur de $P_{Detection}$. Chaque carte de chaleur montre les récompenses obtenues pour 3 valeurs différentes de λ_{attack} et $\lambda_{deception}$.

Nous pouvons remarquer que plus la probabilité de détection des attaques est faible, plus les récompenses pour le défenseur sont faibles. Ceci est logique car si le défenseur n'observe pas les attaques, il ne peut pas choisir une stratégie appropriée. λ_{attack}, qui influence la probabilité de réussite des attaques, a un fort impact sur les performances de notre agent. Plus λ_{attack} est grand, moins la probabilité de réussite des attaques est grande, et nous avons bien la récompense du défenseur qui est plus grande. $\lambda_{deception}$ a aussi un impact sur la probabilité de succès des leurrages. Plus la valeur de $\lambda_{deception}$ est grande, plus la probabilité de réussite

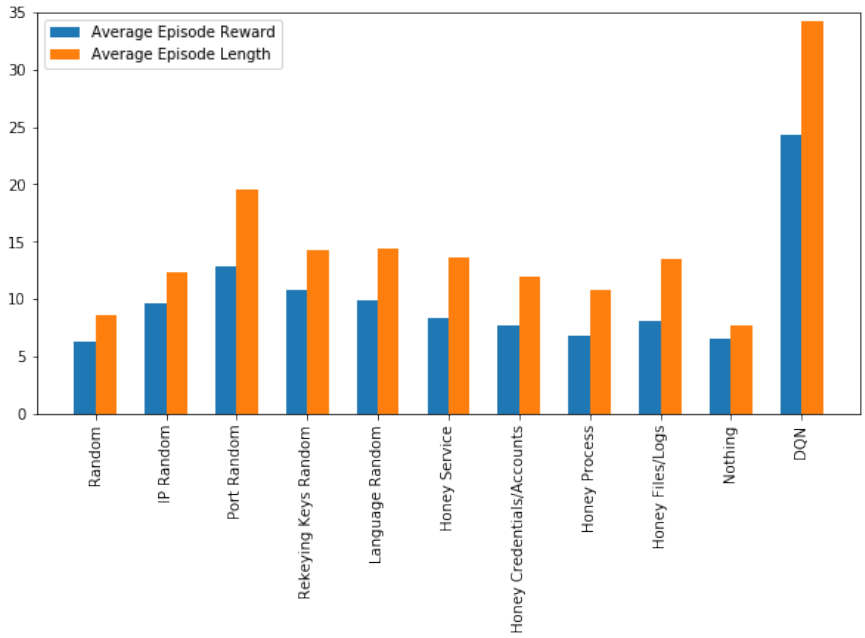

Figure 3.3 Comparaison des performances de différentes stratégies défensives dans des environnements avec $\lambda_{attack} = 0.5$ et $\lambda_{Deception} = 0.5$, $P_{Detection} = 1$

du leurrage est faible. Nous remarquons bien que plus $\lambda_{deception}$ est grande, plus la récompense du défenseur est grande.

Le problème avec l'algorithme DQN est que la politique obtenue n'est pas nécessairement optimale. En effet, il peut s'agir d'un optimum local. Ceci explique certains des résultats de la figure 3.4, par exemple, pour $P_{Detection} = 0.8$ et $\lambda_{attack} = 0.7$, les performances de l'agent pour $\lambda_{deception} = 0.5$ sont plus faibles que pour $\lambda_{deception} = 0.7$ ce qui est anormal selon les explications du paragraphe précédent.

3.5 Discussions

Le modèle proposé est multi-états et multi-étapes. Il permet de considérer un grand nombre d'états possibles pour le système. La modélisation de différentes machines avec différents services est possible. En effet, la représentation des vulnérabilités dans le vecteur V permet de considérer différents types de machines comme les serveurs ou les stations de travail qui auront différents types de vulnérabilités. De plus, grâce à notre représentation des observations du défenseur, nous sommes capables d'utiliser le comportement passé de l'attaquant pour mieux prédire ces actions futures et ainsi choisir une stratégie optimale.

De plus, notre modélisation tient compte de la différence de nature entre les stratégies MTD

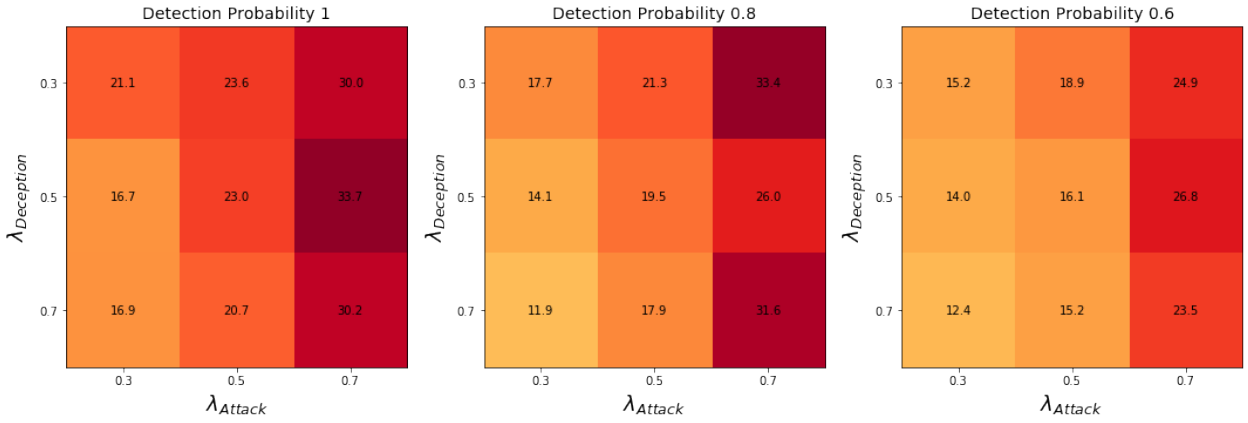

Figure 3.4 Comparaison des récompenses globales sur un épisode d'un agent défenseur DQN dans des environnements avec différentes valeurs pour $P_{Detection}$, λ_{attack} et $\lambda_{deception}$

et les stratégies de leurrage. Celles-ci agissent différemment sur le système et donc sur les observations de l'attaquant. Les stratégies MTD et de leurrage doivent être considérées comme complémentaires. En effet, les expériences montrent que pour contrer une variété d'attaques potentielles comme celles utilisées pour nos tests, il est nécessaire d'avoir accès à une grande variété de stratégies défensives. Les expériences montrent également l'importance d'une bonne coordination de ces stratégies et la nécessité de limiter le déploiement de certaines stratégies aux moments critiques car leur coût pour le système peut être très élevé.

Indépendamment des paramètres du modèle utilisés, nous parvenons à former un agent défenseur qui obtient de meilleures performances que les stratégies unitaires c'est à dire où une seule stratégie est utilisée. La limite de ces résultats est qu'ils sont obtenus contre notre attaquant suivant notre scénario détaillé dans la section 3.2. Nous ne pouvons savoir à quel point ceux-ci sont généralisables. Les avantages de ce scénario d'attaquant sont qu'il est basé sur la perception de l'attaquant et qu'il sépare les attaques à distance des attaques locales telles que l'élévation de privilèges par exemple. Il suit aussi les grandes étapes d'une tentative d'intrusion d'un attaquant comme il est basé sur les principales étapes de la CKC.

Le réalisme du modèle pourrait être amélioré en utilisant des données issues d'expériences. Pour cela, nous pourrions mettre en œuvre plusieurs stratégies défensives sur un système présentant des vulnérabilités connues afin de mesurer la capacité de ces stratégies à empêcher ou ralentir leur exploitation par un attaquant. En particulier, le modèle étant séquentiel, il serait intéressant d'intégrer le temps de chaque action pour l'attaquant et le défenseur dans le calcul de leur coût. De plus, l'efficacité des stratégies défensives contre les attaques ne devrait pas être statique mais dépendre de la progression de l'attaquant dans ses attaques. En effet, par exemple, la stratégie de défense "IP Random" ne sera pas aussi efficace contre un

attaquant situé en dehors du réseau ou contre un attaquant qui a déjà accès au réseau interne. Si l'attaquant est situé en dehors du réseau, il ne sera pas affecté par cette stratégie et celle-ci entraînera uniquement une surcharge pour le réseau. Tandis que si l'attaquant est dans le réseau, il pourrait par exemple perdre l'accès à une vulnérabilité et devrait recommencer certaines étapes antérieures de son attaque.

3.6 Conclusion

Les stratégies de leurrage et les stratégies MTD sont deux types de stratégies qui consistent à apporter de fausses informations ou de l'incertitude à la perception de l'adversaire afin d'augmenter le coût des attaques tout en réduisant leurs impacts. Nous avons vu que pour modéliser de telles confrontations attaquant/défenseur la théorie des jeux était souvent utilisée. Cependant, nous avons montré qu'avec nos critères (plusieurs actions pour chaque joueur, support de plusieurs sources de leurrage, leurrage de différentes natures, asymétrie de perception des joueurs, confrontation multi-étapes et multi-états), nous ne pouvions l'utiliser.

Ainsi, nous avons proposé un modèle de confrontation attaquant/défenseur dans un système informatique considérant l'asymétrie des perceptions et l'impact des stratégies des deux joueurs sur celles-ci. Nous avons ensuite proposé un scénario d'attaquant basé sur la CKC et utilisant sa perception de l'environnement. Grâce à cela, nous avons réalisé des simulations et entraîné avec l'algorithme DQN un agent défensif. Il est capable de choisir les stratégies défensives les plus adaptées pour empêcher ou ralentir la compromission de la machine en utilisant les actions passées observées de l'attaquant. Cela permet d'avoir une idée de l'efficacité de ces stratégies lorsqu'elles sont utilisées ensemble à partir de l'efficacité individuelle de chacune d'elles. Ce cadre de simulation pourrait être utilisé pour mieux sélectionner et mesurer les performances des stratégies déceptives par rapport aux risques d'attaques considérées.

CHAPITRE 4 SELECTION DE LA STRATEGIE DEFENSIVE DANS UN ENVIRONNEMENT REEL

Dans le chapitre précédent, nous avons entraîné un agent défenseur à sélectionner la meilleure stratégie dans le modèle que nous avions proposé. Cependant, pour valider la méthode utilisée, nous devons nous assurer que celle-ci est toujours efficace dans un environnement réel. Dans ce but, nous avons déployé un environnement implémentant des stratégies MTD ainsi que des stratégies de leurrage. De plus, nous avons implémenté les actions de l'attaquant et cette fois à la place d'utiliser un scénario d'attaquant pour simuler son comportement, nous avons entraîné un agent attaquant à pénétrer le système. Nous avons ensuite entraîné un défenseur à sélectionner les meilleures actions face à cet attaquant.

Dans la section suivante (4.1), nous présentons l'environnement que nous avons implémenté. Dans la section 4.2, nous introduisons l'attaquant avec son objectif, les actions qui lui sont possibles, ses observations de l'environnement ainsi que son système de récompense. Nous décrivons le défenseur de manière similaire dans la section 4.3. Le scénario de l'attaquant ayant changé par rapport au chapitre précédent, nous expliquons le processus d'optimisation des stratégies dans la section 4.4. Finalement, nous présentons et discutons des expérimentations et de leurs résultats dans les sections 4.5, 4.6 et 4.7.

4.1 L'environnement

4.1.1 Architecture

L'environnement que nous avons déployé est composé d'un serveur qui héberge des services, un serveur de Contrôle et un pare-feu.

Les services hébergés sont les suivants :

- Un serveur Web muni d'une page d'authentification accessible depuis l'extérieur du réseau.
- Un serveur Web sans page d'authentification également accessible depuis l'extérieur du réseau.
- Une base de données interne contenant des données ainsi que des identifiants.
- Un serveur de fichier interne de type Ftp contenant des données.

Ce sont les services qui doivent être déployés au minimum et dont le fonctionnement normal doit être garanti malgré l'utilisation de stratégies déceptives. A ces derniers s'ajoutent les services potentiellement utilisés pour le leurrage :

— Un serveur Web muni d'une page d'authentification relié à une base de données contenant de faux identifiants
— Une base de données interne contenant de fausses données ainsi que de faux identifiants.
— Un serveur de fichier interne de type Ftp contenant de fausses données.

Nous limitons le nombre de faux services pouvant être déployés à un par type de services.

Le serveur de Contrôle est le serveur qui permet de gérer le déploiement des services. C'est ce dernier qui envoie toutes les instructions au serveur d'hébergement et au pare-feu afin de gérer le déploiement initial des services ainsi que celui des stratégies MTD et de leurrage.

Le pare-feu sépare le réseau interne de l'extérieur. Il empêche toute personne extérieure au réseau d'accéder à des ressources de celui-ci. Le seul trafic autorisé est le trafic à destination des sites Web.

Les contraintes par rapport à l'utilisation de stratégies de MTD ou de leurrage sont que les sites Web soient toujours accessibles à la même adresse IP et sur les mêmes ports depuis l'extérieur du réseau et que la base de données et le serveur Ftp soient toujours accessibles sur les mêmes ports sur le serveur hébergeant les services.

La figure 4.1 représente l'architecture de notre environnement d'expérimentations. Les services essentiels y sont présentés en vert tandis que les faux services de leurrage y sont présentés en rouge.

4.1.2 Déploiement

Notre déploiement est fait à base de machines virtuelles. Chaque serveur est représenté par une machine virtuelle.

De plus, afin de gérer le déploiement de nos services, nous utilisons des conteneurs docker [63]. En effet, leur utilisation nous permet de faciliter le déploiement de nos services notamment en permettant de modifier certains paramètres à chaque déploiement. Par exemple, pour les base de données, le même conteneur peut être utilisé pour déployer plusieurs bases de données en modifiant uniquement le dossier contenant les données. Chaque service ne nécessite pas le même nombre de conteneur docker. En effet, le déploiement d'un site web sans authentification, d'une base de données ou d'un serveur Ftp peut se faire en utilisant un seul docker par service tandis que le déploiement d'un site web avec authentification utilise un conteneur pour gérer le frontend, un autre pour le backend et un dernier pour sa base de données.

Nous dissocions la gestion des données et les services. L'ensemble des données se trouve sur le serveur de contrôle et sont accédées à travers un conteneur Système de Fichiers en

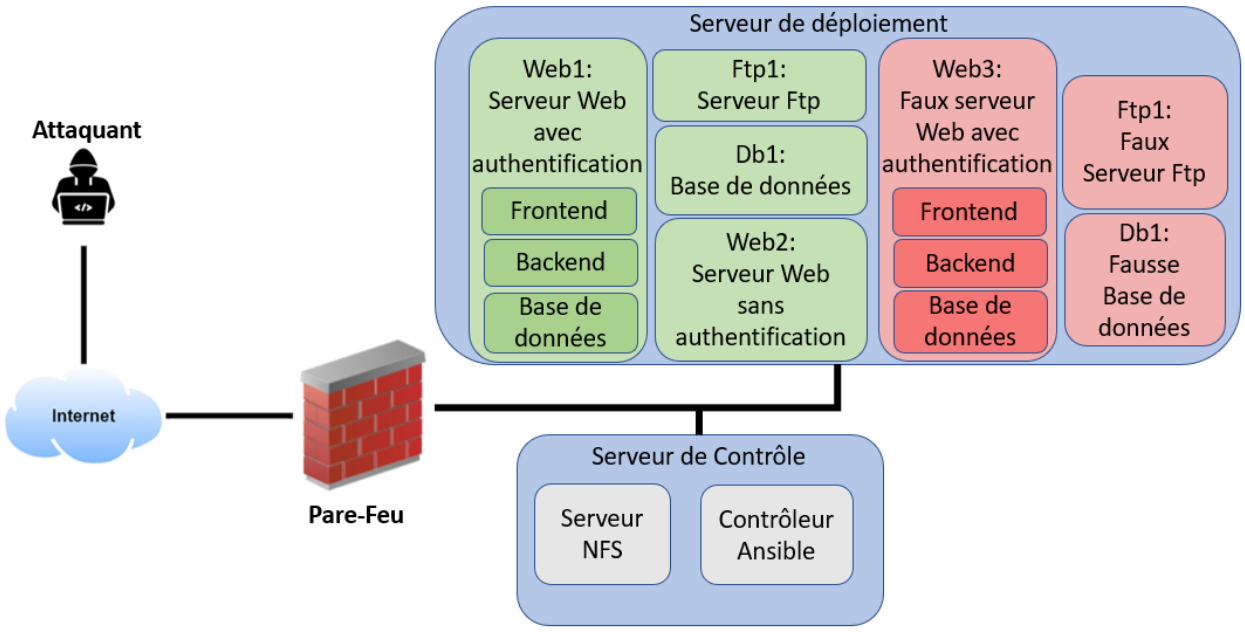

Figure 4.1 Architecture de l'environnement d'expérimentations

Réseau (NFS) présent sur ce dernier.

En plus d'améliorer la sécurité de notre infrastructure car limitant la propagation d'un service à l'autre, l'utilisation de conteneurs facilite le redéploiement des services ce qui sera très utile pour un environnement de tests comme celui dont nous avons besoin.

Nos services sont déployés dans des conteneurs docker mais le déploiement est géré par un conteneur Ansible présent sur le serveur de contrôle. Ansible [64] est une plateforme logicielle permettant d'automatiser le déploiement de conteneurs ou d'autres logiciels sur un ou plusieurs noeuds, la gestion de configuration, ainsi que l'exécution de tâches ad-hoc. Ansible utilise des playbooks contenant les instructions de déploiement de nos services. Cela permet d'automatiser une part des tâches nécessaires au déploiement notamment celles qui vont se répéter.

4.1.3 Description des services

Pour chaque service, nous avons eu besoin de plusieurs versions. En effet, cela fut nécessaire pour réaliser des stratégies MTD sur les versions des services déployés ainsi que pour avoir une variété d'environnement de tests avec des vulnérabilités ni toujours identiques ni toujours sur les mêmes services. Par exemple, pour le frontend du site avec une authentification, nous avons une version nginx ainsi qu'une version apache pour pouvoir passer d'une version à l'autre

grâce à une stratégie MTD mais nous avons aussi plusieurs versions vulnérables utilisant apache pour intégrer une variété de cas à l'environnement.

Dans le tableau 4.1 se trouvent les différentes versions pour chaque service avec les CVE associées. A ces CVE s'ajoutent d'autres vulnérabilités qui ne proviennent pas de la version du logiciel utilisé. Il s'agit par exemple d'injection Sql ou de réutilisation d'identifiants. Nous reviendrons dessus dans la section suivante (4.2) lorsque nous traiterons des actions de l'attaquant.

Services	Versions	CVE
Serveur Web sans authentification	Apache Struts 1	CVE-2017-5638
	Apache Struts 2	CVE-2020-17530
Frontend du serveur Web avec authentification	nginx	
	apache0	
	apache1	CVE-2021-41773 :no-cgid
	apache2	CVE-2021-41773 :with-cgid
Base de données	mysql	
	mongo	
Serveur Ftp	vsftpd	
	proftpd0	
	proftpd1	CVE-2015-3306

Tableau 4.1 Liste des versions des services avec les CVE associées

4.2 L'attaquant

Maintenant que nous avons présenté la structure de notre environnement, nous allons introduire l'attaquant avec ses objectifs, les différentes actions qui lui sont possibles ainsi que ses observations.

L'objectif

Contrairement à la section 3.2 où nous avions introduit un scénario d'attaquant pour simuler ses actions, ici nous assignons un objectif global à l'attaquant et grâce à de l'apprentissage par renforcement, nous allons essayer de lui apprendre à atteindre cet objectif. L'objectif global de l'attaquant est l'extraction de données sensibles pouvant être présentes dans une base de données ou sur un serveur de fichiers Ftp.

Pour atteindre son objectif, l'attaquant qui initialement se situe à l'extérieur du réseau va devoir chercher des vulnérabilités sur les services exposés à l'extérieur donc les sites Web afin

de pénétrer le système. Une fois ce but atteint, l'attaquant devra chercher à exploiter d'autres vulnérabilités sur des services internes pour se propager latéralement dans le système et ainsi être capable d'atteindre son objectif global.

Les actions

L'attaquant a un certain nombre d'actions possibles. Initialement, il peut seulement scanner les services exposés du système mais lorsqu'il découvre un vulnérabilité, il peut l'exploiter ce qui lui permet d'effectuer de nouvelles actions. Par exemple, si l'attaquant découvre une injection Sql dans le système d'authentification du site Web, il peut l'exploiter pour en extraire des identifiants. De plus, pour représenter la durée et la difficulté des actions, certaines actions qui sont plus longues ou plus complexes à réaliser sont divisées en plusieurs états. Plus une action est longue ou difficile à réaliser, plus elle aura d'états. L'état d'une action représente la progression dans cette action. A l'opposé, des actions courtes auront un seul état.

Dans les tableaux 4.2 et 4.3 sont présents le nombre d'états des actions de l'attaquant en fonction de la cible. Une fois les vulnérabilités scannées, la présence d'une vulnérabilité permet à l'attaquant l'exploitation de celle-ci. L'exploitation peut prendre plusieurs formes, cela peut être d'extraire des identifiants, des données ou permettre un mouvement latéral dans le réseau grâce à des scans du réseau interne.

	Serveur Web					
Type d'attaque	File Disclosure	Command Injection	Sql Injection	Mongodb Injection	Command Injection	File Disclosure
Scan	4	4	1	1	1	4
Network Scan		1			1	
Ports Scan		4			1	
Dump Credentials	1	1	3	3		
Dump Data						

Tableau 4.2 Nombre d'états des actions possibles pour l'attaquant sur les serveurs Web

A ces actions s'ajoutent une action permettant une attaque par dictionnaire sur les mots de passe extraits de la base de données car ceux-ci sont chiffrés. Les mots de passe ainsi trouvés pourront être utilisés pour des tentatives de connexion à certains services comme les bases de données ou les serveurs Ftp.

Ces actions, une fois exécutées peuvent retourner plusieurs résultats :

— Un Succès : obtenu lorsque le résultat est celui attendu et s'il est positif. Par exemple,

	Serveur Ftp		Base de données
Type d'attaque	Command Injection	Reuse dumped credentials	Reuse dumped credentials
Scan	2	1	1
Network Scan	1		
Ports Scan	4		
Dump Credentials			1
Dump Data	1	1	1

Tableau 4.3 Nombre d'états des actions possibles pour l'attaquant sur les serveurs Ftp et sur les bases de données

si l'attaquant scanne un serveur Web pour savoir si sa page d'authentification est vulnérable à une injection sql alors c'est un succès si la vulnérabilité est bien présente.

— Un Échec : obtenu lorsque le résultat est celui attendu mais il est négatif. Par exemple, si l'attaquant scanne un serveur Web pour savoir si sa page d'authentification est vulnérable à une injection sql alors c'est un échec si la vulnérabilité n'est pas présente.

— Une Erreur : obtenu lorsque le résultat n'est pas celui attendu. Par exemple, si l'attaquant scanne un serveur Web pour savoir si sa page d'authentification est vulnérable à une injection sql alors si le site n'a pas de page d'authentification le résultat est une erreur.

Les récompenses

Comme nous voulons entraîner un agent attaquant à pénétrer le système, nous devons aussi mettre en place un système de récompense pour les actions prises. Voici les différentes récompenses :

— Lorsque l'attaquant atteint son objectif global c'est à dire l'exfiltration de données sensibles d'une base de données ou bien d'un serveur de fichier Ftp, alors il obtient une récompense de 50.

— Lorsque l'attaquant réalise une action avec succès et que ce résultat est nouveau par rapport à ce qu'il connaissait précédemment, il obtient une récompense de 5. Par exemple, si l'attaquant scanne une vulnérabilité pour la première fois et que le résultat est positif et sa récompense est de 5. En revanche, si l'attaquant scanne de nouveau cette même vulnérabilité alors le résultat est toujours positif mais la récompense est nulle.

— Aux deux récompenses précédentes s'ajoutent une récompense négative. En effet, actuellement si l'attaquant scanne une vulnérabilité et que le résultat est négatif alors la

récompense est nulle. Hors, nous souhaitons que l'attaquant cherche à atteindre son objectif le plus rapidement possible. Nous ajoutons donc -1 à toutes les récompenses pour représenter le temps qui s'écoule.

Il faut aussi dissocier la récompense perçue de la récompense réelle de l'attaquant. En effet certains services du système sont des éléments de leurrage. Si l'attaquant obtient une récompense positive en scannant un élément de leurrage alors celle-ci est une récompense perçue mais non réelle. Il est important de séparer les deux pour être capable de mesurer la récompense réelle globale de l'attaquant ainsi que l'influence et l'efficacité des éléments de leurrage.

Les observations

Enfin, nous avons besoin de définir les observations de l'attaquant. Celles-ci sont séparées entre les observations spécifiques aux services et les observations plus globales.

Concernant les observations spécifiques aux services, elles sont composées d'un booléen indiquant si l'attaquant et conscient de l'existence du service, du type du service s'il est connu et pour chaque action de chaque service, l'attaquant possède deux observations : la progression de l'action et son dernier résultat.

Nous allons maintenant détailler chacune des observations. A certains moments, l'attaquant n'a même pas conscience de l'existence de certains services. Par exemple, tant que l'attaquant n'a pas réussi à pénétrer à l'intérieur du système, celui-ci n'a pas conscience de l'existence du serveur de fichiers Ftp ou d'une base de données. C'est pourquoi, une observation pour chaque service correspond au fait que l'attaquant soit conscient de son existence. Tant qu'un service n'est pas connu, cette valeur est à 0 et les observations pour toutes les actions sur ce dernier sont à 0. Une fois le service connu, cette valeur est à 1.

Une autre observation concernant chaque service est son type, s'il s'agit d'un serveur Web, d'une base de données ou d'un serveur de fichiers Ftp. Nous représentons cette valeur par de l'encodage un parmi n. Nous considérons aussi que c'est le scan des ports d'un conteneur qui permet de déterminer le type de service qu'il héberge.

Concernant la progression des actions de chaque service, il s'agit de la fraction du nombre d'état de l'action dans lequel l'attaquant a déjà été. Par exemple, pour un scan de ports qui possède 4 états différents, si l'attaquant a déjà réalisé deux scans alors la progression de l'action est de 1/2. Pour certaines actions telles que la tentative de connexion à des services par la réutilisation de mots de passe obtenus précédemment, la progression de l'action ne correspond pas à la même chose. Si l'attaquant a déjà trouvé un mot de passe qui fonctionne,

alors la progression de l'action est à 1. S'il n'a pas encore trouvé un mot de passe qui fonctionne et que tous les mots de passe qu'il possède ont été testés, alors la progression de l'action est aussi à 1. En revanche, s'il n'a pas encore trouvé un mot de passe qui fonctionne et qu'un mot de passe n'a pas encore été testé, la progression est de 0.

La seconde observation concernant chaque action de chaque service est le dernier résultat obtenu pour cette action sur ce service. Si l'action n'a jamais été réalisée alors cette valeur est à 0. Et ensuite, si le dernier résultat obtenu est un succès, cette valeur est à 1, si c'est un échec, cette valeur est à 0 et s'il s'agit d'une erreur alors cette valeur est à -1.

A ces observations concernant les services spécifiques s'ajoutent des observations plus générales. Pour chaque vulnérabilité potentielle du système, un vecteur contient celles qui ont déjà été trouvées. Si une vulnérabilité a déjà été trouvée alors son scan même sur un autre service peut se faire en une étape. C'est à dire que même si un scan possède plusieurs états possibles, l'attaquant peut savoir si la vulnérabilité est présente avec un seul scan car nous considérons que si trouver une vulnérabilité peut prendre du temps, uniquement vérifier si celle-ci est présente est rapide.

Enfin, un booléen est à 1 indique si un ou des mots de passe chiffrés ont été exfiltrés d'une base de données alors que l'attaquant n'a pas encore tenté une attaque par dictionnaire dessus. Un autre booléen est à 1 si une ou des adresses IP ont été découvertes dans le réseau alors que leurs ports n'ont pas encore été scannés. Si les conditions ne sont pas remplies, ces booléens sont à 0.

Dans cette section, nous avons donc décrit les actions et les observations de l'attaquant. Pour chaque service identifié, les dimensions des observations de l'attaquant sont entre 12 et 24 et le nombre d'actions disponibles par service se situe entre 6 et 14. En considérant tous les services déployés, cela peut représenter beaucoup d'actions disponibles. Cela forme une difficulté pour la sélection de la stratégie optimale. Nous reviendrons dessus dans la section 4.4.

4.3 Le défenseur

Dans cette section, nous allons décrire les objectifs, observations et actions du défenseur.

L'objectif

Tout d'abord, tandis que l'objectif de l'attaquant était clair : exfiltrer des données sensibles le plus rapidement du système, concernant le défenseur, son objectif ne peut être défini sans

considérer son adversaire. L'objectif du défenseur est relatif à l'attaquant : il doit chercher à empêcher ou ralentir au maximum l'atteinte de l'objectif par l'attaquant.

Les actions

Tout d'abord, le défenseur ne va pas choisir ses actions à la même fréquence que l'attaquant. En effet, les actions du défenseur ayant un grand impact et une influence sur la qualité des services déployés, nous limitons la quantité d'actions du défenseur. Nous considérons qu'il pourra choisir une action tous les dix pas de temps. Donc quand le défenseur jouera une action, pour commencer, l'attaquant pourra en jouer dix. Nous reviendrons sur cette valeur dans la section 4.6 car nous avons dû la faire évoluer au vu de nos premiers résultats.

Le défenseur peut utiliser deux types de défense : des stratégies MTD ou bien des stratégies de leurrage. Comme expliqué dans la section 4.1.1, le défenseur peut déployer de faux services pour leurrer l'attaquant. Nous limitons leur déploiement à un par type de faux services. L'attaquant peut donc déployer un faux serveur Web muni d'une page d'authentification relié à une base de données contenant de faux identifiants, une fausse base de données interne contenant de fausses données ainsi que de faux identifiants et un faux serveur de fichier interne de type Ftp contenant de fausses données. Si l'attaquant peut déployer ces faux services, il a aussi la possibilité de les supprimer s'ils sont déjà déployés.

En plus de ces stratégies de leurrage, le défenseur peut utiliser des stratégies MTD. Il peut utiliser du MTD sur les adresses IP des conteneurs pour randomiser les adresses IP des conteneurs contenant des services déployés. Il peut aussi changer la version du Frontend des serveurs Web muni d'une page d'authentification ainsi que la version de la base de données utilisée. Cela aura une influence sur la page d'authentification car nous passons de la possibilité d'être vulnérable à une injection Sql à celle d'être vulnérable à une injection NoSql (MongoDb) par exemple. Le défenseur peut aussi changer la version des bases de données internes ou celle des serveurs de fichiers Ftp. Toutes les actions MTD sont disponibles sur les vrais services mais également sur les faux services utilisés pour le leurrage s'ils sont déployés.

Dans le tableau 4.4 sont résumés les actions possibles du défenseur.

Les récompenses

Une première difficulté concernant le défenseur est qu'il ne sait pas si l'action qu'il a choisi est efficace, utile ou si elle a été utilisée au moment opportun. Le défenseur ne sait pas si sa stratégie a été efficace ou utile. Pour entraîner le défenseur, il faut donc mettre en place un système de récompense qui n'est pas basé sur le résultat immédiat d'une action. De

Services	Actions			
	Leurrage		MTD	
Machine			IP docker	
Serveur Web avec authentification			Version Frontend	Version Base de données
Base de données interne			Version	
Serveur Ftp			Version	
Faux Serveur Web avec authentification	Ajouter	Supprimer	Version Frontend	Version Base de données
Base de données interne	Ajouter	Supprimer	Version	
Serveur Ftp	Ajouter	Supprimer	Version	

Tableau 4.4 Actions disponibles pour le défenseur

plus, l'objectif du défenseur est de ralentir ou d'empêcher l'attaquant d'atteindre son objectif d'exfiltration de données. Nous pouvons donc considérer que plus l'attaquant met du temps à atteindre son objectif, meilleur est le défenseur. Nous mettons donc en place un système simple de récompense pour le défenseur : Chaque pas de temps écoulé apporte au défenseur une petite récompense de 1.

Les observations

Maintenant que nous avons décrit l'objectif, les actions et les récompenses du défenseur, nous devons définir ces observations, c'est à dire sur quoi il va se baser pour choisir son action.

Ces observations vont se diviser en deux parties : les observations de son propre environnement et les observations des actions de l'attaquant.

Concernant son propre environnement, le défenseur sait quelles versions des services sont déployés mais il ne sait pas si ces versions sont vulnérables ou non. Par exemple, concernant le Frontend du site Web avec une page de connexion, le défenseur sait s'il s'agit de la version utilisant nginx ou apache qui est déployée mais il ne sait pas si c'est une version vulnérable ou non d'apache ou de nginx. De plus, pour le déploiement des faux services, le défenseur sait s'ils sont déployés ou non.

Concernant les observations des actions de l'attaquant, le défenseur ne les observe pas directement. Il a seulement connaissance de la cible visée. Voici quelques exemples pour illustrer cela :

— Si l'attaquant effectue un scan des adresses IP des conteneurs, alors le défenseur observera qu'une action a été effectuée sur tous les conteneurs.

— Si l'attaquant effectue un scan de ports sur certains conteneurs alors le défenseur observera qu'une action a été effectuée sur ces conteneurs.
— En revanche, si l'attaquant choisit une action spécifique à un service comme par exemple s'il scanne un site web à la recherche d'une vulnérabilité, alors le défenseur observera qu'une seule action a été faite sur le Frontend du serveur Web.
— Si l'attaquant choisit l'attaque par dictionnaire sur les mots de passes chiffrés exfiltrés alors il n'y a pas de cible parmi les services du défenseur, donc le défenseur n'observe aucune action.

De plus, comme l'attaquant joue 10 actions quand le défenseur en joue une seule, les observations du défenseur ne vont pas se limiter à une seule action de l'attaquant. Les observations globales du défenseur seront donc les observations de son propre environnement à un instant donné ainsi que les observations des actions de l'attaquant pour les dix derniers pas de temps.

4.4 Optimisation des stratégies

Maintenant que nous avons décrit les objectifs, les actions, les observations et le système de récompenses pour l'attaquant et le défenseur, nous devons décrire le méthodes utilisées pour l'optimisation des stratégies pour chacun des joueurs.

4.4.1 L'algorithme Proximal Policy Optimization

Pour optimiser la sélection des stratégies des deux joueurs, nous allons utiliser de l'apprentissage profond par renforcement. A l'origine, nous voulions utiliser le même algorithme que pour l'optimisation des stratégies du défenseur dans notre modèle (Section 3.3), c'est à dire l'algorithme DQN [52]. Cependant, pour des raisons techniques, nous avons dû changer d'algorithme. En effet, pour rappel, dans un état donné, l'algorithme DQN associe à chaque action une valeur de satisfaction et l'action choisie est celle maximisant cette valeur. Donc dans un même état, la même action sera toujours choisie. Dans le modèle, cela ne posait pas de problème car chaque action faisait évoluer l'état du modèle. Dans notre environnement et avec les observations que nous avons décidées pour chaque joueur, il est possible qu'une action ne fasse pas évoluer ses observations du système. Par exemple, si l'attaquant scanne un des sites web sans page d'authentification à la recherche d'une injection Sql, alors le résultat de cette action est une erreur et la progression de l'action est à 1. Si l'attaquant répète cette action alors cela ne va pas faire évoluer les observations de l'attaquant et avec l'algorithme DQN, la même action sera toujours répétée.

Afin de résoudre ce problème, nous avons besoin d'un algorithme qui sélectionne la stratégie

dans un état donné de manière probabiliste et non déterministe comme l'algorithme DQN. Une famille d'algorithme permettant cela sont les algorithmes Actor-Critic. Il s'agit d'une classe d'algorithme basée sur la politique. La politique représente la probabilité de choisir une action dans un état donné. Dans ce type d'algorithme, la politique optimale est calculée en manipulant directement la fonction de politique, et la fonction de valeur trouve implicitement la politique optimale en trouvant la fonction de valeur optimale. Comme nous utilisons de l'apprentissage profond par renforcement, la fonction de valeur est représentée par un réseau de neurones et un autre réseau de neurones associe une politique à chaque état.

La fonction de politique joue le rôle de l'acteur : elle choisit les coups à jouer. La fonction de valeur joue le rôle de critique : elle vérifie si l'agent a progressé ou non. C'est ce retour d'information qui guide le processus d'entraînement.

Pour entraîner nos agents défenseurs et attaquants, nous utilisons l'algorithme PPO [65]. Il s'agit d'une famille d'algorithme de méthodes Policy-Gradients. Initialement il fut proposé pour gérer des plages d'actions continues. Cependant il fonctionne également pour des actions discrètes. Il présente aussi l'avantage de s'assurer que la mise à jour de la politique n'est pas trop importante, c'est-à-dire que l'ancienne politique n'est pas trop différente de la nouvelle. Pour cela, il utilise une fonction d'écrêtage (*clipping function* en anglais). Cela permet d'améliorer la stabilité de l'apprentissage.

4.4.2 Optimisation des stratégies de l'attaquant

Pour l'attaquant, l'état du système sera ses observations. Or nous utilisons de l'apprentissage profond par renforcement, Il faut donc représenter les observations sous la forme d'un vecteur. Les observations décrites dans la section 4.2 forment ainsi un vecteur de taille 244. L'ensemble des actions de l'attaquant est de taille 121, il est donc plutôt grand. Cependant, certaines actions sont impossibles. Par exemple, toute action sur une cible que l'attaquant n'a pas encore découverte est impossible. L'exploitation d'une vulnérabilité pour extraire des informations ou se propager latéralement n'est pas non plus permise tant que la vulnérabilité n'a pas été découverte. Ainsi le nombre d'actions réellement disponibles à l'attaquant dans chaque état est bien inférieur au nombre d'actions totales. Cela nous permet d'utiliser un masque sur les actions impossibles, la probabilité de les sélectionner est nulle. Cela résout le problème de la grande dimension de l'espace des actions.

Nous allons maintenant parler de l'architecture des réseaux de neuronnes utilisés pour le réseau Actor qui fournit la politique et pour le réseau Critic qui fournit la valeur de l'état.

L'attaquant peut observer à chaque instant plusieurs services du même type. Par exemple,

si une fausse base de données interne est déployée, l'attaquant peut observer la base de données utilisée par le serveur Web, la base de données interne et la fausse base de données interne. Or les observations de chacune de ses bases de données respectent la même structure. L'attaquant doit être capable d'extraire de l'information de ces observations, c'est la même tâche qu'il s'agisse d'un vrai service ou d'un service de leurrage tant qu'il s'agit du même type de service. Nous pouvons donc améliorer les réseaux de neurones utilisés par notre algorithme PPO. Nous pouvons considérer que les premières couches de ces neurones permettent d'extraire l'information des observations. Nous pouvons réutiliser pour des services du même type le même extracteur d'informations. Cela permet d'améliorer l'apprentissage de notre réseau de neurones et notamment sa vitesse de convergence car nos extracteurs seront utilisés plusieurs fois à chaque sélection de stratégie. Nous entraînons nos extracteurs avec davantage de données que le reste du réseaux de neurones. Nous concaténons ensuite la sortie des réseaux extracteurs avec les observations générales que nous mettons en entrée d'un réseau de neurones plus classique. Cette architecture de réseau de neurones est représentée schématiquement dans la figure 4.2.

Les réseaux de neuronnes utilisés pour la partie Actor et pour la partie Critic ont cette même structure décrite ci-dessus. La seule différence est la sortie. Pour la partie Actor, la sortie est une politique tandis que pour la partie Critic, la sortie est une valeur, une estimation de la valeur de l'état.

4.4.3 Optimisation des stratégies du défenseur

Pour le défenseur, l'état du système sera ses observations. Il faut donc aussi représenter ses observations sous la forme d'un vecteur. Les observations décrites dans la section 4.3 se divisent en des observations de son propre environnement et des observations des actions de l'attaquant. Ensemble, elles forment ainsi un vecteur de taille 109. Le défenseur a beaucoup moins de possibilités d'action. L'ensemble de ses actions est de taille 15. Nous n'avons donc pas le même problème que pour l'attaquant concernant les dimensions de l'ensemble des actions. Nous utilisons tout de même un masque sur les actions car certaines actions sont impossibles comme par exemple le déploiement d'un service de leurrage si un autre service de leurrage du même type a déjà été déployé. De même, le défenseur ne peut supprimer un service qui n'est pas déjà déployé.

Contrairement à l'attaquant, les réseaux de neurones utilisés par l'algorithme PPO pour entraîner le défenseur n'ont pas une partie extracteurs. Ce sont donc juste des réseaux de neurones classiques entièrement connectés (*Fully-connected* en anglais). Nous reviendrons sur l'architecture des réseaux de neurones utilisés dans la section 4.5.3.

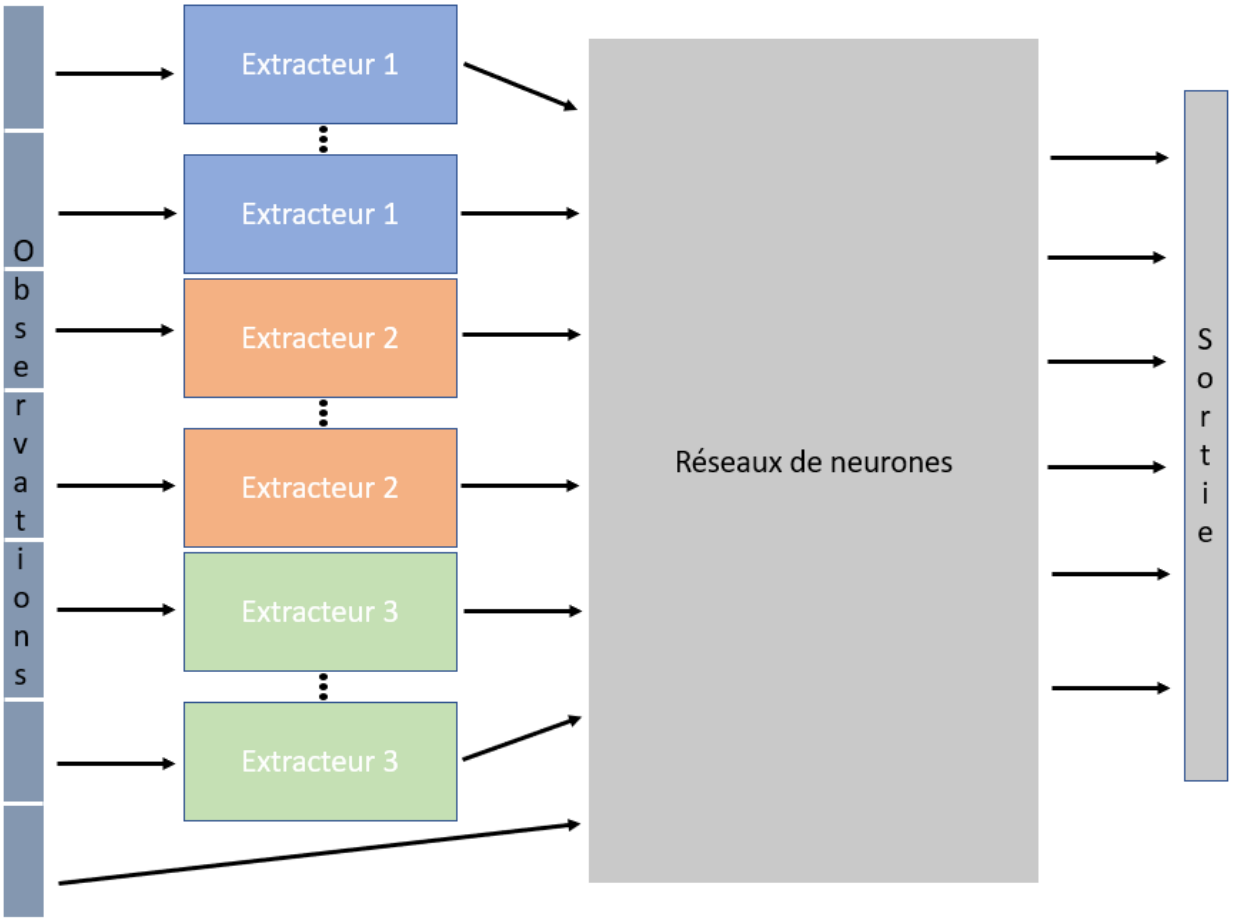

Figure 4.2 Structure des réseaux de neurones utilisés

4.5 Expérimentations

4.5.1 Détails de l'implémentation

Les expérimentations ont été réalisées sur un ordinateur muni d'un processeur CPU Intel Core i7-9750H avec une carte graphique NVIDIA GeForce RTX 2070. Les machines virtuelles utilisées pour simuler nos serveurs sont lancés par le logiciel de virtualisation VirtualBox [66]. Le système d'exploitation utilisé comme base pour nos serveurs est Debian 10.2 [67] avec le noyau Debian 5.10.106-1. Notre agent défenseur est programmé en python et utilise des playbooks Ansible pour envoyer les instructions de déploiement au différents serveurs.

Nous utilisons encore une fois la librairie Python Stable Baselines 3 [62] pour entraîner nos agents attaquants et défenseurs. Nous créons des environnements OpenAi Gym afin de faire le lien entre l'environnement réel et nos agents.

4.5.2 Déroulement des expérimentations

Afin de mesurer les performances de notre optimisation de la sélection des stratégies de notre défenseur, nous suivons la procédure suivante. Tout d'abord, nous entraînons notre agent attaquant dans un environnement sans éléments de leurrage et sans actions du défenseur. L'environnement est donc statique et les performances obtenues de l'attaquant pourront être utilisées comme base de comparaison. Nous entraînons ensuite notre agent attaquant dans un environnement avec tous les éléments de leurrage déployés mais toujours sans actions du défenseur. Cela permettra de mesurer la capacité et l'efficacité des stratégies de leurrage statique à ralentir l'attaquant. Enfin, nous entraînons notre agent défenseur à sélectionner ces actions face aux deux agents attaquants entraînés préalablement afin de mesurer la capacité d'un agent défensive intelligent et adaptatif à ralentir l'adversaire.

4.5.3 Paramètres des expérimentations

Dans cette section, nous décrivons les hyperparamètres utilisés dans l'algorithme d'apprentissage profond par renforcement PPO pour l'entraînement de nos agents attaquant et pour notre agent défenseur.

Les hyperparamètres proposés sont ceux ayant obtenus les meilleurs résultats. Dans le tableau 4.5 se trouvent ceux utilisés pour l'entraînement des agents attaquants.

Paramètre	Valeur
Taux d'apprentissage	0.0001
Fréquence d'entraînement	1000 étapes
Taille du batch	128
Facteur de dévaluation	0.9
Plage de clipping	0.2
Couche cachée de l'extracteur Serveur Web des réseaux Actor et Critic	[32]
Couche cachée de l'extracteur Base de données des réseaux Actor et Critic	[32]
Couche cachée de l'extracteur Serveur Ftp des réseaux Actor et Critic	[32]
Couches cachées globales post-extracteurs des réseaux Actor et Critic	[256x128]
Fonction d'activation	Tanh
Valeur maximale pour le clipping du gradient	0.5

Tableau 4.5 Liste des hyperparamètres de l'algorithme PPO utilisés pour l'entraînement de l'agent attaquant

Dans le tableau 4.6 se trouvent les hyperparamètres utilisés pour l'entraînement de l'agent défenseur.

Paramètre	Valeur
Taux d'apprentissage	0.001
Fréquence d'entraînement	100 étapes
Taille du batch	32
Facteur de dévaluation	0.99
Plage de clipping	0.2
Couches cachées des réseaux Actor et Critic	[256x128x64]
Fonction d'activation	Tanh
Valeur maximal pour le clipping du gradient	0.5

Tableau 4.6 Liste des hyperparamètres de l'algorithme PPO utilisés pour l'entraînement de l'agent défenseur

4.6 Résultats

La première partie des apprentissages fût pour optimiser les hyperparamètres de notre agent. Par exemple, nous nous sommes intéressés aux fonctions d'activations des réseaux de neurones et nous avons comparé l'utilisation des fonctions Relu ou Tanh. La fonction Relu désigne la fonction Unité Linéaire Rectifiée [68]. La fonction Tanh désigne la tangente hyperbolique.

La récompense totale moyenne obtenue par épisode et la longueur moyenne des épisodes est donnée dans les figures 4.3 et 4.4. Ces résultats sont obtenus dans un environnement statique sans éléments de leurrage déployés.

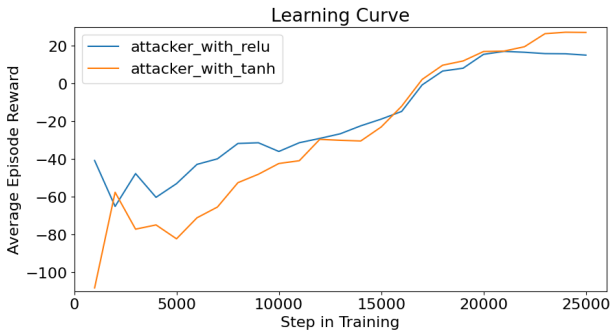

Figure 4.3 Récompense moyenne par épisode obtenue au cours de l'entraînement avec les fonctions d'activations relu et tanh

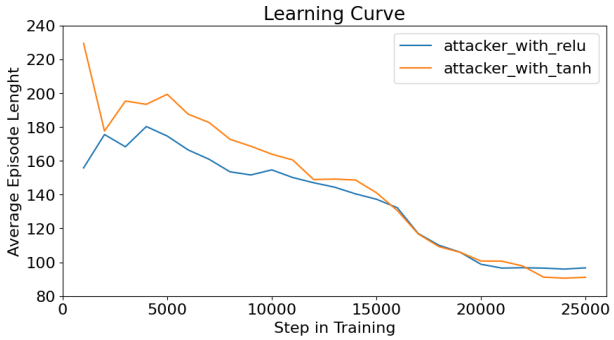

Figure 4.4 Longueur moyenne des épisodes obtenue au cours de l'entraînement avec les fonctions d'activations relu et tanh

Nous avons également comparé les performances de l'attaquant lorsqu'il est dans un environnement statique avec et sans éléments de leurrage déployés. Les courbes d'apprentissages se situent sur les figure 4.3 et 4.4. Attention, les courbes des récompenses ne montrent pas les récompenses réelles mais les récompenses perçues de l'attaquant. Elles permettent de vérifier

que l'agent attaquant apprend bien pendant son entraînement par rapport à ses connaissances et sa perception.

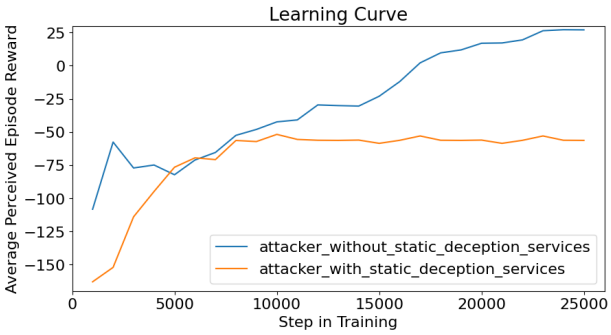

Figure 4.5 Récompense moyenne par épisode obtenue au cours de l'entraînement avec et sans leurrage

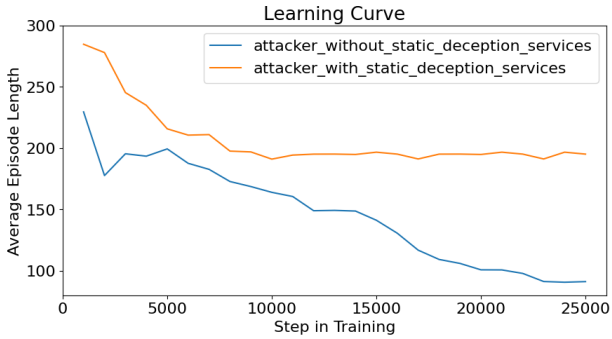

Figure 4.6 Longueur moyenne des épisodes obtenue au cours de l'entraînement avec et sans leurrage

Cependant, quand le système est statique, il est possible que pour l'attaquant, le chemin vers l'objectif n'existe pas. L'apprentissage est réalisé sur l'environnement avec un déploiement aléatoire. Il faut donc mesurer la longueur moyenne des épisodes lorsqu'on garantit l'existence d'un chemin jusqu'à l'objectif de l'attaquant. Ces résultats sont présents dans le tableau 4.7.

	Récompense perçue	Récompense réelle	Longueur Épisode
Sans Leurrage	23.71 ± 45.68	23.71 ± 45.68	$90.14 \pm 47,52$
Avec Leurrage	-29.72 ± 83.85	-21.36 ± 90.76	178.18 ± 84.20

Tableau 4.7 Performance de l'attaquant lorsqu'un chemin vers l'objectif est garanti avec un environnement statique

Les figures 4.7 et 4.8 montrent un exemple d'évolution de la récompense de l'attaquant au cours d'un épisode dans un environnement statique avec ou sans leurrage.

Lorsque le défenseur peut déployer des stratégies MTD et de leurrage, avec un entraînement de seulement quelques itérations de l'agent défenseur, les performances de l'attaquant chutent drastiquement. Nous décidons donc d'agrandir l'intervalle entre chaque action du défenseur. Nous limitons ici la longueur des épisodes à 500 étapes. Les résultats obtenus au bout de dix épisodes d'apprentissages face à un agent attaquant entraîné dans l'environnement statique avec des stratégies de leurrage déployées sont dans le tableau 4.8. Nous analyserons ces résultats dans la section suivante (4.7).

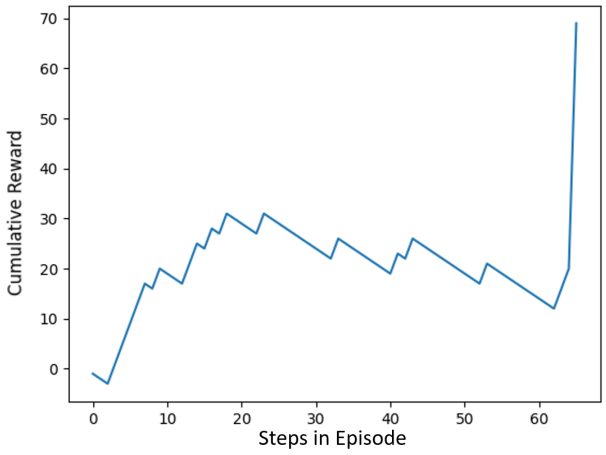

Figure 4.7 Exemple d'évolution de la récompense de l'attaquant au cours d'un épisode dans un environnement statique sans leurrage

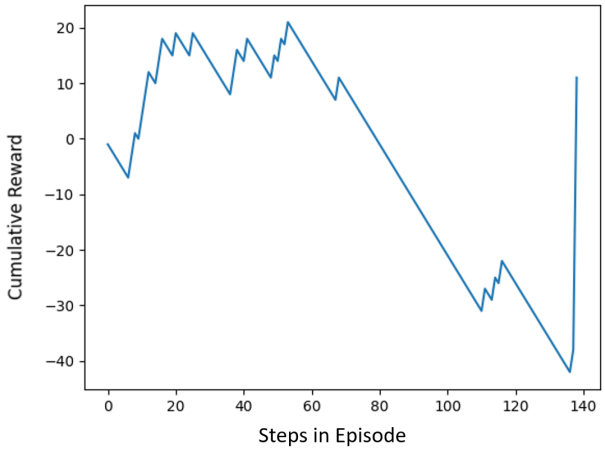

Figure 4.8 Exemple d'évolution de la récompense de l'attaquant au cours d'un épisode dans un environnement statique avec leurrage

Intervalle des actions du défenseur	Récompense perçue	Récompense réelle	Longueur Épisode
10	-185.71 ± 69.54	-247.04 ± 63.29	500 ± 0
20	-324.07 ± 72.32	-363.34 ± 67.85	493.08 ± 10.51
50	$-266.23 \pm 171,81$	-303.76 ± 162.23	409.21 ± 191.42

Tableau 4.8 Performance de l'attaquant face à un défenseur déployant des stratégies à différents intervalles

4.7 Discussions

Dans cette section, nous discutons des résultats obtenus dans la section 4.6.

Entrainement de l'attaquant

Tout d'abord, nous remarquons que l'apprentissage de l'agent attaquant fonctionne très bien dans un environnement statique que des éléments de leurrage soient déployés ou non. Les figures 4.5 et 4.6 l'illustrent.

L'utilisation du masque afin d'empêcher la sélection d'actions impossibles pour l'attaquant fonctionne bien et permet d'éviter d'autres méthodes moins efficaces comme par exemple l'attribution d'une grande récompense négative pour les actions impossibles. Notre méthode est efficace même quand le nombre d'actions disponibles est grand (de l'ordre de la centaine).

En observant la progression de l'attaquant dans un environnement avec ou sans éléments

de leurrage, nous remarquons que les stratégies de leurrage, à cause de l'emplacement où nous les déployons c'est-à-dire dans le réseau interne, n'ont pas d'influence sur la première phase de l'attaque, la phase où l'attaquant cherche à pénétrer le système. Cela est normal car l'attaquant ne les observe pas encore. Nous pouvons observer cela en comparant les figures 4.7 et 4.8. En revanche, une fois l'attaquant dans le réseau interne, les éléments de leurrage sont efficaces. Cela s'explique car l'attaquant a des difficultés à distinguer les vrais des faux services dans le réseau interne. Plus précisément, ce n'est que lorsqu'il essaie d'extraire certaines données qu'il se rend compte s'il s'agit d'un vrai ou faux service. Par exemple, lorsque l'attaquant scanne un serveur de fichiers Ftp, ce n'est que quand il tente d'extraire des données sensibles qu'il se rend compte qu'il s'agit d'un faux serveur Ftp. La situation empire pour l'attaquant lorsqu'il cible les bases de données car qu'elles soient vraies ou non, il peut en extraire des mots de passe chiffrés qu'il tentera d'utiliser. Une part des mots de passe de la fausse base de données sont volontairement faibles et donc l'attaquant peut perdre du temps ensuite à tenter des connexions sur d'autres services avec ces mots de passe.

Entraînement du défenseur

Concernant le défenseur, tout d'abord, il est important de remarquer que le coût des actions du défenseur n'est pas considéré de la même façon dans notre modèle. En effet, plutôt que d'inclure le coût de l'action dans son utilité, nous décidons de limiter directement leur utilisation ce qui limite ainsi le coût. Cela semble plus réaliste puisque nous optimisons la capacité du défenseur à ralentir l'attaquant dans un environnement avec des ressources limitées. Nous limitons donc le déploiement des stratégies de leurrage à un par type de services et nous espaçons les actions du défenseur. Comme nous l'avons dit le défenseur joue initialement dix fois moins de coups par rapport à l'attaquant. Ainsi, en théorie, il est plus difficile d'apprendre au défenseur les stratégies à sélectionner, car il a finalement dix fois moins d'expérience sur chaque épisode.

De plus, il est intéressant de remarquer qu'avec les stratégies MTD sur les versions, nous apportons de la diversité mais nous multiplions aussi les chances d'avoir une vulnérabilité. En effet comme il y a davantage de versions utilisées, nous multiplions donc la probabilité qu'une vulnérabilité soit présente parmi celles-ci. Malgré cela, il se trouve que lorsque le défenseur peut utiliser des stratégies MTD ou de leurrage, les performances de l'attaquant chutent. Nous avons commencé par entraîner un agent défenseur dont l'intervalle des actions était de 10 étapes. Au bout de quelques itérations d'entraînement du défenseur, l'attaquant n'arrivait plus à atteindre son objectif. Les résultats obtenus sont présents dans le Tableau

4.8.

Nous avons trouvé deux types de cas dans nos expérimentations pour expliquer cela.

Influence des stratégies MTD internes Premièrement, lorsqu'une stratégie MTD est utilisée, il faut un temps significatif à l'attaquant pour récupérer des informations. Par exemple, si le défenseur utilise la stratégie MTD sur les adresses IP des dockers, l'attaquant ne le sait pas donc il continue comme si cette stratégie n'avait pas été utilisée. Il lui faut quelques actions retournant une erreur (car les adresses IP cibles ne correspondent plus) pour se rendre compte que les adresses IP ont changé. Il lui faut ensuite une étape pour rescanner les adresses IP du réseau et quelques étapes pour scanner les ports et identifier les machines. Il lui faut donc au moins une dizaine d'actions pour revenir à un état de connaissance similaire à ce qu'il avait avant que les adresses IP changent. Or, cette période correspond à l'intervalle entre chaque action du défenseur. En quelques épisodes d'entraînement, le défenseur apprend à se focaliser sur les stratégies MTD.

Influence des stratégies MTD externes Le second cas est le suivant. Si l'attaquant a trouvé dans un service extérieur une vulnérabilité lui permettant de pénétrer à l'intérieur du réseau interne, si le défenseur change la version de ce service, l'attaquant peut perdre son accès à l'intérieur et il lui faut du temps soit pour en trouver un autre soit pour que le défenseur utilise de nouveau une stratégie MTD sur ce service et que le service vulnérable soit exposé de nouveau. Par exemple, si l'unique service extérieur vulnérable pour pénétrer le système est le serveur web avec authentification déployé avec une version apache vulnérable comme frontend. Si le défenseur utilise une stratégie MTD sur le frontend de ce service qui devient une version nginx non vulnérable, alors l'attaquant perdra l'accès à l'intérieure du réseau tant que cette version sera déployée.

Influence de l'intervalle entre les actions du défenseur Lorsque nous augmentons l'intervalle entre les actions du défenseur à 20 puis à 50, l'attaquant n'est pas tellement meilleur. Ces résultats sont présentés dans le Tableau 4.8. En observant les expérimentations, nous remarquons que le premier cas décrit précédemment pour expliquer les faibles performances de l'attaquant n'est plus présent. En effet, entre deux actions du défenseur, l'attaquant a le temps de rattraper et même dépasser son niveau précédant de connaissance du système. En revanche, le second cas identifié comme une des causes des faibles performances de l'attaquant est toujours présent.

Transfert des connaissances d'un environnement statique à dynamique Avec un intervalle d'action pour le défenseur de 50, nous observons un nouveau cas particulier. Nous remarquons que soit l'attaquant réussit à atteindre son objectif avant que le défenseur ait effectué sa première action sinon il n'arrive pas à atteindre son objectif. Cela explique la grande variance des récompenses et de la longueur d'un épisode dans le Tableau 4.8 pour un intervalle de 50. De cette observation, nous déduisons que les connaissances acquises par l'entraînement de l'attaquant sur un environnement statique ne se transfèrent pas dans un environnement dynamique. Cela est sûrement dû à la forme des observations de l'attaquant. Elles ne sont sûrement pas assez adaptées pour représenter les changements dans les services observés par l'attaquant.

Difficultés

Une des plus grandes difficultés avec l'entraînement de nos agents attaquants comme défenseurs dans un environnement réel est le temps d'apprentissage. En effet, l'attaquant effectue de vraies attaques, donc leur durée peut être assez grande. Cela complique l'apprentissage car il faut beaucoup plus de temps rien que pour exécuter chaque action. La durée réelle d'un épisode est alors très variable et peut atteindre régulièrement l'heure. De ce fait, l'entraînement de nos agents prend beaucoup de temps et peut durer presque une semaine.

De plus, l'utilisation d'un environnement réel implique une grande utilisation de ressources matérielles. En effet, même si le déploiement est fait grâce à des machines virtuelles, héberger plusieurs services demande beaucoup de mémoire vive. Par exemple, le fonctionnement complet de notre environnement demande un minimum de 10Go de mémoire vive partagés principalement entre le serveur hébergé les services et le serveur de contrôle. Cela limite donc le déploiement de plusieurs environnements ce qui aurait pu être utile pour paralléliser l'apprentissage sur plusieurs environnements et ainsi gagner du temps.

Pistes d'améliorations

Les résultats nous montrent que l'amélioration du défenseur passera par l'amélioration de l'attaquant. Il faut notamment un attaquant avec une meilleure perception, c'est à dire un attaquant qui déduit d'avantage de choses avec ses observations. Une piste d'amélioration serait donc d'adapter la représentation des observations et notamment pour mieux y représenter les changements d'observations de l'attaquant. L'équilibre entre la précision et la taille des observations aura aussi un fort impact sur l'apprentissage. Les performances de l'attaquant chutent lorsque l'environnement est dynamique. Pour faire face à ce problème, nous pourrions par exemple intégrer la mémoire non pas aux observations comme nous faisions

mais plutôt au réseau de neurones en utilisant par exemple un réseau de neurones de type Recurrent Neural Network (RNN) ou Long Short-Term Memory (LSTM) qui permet de gérer une séquence d'observation en entrée à la place d'une observation.

De plus, pour obtenir les meilleurs performances de l'agent défensif, de l'apprentissage croisé pourrait être utilisé. Cela consiste à entraîner l'attaquant et le défenseur de manière simultanée. Le problème a géré avec une telle approche serait la durée de l'entraînement qui pourrait être très longue.

4.8 Conclusion

Dans ce chapitre, nous avons commencé par décrire l'environnement que nous avons déployé. Ce dernier permet le déploiement de différentes stratégies MTD et de leurrage que nous avons implémentées. Nous avons ensuite entraîné un agent attaquant à sélectionner les meilleures actions pour pénétrer dans le système et atteindre un objectif d'extraction de données sensibles le plus rapidement possible. Nous avons réalisé cela dans l'environnement statique avec et sans stratégies de leurrage du défenseur déployées. L'entraînement fût un succès grâce à l'algorithme PPO. Nous avons ensuite entraîné un agent défenseur à ralentir ou empêcher l'atteinte de l'objectif par l'attaquant. Cependant, au bout de quelques épisodes d'entraînement les performances de l'attaquant chutent drastiquement. Nous allongeons l'intervalle entre deux actions du défenseur, mais cela n'améliore pas les performances de l'attaquant. Nous donnons plusieurs cas observés dans les expérimentations qui expliquent ces résultats et nous proposons plusieurs pistes pour améliorer les performances de l'attaquant car l'agent défenseur n'aura pas besoin de s'améliorer tant que les performances de l'attaquant ne seront pas meilleurs.

CHAPITRE 5 ANALYSE DES MÉTHODES UTILISÉES

Dans les chapitres 3 et 4, nous avons tenté d'optimiser la sélection des stratégies MTD et de leurrage respectivement dans un modèle et dans un environnement réel. Pour cela, nous avons utilisé de l'apprentissage profond par renforcement. Dans la section 5.1, nous reviendrons sur les différents algorithmes d'apprentissage profond utilisés. Enfin dans la section 5.2, nous comparerons les approches utilisées et nous discuterons des hypothèses utilisées dans les chapitres 3 et 4.

5.1 Les algorithmes d'apprentissage profond par renforcement

Dans les chapitres 3 et 4, lorsque nous avons entraîné nos agents attaquant et défenseur, nous n'avons pas utilisé le même algorithme d'apprentissage profond par renforcement. En effet, nous avons d'abord utilisé l'algorithme DQN pour entraîner notre agent défenseur dans le modèle et nous avons utilisé l'algorithme PPO pour entraîner nos agents défenseurs et attaquants dans l'environnement que nous avons implémenté. Nous avons expliqué dans la section 4.4.1 les raisons de l'utilisation de cet algorithme à la place de DQN.

Choix de l'apprentissage profond par renforcement Plusieurs raisons nous ont poussés à utiliser des algorithmes d'apprentissage profond par renforcement. La nature des confrontations considérées nous a poussé à utiliser des agents, un pour choisir les actions de l'attaquant et un autre pour choisir celles du défenseur. De plus, l'utilisation d'algorithme d'apprentissage par renforcement est très utile pour que ces agents apprennent grâce à leurs propres expériences en utilisant leurs observations de l'environnement et le retour des actions prises. Enfin, l'utilisation d'algorithme d'apprentissage profond par renforcement à la place d'algorithme classique d'apprentissage par renforcement fut décidé par rapport à la nature des observations de nos agents. En effet, pour un agent, ses observations correspondent à l'état dans lequel il se trouve. Or nos observations comportent plus d'une centaine d'attributs. Chaque combinaison d'attributs peut représenter un état donc le nombre d'état aurait été beaucoup trop grand pour un algorithme classique d'apprentissage par renforcement. Parmi les algorithmes DRL, nous avons décidé d'utiliser DQN et PPO. Par la suite, nous expliquons les avantages et inconvénients de chacun et ce qui nous a poussé à les choisir.

Algorithme DQN DQN est un algorithme Off-Policy, c'est-à-dire qu'il est capable d'apprendre sa stratégie indépendamment des actions de l'agent. Si nous possédions des exemples

de partie de jeu dans un environnement, nous pourrions en théorie apprendre la stratégie à l'agent sans interagir directement avec cet environnement. L'avantage d'une telle approche est qu'elle permet l'utilisation d'un tampon de répétition et donc d'apprendre plusieurs fois avec les mêmes données, d'utiliser plusieurs fois les expériences dans l'environnement. Nous obtenons de très bons résultats avec cet algorithme mais en contrepartie il met beaucoup de temps à converger car il a besoin d'énormément de données.

Algorithme PPO A l'opposé PPO est un algorithme On-Policy. Il nécessite donc une interaction directe avec l'environnement. Le premier avantage de cet algorithme est sa simplicité d'implémentation. Il est aussi reconnu comme étant un algorithme dont les hyperparamètres sont faciles à ajuster et optimiser. Ce fut le cas pour l'apprentissage de nos agents dans l'environnement réel, en quelques tests, nous avons réussi à trouver des hyperparamètres pour lesquels l'algorithme semblait converger. Un dernier avantage de PPO est sa vitesse de convergence. Par exemple, l'apprentissage total de nos agents attaquants dans un environnement statique était autour des 25000 étapes ce qui ne correspond finalement qu'à 208 épisodes ce qui est plutôt faible surtout dans un environnement stochastique.

Traitement des observations Nous avons pu observer l'importance et l'influence de la représentation des observations pour entraîner nos agents. En effet les faibles performances de l'agent attaquant face à un agent défenseur utilisant des stratégies MTD sont en partie dues à ses observations. Dans la section 4.4.2, nous avons proposé d'adapter la structure des réseaux de neurones utilisés aux observations pour accélérer et améliorer l'apprentissage. Dans l'environnement de test, nous avons utilisé des sous-réseaux de neurones extracteurs d'informations spécifiques à chaque type de service pour pré-traiter les observations de l'attaquant. Comme plusieurs services du même type sont observés par l'attaquant, ces extracteurs sont davantage entraînés que le reste du réseau de neurones. Cette approche fût par exemple utilisée dans l'article Basbrain et al. [69] et montra une amélioration des performances dans un cadre totalement différent de reconnaissance d'émotions.

5.2 Modèle ou environnement réel

Des hypothèses fortes

Tout d'abord, dans notre modélisation comme dans notre environnement réel, pour envisager d'optimiser la sélection des stratégies, nous avons dû faire des hypothèses assez fortes sur la confrontation de l'attaquant et du défenseur.

La gestion du temps Dans nos confrontations, nous avons discrétisé le temps. L'attaquant et le défenseur choisissent et jouent chacun leurs actions de manière séquentielle. Dans la réalité, cela n'est pas le cas, un attaquant peut par exemple lancer un scan sur un réseau tout en réalisant un scan sur un service spécifique en même temps. De la même façon, un attaquant peut décider de déployer plusieurs stratégies de leurrage simultanément.

La détection des attaques Dans notre travail, nous n'avons pas non plus considéré les systèmes de détection d'intrusion. Le défenseur sélectionne ses actions par rapport à ce qu'il détecte. Or, chaque attaque de l'attaquant n'a pas la même probabilité d'être détectée, certaines attaques sont plus discrètes que d'autres. Leur durée, le bruit qu'elles génèrent et le service visé va beaucoup influer sur leur discrétion. De plus, les systèmes de détection d'intrusion pourraient être perturbés par l'utilisation de stratégies MTD, nous n'avons pas non plus considéré cet aspect.

L'implémentation des stratégies Un réseau d'entreprise peut être séparé en secteurs et certaines stratégies que nous avons considérées agissent finalement sur plusieurs d'entre eux. Par exemple, une entreprise aurait plusieurs réseaux internes, un pour les utilisateurs standards, un pour les services externes, etc... Une stratégie MTD mélangeant les adresses IP devrait donc être spécifique pour chaque réseau. De la même façon, l'endroit où est déployé un service de leurrage a beaucoup d'importance. Enfin la manière dont sont implémentés les stratégies de défense va beaucoup influencer les performances de ces dernières. De la même façon, nous avons fait une sélection des stratégies de défense tout comme des attaques ce qui va influer sur nos résultats car certaines stratégies de défense non considérées seraient peut-être plus efficaces contre certaines attaques considérées ou réciproquement.

Comparaison de l'optimisation des stratégies dans un modèle ou dans un environnement réel

L'optimisation de la sélection des stratégies déceptives dans un modèle ou dans un environnement réel offre chacun divers avantages et inconvénients.

Les données pour les expérimentations A l'origine, si l'on propose un modèle, c'est pour pouvoir généraliser le comportement et les variations d'un objet et de ses attributs notamment à des cas où l'on ne peut pas connaître le comportement réel. Un modèle va tout de même avoir besoin de données réelles. Dans notre cas, pour bien paramétrer le modèle, il faut récupérer des informations d'attaques réelles dans un environnement utilisant des stratégies

MTD pour paramétrer le modèle. Pour cela, il est aussi possible d'utiliser des études sur des stratégies spécifiques. Par exemple, Clark et al. [31] nous permet de connaître l'efficacité d'une mutation sur les adresses IP contre des scanners externes et Luo et al. [33] fournit l'efficacité de mutations sur les ports des machines pour empêcher certaines attaques. De telles études permettent de mieux paramétrer notre modèle. Cependant, il est difficile de comparer l'efficacité des solutions si elles ne sont pas testées dans les mêmes conditions. De plus, utiliser les résultats d'études spécifiques ne considère pas l'influence de chaque stratégie sur les autres, ni des conflits potentiellement créés, notamment par les implémentations, ni les bénéfices de combiner plusieurs stratégies. En revanche, les derniers points énoncés sont un avantage de l'environnement réel. Si certains conflits d'implémentations sont présents entre différentes stratégies déceptives, ils sont directement observables dans les tests. L'environnement réel n'a pas non plus besoin de données antérieures sur les stratégies de défense. L'efficacité de chacune d'entre elles contre chaque attaque est mesurable directement lors des tests mais nous n'avons pas besoin de la connaître. En effet, l'efficacité de chaque stratégie est directement prise en compte par l'agent défenseur lors de son entraînement puisqu'il a un retour sur la progression de l'attaquant et donc sur l'efficacité des stratégies utilisées.

La durée d'entraînement Une autre différence importante entre les deux approches est le temps des expérimentations. Dans l'environnement réel, comme ce sont de vraies attaques qui sont implémentées, leurs durées sont assez significatives ce qui ralentit grandement les expérimentations et les apprentissages. A l'opposé, dans le modèle, les actions ne sont pas réelles, elles ne font qu'agir sur le modèle et non sur un environnement réel. Elles ont donc une période très courte ce qui permet des apprentissages sur beaucoup plus d'épisodes.

L'influence de la perception Dans le chapitre 4, nous avons vu que dans l'environnement réel, nos agents attaquants sont rapidement mauvais contre un défenseur utilisant des stratégies MTD. Cela montre deux choses. D'une part, cela montre l'efficacité des stratégies MTD contre un agent attaquant automatisé entraîné sur un environnement statique. D'autres part, un attaquant humain a davantage d'observations que nos agents et n'utilise pas une version condensée d'informations. Comme nous fournissons les observations à un réseau de neurones pour entraîner nos agents, nous devons représenter celles-ci sous forme vectorielle ce qui induit une perte de données dans les observations. L'humain fait des liens entre les différents éléments qu'il observe et fait des déductions qui sont difficiles à apprendre à une machine. Ces liens qu'il fait vont influencer sa perception et c'est pourquoi un modèle de perception des joueurs est intéressant. En effet, la perception et la capacité de déduction d'un agent attaquant sont bien plus limitées que celles d'un humain. Pour faire face à ce

problème, nous pouvons imaginer des réseaux de neurones plus grands avec des observations plus précises mais les agents seront tout de suite beaucoup plus durs à entraîner. Une telle approche ne semble pas raisonnable dans un environnement réel à cause du temps nécessaire pour l'entraînement et du coût pour déployer plusieurs environnements réels si l'on souhaitait paralléliser l'entraînement. Dans ce cas, si l'on ne peut pas améliorer la perception d'un agent au même niveau de celle de l'attaquant humain, il est peut-être mieux de simuler la perception du joueur grâce à un modèle.

Amélioration du modèle Nos expériences dans un environnement de test nous ont permis de faire certaines observations qui nous permettraient d'améliorer notre modèle. En effet, la fréquence des actions de chaque joueur ne doit pas être la même pour les deux joueurs car les actions du défenseur ont un plus grand impact ainsi qu'une influence sur la qualité des services déployés. Cela devrait être pris en compte pour améliorer notre modèle.

Pour toutes les stratégies, la position des éléments déceptifs devrait aussi être considérée notamment s'il s'agit d'éléments internes ou externes. Cela peut facilement être fait en considérant plusieurs stratégies utilisant la même technique mais a des positions différentes dans le système. Il suffirait alors de modifier l'efficacité face aux stratégies de d'attaques de chacune d'entre elles.

De plus, nous avons considéré que les stratégies de leurrage pouvaient être utilisées comme les stratégies MTD même si elles ont un impact différent sur la perception des joueurs et sur l'environnement. Pour être plus proche d'une utilisation réelle de ces stratégies, il faudrait plutôt considérer comme stratégie de l'attaquant d'un côté le déploiement de la stratégie et de l'autre l'arrêt de ces stratégies comme nous avons fait dans notre environnement de test. Le coût de déploiement d'une stratégie devrait alors être proportionnel au temps pendant laquelle celle-ci a été déployée. Le modèle pourrait aussi être complété en considérant d'autres stratégies déceptives comme l'utilisation de perturbations par exemple des entêtes d'une requête web.

CHAPITRE 6 CONCLUSION

6.1 Synthèse des travaux

Nos travaux avait pour but l'optimisation de la sélection des stratégies MTD et des stratégies de Leurrage afin d'améliorer la cyber-résilience et la sécurité d'un système. Dans le chapitre 3 nous nous sommes intéressés à l'optimisation de ces stratégies dans un modèle attaquant/défenseur considérant l'asymétrie de perception entre chacun des joueurs et la différence de nature et d'effets entre les stratégies MTD et celles de leurrage. L'utilisation de l'algorithme DQN a permis d'entraîner un agent défenseur face à un scénario d'attaquant assez simple et basé sur la Cyber Kill Chain.

Dans le chapitre 4, nous avons vérifié si l'optimisation des stratégies du défenseur était toujours possible dans un environnement réel. Pour cela, nous avons utilisé l'algorithme PPO. Ici, au lieu d'utiliser un scénario d'attaquant, nous avons entraîné un attaquant à pénétrer le système et à atteindre un objectif d'exfiltration de données sensibles. Nous avons ensuite entraîné un défenseur à le contrer et nous avons montré que les stratégies MTD étaient très efficaces pour contrer un agent attaquant entraîné sur un environnement statique avec ou sans éléments de leurrage. Les résultats restent similaire lorsque l'intervalle entre les actions du défenseur est plus grand. Nous avons déduit de cela que l'entraînement de notre agent attaquant dans un environnement statique ne lui permettait pas du tout de s'adapter à un environnement mouvant. L'amélioration de l'agent défenseur passera donc forcément par une amélioration de l'agent attaquant.

6.2 Limitations

Nous avons déjà discuté des limitations de nos travaux dans les sections 3.5 et 4.7 lors des discussions sur les approches utilisées.

Concernant le modèle, la principale limitation est la nécessité de récupérer des données réelles pour permettre d'ajuster les paramètres du modèle et pour le valider. Nous avons montré que dans notre modèle avec des paramètres fixés arbitrairement, l'entraînement de l'agent défenseur est possible.

Concernant l'environnement implémenté, notre travail cherche à montrer que l'entraînement du défenseur est possible dans un environnement réel. Pour nos tests, nous avons considéré seulement quelques types de vulnérabilités. De plus, même si les actions de l'attaquant sont

réellement exécutées, nous utilisons une échelle de temps discrète. Nous ne considérons donc pas le fait que plusieurs actions puissent être exécutées en parallèle. De plus, la détection des attaques a été considérée comme assurée dans cette partie ce qui constitue une autre limite de notre travail.

6.3 Améliorations futures

Au vue des limitations de notre travail, une première amélioration serait d'intégrer la détection des attaques à notre environnement implémenté. Les stratégies de leurrage peuvent être un outil utile pour détecter des attaques. Il faudrait donc optimiser le déploiement de ce type de stratégie en considérant à la fois leur capacité à ralentir l'attaquant et leur capacité à détecter des attaques.

De plus, dans notre travail, nous avons entraîné nos agents défenseurs face à un type d'attaquant. L'objectif de ce dernier était de pénétrer dans le système pour par exemple exfiltrer des informations. Cependant, les objectifs d'un attaquant pourraient être très différents comme réduire la disponibilité d'un service par exemple. Pour poursuivre notre travail, il faudrait donc entraîner notre agent défensif contre une variété d'attaques informatiques dont les objectifs sont différents. De la même façon, nous nous sommes concentrés sur l'utilisation de quelques stratégies déceptives mais il en existe d'autres qui pourraient être davantage adaptées contre certains types d'attaques.

Nous avons aussi vu qu'une limite à l'utilisation d'un environnement réel est l'absence de généralisation possible. A l'opposé un modèle est généralisable mais les difficultés concerne la validation de ce dernier. Pour faire face à ces problèmes, une approche intermédiaire serait d'optimiser la sélection de stratégie MTD et de leurrage dans un environnement d'entreprise simulé comme par exemple CyberBattleSim [70]. Cette plateforme propose une représentation réaliste des éléments d'un réseau d'entreprises. Certains éléments de leurrage statiques y sont déjà intégrés mais à ma connaissance, ce n'est pas le cas de stratégies MTD. De plus, pour le moment cette solution se concentre sur l'entraînement des agents attaquants, il faudrait essayer d'introduire un agent défenseur dans cet environnement de simulation et voir comment la notion de perception pourrait y être ajoutée pour influencer la prise de décision des deux joueurs.

Finalement, dans un environnement d'entreprise, de nombreux autres systèmes sont présents et pourraient rentrer en conflit avec des stratégies MTD ou de leurrage. Les outils de monitoring ou les systèmes de détection d'intrusions pourraient être perturbés par de telles stratégies. Pour prolonger notre travail, nous pourrions chercher comment intégrer celles-ci

à un environnement d'entreprise pré-existant. Par exemple, un système de détection d'intrusion réseau serait inefficace si les adresses IP d'un réseau changent régulièrement. Il faudrait alors trouver une méthode pour indiquer à ce système de détection la correspondance entre les anciennes adresses IP et les nouvelles.

RÉFÉRENCES

[1] C. Sécurité, "La sécurité dans les petite et moyenne entreprise," Cisco Sécurité, Rapport technique, 2020.

[2] R. Ross, V. Pillitteri, K. Dempsey, M. Riddle et G. Guissanie, "Protecting controlled unclassified information in nonfederal systems and organizations," National Institute of Standards and Technology, Rapport technique, 2019.

[3] F. Björck, M. Henkel, J. Stirna et J. Zdravkovic, "Cyber resilience–fundamentals for a definition," dans *New contributions in information systems and technologies*. Springer, 2015, p. 311–316.

[4] I. Linkov et A. Kott, "Fundamental concepts of cyber resilience : Introduction and overview," *Cyber resilience of systems and networks*, p. 1–25, 2019.

[5] D. L. Schuh, "The cyberspace advantage : Inviting them in-how cyber deception enables better resilience," MITRE CORP MCLEAN VA, Rapport technique, 2020.

[6] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau et R. McQuaid, "Developing cyber resilient systems : a systems security engineering approach," National Institute of Standards and Technology, Rapport technique, 2019.

[7] E. Peter et T. Schiller, "A practical guide to honeypots," *Washington Univerity*, 2011.

[8] D. Fraunholz, S. D. Anton, C. Lipps, D. Reti, D. Krohmer, F. Pohl, M. Tammen et H. D. Schotten, "Demystifying deception technology : A survey," *arXiv preprint arXiv :1804.06196*, 2018.

[9] X. Han, N. Kheir et D. Balzarotti, "Deception techniques in computer security : A research perspective," *ACM Computing Surveys (CSUR)*, vol. 51, n°. 4, p. 1–36, 2018.

[10] J. Pawlick, E. Colbert et Q. Zhu, "A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy," *ACM Computing Surveys (CSUR)*, vol. 52, n°. 4, p. 1–28, 2019.

[11] J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim et F. F. Nelson, "Toward proactive, adaptive defense : A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, vol. 22, n°. 1, p. 709–745, 2020.

[12] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang et S. Kambhampati, "A survey of moving target defenses for network security," *IEEE Communications Surveys & Tutorials*, vol. 22, n°. 3, p. 1909–1941, 2020.

[13] J. Zheng et A. S. Namin, “A survey on the moving target defense strategies : An architectural perspective,” *Journal of Computer Science and Technology*, vol. 34, n°. 1, p. 207–233, 2019.

[14] K. Bissell, J. Fox, R. M. LaSalle et P. D. Cin, “State of cybersecurity resilience 2021,” Accenture, Rapport technique, 2021.

[15] A. Juels et R. L. Rivest, “Honeywords : Making password-cracking detectable,” dans *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, p. 145–160.

[16] M. H. Almeshekah, “Using deception to enhance security : A taxonomy, model, and novel uses,” Thèse de doctorat, Purdue University, 2015.

[17] A. Clark, Q. Zhu, R. Poovendran et T. Başar, “Deceptive routing in relay networks,” dans *International Conference on Decision and Game Theory for Security*. Springer, 2012, p. 171–185.

[18] A. Clark, K. Sun, L. Bushnell et R. Poovendran, “A game-theoretic approach to ip address randomization in decoy-based cyber defense,” dans *International conference on decision and game theory for security*. Springer, 2015, p. 3–21.

[19] X. Feng, Z. Zheng, P. Mohapatra et D. Cansever, “A stackelberg game and markov modeling of moving target defense,” dans *International Conference on Decision and Game Theory for Security*. Springer, 2017, p. 315–335.

[20] J. Pawlick et Q. Zhu, “Deception by design : evidence-based signaling games for network defense,” *arXiv preprint arXiv :1503.05458*, 2015.

[21] H. Çeker, J. Zhuang, S. Upadhyaya, Q. D. La et B.-H. Soong, “Deception-based game theoretical approach to mitigate dos attacks,” dans *International conference on decision and game theory for security*. Springer, 2016, p. 18–38.

[22] Q. D. La, T. Q. Quek, J. Lee, S. Jin et H. Zhu, “Deceptive attack and defense game in honeypot-enabled networks for the internet of things,” *IEEE Internet of Things Journal*, vol. 3, n°. 6, p. 1025–1035, 2016.

[23] I. Mokube et M. Adams, “Honeypots : concepts, approaches, and challenges,” dans *Proceedings of the 45th annual southeast regional conference*, 2007, p. 321–326.

[24] L. Spitzner, “Honeypots : Catching the insider threat,” dans *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE, 2003, p. 170–179.

[25] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil et J. Schönfelder, “A survey on honeypot software and data analysis,” *arXiv preprint arXiv :1608.06249*, 2016.

[26] M. Albanese, E. Battista et S. Jajodia, “A deception based approach for defeating os and service fingerprinting,” dans *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2015, p. 317–325.

[27] F. Araujo, K. W. Hamlen, S. Biedermann et S. Katzenbeisser, “From patches to honey-patches : Lightweight attacker misdirection, deception, and disinformation,” dans *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, p. 942–953.

[28] J. Avery et E. H. Spafford, “Ghost patches : Fake patches for fake vulnerabilities,” dans *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2017, p. 399–412.

[29] L. Zhang et V. L. Thing, “Three decades of deception techniques in active cyber defense-retrospect and outlook,” *Computers & Security*, vol. 106, p. 102288, 2021.

[30] E. Al-Shaer, Q. Duan et J. H. Jafarian, “Random host mutation for moving target defense,” dans *International Conference on Security and Privacy in Communication Systems*. Springer, 2012, p. 310–327.

[31] A. Clark, K. Sun et R. Poovendran, “Effectiveness of ip address randomization in decoy-based moving target defense,” dans *52nd IEEE Conference on Decision and Control*. IEEE, 2013, p. 678–685.

[32] J. H. Jafarian, E. Al-Shaer et Q. Duan, “Openflow random host mutation : transparent moving target defense using software defined networking,” dans *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, p. 127–132.

[33] Y.-B. Luo, B.-S. Wang et G.-L. Cai, “Effectiveness of port hopping as a moving target defense,” dans *2014 7th International Conference on Security Technology*. IEEE, 2014, p. 7–10.

[34] A. R. Chavez, W. M. Stout et S. Peisert, “Techniques for the dynamic randomization of network attributes,” dans *2015 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2015, p. 1–6.

[35] S. W. Boyd, G. S. Kc, M. E. Locasto, A. D. Keromytis et V. Prevelakis, “On the general applicability of instruction-set randomization,” *IEEE Transactions on Dependable and Secure Computing*, vol. 7, n°. 3, p. 255–270, 2008.

[36] S. W. Boyd et A. D. Keromytis, “Sqlrand : Preventing sql injection attacks,” dans *International conference on applied cryptography and network security*. Springer, 2004, p. 292–302.

[37] M. Taguinod, A. Doupé, Z. Zhao et G.-J. Ahn, "Toward a moving target defense for web applications," dans *2015 IEEE International Conference on Information Reuse and Integration*. IEEE, 2015, p. 510–517.

[38] M. Zhu, A. H. Anwar, Z. Wan, J.-H. Cho, C. Kamhoua et M. P. Singh, "A survey of defensive deception : Approaches using game theory and machine learning," *IEEE Communications Surveys & Tutorials*, 2021.

[39] S. Sengupta et S. Kambhampati, "Multi-agent reinforcement learning in bayesian stackelberg markov games for adaptive moving target defense," *arXiv preprint arXiv :2007.10457*, 2020.

[40] M. A. Rahman, M. H. Manshaei et E. Al-Shaer, "A game-theoretic approach for deceiving remote operating system fingerprinting," dans *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2013, p. 73–81.

[41] J. Pawlick, E. Colbert et Q. Zhu, "Modeling and analysis of leaky deception using signaling games with evidence," *IEEE Transactions on Information Forensics and Security*, vol. 14, nº. 7, p. 1871–1886, 2018.

[42] P. K. Manadhata, "Game theoretic approaches to attack surface shifting," dans *Moving Target Defense II*. Springer, 2013, p. 1–13.

[43] A. H. Anwar, C. Kamhoua et N. Leslie, "Honeypot allocation over attack graphs in cyber deception games," dans *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2020, p. 502–506.

[44] K. Horák, Q. Zhu et B. Bošanskỳ, "Manipulating adversary's belief : A dynamic game approach to deception by design for proactive network security," dans *International Conference on Decision and Game Theory for Security*. Springer, 2017, p. 273–294.

[45] J.-H. Cho, M. Zhu et M. Singh, "Modeling and analysis of deception games based on hypergame theory," dans *Autonomous Cyber Deception*. Springer, 2019, p. 49–74.

[46] N. S. Kovach, A. S. Gibson et G. B. Lamont, "Hypergame theory : a model for conflict, misperception, and deception," *Game Theory*, vol. 2015, 2015.

[47] H. Hu, J. Liu, J. Tan et J. Liu, "Socmtd : selecting optimal countermeasure for moving target defense using dynamic game," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 14, nº. 10, p. 4157–4175, 2020.

[48] C. Lei, D.-H. Ma et H.-Q. Zhang, "Optimal strategy selection for moving target defense based on markov game," *IEEE Access*, vol. 5, p. 156–169, 2017.

[49] H. Zhang, K. Zheng, X. Wang, S. Luo et B. Wu, "Strategy selection for moving target defense in incomplete information game," *Computers, Materials & Continua*, vol. 62, nº. 2, p. 763–786, 2020.

[50] J.-l. Tan, C. Lei, H.-q. Zhang et Y.-q. Cheng, "Optimal strategy selection approach to moving target defense based on markov robust game," *computers & security*, vol. 85, p. 63–76, 2019.

[51] H. Zhang, K. Zheng, X. Wang, S. Luo et B. Wu, "Efficient strategy selection for moving target defense under multiple attacks," *IEEE Access*, vol. 7, p. 65 982–65 995, 2019.

[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra et M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv :1312.5602*, 2013.

[53] X. Chai, Y. Wang, C. Yan, Y. Zhao, W. Chen et X. Wang, "Dq-motag : Deep reinforcement learning-based moving target defense against ddos attacks," dans *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2020, p. 375–379.

[54] Q. Jia, K. Sun et A. Stavrou, "Motag : Moving target defense against internet denial of service attacks," dans *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2013, p. 1–9.

[55] T. Eghtesad, Y. Vorobeychik et A. Laszka, "Deep reinforcement learning based adaptive moving target defense," *arXiv preprint arXiv :1911.11972*, 2019.

[56] A. Prakash et M. P. Wellman, "Empirical game-theoretic analysis for moving target defense," dans *Proceedings of the second ACM workshop on moving target defense*, 2015, p. 57–65.

[57] S. Wang, Q. Pei, J. Wang, G. Tang, Y. Zhang et X. Liu, "An intelligent deployment policy for deception resources based on reinforcement learning," *IEEE Access*, vol. 8, p. 35 792–35 804, 2020.

[58] G. F. Lyon, *Nmap network scanning : The official Nmap project guide to network discovery and security scanning*. Insecure. Com LLC (US), 2008.

[59] CAPEC, "Common attack pattern enumeration and classification (capec)," 2022. [En ligne]. Disponible : https://capec.mitre.org/

[60] CVE, "Common vulnerabilities and exposures (cve)," 2022. [En ligne]. Disponible : https://cve.mitre.org/

[61] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang et W. Zaremba, "Openai gym," *arXiv preprint arXiv :1606.01540*, 2016.

[62] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus et N. Dormann, "Stable-baselines3 : Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, n°. 268, p. 1–8, 2021. [En ligne]. Disponible : http://jmlr.org/papers/v22/20-1364.html

[63] D. Merkel, “Docker : lightweight linux containers for consistent development and deployment,” *Linux journal*, vol. 2014, n°. 239, p. 2, 2014.

[64] R. H. Ansible, “Ansible is simple it automation.” [En ligne]. Disponible : https://www.ansible.com/

[65] J. Schulman, F. Wolski, P. Dhariwal, A. Radford et O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv :1707.06347*, 2017.

[66] “Oracle vm virtualbox.” [En ligne]. Disponible : https://www.oracle.com/virtualization/virtualbox/index.html

[67] “Debian.” [En ligne]. Disponible : https://www.debian.org/index.fr.html

[68] A. F. Agarap, “Deep learning using rectified linear units (relu),” *arXiv preprint arXiv :1803.08375*, 2018.

[69] A. M. Basbrain, J. Q. Gan, A. Sugimoto et A. Clark, “A neural network approach to score fusion for emotion recognition,” dans *2018 10th Computer Science and Electronic Engineering (CEEC)*. IEEE, 2018, p. 180–185.

[70] M. D. R. Team., “Cyberbattlesim,” https://github.com/microsoft/cyberbattlesim, 2021, created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei.