Améliorations des outils de détection de malwares par analyse statique grâce à des mécanismes d'intelligence artificielle

Benjamin Marais

Normandie Université

UNIVERSITÉ CAEN NORMANDIE

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité **MATHEMATIQUES**

Préparée au sein de l'**UNIVERSITE CAEN NORMANDIE**

Améliorations des outils de détection de malwares par analyse statique grâce à des mécanismes d'intelligence artificielle

Présentée et soutenue par

BENJAMIN MARAIS

Thèse soutenue le 18/12/2023

devant le jury composé de :

M. REMI BADONNEL	Professeur des universités - UNIVERSITE DE LORRAINE	Rapporteur du jury
M. BASARAB MATEI	Maître de conférences HDR - UNIVERSITE PARIS 4 PARIS-SORBONNE	Rapporteur du jury
MME ELISA FROMONT	Professeur des universités - IRISA/INRIA Rennes	Membre du jury
M. CLEMENT LEVRARD	Professeur des universités - Université de Rennes (EPE)	Membre du jury
M. TONY QUERTIER	ASSISTANT INGENIEUR DE RECH.ET FORMATION - ORANGE INNOVATION	Membre du jury
M. MOHAMED DIDI BIHA	Professeur des universités - UNIVERSITE CAEN NORMANDIE	Président du jury
M. CHRISTOPHE CHESNEAU	Maître de conférences - UNIVERSITE CAEN NORMANDIE	Directeur de thèse

Thèse dirigée par **CHRISTOPHE CHESNEAU** (Laboratoire de Mathématiques 'Nicolas Oresme' (Caen))

Remerciements

Dans un premier temps, je tiens à remercier mon directeur de thèse, Christophe Chesneau, pour avoir été présent tout au long de cette thèse, pour ces conseils et le temps qu'il m'a accordé.

Je tiens également à remercier très sincèrement Tony Quertier. Je te remercie de m'avoir fait confiance et pour m'avoir accompagné dans cette aventure. Merci pour ta bienveillance, tes encouragements et tes conseils. Je ne pense pas que j'aurais pu espérer un meilleur encadrement. Merci de m'avoir donné goût à la recherche scientifique, et de m'avoir aidé à développer mes qualités professionnelles, mais aussi personnelles. Je souhaite également remercier mes collègues de Orange Innovation, en particulier Philippe Calvet, Stéphane Morucci et Sok-Yen Loui, pour leur soutien et leur confiance.

J'adresse également mes remerciements à Rémi Badonnel et Basarab Matei qui ont accepté de rapporter mon manuscrit de thèse, ainsi qu'à Mohamed Didi Biha, Elisa Fromont et Clément Levrard qui ont accepté de faire partie de mon jury de soutenance. Je vous remercie pour l'intérêt que vous portez pour mon sujet de thèse et mes travaux.

Je tiens aussi à remercier chaleureusement mes amis qui ont été présents tout au long de ma thèse, et qui ont su m'épauler dans des périodes parfois troubles. Merci à mes amis de Rennes pour le temps passé avec vous, pour avoir été là dans les moments les plus compliqués, comme les plus joyeux. Merci à mes amis de Nantes pour m'avoir toujours accueilli les bras ouverts, souvent à l'improviste, et pour m'avoir permis de souffler et de décompresser le temps d'un week-end. Merci à mes amis de Caen de toujours trouver un peu de temps pour moi à chacun de mes passages. Enfin, merci à mes amis de la Manche pour être toujours là après les années.

Je remercie également, et du fond du cœur, mes parents et mon frère, pour leur soutien tout au long de mes études, en particulier au cours de ces trois ans de thèse, mais aussi pour leur bienveillance et leur amour au quotidien.

Enfin, je souhaite remercier mon grand-père, Yves. Je te remercie pour les moments passés au coin du feu à parler des sujets les plus anodins comme des plus sérieux. Tu as toujours été à l'écoute et tes conseils ont toujours été d'une aide précieuse. Je te remercie pour le temps passé ensemble.

Table des matières

Introduction

La sécurité des systèmes d'information et des données s'est rapidement imposée comme un domaine à part entière dans les métiers de l'informatique et des technologies de l'information. En effet, avec l'expansion du réseau internet, des technologies connectées et des services de cloud, le nombre de menaces n'as jamais été aussi grand [Asl+23]. Malgré tout, les motivations des cybercriminels et des hackers restent principalement les mêmes au fil des années selon l'ANSSI [ANS23] : l'extorsion de fonds, l'espionnage et le vol de données, la déstabilisation d'entreprise ou de structures étatiques. En plus de menacer le bon fonctionnement des entreprises et des services publics, la cybercriminalité à un impact non négligeable sur l'économie mondiale avec un coût total des pertes occasionnées par des cyber-attaques estimé à 6 000 milliards de dollars en 2021.

Parmi les vecteurs d'attaques et d'infections existants, l'utilisation de logiciels malveillants (malicious software, ou malware) reste une des méthodes privilégiées. En effet, ce sont pas moins de 100 millions de nouveaux fichiers malveillants découverts en 2022 selon le site AV-Test. Malgré une baisse de 33% par rapport à l'année 2021, principalement due au fait que la guerre en Ukraine à grandement influencé les activités cybercriminelles mondiales [Hun23], la menace des fichiers malveillants reste tout de même présente et devrait davantage se développer dans les années à venir. Depuis le début de l'épidémie du COVID-19, les campagnes massives de cyber-attaques se sont multipliées [Lal+21] et un nouveau marché à vu le jour nommé "Malware-as-a-Service" (MaaS) [Sop2323]. Ce marché permet d'avoir accès à un panel de solutions d'attaques basées sur des fichiers malveillants et en particulier des fichiers de type ransomware (Ransomware-as-a-Service, RaaS). Ainsi, les attaques par ransomware ne sont plus limitées à des individus ou des groupes spécialisés, mais deviennent un produit à part entière sur le marché du cybercrime et sont accessibles à n'importe qui sans avoir besoin de connaissances poussées en programmation. L'essor du marché des RaaS est d'autant plus inquiétant quand on sait qu'un pays comme le Costa-Rica a été partiellement paralysé après une attaque de plusieurs structures gouvernementales par les ransomwares Conti [Dre] et Hive [Bur22].

Malgré les avancées et innovations dans les domaines de l'analyse et de la détection de fichiers malveillants, les attaques restent nombreuses et dommageables. Les attaquants et les créateurs de malware ont une longueur d'avance sur les éditeurs de solutions de détection et exploitent souvent des vulnérabilités inconnues ou bien contournent des mises à jour de sécurité pour améliorer l'efficacité de leurs fichiers malveillants [Cro23]. Avec l'essor de l'intelligence artificielle (IA) dans les dernières années, de nombreux travaux de recherche ont été conduits afin de proposer des solutions de détection basées sur cette technologie [UAB19 ; GMP20 ; Tay+22]. Malgré des résultats souvent prometteurs, l'utilisation de l'intelligence artificielle dans le monde de la

cybersécurité reste limitée. Tout d'abord, le décalage de point de vue entre les experts en cybersécurité et les chercheurs en IA résultent en une déconnexion de ces deux communautés [Smi+20]. De plus, les modèles de détection basés sur l'IA sont sujets à des défaillances connues comme les exemples adverses [GSS15] ou la dérive conceptuelle [Žli10]. La recherche en cybersécurité, en particulier dans le domaine de l'analyse de fichiers malveillants, est un terreau fertile et mérite toute notre attention afin d'améliorer la sécurité des personnes et des entreprises.

Plan et contributions

Le prochain chapitre présente les notions importantes sur les sujets de l'analyse de malware et de l'intelligence artificielle. La suite du document est séparé en trois parties qui peuvent être lues de façon indépendante :

- La partie I est une partie orientée analyse défensive. Elle regroupe différents travaux sur l'analyse et la détection de fichiers malveillants. En autre, nous présentons trois outils qui permettent de faire un examen préliminaire d'un fichier PE afin d'avoir une idée de sa nature et de son fonctionnement. Nous proposons aussi différents modèles de détection, entraînés à partir d'algorithmes d'apprentissage supervisé classiques et profonds et différentes techniques de prétraitement,
- la partie II est, quant à elle, orientée analyse offensive. Nous y présentons des modèles capables de contourner les antivirus classiques et les solutions de détection basées sur l'IA. L'objectif est d'identifier leurs faiblesses afin de fournir un rapport de vulnérabilités qui sera utile pour améliorer ces outils,
- la partie III regroupe différents travaux réalisés au cours de la thèse. Entre autres, nous abordons le sujet de la dérive conceptuelle qui rend les modèles obsolètes avec le temps, ou la détection de fichiers chiffrés ou compressés.

Chapitre 1

Chapitre Préliminaire

Dans ce chapitre, nous introduisons les concepts et les notions relatifs à l'analyse et la détection de fichiers malveillants, ainsi qu'à l'intelligence artificielle, en particulier à l'apprentissage supervisé. Nous faisons aussi le lien entre les méthodes classiques d'analyse de malware et les méthodes utilisant l'intelligence artificielle.

1.1 Les fichiers malveillants

McGraw et Morrisett [MM00] (def. 1) définissent du code malveillant comme tout ou partie d'un fichier dont le but est de réaliser un exploit informatique, c'est-à-dire d'infiltrer ou d'endommager un système informatique à l'insu de ses utilisateurs. On définit donc un logiciel malveillant (malicious software, ou malware) comme un fichier dont le but est de compromettre un système informatique à des fins peu scrupuleuses.

Définition 1 *Malicious code is any code added, changed, or removed from a software system to intentionally cause harm or subvert the system's intended function. - G. McGraw et G. Morrisett [MM00]*

Il existe différentes catégories de fichiers malveillants selon les objectifs des attaquants. Par exemple, un fichier de type **ransomware** (rançongiciel) chiffre une machine ciblée afin d'en empêcher l'utilisation, et demande une rançon à l'utilisateur pour la débloquer. Un **spyware** (logiciel espion) permet d'espionner un utilisateur et de voler des informations personnelles comme des mots de passe ou des informations bancaires. En général, le mobile des cybercriminels est l'extorsion de fond, l'espionnage industriel ou politique et/ou la déstabilisation économique et politique. La table 1.1 résume les différentes catégories de fichiers malveillants et leurs usages les plus connus.

Les vecteurs d'infections sont nombreux et différents selon les systèmes d'exploitation (ou OS) ciblés. On retrouvera par exemple les fichiers PDF (.pdf), les documents de la suite Office type Word (.doc ou .docx), Excel (.xls ou .xlsx) ou Powerpoint (.ppt ou pptx), ou bien les fichiers exécutables comme les fichiers ELF sous Linux, APK sous Androïde ou Portable Executable (PE) sous Windows. Au cours de nos travaux, nous nous sommes particulièrement intéressés à l'analyse et la détection des fichiers Portable Executable car Windows reste un des OS les plus utilisés à l'heure actuelle.

		Usages		
		Extorsion de fonds	Espionnage	Déstabilisation
Type de fichier	Ransomware	×		
	Trojan		×	
	Backdoor		×	
	Worms			×
	Bots			×
	Keylogger		×	
	Rootkits		×	
	Adware			×
	Spyware		×	
	Virus			×

TABLE 1.1 – Type de fichiers malveillants et leurs usages

1.1.1 Le format Portable Executable

Le format PE est un format de fichier hérité du format COFF (Common Object File Format) principalement utilisé par les systèmes UNIX. Ce format a été créé en 1993 par Microsoft [Mic] pour être utilisé par les fichiers des systèmes d'exploitation Windows. On retrouve principalement les fichiers DLL (.dll) et exécutable (.exe). Comme le montre la figure 1.1, un fichier PE est séparé en deux parties : l'en-tête (header) et les sections. L'en-tête décrit le fichier et son contenu. Elle contient, en autre, des informations comme la date de création du fichier, des informations sur le chargement du fichier en mémoire ou le nombre de sections. Chaque section est décrite par un en-tête spécifique où l'on retrouve son nom, sa taille et son emplacement en mémoire virtuelle. Les sections contiennent généralement le code de l'exécutable (.text), les variables utilisées et leurs valeurs par défaut (.data) ou bien les informations sur les fonctions (DLL) nécessaires à l'exécution du fichier (.idata).

L'étude de ces informations contenues dans l'en-tête et les sections d'un fichier, ainsi que de son comportement (et de son objectif), permet de déterminer si un fichier est potentiellement malveillant. Cette pratique est essentielle puisqu'elle permet de détecter et contrer les attaques de malware qui pourraient compromettre la sécurité d'un système d'information.

1.1.2 Analyse et détection de fichiers malveillants

Analyse de logiciels malveillants

L'analyse de fichiers malveillants est une étape qui consiste à analyser un malware afin d'en comprendre le comportement ainsi que les principaux dégâts qu'il peut provoquer. L'analyse de malware permet aussi aux chercheurs et aux analystes en cybersécurité de développer de nouveaux outils de détection, d'identifier les vecteurs d'infection actuels et d'améliorer les solutions de sécurité afin de protéger les systèmes et réseaux d'éventuelles menaces. On distingue trois méthodes d'analyse qui sont l'analyse statique, l'analyse dynamique et l'analyse hybride [SOA18]. L'analyse statique consiste à examiner un fichier sans l'exécuter. Il s'agit en général de se concentrer sur l'analyse du

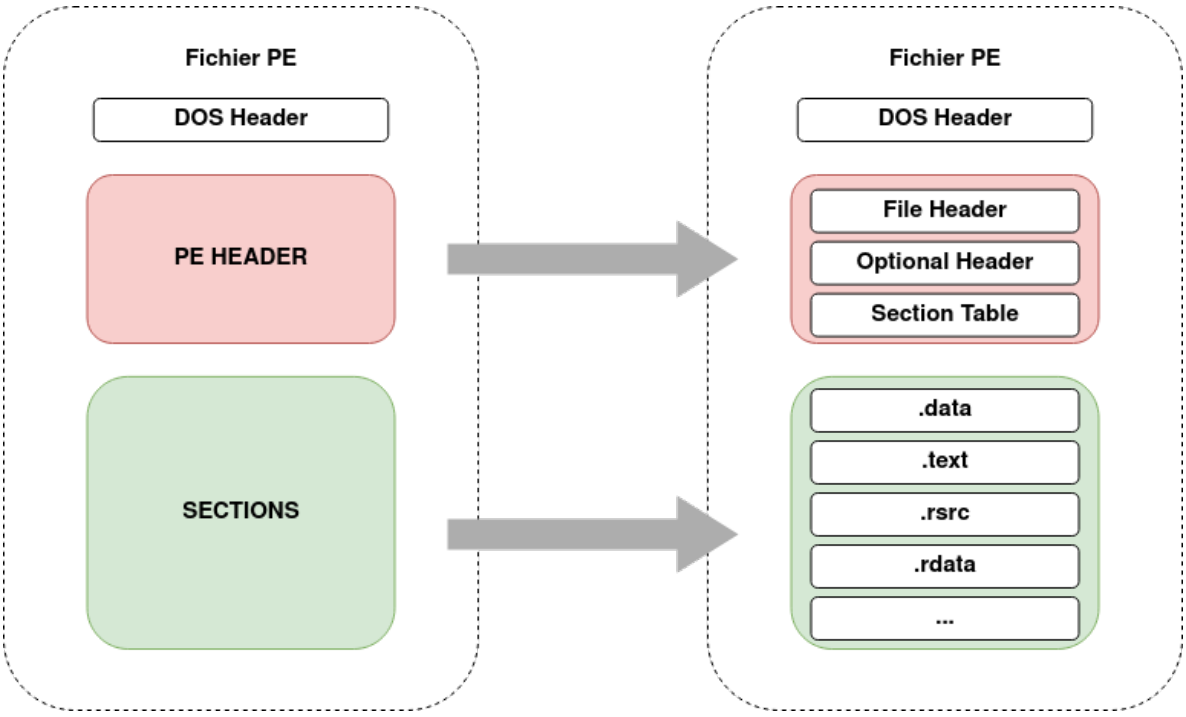


FIGURE 1.1 – Représentation simplifiée d'un fichier au format Portable Executable

code, de la structure du fichier et des métadonnées. Bien que l'analyse statique puisse fournir des premières informations sur la nature d'un fichier, cette méthode possède des failles [MKK07]. Si le fichier est compressé ou chiffré, l'accès à certaines parties ou à la totalité du fichier peut être compromis empêchant une analyse du code ou des données qu'il contient. D'autres moyens comme des méthodes d'obfuscation ou des techniques anti-désassemblages peuvent rendre le code illisible pour en empêcher sa compréhension. Enfin, certaines données nécessaires au fonctionnement du logiciel peuvent être téléchargées au moment de son exécution et donc non-identifiables par analyse statique.

L'analyse dynamique implique l'exécution d'un fichier dans un environnement sécurisé. En général, il s'agit d'un environnement virtuel qui simule un système d'exploitation (sandbox) ou d'une machine isolée du réseau. L'analyse dynamique permet d'observer l'exécution et le comportement d'un fichier, en se concentrant en particulier sur les appels systèmes, les communications réseaux et les opérations mémoires. L'analyse dynamique permet de mieux comprendre le comportement et l'impact du fichier sur le système. Néanmoins, cette méthode possède aussi des faiblesses [Or+19]. En particulier, certains logiciels peuvent détecter s'ils sont exécutés dans un environnement virtuel et ne pas fonctionner le cas échéant. Si le fichier a besoin de ressources stockées sur un serveur et que la sandbox n'est pas connectée au réseau, pour des raisons de sécurité, le fichier ne peut pas fonctionner. De plus, dans le cas d'analyse massive de fichiers, le temps d'exécution peut devenir une contrainte non négligeable.

Détection de logiciels malveillants

Le but est d'identifier un fichier malveillant avant qu'il n'ait pu agir et compromettre un système ou un réseau. On distingue plusieurs méthodes qui peuvent permettre la détection de fichiers malveillants. La méthode la plus connue est la détection par signa-

ture. Cette solution consiste à comparer un fichier à une base de données de signatures (une portion de code qui permet d'identifier un fichier spécifique). Cette solution est très efficace contre les fichiers malveillants connus et déjà rencontrés, mais peu efficace face à un malware inconnu ou bien face à des logiciels polymorphes [BKM05] ou métamorphiques [BM08], deux types de logiciel malveillant qui peuvent modifier leur "apparence" sans modifier leur mode d'exécution.

La détection de malware basée sur le comportement est une méthode qui analyse le fonctionnement et les actions d'un logiciel malveillant. Cette méthode peut s'appuyer sur des caractéristiques issues de l'analyse statique comme les API calls ou sur l'analyse dynamique via l'exécution du logiciel dans une sandboxe ou une machine virtuelle. Le but est d'identifier si un fichier a un comportement déviant ou suspect, comme par exemple l'élévation des privilèges, où un logiciel essaye d'obtenir des droits administrateur, ou la manipulation de processus afin d'exécuter des fonctions malveillantes à la place de processus légitimes (code injection, hooking [SH12, p. 259]). Ces comportements suspects sont nombreux et différents selon les catégories de malware. Cette méthode de détection permet, entre autres, d'identifier des logiciels inconnus, ou des variants connus ayant outrepassé un outil de détection par signature. Néanmoins, les outils basés sur l'analyse comportemental peuvent détecter certains fichiers bénins comme malveillants si ces derniers ont un fonctionnement peu conventionnel.

Les solutions de détection basées sur des heuristiques (règles) sont principalement utilisées pour la détection de nouveaux variants ou de nouveaux logiciels malveillants [Baz+13]. Ces solutions s'appuient sur des règles mises en place à partir de l'analyse de fichiers malveillants (analyse statique et analyse dynamique). Les analystes vont chercher à mettre en place des règles qui caractérisent un fichier malveillant comme des séquences particulières de code ou des comportements suspects. Cette méthode permet de tirer parti de l'analyse statique et de l'analyse dynamique, mais nécessite l'intervention d'analystes experts afin de créer les règles. De plus, si les règles sont trop spécifiques, cela peut engendrer un certain nombre de faux négatifs. Au contraire, si elles sont trop générales, cela peut entraîner une augmentation du nombre de faux positifs.

1.2 L'intelligence artificielle

L'intelligence artificielle (IA) est une branche de l'informatique qui vise à modéliser l'intelligence humaine pour résoudre des problèmes complexes nécessitant normalement des connaissances spécifiques et une capacité à raisonner ainsi qu'à faire des liens logiques. Depuis plusieurs années, l'IA est au centre de nombreux sujets de recherche et ne cesse de se développer dans des domaines comme la vision par ordinateur, la compréhension du langage et la robotique. Malgré les avancées majeures ces dernières années, jusqu'à l'arrivée des IA génératives comme ChatGPT et Midjourney, qui ont marqué un tournant dans les rapports entre humains et machines, nous sommes encore loin des scénarios véhiculés par les films et les livres de science-fiction d'une IA omnisciente et omnipotente. En effet, en dépit de l'aspect ésotérique de l'IA, ces technologies reposent principalement sur des règles définies par l'humain (IA symbolique) ou sur des algorithmes et modèles mathématiques (apprentissage automatique).

L'apprentissage automatique (machine learning, ML) est un domaine de l'intelli-

gence artificielle et de la science des données qui consiste à développer des programmes capables d'apprendre sur de larges jeux de données afin de répondre à des problématiques spécifiques. On parle d'apprentissage quand un programme n'est pas défini par des règles strictes et que sa capacité à prendre des décisions s'améliore avec le temps et l'expérience (def. 2).

Définition 2 *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E - T. Mitchell [Mit97]*

Le machine learning regroupe plusieurs sous-domaines qui sont l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement. Dans ce chapitre, nous présentons principalement les notions liées à l'apprentissage supervisé. Dans la partie II, nous définissons plus en détails l'apprentissage par renforcement.

1.2.1 Apprentissage Supervisé

L'apprentissage supervisé est une branche du machine learning où l'on cherche à modéliser le lien entre deux variables $x \in \mathcal{X}$ et $y \in \mathcal{Y}$, en supposant qu'il existe une fonction g inconnue telle que $g(x) = y$. Pour ce faire, on approche la fonction g par une autre fonction f. Cette approximation se fait en "entraînant" la fonction f à prédire la valeur de y telle que $f(x) \approx y$. La fonction f est appelée modèle, modèle prédictif ou prédicteur. L'apprentissage supervisé repose donc sur trois points essentiels qui sont les données, le modèle et l'entraînement.

Données

Un couple (x, y) représente une instance, où x est appelée variable d'entrée (ou observation, variable explicative) et y est appelée variable de sortie (ou variable cible, étiquette). En général, $\mathcal{X} = \mathbb{R}^d$ où d représente la dimension de l'observation x, ou le nombre de caractéristiques de x. Ainsi $x = [x_1, x_2, \ldots, x_d]$ où x_j représente la j-ième caractéristique de l'observation x. Pour l'ensemble $\mathcal{Y}$, on distingue plusieurs cas différents. Si l'on cherche à prédire une valeur continue, nous sommes dans le cas de la régression et $\mathcal{Y} = \mathbb{R}$. Dans le cas où l'on veut séparer les instances en deux groupes, on parle de classification binaire et $\mathcal{Y} = \{0, 1\}$. Enfin, s'il l'on cherche à séparer les instances en plus de deux groupes, on parle de classification multi-classe et $\mathcal{Y} = \{1, 2, \ldots, C\}$ où $C > 2$ est le nombre de groupes (ou classes) différents. Comme la détection de fichiers malveillants peut être considérée comme un problème de classification binaire, nous nous concentrons particulièrement sur ce cas dans la suite.

Pour estimer la fonction f, on dispose de deux ensembles contenant n valeurs, l'ensemble des observations $X = [x^{(1)}, x^{(2)}, ..., x^{(n)}] \in \mathbb{R}^{n \times d}$ et l'ensemble d'étiquettes $Y = [y^{(1)}, y^{(2)}, \ldots, y^{(n)}]$. Dans la suite, $X_i = x^{(i)}$ désigne la i-ième observation et $X_{ij} = x_j^{(i)}$ désigne la j-ième caractéristique de la i-ième observation. On note $S = \{X, Y\}$ l'ensemble de toutes les instances et, en principe, on sépare S en trois sous-ensembles S_{train}, S_{test} et S_{val} qui servent respectivement à entraîner, évaluer et régulariser le modèle f.

Modèle

En se plaçant dans le cas de la classification binaire, il est possible d'approcher g par une fonction $f \to \{0, 1\}$. En générale, on utilise une fonction intermédiaire $h : \mathcal{X} \to \mathbb{R}$ et on définit f selon une règle de décision :

$$f(x) = \begin{cases} 1 & \text{si } h(x) \geq \tau \\ 0 & \text{sinon,} \end{cases} \tag{1.1}$$

où $\tau \in \mathbb{R}$ est une valeur de seuil. On pourra aussi utiliser la règle de décision suivante :

$$f(x) = \begin{cases} 1 & \text{si } \sigma \circ h(x) \geq \tau \\ 0 & \text{sinon,} \end{cases} \tag{1.2}$$

avec $\tau \in]0, 1[$ et où σ est la fonction sigmoïde définie par :

$$\begin{aligned} \sigma : \quad & \mathbb{R} \to [0, 1] \\ & x \mapsto \frac{1}{1 + \mathrm{e}^{-x}} \end{aligned} \tag{1.3}$$

L'utilisation de la fonction sigmoïde permet de définir $p = \sigma \circ h(x)$. La valeur $p \in [0, 1]$ est appelée prédiction du modèle et est fortement utile dans le cas de la détection pour déterminer l'ampleur et la dangerosité de la menace. En effet, plus p est proche de 1, plus les chances qu'un fichier soit malveillant sont élevées et plus la menace est importante.

Le modèle f peut être une fonction paramétrée, noté f_θ, où $\theta = [\theta_1, \theta_2, \ldots, \theta_{n_\theta}]$ est vecteur de n_θ paramètres (ou coefficients, poids).

Entraînement, Évaluation et Régularisation

Un problème d'apprentissage est en fait un problème d'optimisation où nous cherchons la fonction f qui approche le mieux la fonction g. Pour ce faire, nous utilisons une fonction de perte (ou de coût) notée L. La valeur $L(f(x), y)$ correspond à l'erreur de prédiction du modèle f pour le couple (x, y) et nous pouvons généraliser à l'ensemble $S = (X, Y)$ de la façon suivante :

$$L(X, Y) = \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)})). \tag{1.4}$$

Si la valeur de L [1] est grande, le modèle f a une mauvaise capacité de prédiction, et inversement, si la valeur de L est petite alors f est un bon estimateur de g. Au cours de l'entraînement, nous cherchons donc la fonction f^* qui minimise L :

$$\begin{aligned} f^* &= \arg\min_f L(X, Y) \\ &= \arg\min_f \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)})). \end{aligned} \tag{1.5}$$

1. sous-entendu $L = L(X, Y)$

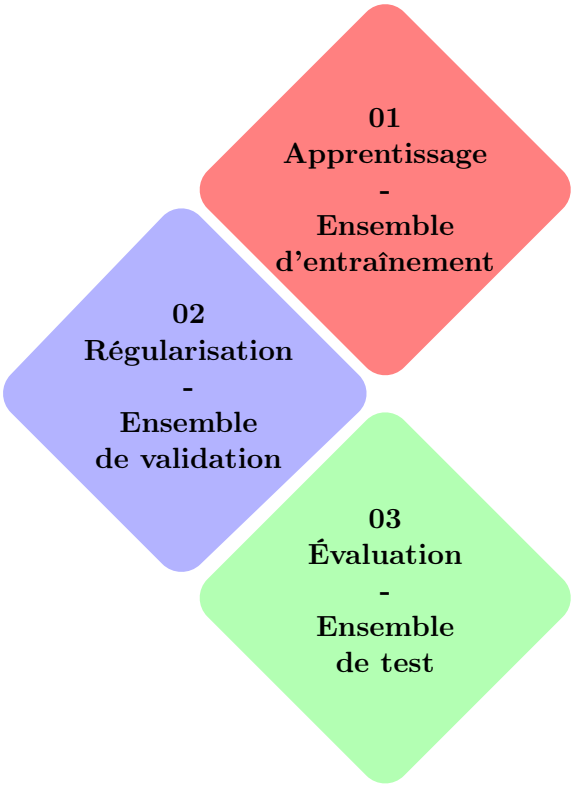


FIGURE 1.2 – Étapes de l'apprentissage

Si le modèle prédictif est une fonction paramétrée f_θ alors nous cherchons θ^* qui minimise L_θ :

$$\begin{aligned} \theta^* &= \arg\min_\theta L_\theta(X, Y) \\ &= \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f_\theta(x^{(i)})). \end{aligned} \tag{1.6}$$

Nous utiliserons généralement l'algorithme d'optimisation appelé descente de gradient stochastique (SGD) (alg. 1) qui consiste à mettre à jour θ de manière itérative jusqu'à que L_θ soit minimum, c'est-à-dire jusqu'à que θ converge, en appliquant la règle de mise à jour suivante :

$$\begin{aligned} \theta &= \theta - \eta \nabla L_\theta(X, Y) \\ &= \theta - \eta \nabla \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f_\theta(x^{(i)})), \end{aligned} \tag{1.7}$$

où η est un paramètre appelé taux d'apprentissage, et $\nabla L_\theta(X, Y)$ est le gradient de L_θ par rapport à θ, c'est-à-dire :

$$\nabla L_\theta(X, y) = \left[\frac{\partial L_\theta(X, Y)}{\partial \theta_i} \right]_{i \in \{1, \ldots, n_\theta\}}. \tag{1.8}$$

Au cours de l'entraînement, le modèle f peut souffrir de sur-apprentissage (overfitting). Ce phénomène arrive quand le modèle apprend trop bien et a de très bonnes

Algorithme 1 Algorithme de la descente de gradient stochastique

Entrée :
L_θ une fonction de perte différentiable
Initialisation :
$\theta \in \mathbb{R}^?$ (p. ex. 0 ou aléatoire)
$\delta \leftarrow \infty$
Paramètre :
$\eta \in]0, 1]$, le taux d'apprentissage
ϵ petit
Boucle : tant que $\delta \geq \epsilon$:
$L_{-1} \leftarrow L_\theta$
$\theta \leftarrow \theta - \eta \nabla L_\theta(X, Y)$
$\delta \leftarrow |L_\theta - L_{-1}|$

capacités de prédiction sur le jeu de données d'apprentissage S_{train}, mais ses performances sont médiocres sur de nouvelles données.

Pour éviter ce phénomène, nous utilisons un ensemble de données de validation S_{val} qui permet, au cours de l'entraînement, de comparer les performances de différents modèles pour minimiser l'erreur de prédiction sur S_{val}. Nous cherchons donc θ^* tel que :

$$\theta^* = \arg\min_\theta \frac{1}{n_{val}} \sum_{(x,y)\in S_{val}} L(y, f_\theta(x)). \tag{1.9}$$

Cette ensemble de validation peut aussi être utilisé pour ajuster les hyperparamètres d'un modèle. En résumé, l'ensemble de validation fournit une évaluation indépendante des données utilisées au cours du processus d'apprentissage.

Enfin, l'ensemble de test (ou d'évaluation) permet d'étudier les performances. L'efficacité d'un modèle se mesure par sa capacité à correctement prédire la nature d'une instance. Nous pouvons tout d'abord étudier le nombre d'instances correctement classifiées à l'aide d'une matrice de confusion (figure 1.3). Pour quantifier la performance d'un modèle, il est courant d'utiliser des mesures de performance comme l'accuracy [2] (ACC) qui permet d'évaluer le nombre de prédiction correcte du modèle, définie par :

$$ACC = \frac{VP + VN}{VP + FP + VN + FN}. \tag{1.10}$$

Une autre mesure couramment utilisée est le score F1, en particulier dans le cas d'un jeu de données déséquilibrées. Le score F1 est la moyenne harmonique de la précision et du rappelle, deux mesures employées pour identifier la part de faux négatif et la part de faux positif dans les prédictions. Le score F1 est définie par :

$$F1 = 2 \cdot \frac{\text{Précision} \cdot \text{Rappel}}{\text{Précision} + \text{Rappel}} \tag{1.11}$$

avec :

$$\text{Précision} = \frac{TP}{TP + FP} \quad \text{et} \quad \text{Rappel} = \frac{TP}{TP + FN}. \tag{1.12}$$

2. On pourrait traduire le terme accuracy par "précision" mais il existe une autre mesure qui porte ce nom

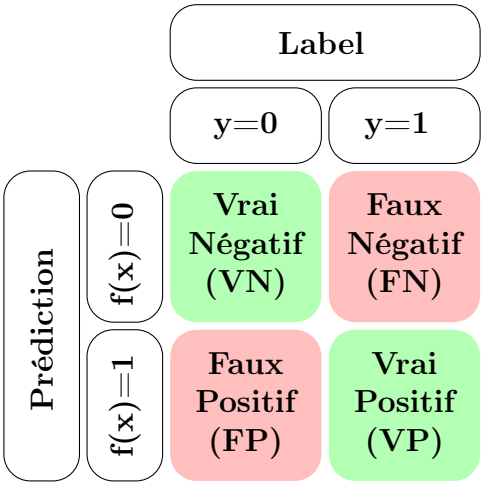


FIGURE 1.3 – Matrice de confusion

Enfin, il est courant de vouloir minimiser le nombre de fausses prédictions donc de s'attarder sur le Taux de Faux Positifs (TFP) et le Taux de Faux Négatifs (TFN) :

$$TFN = \frac{FN}{FN + TP} \quad \text{et} \quad TFP = \frac{FP}{FP + TN}. \tag{1.13}$$

1.2.2 Apprentissage Profond

L'apprentissage profond (deep learning, DL) est un sous-domaine de l'apprentissage automatique qui repose principalement sur les réseaux de neurones.

Neurones

Un neurone est simplement une fonction qui prend un certain nombre de valeurs en entrée et renvoie une nouvelle valeur. La figure 1.4 représente l'architecture d'un neurone, et nous pouvons définir mathématiquement un neurone de la manière suivante :

$$x' = \phi(\sum_{i=1}^{n} w_i x_i + b) \tag{1.14}$$

où $\{x_i\}_{i\in\{1,\dots,n\}}$ sont les variables d'entrée du neurone, provenant de données brutes où d'autres neurones, x' est la sortie du neurone. Les poids $\{w_i\}_{i\in\{1,\dots,n\}}$ déterminent la contribution de chaque variable à l'activité du neurone et le biais b permet d'ajuster l'activité du neurone. Enfin, la fonction ϕ est appelée fonction d'activation et permet d'ajouter de la non-linéarité dans le modèle.

Réseau de neurones

Un réseau de neurones artificiel (ANN) est un ensemble de neurones organisés en différentes couches (figure 1.5). La couche d'entrée transmet les observations au réseau, la couche de sortie renvoie une ou plusieurs informations à l'utilisateur selon la tâche attribuée au réseau (classification, régression...). Entre ces deux couches, il peut y avoir un certain nombre de couches dites cachées. Le nombre de couches cachées détermine

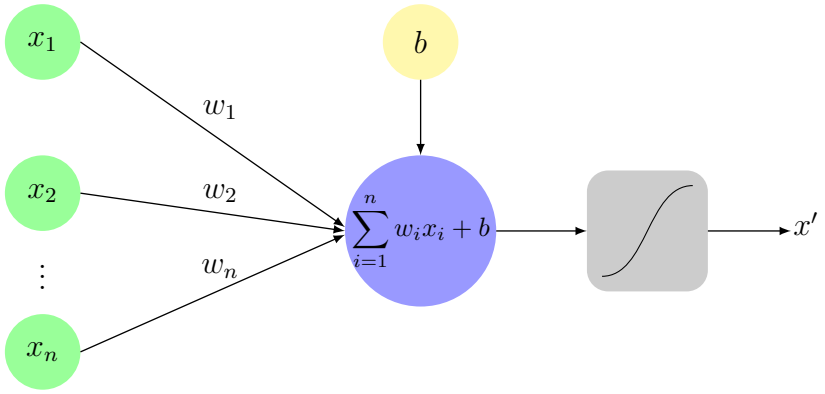


FIGURE 1.4 – Neurone

la profondeur d'un réseau, mais aussi sa capacité à modéliser les relations entre les observations et les prédictions.

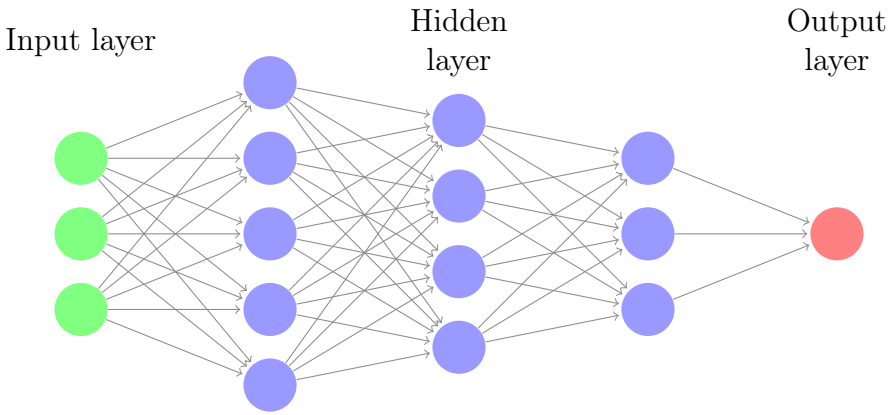


FIGURE 1.5 – Réseau de neurones de profondeur 3

Comme dans le cas de l'apprentissage supervisé non-profond, nous cherchons à modéliser la relation entre deux variables x et y. Dans le cas de l'apprentissage profond, un réseau de neurones est donc une fonction paramétrée f_θ où θ est l'ensemble des poids w et des biais b du modèle.

Plus généralement, pour tout $x \in \mathcal{X}$, $f_\theta(x) = \phi \circ h_\theta(x)$ où $h_\theta(x)$ est la sortie du réseaux et ϕ est une fonction d'activation qui prendra plusieurs valeurs selon l'ensemble $\mathcal{Y}$. Pour le cas de la classification binaire ($\mathcal{Y} = \{0, 1\}$), auquel on s'intéresse particulièrement dans la suite, la fonction ϕ est la fonction sigmoïde définie précédemment (équation (1.3)).

Différentes couches

Un réseau de neurones est donc composé d'une couche d'entrée, de plusieurs couches cachées et d'une couche de sortie. Les couches cachées permettent au réseau de modéliser des relations entre les variables d'entrée et d'identifier certains motifs dans les données. Plus le réseau est profond, plus les relations et les motifs seront nombreux et précis, mais plus le modèle sera complexe et long à entraîner.

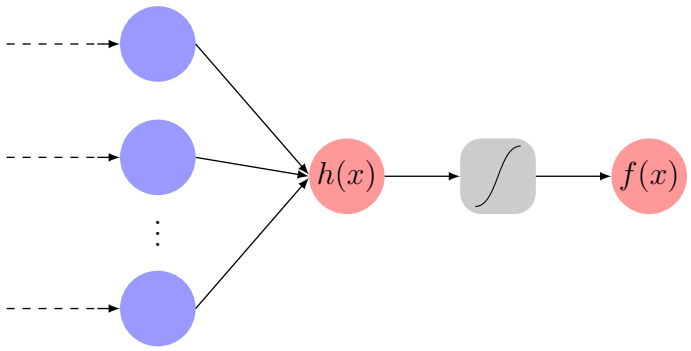
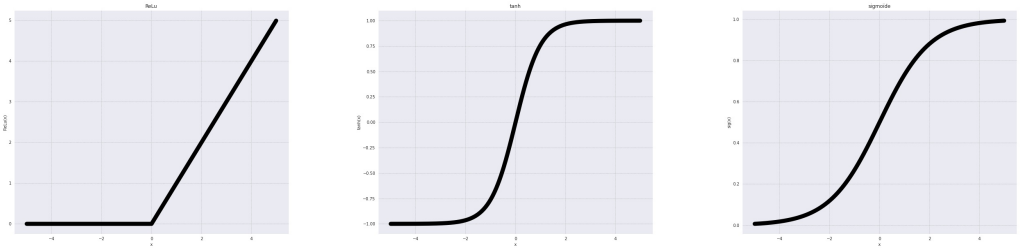


FIGURE 1.6 – Dernières couches d'un réseau de neurones avec la fonction d'activation sigmoïde.

Parmi les couches les plus utilisées, on retrouve la couche **dense** (ou entièrement connectée). Elle est composée de plusieurs neurones qui prennent en entrée tous les neurones de la couche précédente. Cette couche est souvent suivie d'une couche d'**activation** qui permet d'introduire de la non-linéarité dans le modèle avec des fonctions comme ReLu (fig. 1.7a), tanh (fig. 1.7b) ou sigmoïde (fig. 1.7c).



(a) ReLu (b) tanh (c) sigmoïde

FIGURE 1.7 – Différentes fonctions d'activation

Pour éviter le sur-apprentissage, et pour améliorer la convergence du modèle, il est possible d'utiliser des couches de **régularisation** comme la couche **dropout** [Sri+14] qui, pendant l'entraînement, désactive un certain nombre de neurones de la couche concernée. Une autre couche de régularisation est la couche de **batch normalization** (ou batchnorm) [IS15] qui permet de normaliser les sorties d'une couche dense avant d'y appliquer la fonction d'activation. La figure 1.8 présente un exemple d'architecture possible au sein d'un réseau de neurones.

Pour certaines tâches spécifiques, il est possible d'utiliser des architectures de réseaux de neurones utilisant des couches particulières. C'est le cas de la reconnaissance d'images qui utilise des réseaux de neurones dits **convolutifs** (CNN) [Lec+98 ; KSH12]. L'architecture d'un CNN repose principalement sur la couche de **convolution**. Cette couche est composée de différents filtres qui permettent d'extraire des caractéristiques d'une image comme les bords ou les angles. Chaque couche renvoie des cartes d'activation qui dépendent des filtres utilisés. En augmentant le nombre de couches de convolutions, il est possible pour un modèle de combiner les informations extraites pour identifier des formes plus complexes ou des textures. Un autre type de couche utilisée, conjointement avec la convolution, est la couche de **pooling**. Cette couche est

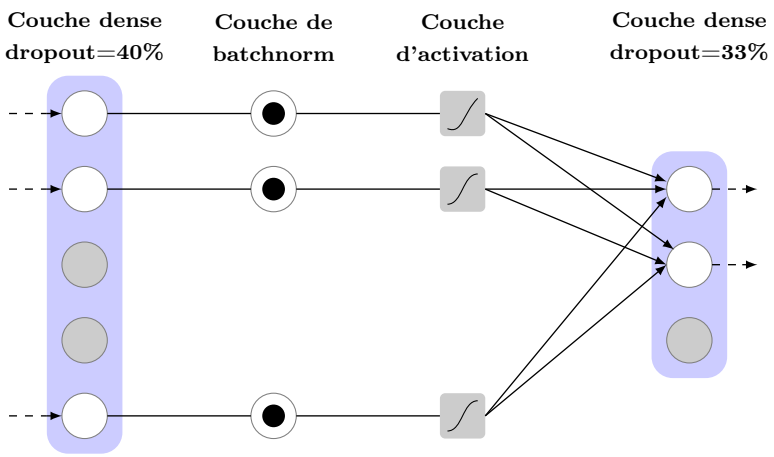


FIGURE 1.8 – Exemple d'architecture au sein d'un réseau de neurones

généralement située à la suite d'une couche de convolution et permet de réduire la taille des cartes d'activation, tout en conservant l'information extraite de l'image.

Généralement, un CNN est composé de plusieurs couches de convolution et de pooling afin d'extraire de l'information de l'image, puis de couches denses pour générer une prédiction comme le montre la figure 1.9.

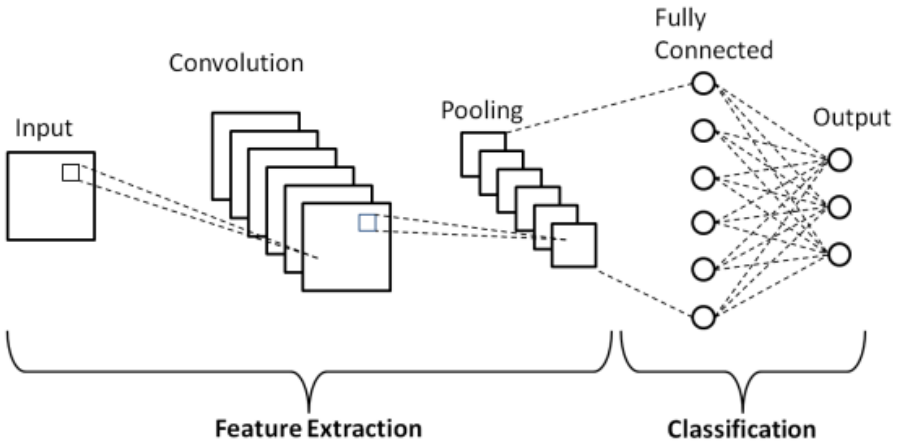


FIGURE 1.9 – Exemple de réseau de neurones convolutif [PR18]

1.3 L'IA pour la détection de fichiers malveillants

Dans cette section, nous présentons le processus pour créer un outil d'analyse de fichiers malveillants à l'aide de modèles d'apprentissage supervisé. La figure 1.10 montre les différentes étapes nécessaires à la construction d'un outil de détection de malware. Cette démarche est valable pour d'autres problématiques comme la classification de différentes familles (trojan, ransomware ...) ou bien la détection d'un type de fichier en particulier.

La première étape consiste à collecter des données. Dans notre cas, il s'agit de fichiers PE bénins et malveillants. Il existe plusieurs bases de données mises à disposition

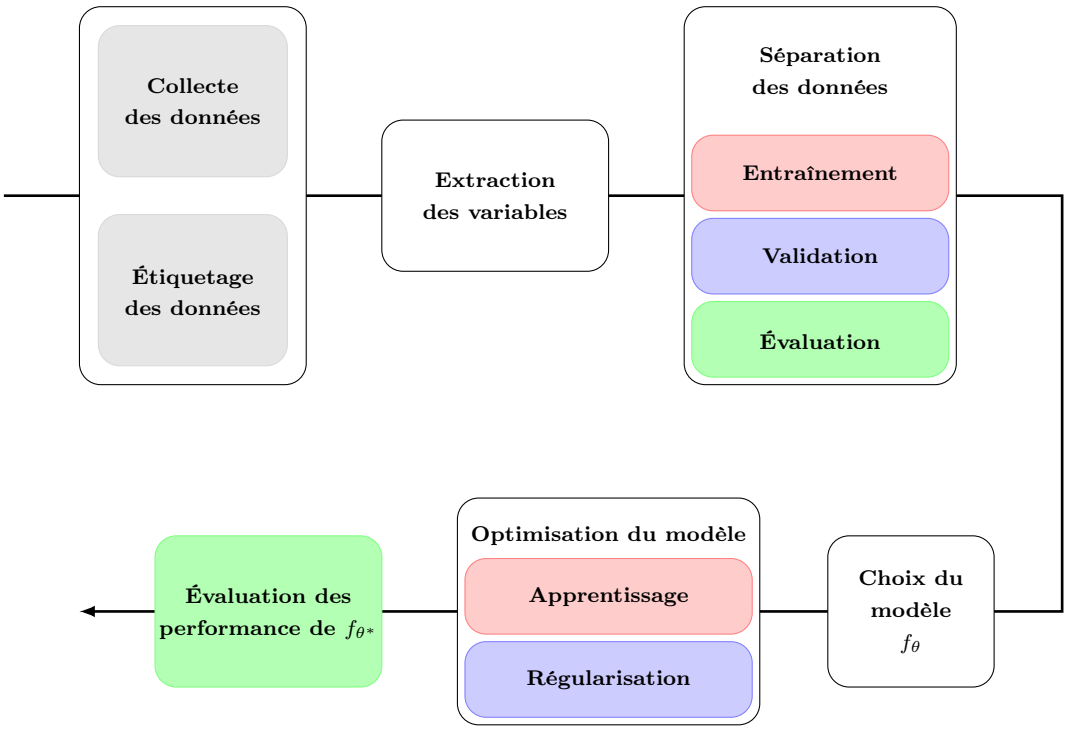


FIGURE 1.10 – Étapes principales de la construction d'un modèle de détection de fichier malveillants

par des entreprises ou des chercheurs pour améliorer la recherche sur des sujets liés à l'analyse de malware. Par exemple, Microsoft lance en 2015 le challenge BIG 2015 via le site Kaggle [Pan+15]. Ils mettent à disposition une base de données constituée de fichiers malveillants et le but est de les classifier selon leurs familles. Dans d'autres bases de données comme EMBER [AR18] ou BODMAS [Yan+21], les fichiers sont directement transformés sous forme de vecteurs, et les auteurs ne fournissent pas nécessairement les fichiers PE originaux, souvent pour des raisons de licences et de droits d'auteur. Il existe des sites comme VirusTotal ou MalwareBaazar qui permettent de récupérer des fichiers malveillants, mais la provenance et la nature de ces données sont souvent incertaines. Cela reste compliqué de créer une base de fichiers bénins pour les raisons de copyright. Pour que la base de données soit viable, il faut s'assurer que les fichiers correspondent au cas d'usage prévu pour l'outil de détection. Ils doivent être suffisamment récents pour représenter l'état du marché au moment de l'entraînement du modèle et géographiquement adaptée au pays où sera déployé l'outil. En effet, les fichiers malveillants en circulation ne sont pas nécessairement les mêmes dans chaque région du globe [SH17 ; MCC15]. Enfin, pour des problématiques spécifiques comme la classification, il sera peut-être nécessaire d'analyser les fichiers en amont pour pouvoir les étiqueter selon leur nature, famille ou catégorie. Les étapes de collecte et de labellisation sont sûrement les plus importantes puisque l'efficacité du modèle dépendra fortement de la quantité de données, ainsi que de la justesse de l'étiquetage.

Dans le cas de l'analyse de fichiers malveillants, les données brutes, c'est-à-dire les fichiers PE, n'ont pas un format adapté pour être utilisées avec des modèles d'apprentissage supervisé. Pour régler ce problème, il faut souvent extraire de l'information des fichiers pour se ramener à une forme plus usuelle. Comme nous l'avons vu dans la section

1.1.2, les deux techniques principalement utilisées sont l'analyse statique, qui consiste à examiner le contenu et le code du fichier sans l'exécuter, et l'analyse dynamique qui permet d'observer le comportement du fichier. Avec ces deux méthodes, il est possible de créer un vecteur de valeurs qui décrit l'instance comme la distribution des octets, l'entropie du fichier et le contenu de la table des imports (statique) ou bien les accès aux registres, à certains fichiers et les communications réseaux (dynamique). D'autres transformations sont possibles comme par exemple, RAFF et al. [Raf+17] qui utilisent directement la séquence d'octets brute qui constitue le fichier pour entraîner un modèle, ou NATARAJ et al. [Nat+11] qui transforme le fichier en image.

Il est ensuite conseillé d'entraîner plusieurs modèles à partir de différents algorithmes d'apprentissage (forêt aléatoire, K-NN, réseaux de neurones...) pour ensuite comparer les performances des modèles entre eux et choisir le plus efficace. Si les différents modèles n'ont pas de bons résultats suite à la phase d'évaluation, le problème peut venir de la base de données (trop peu d'instances, une mauvaise labellisation des données...), du choix des variables qui ne sont peut-être pas représentatives de la problématique, ou de l'algorithme d'apprentissage qui n'est pas adapté aux données.

Bibliographie

[ANS23] ANSSI. *Panorama de la cybermenace 2022*. https://www.cert.ssi.gouv.fr/cti/CERTFR-2023-CTI-001/. 2023.

[AR18] Hyrum S. Anderson et Phil Roth. *EMBER : An Open Dataset for Training Static PE Malware Machine Learning Models*. 2018. arXiv : 1804.04637 [cs.CR].

[Asl+23] Ömer Aslan et al. « A Comprehensive Review of Cyber Security Vulnerabilities, Threats, Attacks, and Solutions ». In : *Electronics* 12.6 (mars 2023), p. 1333. doi : 10.3390/electronics12061333.

[Baz+13] Zahra Bazrafshan et al. « A survey on heuristic malware detection techniques ». In : *The 5th Conference on Information and Knowledge Technology*. 2013, p. 113-120. doi : 10.1109/IKT.2013.6620049.

[BKM05] Guillaume Bonfante, Matthieu Kaczmarek et Jean-Yves Marion. « Toward an abstract computer virology ». In : *Second International Colloquium on Theoretical Aspects of Computing - ICTAC 2005*. Sous la dir. de Dang Van Hung et Martin Wirsing. T. 3722. Lecture Notes in Computer Science. Hanoï/Vietnam : Springer, oct. 2005, p. 579-593. doi : 10.1007/11560647_38.

[BM08] Jean-Marie Borello et Ludovic Mé. « Code obfuscation techniques for metamorphic viruses ». In : *Journal in Computer Virology* 4.3 (fév. 2008), p. 211-220. doi : 10.1007/s11416-008-0084-2.

[Bur22] Florian Burnel. *L'agence de santé publique du Costa Rica victime du ransomware Hive*. https://www.it-connect.fr/lagence-de-sante-publique-du-costa-rica-victime-du-ransomware-hive/. En ligne, publié le 01/06/2022. 2022.

[Cro23] Crowdstrike. *2023 Globale Threat Report*. 2023.

[Dre] Todd Drew. *Costa Rica Declares State of Emergency After Conti Ransomware Attack*. https://www.secureworld.io/industry-news/costa-rica-emergency-ransomware. En ligne, publié le 12/05/2022.

[GMP20] Daniel Gibert, Carles Mateu et Jordi Planes. « The rise of machine learning for detection and classification of malware : Research developments, trends and challenges ». In : *Journal of Network and Computer Applications* 153.January (2020), p. 102526. issn : 10958592. doi : 10.1016/j.jnca.2019.102526.

[GSS15] Ian J. Goodfellow, Jonathon Shlens et Christian Szegedy. « Explaining and harnessing adversarial examples ». In : *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), p. 1-11. arXiv : 1412.6572.

[Hun23] Shane Huntley. *Fog of war : how the Ukraine conflict transformed the cyber threat landscape*. https://blog.google/threat-analysis-group/. 2023.

[IS15] Sergey Ioffe et Christian Szegedy. « Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift ». In : *Proceedings of the 32nd International Conference on Machine Learning*. Sous la dir. de Francis Bach et David Blei. T. 37. Proceedings of Machine Learning Research. Lille, France : PMLR, juill. 2015, p. 448-456.

[KSH12] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton. « ImageNet Classification with Deep Convolutional Neural Networks ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de F. Pereira et al. T. 25. Curran Associates, Inc., 2012. url : https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[Lal+21] Harjinder Singh Lallie et al. « Cyber security in the age of COVID-19 : A timeline and analysis of cybercrime and cyber-attacks during the pandemic ». In : *Computers & Security* 105 (2021), p. 102248. issn : 0167-4048. doi : https://doi.org/10.1016/j.cose.2021.102248.

[Lec+98] Y. Lecun et al. « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324. doi : 10.1109/5.726791.

[MCC15] Ghita Mezzour, Kathleen M. Carley et L. Richard Carley. « An Empirical Study of Global Malware Encounters ». In : *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*. HotSoS '15. Urbana, Illinois : Association for Computing Machinery, 2015. isbn : 9781450333764. doi : 10.1145/2746194.2746202.

[Mic] Microsoft. *PE Format*. https://learn.microsoft.com/en-us/windows/win32/debug/pe-format. En ligne, consulté le 03/02/2021.

[Mit97] Tom M Mitchell. *Machine learning*. McGraw-Hill Science/Engineering/Math, 1997. isbn : 0070428077.

[MKK07] Andreas Moser, Christopher Kruegel et Engin Kirda. « Limits of Static Analysis for Malware Detection ». In : *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE, déc. 2007. doi : 10.1109/acsac.2007.21.

[MM00] Gary McGraw et Greg Morrisett. « Attacking Malicious Code : A Report to the Infosec Research Council ». In : *IEEE Software* 17.5 (sept. 2000), p. 33-41. doi : 10.1109/52.877857.

[Nat+11] L Nataraj et al. « Malware images : Visualization and automatic classification ». In : *ACM International Conference Proceeding Series*. 2011. isbn : 9781450306799. doi : 10.1145/2016904.2016908.

[Or+19] Ori Or-Meir et al. « Dynamic Malware Analysis in the Modern Era—A State of the Art Survey ». In : *ACM Computing Surveys* 52.5 (sept. 2019), p. 1-48. doi : 10.1145/3329786.

[Pan+15] Alessandro Panconesi et al. *Microsoft Malware Classification Challenge (BIG 2015)*. 2015. url : https://kaggle.com/competitions/malware-classification.

[PR18] Van Hiep Phung et Eun Joo Rhee. « A Deep Learning Approach for Classification of Cloud Image Patches on Small Datasets ». In : *Journal of information and communication convergence engineering* 3.3 (sept. 2018). doi : 10.6109/jicce.2018.16.3.173. url : http://dx.doi.org/10.6109/jicce.2018.16.3.173.

[Raf+17] Edward Raff et al. « Malware detection by eating a whole EXE ». In : *arXiv* (2017). doi : 10.13016/m2rt7w-bkok. arXiv : 1710.09435.

[SH12] Michael Sikorski et Andrew Honig. *Practical Malware Analysis : The Hands-On Guide to Dissecting Malicious Software*. 1st. USA : No Starch Press, 2012. isbn : 1593272901.

[SH17] Olga Smirnova et Thomas J. Holt. « Examining the Geographic Distribution of Victim Nations in Stolen Data Markets ». In : *American Behavioral Scientist* 61.11 (2017), p. 1403-1426. doi : 10.1177/0002764217734270.

[Smi+20] Michael R. Smith et al. « Mind the Gap : On Bridging the Semantic Gap between Machine Learning and Malware Analysis ». In : *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*. New York, NY, USA : Association for Computing Machinery, 2020, p. 49-60. isbn : 9781450380942.

[SOA18] Rami Sihwail, Khairuddin Omar et Khairul Akram Zainol Ariffin. « A Survey on Malware Analysis Techniques : Static, Dynamic, Hybrid and Memory Analysis ». In : *International Journal on Advanced Science, Engineering and Information Technology* 8.4-2 (sept. 2018), p. 1662. doi : 10.18517/ijaseit.8.4-2.6827.

[Sop2323] Sophos X-Ops. *Sophos 2023 Threat Report - Maturing criminal marketplaces present new challgenges to defenders*. https://www.sophos.com/en-us/content/security-threat-report. 2023.

[Sri+14] Nitish Srivastava et al. « Dropout : A Simple Way to Prevent Neural Networks from Overfitting ». In : *Journal of Machine Learning Research* 15.56 (2014), p. 1929-1958. url : http://jmlr.org/papers/v15/srivastava14a.html.

[Tay+22] Umm-e-Hani Tayyab et al. « A Survey of the Recent Trends in Deep Learning Based Malware Detection ». In : *Journal of Cybersecurity and Privacy* 2.4 (2022), p. 800-829. issn : 2624-800X. doi : 10.3390/jcp2040041.

[UAB19] Daniele Ucci, Leonardo Aniello et Roberto Baldoni. « Survey of machine learning techniques for malware analysis ». In : *Computers and Security* 81 (2019), p. 123-147. issn : 01674048. doi : 10.1016/j.cose.2018.11.001. arXiv : 1710.08189.

[Yan+21] Limin Yang et al. « BODMAS : An Open Dataset for Learning based Temporal Analysis of PE Malware ». In : *Proceedings - 2021 IEEE Symposium on Security and Privacy Workshops, SPW 2021*. 2021, p. 78-84. isbn : 9781728189345. doi : 10.1109/SPW53761.2021.00020.

[Žli10] Indrė Žliobaitė. *Learning under Concept Drift : an Overview*. 2010. arXiv : 1010.4784 [cs.AI].

Première partie

Analyse et Détection de Logiciels Malveillants

Chapitre 2

Introduction

Cette partie regroupe des travaux réalisés dans le cadre de l'analyse et de la détection de logiciels malveillants au format Portable Executable. Dans ce chapitre, nous décrivons les problématiques liées à la menace des fichiers malveillants, et nous faisons un panorama des solutions de détection basées sur l'intelligence artificielle.

2.1 Contexte et problématique

Au cours des dernières années, la sécurité informatique est devenue un domaine à part entière au vu des menaces de plus en plus nombreuses. Bien que les technologies de cybersécurité soient de plus en plus développées, le nombre d'attaques ne diminue pas et de plus en plus de structures privées comme publiques sont prises pour cible. Parmi les vecteurs d'attaques, l'utilisation de logiciels malveillants reste prépondérante, avec des techniques de plus en plus développées pour tromper les solutions de sécurité et les antivirus, et infiltrer les systèmes d'informations.

Le marché des logiciels malveillants a pris un tout autre tournant au cours des dernières années avec la création, par des groupes de hackers et des éditeurs de malware, d'une nouvelle activité, appelée Malware-as-a-Service (MaaS), qui consiste à vendre des fichiers malveillants prêt à l'emploi, et qui ne nécessitent aucune connaissance préalable. Cette évolution du paysage des menaces vient amplifier les craintes des acteurs de la cybersécurité, dans un contexte déjà tendu avec la multiplication des attaques par ransomware qui ont été sources de préjudices à certains niveaux étatiques, comme pour le Costa-Rica, ou en compromettant des établissement d'utilité publique comme des hôpitaux ou autres établissements de santé.

Les solutions de détection classiques se retrouvent dépassées par le nombre de nouveaux fichiers malveillants mis en ligne chaque année et par les technologies utilisées par les éditeurs de malware. C'est pourquoi certains chercheurs en cybersécurité pensent que l'intelligence artificielle pourrait offrir un nouveau paradigme pour améliorer ces solutions de détection classiques. L'IA se distingue par sa capacité à analyser de larges volumes de données et à identifier des schémas malveillants au-delà des capacités d'analyses humaines.

L'utilisation de l'IA et d'algorithmes d'apprentissage supervisé et profond a permis de concevoir de nouvelles approches, telles que l'analyse d'un fichier interprété comme une séquence de caractères, ou comme une image. Ces approches révolutionnaires, com-

binées aux méthodes de détection classiques, pourraient permettre, à l'avenir, d'améliorer les solutions de sécurité en anticipant les évolutions des logiciels malveillants.

2.2 État de l'art

L'intelligence artificielle est devenue une composante majeure dans de nombreux domaines, dont la cybersécurité, où cette technologie peut contribuer à réduire les risques et les coûts engendrés par des failles de sécurité [Cap+22 ; Cha+19]. Entre autres, l'IA montre un potentiel prometteur dans le cadre de l'analyse et de la détection de fichiers malveillants, où elle peut aider à améliorer les modèles de détection classiques basés sur l'analyse par signature, l'analyse comportementale ou des méthodes heuristiques [Ach+21 ; Fir+10 ; Cho+23].

Pour créer des outils de détection de fichiers malveillants reposant sur l'IA, il est souvent nécessaire d'extraire des variables à partir des fichiers binaires. Généralement, les méthodes d'extraction statiques sont préférées aux méthodes dynamiques en raison de leur rapidité et de leur faible coût en ressources. La performance d'un modèle dépend souvent de la qualité du prétraitement utilisé et des données collectées. À cet égard, Anderson et Roth [AR18] ont mis en ligne une base de données contenant un million d'instances de fichiers malveillants transformés en vecteurs à l'aide de plusieurs méthodes de pré-traitement. D'autre part, [Raf+17] utilise directement les fichiers bruts, sous forme de séquences d'octets, et crée Malconv, un modèle de détection basé sur des techniques de traitement du langage naturel (NLP). Néanmoins, il a été montré que MalConv est sensible à certaines méthodes d'obfuscation comme le padding, ainsi qu'aux exemples adverses [Kre+18]. Pour compenser ces faiblesses, Fleshman et al. [Fle+18] ont développé Non-Negative MalConv qui améliore la robustesse du modèle original, mais réduit ses performances de détection.

En 2011, Nataraj et al. [Nat+11] proposent une méthode de pré-traitement avec laquelle un fichier binaire est converti en image en nuances de gris. Dans leur article, les auteurs utilisent l'algorithme GIST pour extraire les caractéristiques importantes de l'image. Á partir de ces caractéristiques, ils entraînent un K-NN dont le but est de classifier les fichiers malveillants en fonction de leur famille. Cet outil atteint un taux de succès de $97,25\%$, et grâce à la transformation en image, il est plus résistant aux techniques d'obfuscation telles que le chiffrement ou la compression. Par la suite, d'autres méthodes de pré-traitement ont été proposées, notamment en générant des images basées sur l'entropie du fichier [Han+14] ou sur les informations sémantiques contenus dans ses différentes sections [Vu+20]. Néanmoins, ces dernières méthodes requièrent davantage de temps pour convertir un fichier binaire en image en raison de la complexité de la transformation.

Avec l'essor des réseaux de neurones convolutifs, suite à la victoire de AlexNet à la compétition Imagenet de 2012 [KSH12 ; Den+09 ; Alo+18], le sujet de la transformation d'un fichier binaire en image devient une solution potentielle pour créer la nouvelle génération de solutions de détection. Rezende et al. [Rez+17] utilise l'apprentissage par transfert (transfer learning) pour entraîner un outil de classification de malware à partir de ResNet-50 [He+15], un réseau de neurones résiduel composé de 50 couches. Cet outil atteint 98.62% de réussite. Pour aller plus loin, Yakura et al. [Yak+18] utilisent des mécanismes d'attention pour identifier les zones d'intérêts de l'image et

pour aider à la compréhension de la réponse du modèle.

Même si l'utilisation d'algorithmes d'apprentissage supervisé et profond montre de bon résultats pour la détection ou la classification de fichiers malveillants, ces technologies restent confrontées à quelques limites comme l'obfuscation avec des méthodes sophistiquées de chiffrement et de compressions [AF22]. De plus, les experts en cybersécurité et les chercheurs en intelligence artificielle ont souvent des perspectives différentes et abordent ce genre de problématique de manière distincte, ce qui crée une séparation entre ces deux communautés [Smi+20].

2.3 Contributions et plan

Contributions

Dans le cadre de nos recherches, nous avons créé plusieurs outils qui permettent d'analyser un fichier PE afin d'extraire des informations utiles à la compréhension de sa nature et de son comportement. Nous avons ensuite développé plusieurs modèles de détection de fichiers malveillants à partir de modèles d'apprentissage supervisé et profond. Nous avons eu une attention toute particulière à la détection de logiciels de type ransomware au vu du contexte actuel marqué par l'utilisation massive de ces fichiers. Nous proposons, en autre, un modèle multi-tâches composé de plusieurs sous-modèles entraînés sur des tâches spécifiques (détection de malware, détection de ransomware). Ce modèle multi-couches montre de très bon résultats dans ces différentes tâches tout en permettant une certaine adaptation et une certaine flexibilité face à de nouvelles problématiques. Les recherches abordées dans cette partie sont principalement issues de deux articles présentés lors de conférences et diffusés en ligne :

- Benjamin Marais, Tony Quertier et Christophe Chesneau. « Malware Analysis with Artificial Intelligence and a Particular Attention on Results Interpretability ». In : *Distributed Computing and Artificial Intelligence, Volume 1 : 18th International Conference*. Sous la dir. de Kenji Matsui et al. Cham : Springer International Publishing, 2022, p. 43-55. isbn : 978-3-030-86261-9

- Benjamin Marais, Tony Quertier et Stéphane Morucci. « AI-based Malware and Ransomware Detection Models ». In : *Conference on Artificial Intelligence for Defense*. Actes de la 4ème Conference on Artificial Intelligence for Defense (CAID 2022). DGA Maîtrise de l'Information. Rennes, France, nov. 2022. url : `https://hal.science/hal-03881198`

Plan

Tout d'abord nous présentons les outils d'analyse développés au cours de nos travaux dans le chapitre 3, nous présentons ensuite le cadre des expérimentations avec les données et les algorithmes utilisés dans le chapitre 4. Dans le chapitre 5, nous parlons des différents modèles entraînés au cours de nos expérimentations et de leurs performances, puis nous présentons un outil d'analyse avancée dans le chapitre 6. Enfin, nous discutons de nos travaux et des possibles axes de recherches futurs dans le chapitre 7.

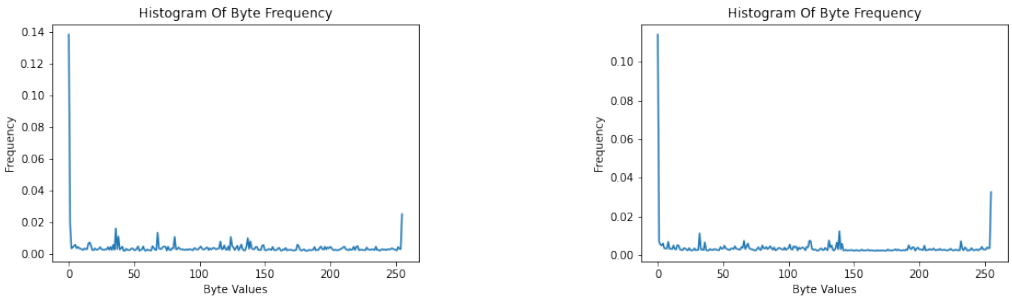
Chapitre 3

Outils d'Analyse de Fichiers Binaires

Au cours de nos travaux, nous avons développé trois outils qui permettent d'effectuer une analyse préliminaire de fichier binaire. Ils peuvent être utilisés en amont d'une analyse approfondie pour estimer la nature d'un fichier à l'aide de méthodes statiques. L'avantage de ces outils est qu'ils sont particulièrement rapides, et qu'ils retournent des résultats simples à interpréter même pour un analyste en malware débutant.

3.1 Histogramme de la fréquence des octets

L'histogramme de la fréquence des octets permet de visualiser la distribution des valeurs d'octets (comprises entre 0 et 255) dans un fichier binaire. Cette distribution permet de faire de premières hypothèses sur la nature du fichier étudié, s'il est malveillant ou bénin. Pour un fichier bénin, l'histogramme présentera une distribution uniforme, avec quelques valeurs qui se démarque comme 0 ou 255. Si le fichier a subi du padding [1], l'histogramme présentera un pic plus important pour la valeur de padding utilisée, ou aura une forme plus chaotique s'il s'agit de padding aléatoire.



(a) Fichier bénin (b) Fichier malveillants

FIGURE 3.1 – Histogrammes de différents fichiers binaires

1. une méthode d'obfuscation qui consiste à augmenter artificiellement la taille d'un fichier en ajoutant de l'information (généralement des séquences d'octets de valeurs 0)

3.2 Transformation de fichiers binaires en image

Comme nous l'avons évoqué précédemment, en considérant un fichier comme une séquence d'octets (8 bits), dont chaque élément à une valeur comprise entre 0 et 255, il est possible de redimensionner cette séquence sous forme de matrice. En interprétant chaque valeur de cette matrice comme un pixel, il est possible de représenter le fichier sous forme d'image en niveau de gris. D'un point de vue mathématique, soit $u \in U$, un fichier binaire au format PE. Nous définissons $b \in \{0, 255\}^m$ comme la représentation du fichier u sous forme de vecteurs, où m est le nombre d'octets de u. Soit $A \in \{0, 255\}^{h \times v}$, la représentation de u sous forme de matrice définie par :

$$A_{i \times j} = b_{(i-1)w+j} \ \forall (i, j) \in \{1, ..., h\} \times \{1, ..., w\}$$

où $w = \lfloor m^{\frac{1}{2}} \rfloor$, $h = \lfloor \frac{m}{w} \rfloor$ où $\lfloor \cdot \rfloor$ est la fonction partie entière définie par $\lfloor x \rfloor = max\{n \in \mathbb{Z} | n \leq x\}$. La figure 3.2 représente la transformation d'un fichier bénin et d'un fichier malveillant sous forme d'images.

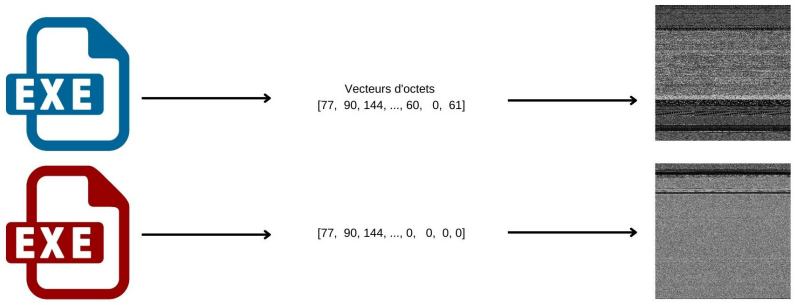


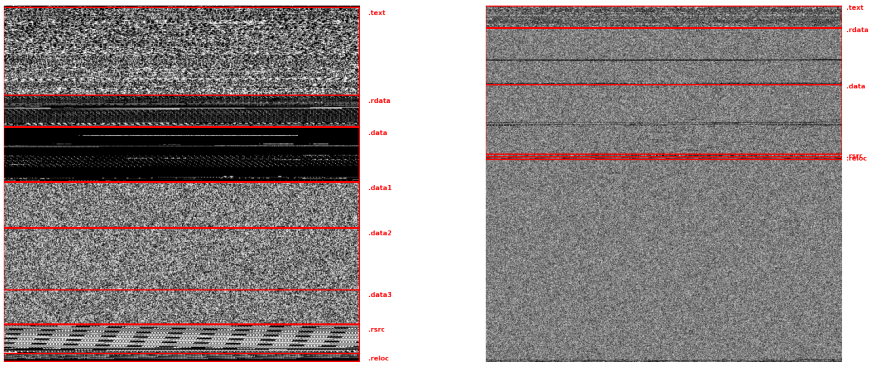
FIGURE 3.2 – Transformation d'un fichier binaire en image avec un fichier bénin (en haut) et un malware de type TrojanVBKrypt (en bas)

Nous utilisons cette méthode car la transformation d'un fichier PE en image permet de visualiser les différentes sections du fichier.

Cette méthode permet aussi de faire correspondre les zones de l'image aux différentes sections du fichiers, ce qui permet d'identifier des zones suspectes (voir 3.3). De plus, les éditeurs de fichiers malveillants ont tendance à appliquer des modifications à des fichiers déjà connus pour changer leur signature afin de les rendre indétectables par certains AVs. L'injection de changements mineures ne modifie pas la structure générale du fichier, donc de l'image associée. Par exemple, nous pouvons voir dans la figure 3.4 trois représentations de trois fichiers différents selon un antivirus par signature, mais qui s'avèrent être de la même famille msil.krypt.

3.3 Taux de malveillance et score de malveillance

Dans cette section, nous nous intéressons au contenu de la table des imports. Cette table contient des informations sur les fonctions ou les symboles externes dont le fichier PE dépend pour son exécution. En particulier, cette table répertorie les bibliothèques (Dynamic Link Lybrary) et les fonctions spécifiques nécessaires au programme. On s'intéresse particulièrement à ces fonctions car elles peuvent caractériser le comportement



(a) Fichier bénin (b) Fichier malveillant

FIGURE 3.3 – Représentation de fichiers binaires en images avec les sections contenues dans chaque fichier

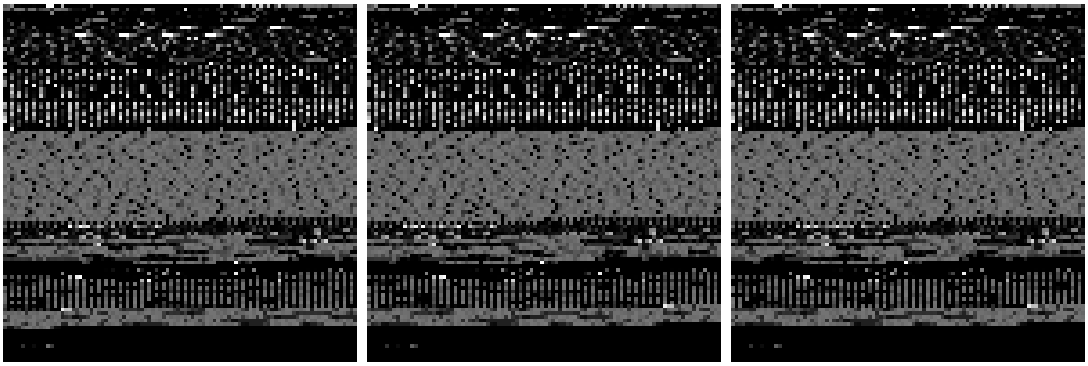


FIGURE 3.4 – Trois variants appartenant à la famille de trojan msil.krypt

d'un fichier et donc permettre de déduire si un fichier a vocation à réaliser un exploit ou non.

3.3.1 Création d'un corpus d'imports I

Soit $U = U_m \cup U_b$, l'ensemble des fichiers PE, avec U_m, l'ensemble des fichiers malveillants et U_b, l'ensemble des fichiers bénins. Pour construire un corpus d'imports I représentatif, on sélectionne un sous-ensemble de fichier malveillant $U'_m \subset U_m$ et on définit I_m comme l'ensemble des k imports les plus présents dans les fichiers de U'_m ($|I_m| = k$). De même, on définit I_b, l'ensemble des k imports les plus présents dans les fichiers de U'_b, avec $U'_b \subset U_b$, un sous-ensemble de fichiers bénins. On définit donc notre corpus d'imports I par $I = I_m \cup I_b$. En choisissant k assez grand, nous pouvons supposer que le corpus I est représentatif des fonctions utilisées par les fichiers binaires. Comme les fichiers bénins et malveillants partagent les mêmes imports $|I| \leq 2k$. Dans la suite, nous noterons d la dimension de I.

Maintenant que nous avons fixé I, nous transformons notre ensemble U de fichiers

binaires en un ensemble $X = \{x^{(i)} \in \{0,1\}^d \mid i \in \{1,...,N\}\}$ où $x^{(i)}$ est la représentation, sous forme de vecteur, d'un fichier binaire $u^{(i)} \in U$ tel que :

$$x_j^{(i)} = \begin{cases} 1 & \text{si } I_j \text{ est un import de } u^{(i)} \\ 0 & \text{sinon.} \end{cases}$$

L'extraction d'import peut aider à conceptualiser le comportement d'un fichier, et peut aussi servir à catégoriser des fichiers selon leur famille.

3.3.2 Définition du taux de malveillance

Afin d'aider à l'analyse de fichiers malveillants, nous souhaitons déterminer si un import peut être considéré comme plutôt malveillant ou plutôt bénin. Pour cela nous définissons le taux de malveillance t_j, qui caractérise la dangerosité d'un certain import $I_j \in I$. Ainsi $T = \{t_j \in [-1,1] \mid j \in \{1,...,d\}\}$ avec :

$$t_j = \frac{\sum\limits_{x^{(i)} \in X_m} x_j^{(i)} - \sum\limits_{x^{(i)} \in X_b} x_j^{(i)}}{\sum\limits_{x^{(i)} \in X} x_j^{(i)}}$$

où $X_m = \{x^{(i)} \in X | u^{(i)} \in U_m\}$ et $X_b = \{x^{(i)} \in X | u^{(i)} \in U_b\}$. Dans la figure 3.5, nous pouvons visualiser une partie des imports d'un fichier. Comme nous pouvons le voir, le taux de malveillance permet d'avoir une représentation rapide des imports potentiellement malveillants contenus dans un fichier PE.

3.3.3 Définition du score de malveillance

Nous définissons aussi $S(u^{(i)}) \in [-1,1]$ qui associe un score de malveillance à un fichier $u^{(i)}$ en se basant sur les imports qu'il contient et le taux de malveillance définit précédemment :

$$S(u^{(i)}) = \begin{cases} \dfrac{\sum\limits_{j=0}^{d} x_j^{(i)} \times t_j}{\|x^{(i)}\|_1} & \text{si } \|x^{(i)}\|_1 \neq 0 \\ 0 & \text{sinon.} \end{cases}$$

avec $\|\cdot\|_1$ la norme L1 définie par $\|x\|_1 := \sum\limits_{j=0}^{d} |x_j|$.

Le but de ce score est de donner une première idée générale sur un fichier potentiellement malveillant. Dans leurs travaux sur la détection de malware via des algorithmes de ML quantique, [BQ23] utilise ce score de malveillance comme une variable pour entraîner leurs modèles de détection.

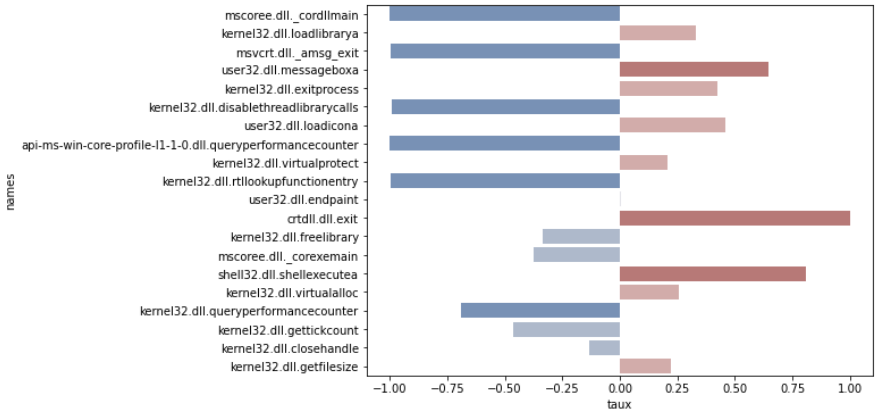


FIGURE 3.5 – Imports et taux de malveillance

Chapitre 4

Cadre et Méthodes

4.1 Bases de données

L'apprentissage automatique nécessite un certain nombre de données pour modéliser les liens qui existent entre plusieurs variables. C'est d'autant plus vrai pour l'apprentissage profond qui permet de répondre à des problématiques plus complexes. Nous avons donc collecté un certain nombre de données à partir de plusieurs sources. Certaines sources fournissent directement des fichiers malveillants ou bénins au format PE (.exe). Pour d'autres, les fichiers ont été transformés sous forme de vecteurs. La table 4.1 précise le nombre et le format des fichiers pour chaque base de données.

Table 4.1 – Information sur les différentes sources de données

Bases de données	Fichiers malveillants	Fichiers bénins	Format	Années de collecte
EMBER	400 000	400 000	Vecteurs	< 2018
BODMAS	57 293	77 142	Vecteurs	sept 2019 - août 2020
	57 293	0	Fichiers PE	
PEMachine Learning	114 737	86 812	Fichiers PE	fev 2018 - oct 2020

EMBER

EMBER est une base de données mise à disposition par Anderson et Roth [AR18]. Nous utilisons la version publiée en 2018 contenant 1 million d'instances, dont 400k fichiers malveillants, 400k fichiers bénins et 200k fichiers non étiquetés, pour encourager l'analyse de malware via des méthodes d'apprentissage non supervisé. Les fichiers malveillants ont été collectés avant 2018. Les instances sont sous forme de vecteurs, les fichiers ont été transformés pour faciliter l'utilisation d'algorithmes de machine learning et deep learning. L'objectif, selon les auteurs, est de fournir une base de données qui peut être utilisée pour l'entraînement de modèles de détection de logiciels malveillants. De plus, ils espèrent que ces données pourront contribuer à renforcer les outils

de détection face à différents problèmes comme la dérive conceptuelle ou les exemples adverses.

BODMAS

BODMAS [Yan+21] est une base de données contenant 134 435 fichiers binaires (PE) dans le même format que EMBER. De plus, les auteurs de BODMAS ont partagé avec nous leur base de 57 293 fichiers malveillants au format PE (ils ne pouvaient pas partager les fichiers bénins pour des raisons de propriété intellectuelle). Ce jeu de données a plusieurs propriétés intéressantes. Tout d'abord, les fichiers ont été collectés sur la période d'août 2019 à septembre 2020, donc les fichiers sont relativement récents. Ensuite, les instances malveillantes ont été étiquetées selon leurs familles et leurs catégories, ce qui permet d'aborder la problématique de la classification de malware. La table 4.2 présente la distribution des fichiers selon leurs catégories. Le terme "autre" regroupe environ une dizaine de catégories sous-représentées dans la base de données BODMAS comme des fichiers de type "dropper" ou "downloader", mais ces fichiers ne sont pas pour autant moins dangereux.

TABLE 4.2 – Répartition des fichiers malveillants de BODMAS selon leurs catégories

Catégories	Nombre de fichiers
Trojan	29,972
Worm	16,697
Backdoor	7,331
Ransomware	821
Autre	2,471
Total	57,293

PEMachineLearning

Nous utilisons aussi le jeu de données PEMachineLearning (PEML), créé et rendu disponible par LESTER [Les]. Ce dataset contient 200 549 fichiers binaires, dont 114 737 fichiers malveillants. Ces derniers ont été collectés sur des sites comme VirusShare [1], MalShare [2] et TheZoo [3]. Les fichiers bénins proviennent de différentes versions du système d'exploitation Windows (à partir de Windows 7).

4.2 Variables et prétraitements

Après avoir récolté les données, nous cherchons à les transformer dans une forme fonctionnelle pour l'entraînement de modèles de ML/DL. Cette partie du travail est appelée prétraitement des données (pre-processing) et consiste à transformer un ensemble

1. https://virusshare.com/
2. https://malshare.com/
3. https://github.com/ytisf/theZoo

de données brutes en un ensemble de données uniformes pour permettre l'entraînement des algorithmes d'apprentissage supervisé. Dans notre cas, il s'agira d'extraire un maximum d'informations des fichiers PE collectés. D'un point vue théorique, il s'agit de trouver une transformation qui, pour chaque fichier binaire, retourne un vecteur dans un espace $\mathcal{X}$ pour que chaque instance soit de la même forme.

Soit U l'ensemble des fichiers PE. On définit ϕ une certaine transformation de sorte que :

$$\begin{aligned} \phi \colon U &\longrightarrow \mathcal{X} \\ u &\longmapsto x, \end{aligned}$$

où $\mathcal{X}$ est l'ensemble des caractéristiques extraites des fichiers binaires, et x est la représentation de u dans $\mathcal{X}$. Au cours de nos travaux, nous utilisons particulièrement la transformation en vecteur "EMBER" et la transformation en image "Grayscale".

Prétraitement EMBER

Comme indiqué précédemment, Anderson et Roth [AR18] ont mis à disposition une base de données d'un million de fichiers malveillants et bénins. Les instances de ce jeu de données sont déjà transformées sous forme de vecteurs, donc utilisables pour l'entraînement de modèles de ML et DL. De plus, les auteurs ont fourni un script qui permet de reproduire cette transformation sur de nouveaux fichiers. Ils se basent sur plusieurs techniques d'extraction d'informations [KM04 ; SB15 ; Sch+ ; Ram12 ; Sha+09] pour transformer un fichier PE en un vecteur contenant 2381 variables réparties dans 9 catégories (voir table 4.3).

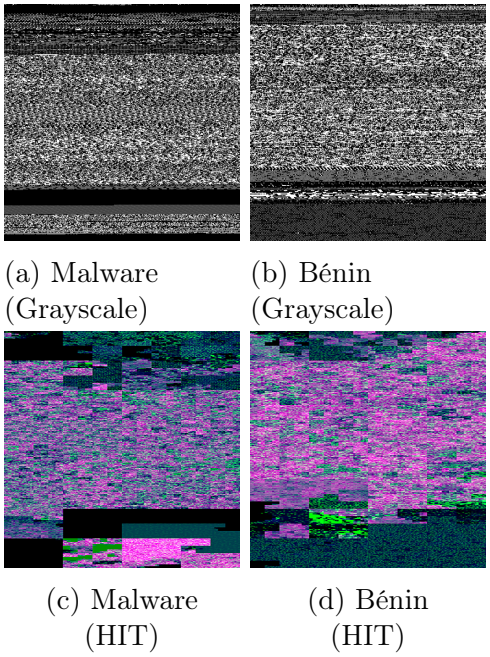
Table 4.3 – Les différents types de variables de EMBER

Nom des variables	Index
Byte histogram	1-256
Byte entropy	257-512
Strings	513-616
General information	617-626
Header information	627-688
Section	689-943
Imports	944-2223
Exports	2224-2350
Data directory information	2251 - 2381

Prétraitement Grayscale

Nous reprenons la transformation de fichiers binaires en image, présentée dans la section 3.2. Cette transformation permet d'interpréter un fichier comme une matrice est donc d'utiliser certains modèles spécifiques comme les réseaux de neurones convolutifs pour créer des outils de détection de fichiers malveillants. D'autres méthodes existent

comme Vu et al. [Vu+19] qui propose une transformation en couleur, appelée HIT, où le fichier est interprété comme un tenseur de taille $h \times w \times 3$, où h et w sont respectivement le nombre de lignes et de colonnes des trois matrices qui codent les niveaux de rouge, de bleu et de vert de l'image. Les auteurs partent du principe que le canal vert de l'œil humain est le plus sensible, ils encodent donc les informations syntaxiques issues du fichier dans ce canal. Les deux canaux servent à capter l'entropie du fichier. La figure 4.1 montrent les transformations Grayscale et HIT. Même si la méthode HIT a montré son efficacité pour la détection de malware, le gain de performance est souvent minime par rapport à la transformation Grayscale, pour un temps de calcul plus long, que ce soit pour la transformation des fichiers en images, ou l'entraînement d'un réseau convolutif. Ces contraintes rendent la méthode difficile d'utilisation dans le cas où il faudrait analyser plusieurs milliers de fichiers par jour.



(a) Malware (Grayscale) (b) Bénin (Grayscale)

(c) Malware (HIT) (d) Bénin (HIT)

Figure 4.1 – Transformation Grayscale (en haut) et transformation HIT (en bas) de fichiers bénins et malveillants

4.3 Modèles

Pour concevoir des outils de détection, nous utilisons des algorithmes d'apprentissage classique (ML) et profond (DL). Pour les données au format EMBER, nous utilisons les modèles XGBoost, LightGBM et des réseaux de neurones (DNN). Pour les images, nous utilisons des réseaux de neurones convolutifs (CNN) car ce sont des modèles adaptés à ce format.

XGBoost

L'algorithme XGBoost, pour "eXtreme Gradient Boosting", est un algorithme d'apprentissage supervisé introduit par Chen et Guestrin [CG16]. Il appartient à la famille

des méthodes d'ensemble qui agrège plusieurs modèles faibles, généralement des arbres de décision, pour créer un modèle fort. Il repose sur le principe de Gradient Boosting où le modèle est entraîné de manière itérative, et à chaque itération, un nouveau modèle faible vient s'ajouter pour corriger l'erreur générale de prédiction du modèle fort. L'algorithme XGBoost est particulièrement efficace dans des tâches de classification et de régression. Bien qu'il soit un modèle de choix pour les ingénieurs et chercheurs en apprentissage automatique, XGBoost a quelques limites comme un temps d'entraînement qui peut devenir relativement long sur des jeux de données volumineux (nombre d'instances et variables), mais aussi une consommation intensive de la mémoire, surtout quand le nombre et la profondeur des arbres sont augmentés. Pour l'entraînement de modèles avec l'algorithme XGBoost, nous utilisons la librairie Python correspondante [4].

LightGBM

L'algorithme LightGBM [Ke+17], pour "Light Gradient Boosting Machine", fait aussi partie des méthodes d'ensemble de type Gradient Boosting. Il a pour avantage d'être particulièrement rapide pour l'entraînement de nouveaux modèles, même face à de larges jeux de données, grâce à plusieurs méthodes comme le regroupement des variables. De plus, il est optimisé pour utiliser moins de mémoire au cours de l'apprentissage. LightGBM montre en plus de très bon résultats, ce qui en fait un modèle concurrentiel à XGBoost, mais il possède néanmoins quelques faiblesses non négligeables comme une certaine sensibilité aux valeurs aberrantes ou des performances réduites face à des données déséquilibrées. L'utilisation de l'un ou l'autre de ces modèles dépendra donc du jeu de données, des ressources disponibles et de la problématique étudiée. Pour l'entraînement de modèle avec l'algorithme LightGBM, nous utilisosn la librairie Python correspondante [5].

Réseaux de neurones artificiels

Comme décrit dans la section 1.2, les réseaux de neurones sont particulièrement utiles pour gérer de grandes quantités de données et de grands espaces de caractéristiques. Nous construisons des modèles de détection à partir de réseaux de neurones denses, ou entièrement connectés, (DNN) entraînés à l'aide de données au format de EMBER. Ces réseaux de neurones permettent généralement d'obtenir de bonnes performances pour la détection de fichiers malveillants, mais ont le désavantage d'être opaques, ce qui rend complexe la compréhension des décisions qu'ils prennent et des résultats qu'ils renvoient. La figure 4.2 montre un exemple d'architecture type utilisée au cours de nos expérimentations.

Nous construisons aussi des modèles de détection à partir de réseaux de neurones convolutifs (CNN). Ces réseaux ont la particularité de pouvoir apprendre à partir d'images, et sont donc entraînés à l'aide des données transformées en image en niveau de gris à partir du prétraitement Grayscale. Même si ce type de réseau reste opaque, comme les DNN, il est possible d'identifier quelles parties d'une image ont aidé le modèle dans sa prise de décision, en plus d'identifier des patterns récurrents dans les fichiers

4. `https://xgboost.readthedocs.io/en/stable/python/index.html`
5. `https://lightgbm.readthedocs.io/en/v3.3.2/index.html`

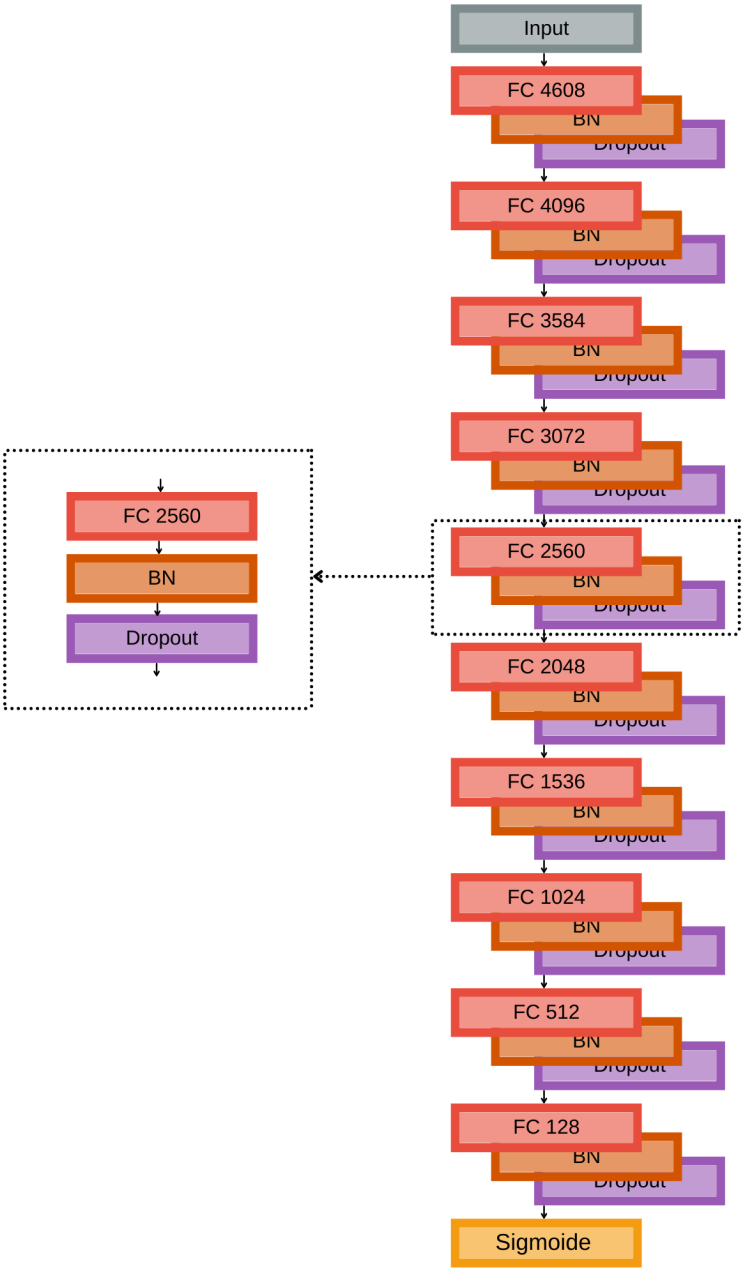


FIGURE 4.2 – DNN architecture

malveillants ou bénins. La figure 4.3 montre une architecture de réseau de neurones convolutifs utilisé pour entraîner un modèle de détection de fichiers malveillants.

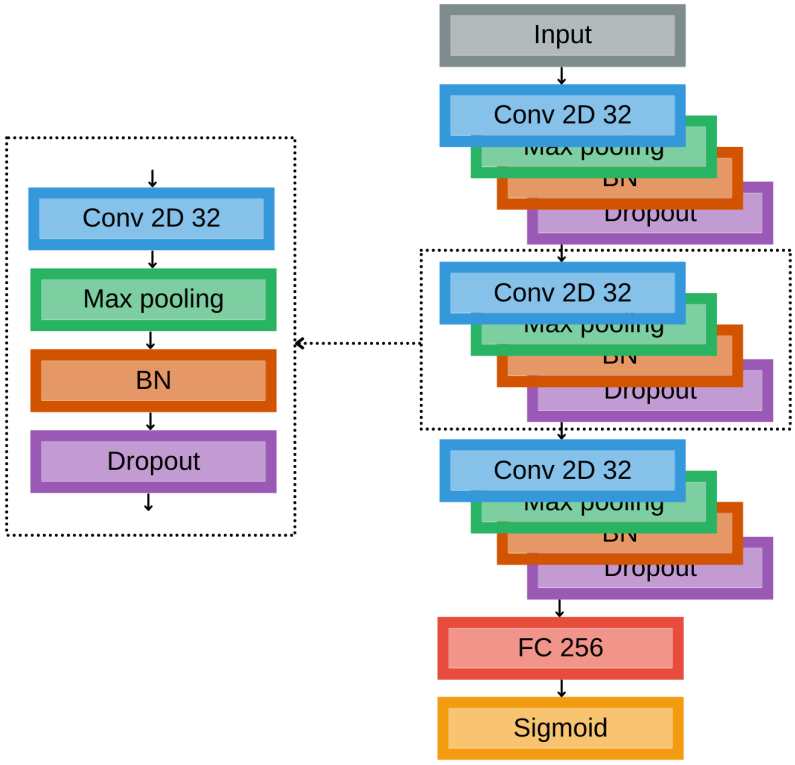


FIGURE 4.3 – CNN architecture

Chapitre 5

Expérimentation

Dans ce chapitre, nous présentons nos travaux et nos expérimentations sur la détection et la classification de fichiers malveillants en comparant différents types de modèles ainsi que plusieurs méthodes de prétraitement. Nous abordons aussi le sujet des fichiers de type ransomware et nous proposons un modèle multi-tâches pour la détection de malware et la détection de ransomware.

5.1 Détection

Dans cette section, nous entraînons différents modèles d'apprentissage supervisé pour la détection de fichiers malveillants. Pour entraîner, valider et évaluer nos modèles, nous utilisons la base de données PEML puisqu'elle contient le plus de fichiers au format PE et est également la plus récente. La table 5.1 résume les performances de chaque modèle à l'aide du score F1 et de l'accuracy. Le modèle XGboost obtient les meilleurs résultats, bien que les performances des modèles LightGBM et DNN soient relativement proches. Nous observons que le CNN est moins performant que les trois autres modèles même si les scores sont proches de $0,95$.

Table 5.1 – Performances des modèles de détection - Test sur le jeu de données PEML

	LGBM	XGBoost	DNN	CNN
Accuracy	0.990	**0.993**	0.9902	0.9458
F1 Score	0.991	**0.994**	0.991	0.95

En complément, nous testons nos modèles sur les fichiers malveillants issus de BODMAS pour évaluer leur capacité de généralisation sur des données issus d'autre sources. Comme le jeu de données BODMAS ne contient que des fichiers malveillants, nous estimons les performances des modèles à l'aide du FNR, qui est une mesure importante dans notre cas. En effet, si le FNR est haut alors le modèle ne détecte pas assez de fichiers malveillants, ce qui peut conduire à des problèmes de sécurité au moment du déploiement d'un outil de détection. Pour chaque modèle, le FNR est indiqué dans le tableau 5.2. Nous ajoutons également le nombre de logiciels malveillants non détectés sur les 57293 fichiers de BODMAS. Le modèle CNN a un FNR élevé par rapport aux autres modèles, il ne détecte pas suffisamment de logiciels malveillants dans BODMAS,

ce qui signifie que ce modèle n'a pas une bonne capacité de généralisation. En revanche, les modèles XGBoost et LightGBM ont des FNR très proches et faibles. Ils obtiennent de bonnes performances pendant la phase de test et ont de bonnes capacités de généralisation. Enfin, le DNN obtient un score parfait de zéro logiciel malveillant non détecté. Nous comparons nos résultats à un modèle connu de l'état de l'art. Il s'agit d'un modèle de type LightGBM entraîné sur le jeu de données EMBER [AR18]. Ce modèle renvoie un faible FNR aussi. À l'exception du CNN, les modèles que nous proposons sont légèrement meilleurs que le modèle EMBER, même s'ils sont entraînés avec des données moins nombreuses mais plus récentes.

TABLE 5.2 – Taux de faux négatifs (FNR) et nombre de malwares non détectés - BODMAS

	LGBM	XGBoost	DNN	CNN	EMBER LGBM
FNR	$1.56 \cdot 10^{-3}$	$0.96 \cdot 10^{-3}$	**0**	0.13	$1.42 \cdot 10^{-2}$
Malware non détectés	84	55	**0**	7448	816

Au vu des résultats du tableau 5.1 et du tableau 5.2, le modèle XGBoost semble être le modèle le plus efficace pour détecter les fichiers malveillants pendant les phases d'apprentissage et de test. En outre, il présente une bonne capacité de généralisation avec un faible FNR sur l'ensemble de données BODMAS. Le modèle LightGBM fournit des résultats légèrement inférieurs mais très proches. Son principal avantage par rapport à XGBoost est son temps de calcul plus rapide [Ke+17]. Même si les résultats du modèle CNN sont assez bons, sa mauvaise performance FNR indique une faible capacité de généralisation. Enfin, le DNN semble être plus performant que les autres avec des résultats proches de XGBoost sur le sous-ensemble de test et le FNR le plus faible, égal à zéro, sur l'ensemble de données BODMAS. D'un autre côté, il nécessite plus de puissance de calcul pour une augmentation de performance assez faible. Le modèle XGBoost peut donc être considéré comme un bon compromis entre le temps de calcul et les performances.

5.2 Classification

Certains logiciels malveillants sont plus dommageables que d'autres. Pour cela, en plus de pouvoir détecter des logiciels malveillants, il est important de pouvoir catégoriser ces logiciels afin de hiérarchiser le niveau de menace. Nous proposons dans cette partie des modèles basés sur l'apprentissage supervisé afin de classifier les fichiers malveillants. Nous nous concentrons en particulier sur les Trojan, Worm, Backdoor, et Ransomware qui sont les fichiers les plus courants. Nous utilisons la base de données fournie par BODMAS pour nos expérimentations. Les instances de cette base de données sont étiquetées selon leurs catégories et leurs familles. Néanmoins, BODMAS ne contient que 821 fichiers de type ransomware, ce qui est trop peu par rapport aux autres catégories (voir table 4.2). Pour pallier ce manque de fichiers, nous avons analysé un certain nombre de fichiers malveillants issus de PEML à l'aide de VirusTotal. Nous considérons qu'un

fichier est un ransomware si la majorité des AV de VT l'identifie comme tel. Cela nous a permis de constituer une base de 10000 fichiers, 2000 de chaque catégorie pour avoir un jeu de données équilibré. L'utilisation d'un dataset équilibré a pour but de limiter le sur-apprentissage.

Nous entraînons plusieurs modèles de ML et DL, les résultats sont présentés dans la table 5.3. Le modèle le plus performant est le modèle LightGBM, suivi des modèles XGBoost et DNN. Avec une précision moyenne de 0.9413 et un score F1 moyen de 0.9412, ces trois modèles semblent être de bons candidats pour la classification des logiciels malveillants, même si ces résultats doivent être améliorés. En revanche, le CNN n'est pas aussi performant et ses résultats ne peuvent pas être considérés comme fiables. Nous pensons que le manque de données pendant l'entraînement et la faible résolution des images (seulement 64x64 pixels) limitent les performances générales de ce genre de modèles.

TABLE 5.3 – Performances des modèles de classification de fichiers malveillants

	LGBM	XGBoost	DNN	CNN
Accuracy	**0.9442**	0.9427	0.9369	0.7270
F1 Score	**0.9440**	0.9425	0.9370	0.7197

5.3 Détection de Fichiers de Type Ransomware

Dans la continuité des problématiques de détection et de classification de fichiers malveillants, nous montrons dans cette section la nécessité de développer des outils spécifiques pour la détection de ransomware et nous proposons des modèles basés sur l'IA pour détecter ce type de logiciels.

5.3.1 Problématiques liées aux rançongiciels

Les logiciels de type rançongiciel sont des fichiers malveillants de plus en plus utilisés par les cyber-attaquants. Leur rôle est d'infiltrer et de rendre inutilisable une machine ou un système d'information en chiffrant l'ensemble des données (cryptographic-ransomware), comme le fait le logiciel Wannacry [YW19 ; MP17], ou bien en bloquant l'accès au système (locker-ransomware) comme le ransomware Reveton [Kha+15]. Pour déchiffrer l'ensemble de ses données ou récupérer le contrôle de sa machine, un utilisateur victime de ce type d'attaque doit généralement payer une rançon aux attaquants. Bien que cette menace existe depuis plusieurs années, les attaques par ransomware se sont intensifiées avec l'essor des cryptomonnaies, qui permettent de recevoir un paiement avec un certain niveau d'anonymat. De plus, des activités parallèles sont apparues comme le marché Ransomware-as-a-Service (RaaS) [MBS20], où n'importe qui peut se procurer des logiciels malveillants de type ransomware et les utiliser sans connaissance préalable. Même si de nombreuses méthodes défensives et préventives existent contre les attaques par ransomware [RN17 ; Kol+17 ; Oz+22 ; MPS19 ; McA16], 54% de ces attaques contre des entreprises restent un succès en 2021, pour un coût moyen de 1.85

million de USD par offensive [Sop2121]. De plus, même si le nombre d'attaques a diminué par rapport à 2021, le coût pour les entreprises a augmenté d'environ 1 million de USD car davantage ont choisi de payer une rançon pour récupérer leurs données. Cette augmentation de l'impact des attaques par ransomware dans le panorama des cybermenaces montre qu'il s'agit d'un problème d'actualité. Cette menace nécessite une attention particulière, aussi bien sur le plan préventif que défensif, en éduquant les utilisateurs ainsi qu'en améliorant les outils de détection de ransomware pour les rendre plus robustes et plus efficaces [Bea+21].

5.3.2 Détection de ransomware

Pour étudier la problématique de la détection de ransomware, nous utilisons le jeu de données construit dans le cadre de la classification de malware, présenté précédemment dans la section 5.2, contenant 10 000 fichiers malveillants dont 2000 fichiers ransomware. Si une instance est de type ransomware, nous notons $y = 0$, sinon nous notons $y = 1$. Nous séparons l'ensemble des données en trois parties pour l'entraînement (70%), la validation (15%) et l'évaluation (15%).

Nous utilisons les deux méthodes de pré-traitement EMBER et Grayscale pour entraîner différents modèles de détection de ransomware basés sur les algorithmes LightGBM, XGBoost, DNN et CNN et les architectures validées au cours de la détection de malware dans la section 5.1. Les performances des quatre modèles sont présentées dans la table 5.4. Comme nous pouvons l'observer, le modèle LightGBM a une efficacité de plus de 99% dans la tâche de détection de ransomware, tout comme les modèles XGBoost et DNN. Pour ce qui est du modèle CNN, entraîné à partir des images, il montre de bonnes performances aussi avec un score de détection de plus de 98%. Il semblerait qu'il ait bien appris à différencier les fichiers de type ransomware par rapport aux autres familles de malware.

TABLE 5.4 – Performances des modèles de détection de ransomware

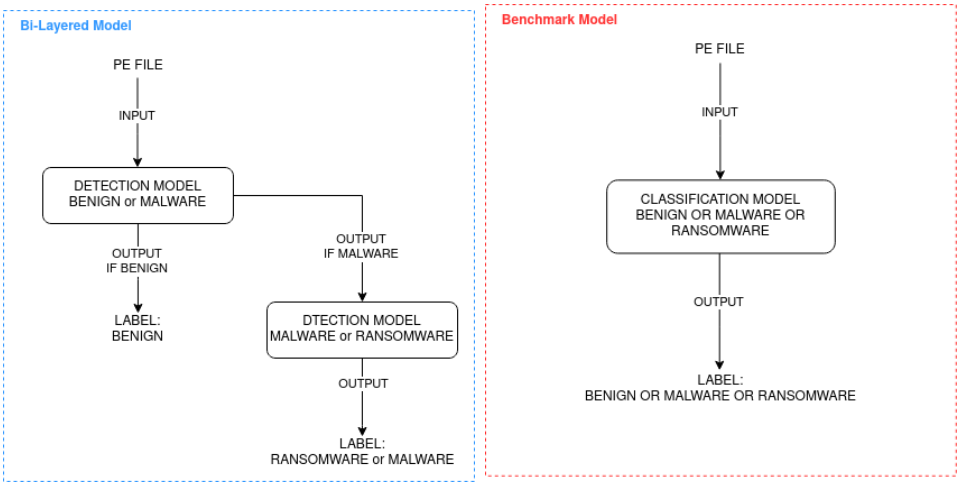
	LGBM	XGBoost	DNN	CNN
ACC	**0.9954**	0.9948	0.9938	0.9830
F1 Score	**0.9971**	0.9969	0.9967	0.9823

5.4 Modèles bi-couches pour la détection de fichiers malveillants et de ransomware

Dans cette section, nous présentons un modèle dont l'objectif est de détecter les fichiers malveillants, et parmi eux, les fichiers de type ransomware. Nous l'appelons **modèle bi-couches** car il est constitué de deux sous-modèles ayant chacun un objectif spécifique. Le premier modèle a été entraîné à détecter les fichiers malveillants (section 5.1) tandis que le second a pour but d'identifier les fichiers de type ransomware (section 5.3.2). Nous combinons ces deux outils afin de construire un modèle unique qui permet de classifier un fichier selon les labels bénin, malveillant ou ransomware, comme présenté

dans la figure 5.1a. Nous construisons donc trois modèles basés sur les algorithmes LightGBM, XGBoost et DNN. Nous n'utilisons pas les modèles CNN car les résultats sont généralement inférieurs aux autres outils de détection.

Pour évaluer les performances de ces modèles multi-couches, nous entraînons d'autres modèles dits **benchmark**. Ils sont entraînés à classifier des fichiers PE selon les trois catégories bénin, malware ou ransomware (figure 5.1b). Nous entraînons trois modèles à partir des algorithmes LightGBM, XGBoost et DNN en utilisant le jeu de données de la section 5.2 et 8000 fichiers bénins issus du dataset PEML.



(a) Modèle bi-couches (b) Modèle benchmark

FIGURE 5.1 – Schéma des modèles de détection de malware et de ransomware

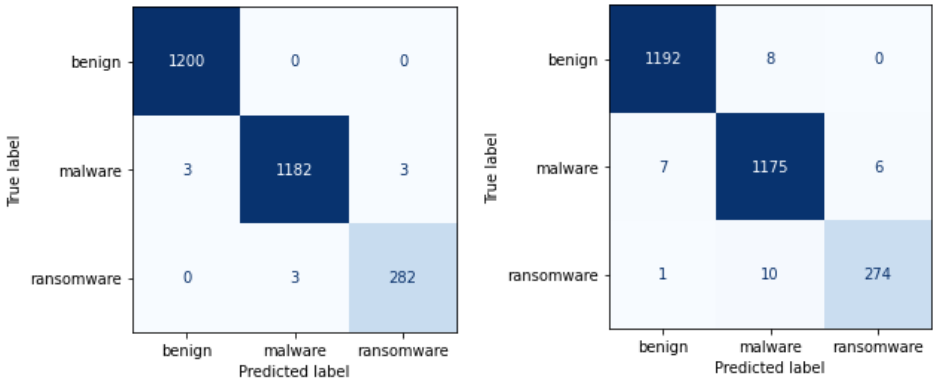
Pour chacun des modèles bi-couches et benchmark, nous présentons les performances de détection et de classification dans la table 5.5. Tout d'abord, nous pouvons voir que les modèles bi-couches sont meilleurs que leurs homologues de référence. En observant plus en détails la matrice de confusions pour les modèles XGBoost bi-couches et benchmark (figure 5.2), nous constatons que le modèle bi-couches produit moins d'erreurs que le modèle benchmark, avec aucun fichier bénin mal classifié, et moins d'erreurs de prédiction sur les fichiers malveillants.

TABLE 5.5 – Performances des modèles bi-couches et de référence

	Accuracy		F1 Score	
	Benchmark	Bi-couches	Benchmark	Bi-couches
LGBM	0.9896	0.9932	0.9895	0.9934
XGBoost	0.988	**0.9960**	0.988	**0.9966**
DNN	0.9867	0.9913	0.9868	0.9915

Les modèles bi-couches, composés de deux sous-modèles entraînés pour séparer deux classes, sont légèrement plus efficaces que les modèles de classifications multi-classes sur la problématique. Ils ont aussi l'avantage d'être plus flexible puisque chaque couche peut être entraînée et optimisée pour une tâche en particulier. Si de nouvelles données sont disponibles, il est possible de ré-entraîner ou d'actualiser le modèle concerné ce qui peut-être un gain en temps et en ressources pour les entreprises. De plus, la construction

par couche permet d'ajouter des modèles spécifiques pour gérer de nouvelles tâches de classification ou de détection.



(a) XGBoost Bi-couches (b) XGBoost Benchmark

FIGURE 5.2 – Matrices de confusion des modèles XGBoost bi-couches et benchmark sur l'ensemble de test

Chapitre 6

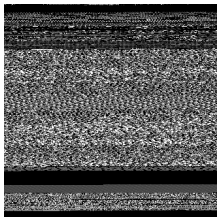
Analyse Approfondie

6.1 Explicabilité du modèle CNN

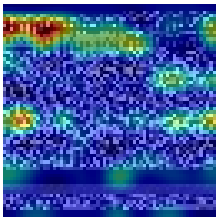
L'un des défauts des réseaux de neurones artificiels, et en particulier les réseaux de neurones convolutifs, est leur manque de transparence. Il s'agit de modèles dits "boite noire" pour lesquels l'interprétation des résultats et des décisions qu'ils renvoient est souvent difficile voir impossible [Loy19]. Certaines méthodes ont été développées pour aider à comprendre ces modèles afin de générer des cartes de saillance [Spr+15] (saliency map) ou des cartes d'activations (activation map) [Zho+15]. Ces techniques sont particulièrement utilisées en imagerie médicale pour aider à identifier des anomalies non visibles à l'œil nu ou pour aider à valider un diagnostic [Aru+21].

Au cours de nos travaux, nous avons voulu ajouter une couche de compréhension à nos modèles. Nous utilisons donc l'algorithme Grad-CAM++ [Cha+18] qui permet de générer des cartes d'activation, associé au modèle CNN présenté dans la section 4.3, et utilisé au pour nos expérimentations. Grâce à l'outil Grad-CAM++, nous pouvons identifier les zones d'un fichier qui ont aidé le modèle à faire sa prédiction. Par exemple, avec la figure 6.1, nous pouvons observer un malware, correctement détecté par le CNN et sa carte d'activation produite à l'aide de Grad-CAM++. Plus la couleur de la carte est chaude (rouge), plus les pixels correspondant ont contribué à la prise de décision du modèle, au contraire, plus la zone est froide (bleu), moins elle a contribué. Nous pouvons observer que pour le malware, il y a une zone particulièrement d'intérêt, contrairement au fichier bénin, où le CNN a besoin de plus d'information pour prendre une décision.

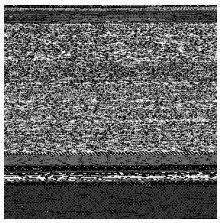
Cette représentation peut aussi aider à identifier un problème d'apprentissage du modèle. La figure 6.2 montre par exemple un fichier bénin et un fichier malveillant, détectés tous les deux comme malware par le CNN. Nous pouvons remarquer que la zone d'intérêt des deux images est une zone de padding, c'est-à-dire une zone artificiellement remplie de séquences d'octets nuls. Nous pouvons en déduire que le CNN considère qu'une zone de padding comme un indice de compromission ou de malveillance. Bien que cette technique soit particulièrement utilisée par les éditeurs de fichiers malveillants, souvent pour contourner les AVs par signature ou heuristique, elle peut aussi être utilisée par des éditeurs de logiciels bénins pour respecter certains standards dans la structure et la longueurs des données. Le CNN a sur-interprété en considérant que les zones de padding étaient seulement présentes dans les fichiers malveillants.



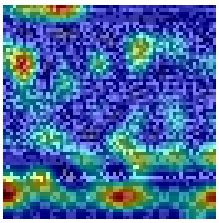
(a) Image grayscale (malware)



(b) Carte d'activation (malware)

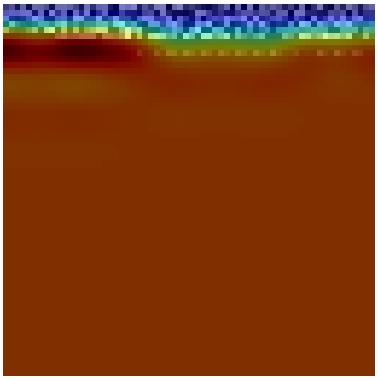


(c) Image grayscale (bénin)

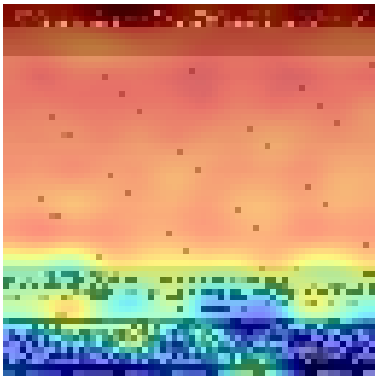


(d) Carte d'activation (bénin)

FIGURE 6.1 – Transformation de fichiers en images et cartes d'activation renvoyées par le modèle CNN et l'outil Grad-CAM++



(a) Fichier malveillant

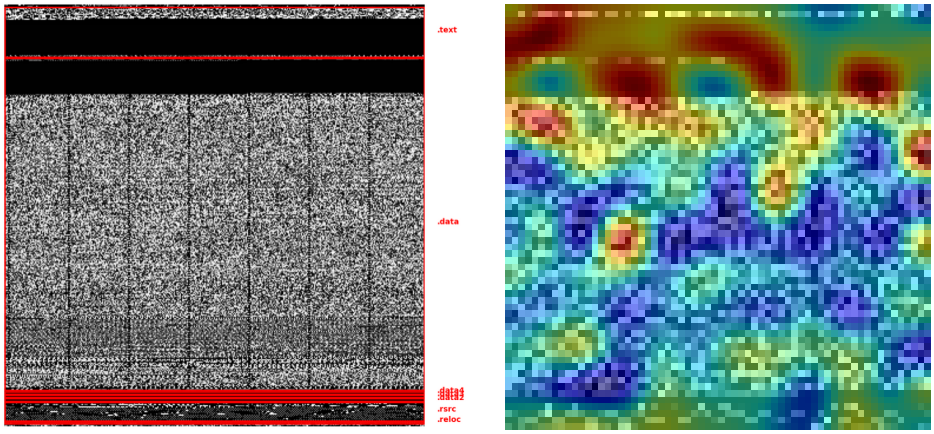


(b) Fichier bénin

FIGURE 6.2 – Cartes d'activation de deux fichiers détectés comme malware à cause du padding

6.2 Exemple d'analyse approfondie

Dans cette section, nous montrons un exemple d'utilisation de nos outils afin de faire une analyse approfondie d'un fichier exécutable potentiellement malveillant. Il s'agit d'un fichier ayant été détecté par le modèle CNN avec un taux de détection de 0.943. Nous pouvons commencer par comparer sa carte d'activation et les sections. La figure 6.3 montre le fichier en version grayscale, les différentes sections qui le compose et sa carte d'attention renvoyée par le CNN, à l'aide de Grad-CAM++. Nous retrouvons le phénomène de padding présenté dans la section précédente, qui a contribué à la décision du modèle. Les zones de padding sont situées à la fin de la section .text et au début de la section .data. Nous pouvons supposer que l'éditeur de ce malware à voulu tromper un ou plusieurs antivirus en changeant la signature de ce fichier. Nous pouvons aussi voir que la section .data a une structure très uniforme et régulière, il s'agit



(a) Image grayscale et section du fichier

(b) Carte d'attention du fichier renvoyée par le CNN

FIGURE 6.3 – Comparaison de l'image grayscale et de la carte d'attention du fichier étudié

d'une autre forme d'obfuscation où le développeur du logiciel répète un pattern pour dissimulation de l'information au sein d'une section. Le pattern est souvent aléatoire. En ce qui concerne la carte d'activation, il semblerait que le CNN ait trouvé des zones d'intérêt dans la section .data, donc qu'il y est bien de l'information dissimulée au sein de cette section.

Pour aller plus loin, nous pouvons regarder les imports de ce fichier avec la figure 6.4. Nous pouvons voir que ce logiciel possède plusieurs imports considérés comme malveillants selon la définition de notre taux de malveillance. Entre autres, LoadLibraryA, GetModuleHandle et VirtualAlloc sont souvent utilisés pour effectuer une interception (hooking). C'est une attaque qui consiste à manipuler ou intercepter des appels fonctions afin de réaliser un exploit (vol d'information, compromission du système). La fonction CreateMutexA est souvent utilisée pour vérifier qu'une instance du fichier n'est pas déjà en cours d'exécution. Enfin, la fonction ShellEcexuteA a pour but d'exécuter d'autres programmes. Nous pouvons supposer que le but de ce fichier est d'abord de réaliser un hooking afin de détourner les fonctions système, d'augmenter les privilèges du fichier, pour ensuite exécuter un autre programme sûrement malveillant.

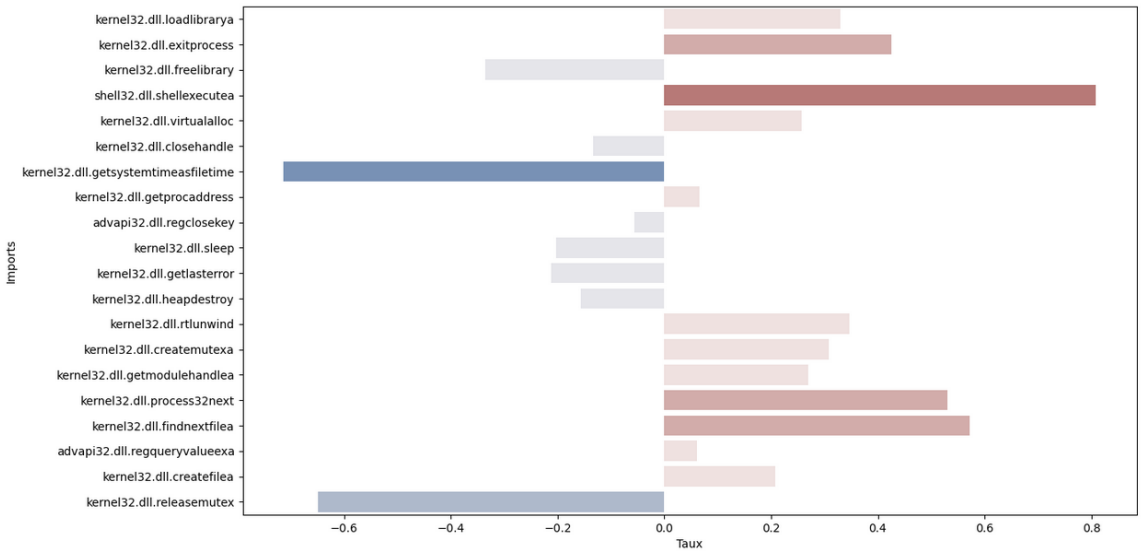


FIGURE 6.4 – Certains imports extrait de l'exemple

Chapitre 7

Conclusion et Travaux Futurs

7.1 Conclusion

Dans cette partie, nous avons présenté plusieurs outils relatifs à l'analyse et la détection de fichiers malveillants. Les outils d'analyse préliminaire permettent d'avoir un aperçu de la nature et du comportement d'un fichier. Entre autres, l'extraction d'imports, le taux de malveillance et le score de malveillance sont de très bons indicateurs de compromission. Pour ce qui est des outils de détection, nous pouvons observer que les modèles entraînés à l'aide de la méthode de prétraitement EMBER sont particulièrement efficaces. Les réseaux de neurones convolutifs produisent des résultats relativement corrects dans l'ensemble, si l'on exclut la classification. Nous pensons que la principale faiblesse de nos travaux repose sur le manque de données, ce qui est une des briques les plus importantes pour l'entraînement de CNN. Notre contribution originale est le modèle bi-couches qui permet de répondre à plusieurs problématiques, comme la détection de malware et la détection de ransomware, à l'aide de sous-modèles optimisés pour ces tâches. De plus, nous pouvons constater qu'il est plus efficace qu'un modèle de classification classique. Le principal avantage de ce modèle est sa flexibilité, qui permet de ré-entraîner ou d'actualiser un des sous-modèles si l'on dispose de nouvelles données. Il est aussi possible de s'attarder sur d'autres problématiques et d'ajouter d'autres couches à ce modèle selon les besoins. Nous nous sommes principalement attardés sur la détection de ransomware, mais il serait possible d'adapter la méthode à d'autres catégories de malware comme les fichiers trojan, worm ou backdoor.

7.2 Travaux futurs

Nous pensons que les outils classiques de détection de malware peuvent être grandement améliorés à l'aide des modèles d'IA, grâce à leur capacité à gérer de grandes quantité de données et à extrapoler des règles de décisions. Néanmoins, il est souvent difficile pour un humain de comprendre comment et pourquoi une décision a été prise par un modèle, c'est pourquoi il est nécessaire de se tourner vers des méthodes qui pourront aider à rendre intelligible le fonctionnement de ces outils, comme les méthodes XAI (Explainable Artificial Intelligence) ou IML (Interpretable Machine Learning). De plus, il serait nécessaire d'étudier la résilience de nos modèles face à des problématiques comme les exemples adverses ou la dérive conceptuelle, afin d'améliorer leur efficacité

à long terme, sans avoir à les ré-entraîner de manière trop régulière, ce qui est un coût en temps et en ressources.

Références

[Ach+21] Jatin Acharya et al. « Detecting Malware, Malicious URLs and Virus Using Machine Learning and Signature Matching ». In : *2021 2nd International Conference for Emerging Technology (INCET)*. 2021, p. 1-5. doi : 10.1109/INCET51464.2021.9456440.

[AF22] Muhammad Shoaib Akhtar et Tao Feng. *Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time*. 2022. doi : 10.3390/sym14112308.

[Alo+18] Md Zahangir Alom et al. « The history began from alexnet : A comprehensive survey on deep learning approaches ». In : *arXiv preprint arXiv :1803.01164* (2018).

[AR18] Hyrum S Anderson et Phil Roth. *EMBER : An open dataset for training static pe malware machine learning models*. 2018. arXiv : 1804.04637.

[Aru+21] Nishanth Arun et al. « Assessing the Trustworthiness of Saliency Maps for Localizing Abnormalities in Medical Imaging ». In : *Radiology : Artificial Intelligence* 3.6 (2021), e200267. doi : 10.1148/ryai.2021200267.

[Bea+21] Craig Beaman et al. « Ransomware : Recent advances, analysis, challenges and future research directions ». In : *Computers and Security* 111 (2021), p. 102490. issn : 01674048. doi : 10.1016/j.cose.2021.102490.

[BQ23] Grégoire Barrué et Tony Quertier. *Quantum Machine Learning for Malware Classification*. 2023. arXiv : 2305.09674 [cs.CR].

[Cap+22] Nicola Capuano et al. « Explainable Artificial Intelligence in CyberSecurity : A Survey ». In : *IEEE Access* 10 (2022), p. 93575-93600. doi : 10.1109/ACCESS.2022.3204171.

[CG16] Tianqi Chen et Carlos Guestrin. « Xgboost : A scalable tree boosting system ». In : *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, p. 785-794.

[Cha+18] Aditya Chattopadhay et al. « Grad-CAM++ : Generalized gradient-based visual explanations for deep convolutional networks ». In : *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, p. 839-847.

[Cha+19] Leong Chan et al. « Survey of AI in Cybersecurity for Information Technology Management ». In : *2019 IEEE Technology & Engineering Management Conference (TEMSCON)*. 2019, p. 1-8. doi : 10.1109/TEMSCON.2019.8813605.

[Cho+23] Md Naseef-Ur-Rahman Chowdhury et al. *Android Malware Detection using Machine learning : A Review*. 2023. arXiv : 2307.02412 [cs.CR].

[Den+09] Jia Deng et al. « Imagenet : A large-scale hierarchical image database ». In : *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, p. 248-255.

[Fir+10] Ivan Firdausi et al. « Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection ». In : *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*. 2010, p. 201-203. doi : 10.1109/ACT.2010.33.

[Fle+18] William Fleshman et al. « Non-negative networks against adversarial attacks ». In : *arXiv preprint arXiv :1806.06108* (2018).

[Han+14] Kyoung Soo Han et al. « Malware analysis using visualized images and entropy graphs ». In : *International Journal of Information Security* 14.1 (avr. 2014), p. 1-14. doi : 10.1007/s10207-014-0242-0. url : https://doi.org/10.1007/s10207-014-0242-0.

[He+15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv : 1512.03385 [cs.CV].

[Ke+17] Guolin Ke et al. « LightGBM : A highly efficient gradient boosting decision tree ». In : *Advances in Neural Information Processing Systems*. T. 2017-Decem. 2017, p. 3147-3155.

[Kha+15] Amin Kharraz et al. « Cutting the gordian knot : A look under the hood of ransomware attacks ». In : *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. T. 9148. Springer Verlag, 2015, p. 3-24. isbn : 9783319205496. doi : 10.1007/978-3-319-20550-2_1.

[KM04] Jeremy Z. Kolter et Marcus A. Maloof. « Learning to detect malicious executables in the wild ». In : Association for Computing Machinery (ACM), 2004, p. 470-478. isbn : 1581138881. doi : 10.1145/1014052.1014105.

[Kol+17] Eugene Kolodenker et al. « PayBreak : Defense against cryptographic ransomware ». In : *ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*. 2017, p. 599-611. isbn : 9781450349444. doi : 10.1145/3052973.3053035.

[Kre+18] Felix Kreuk et al. « Adversarial examples on discrete sequences for beating whole-binary malware detection ». In : *arXiv preprint arXiv :1802.04528* (2018), p. 490-510.

[KSH12] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton. « ImageNet Classification with Deep Convolutional Neural Networks ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de F. Pereira et al. T. 25. Curran Associates, Inc., 2012. url : https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[Les] Michael Lester. *Pratical Security Analytics - PE Malware Machine Learning Dataset*. https://practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/.

[Loy19] Octavio Loyola-González. « Black-Box vs. White-Box : Understanding Their Advantages and Weaknesses From a Practical Point of View ». In : *IEEE Access* 7 (2019), p. 154096-154113. doi : 10.1109/ACCESS.2019.2949286.

[MBS20] Per Håkon Meland, Yara Fareed Fahmy Bayoumy et Guttorm Sindre. « The Ransomware-as-a-Service economy within the darknet ». In : *Computers and Security* 92 (mai 2020). issn : 01674048. doi : 10.1016/J.COSE.2020.101762.

[McA16] McAfee Labs. « Understanding Ransomware and Strategies to Defeat it ». In : *McAfee Labs* (2016), p. 1-18.

[MP17] Savita Mohurle et Manisha Patil. « A brief study of Wannacry Threat : Ransomware Attack 2017 ». In : *International Journal of Advanced Research in Computer Science* 8.5 (2017), p. 1938-1940. issn : 0976-5697.

[MPS19] Sumith Maniath, Prabaharan Poornachandran et V. G. Sujadevi. *Survey on prevention, mitigation and containment of ransomware attacks*. T. 969. Springer Singapore, 2019, p. 39-52. isbn : 9789811358258. doi : 10.1007/978-981-13-5826-5_3.

[MQC22] Benjamin Marais, Tony Quertier et Christophe Chesneau. « Malware Analysis with Artificial Intelligence and a Particular Attention on Results Interpretability ». In : *Distributed Computing and Artificial Intelligence, Volume 1 : 18th International Conference*. Sous la dir. de Kenji Matsui et al. Cham : Springer International Publishing, 2022, p. 43-55. isbn : 978-3-030-86261-9.

[MQM22] Benjamin Marais, Tony Quertier et Stéphane Morucci. « AI-based Malware and Ransomware Detection Models ». In : *Conference on Artificial Intelligence for Defense*. Actes de la 4ème Conference on Artificial Intelligence for Defense (CAID 2022). DGA Maîtrise de l'Information. Rennes, France, nov. 2022. url : https://hal.science/hal-03881198.

[Nat+11] L Nataraj et al. « Malware images : Visualization and automatic classification ». In : *ACM International Conference Proceeding Series*. 2011. isbn : 9781450306799. doi : 10.1145/2016904.2016908.

[Oz+22] Harun Oz et al. « A Survey on Ransomware : Evolution, Taxonomy, and Defense Solutions ». In : *ACM Computing Surveys* (2022). issn : 0360-0300. doi : 10.1145/3514229. arXiv : 2102.06249.

[Raf+17] Edward Raff et al. « Malware detection by eating a whole EXE ». In : *arXiv* (2017). doi : 10.13016/m2rt7w-bkok. arXiv : 1710.09435.

[Ram12] Karthik Raman. « Selecting Features to Classify Malware ». In : 2012.

[Rez+17] Edmar Rezende et al. « Malicious software classification using transfer learning of resnet-50 deep neural network ». In : *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2017, p. 1011-1014.

[RN17] Ronny Richardson et Max North. « Ransomware : Evolution, Mitigation and Prevention ». In : *International management review* 13.1 (2017), p. 10-21. issn : 1551-6849.

[SB15] Joshua Saxe et Konstantin Berlin. « Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features ». In : (2015).

[Sch+] M.G. Schultz et al. « Data mining methods for detection of new malicious executables ». In : *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE Comput. Soc. doi : 10.1109/secpri.2001.924286.

[Sha+09] Muhammad Zubair Shafiq et al. « A Framework for Efficient Mining of Structural Information to Detect Zero-Day Malicious Portable Executables ». In : 2009.

[Smi+20] Michael R. Smith et al. « Mind the Gap : On Bridging the Semantic Gap between Machine Learning and Malware Analysis ». In : *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*. New York, NY, USA : Association for Computing Machinery, 2020, p. 49-60. isbn : 9781450380942.

[Sop2121] Sophos Ltd. « The State of Ransomware 2021 ». In : *Sophos Whitepaper* (2021).

[Spr+15] Jost Tobias Springenberg et al. *Striving for Simplicity : The All Convolutional Net*. 2015. arXiv : 1412.6806 [cs.LG].

[Vu+19] Duc-Ly Vu et al. « A convolutional transformation network for malware classification ». In : *2019 6th NAFOSTED Conference on Information and Computer Science (NICS)*. IEEE. 2019, p. 234-239.

[Vu+20] Duc-Ly Vu et al. « HIT4Mal : Hybrid image transformation for malware classification ». In : *Transactions on Emerging Telecommunications Technologies* 31.11 (2020), e3789.

[Yak+18] Hiromu Yakura et al. « Malware analysis of imaged binary samples by convolutional neural network with attention mechanism ». In : *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. 2018, p. 127-134.

[Yan+21] Limin Yang et al. « BODMAS : An Open Dataset for Learning based Temporal Analysis of PE Malware ». In : *Proceedings - 2021 IEEE Symposium on Security and Privacy Workshops, SPW 2021*. 2021, p. 78-84. isbn : 9781728189345. doi : 10.1109/SPW53761.2021.00020.

[YW19] Lena Y. Connolly et David S. Wall. « The rise of crypto-ransomware in a changing cybercrime landscape : Taxonomising countermeasures ». In : *Computers and Security* 87 (nov. 2019). issn : 01674048. doi : 10.1016/j.cose.2019.101568.

[Zho+15] Bolei Zhou et al. *Learning Deep Features for Discriminative Localization*. 2015. arXiv : 1512.04150 [cs.CV].

Deuxième partie

Évasion avec de l'apprentissage par renforcement

Chapitre 8

Introduction

La détection de logiciels malveillants est devenue une priorité pour les entreprises, ainsi qu'un sujet de recherche d'importance au vu du nombre d'attaques grandissant au cours des dernières années. Selon Sophos [Sop2020], en 2020, 73% des attaques via un fichier de type "ransomware" ont été un succès. AV-TEST [AV-Test] estime que 450 000 nouveaux logiciels malveillants sont découverts chaque jour, dont 93% sont des fichiers Windows au format Portable Executable (PE). De plus, les méthodes de détection de fichiers malveillants, comme l'analyse par signature par exemple, deviennent inefficaces face à l'augmentation du nombre de fichiers malveillants sur le marché, variants comme nouveaux malwares [Son+20]. Certains logiciels malveillants sont d'ailleurs spécialisés dans la propagation de nouveaux variants, comme les fichiers malveillants polymorphes ou métamorphiques. Comme de nombreux éditeurs de logiciels antivirus utilisent des méthodes de détection traditionnelles basées sur les signatures pour détecter et bloquer les fichiers malveillants, lorsqu'un nouveau variant ou un nouveau type de malware est identifié, il a eu le temps nécessaire pour se propager et de muter.

D'un point de vue offensif, outrepasser un AV est indispensable pour réaliser un exploit. D'un point de vue défensif, il est indispensable de s'assurer qu'un AV soit résistant face aux différentes techniques d'évasion connues. Nos travaux ont donc pour but de créer un agent autonome capable de modifier des fichiers PE afin de passer à travers des outils de détection de logiciels malveillants. Cela permettra d'identifier les faiblesses de ces outils et d'améliorer en conséquence leurs modules de détection.

Nous utilisons une approche proposée par Anderson, Filar et Roth [AFR17] en 2017, qui consiste à attaquer des solutions de détection dîtes « boîte noire », c'est-à-dire des systèmes qui ne renvoient pas d'autre information que la nature du fichier PE, sous forme de label (malveillant ou bénin). Ce type d'attaque est le plus générique possible, elle ne nécessite aucune connaissance préalable sur le fonctionnement d'un AV, c'est-à-dire la méthode qu'il emploie pour la détection, et les caractéristiques utilisées pour classifier un fichier PE. À partir d'un logiciel normalement étiqueté comme malveillant, nous générons une version modifiée qui n'est plus détectée. La méthode utilisée ne détériore pas et ne modifie pas le comportement du logiciel malveillant, il garde ses fonctionnalités et est toujours utilisable.

Dans nos travaux, nous nous concentrons sur des solutions de détection statiques. Ce sont ces outils qui sont largement utilisés et qui peuvent procéder à un grand nombre d'analyses en peu de temps. Nos travaux montrent qu'il est facile d'outrepasser ces solutions en appliquant certaines modifications à des fichiers malveillants, et ce de

manière automatique. Ce processus est nécessaire pour identifier les faiblesses d'un outil de détection et le rendre plus efficace face aux nouveaux variants de fichiers malveillants.

8.1 Contexte et travaux connexes

La détection de logiciels malveillants est devenue un des sujets prioritaires dans le domaine de la cyber-sécurité au vu des conséquences économiques et des dommages que peuvent engendrer ce genre d'attaque [And+13]. Les avancées dans les domaines de l'intelligence artificielle, de l'apprentissage supervisé et de l'apprentissage profond ont permis d'améliorer la détection et la classification des fichiers malveillants à l'aide de ces technologies [RN20 ; UAB19]. En effet, plusieurs chercheurs en cyber-sécurité et en intelligence artificielle ont publié des datasets de fichiers malveillants comme EMBER [AR18a], SOREL-20M [HR20] et BODMAS [Yan+21]. Ces datasets ont permis à d'autres chercheurs d'améliorer les outils de détection basés sur des modèles de machine learning et de deep learning, tout en les comparant avec ce qui existe déjà afin de suivre les progrès réalisés par la communauté. En plus de ces bases de données, certains chercheurs mettent à disposition des protocoles pour entraîner des modèles. C'est le cas de Anderson et Roth [AR18a] avec la méthode d'extraction utilisée pour générer la base de données EMBER ou de Raff et al. [Raf+17] avec MalConv.

Néanmoins, les modèles de détection basés sur des algorithmes d'apprentissage supervisé (ML ou DL), bien que performants, ne sont pas forcément résistants face à certaines attaques qui consistent à générer de fausses instances, appelées exemples adverses. Les exemples adverses sont introduits pour la première fois dans les travaux de Szegedy et al. [Sze+13]. Ces exemples sont spécialement créés dans le but de tromper des modèles de ML ou DL et perturber la réponse de l'algorithme. Goodfellow et al. [Goo+14] formalisent le principe avec la classification d'images, mais la méthode peut être généralisée à tous types de format de données. Dans le cas de l'analyse de fichiers malveillants, plusieurs approches ont été proposées afin de générer des variants de malware qui trompe les outils de détection, en étant identifiés comme des fichiers bénins. Parmi les approches existantes, nous retrouvons les GAN (Generative Adversarial Network) avec MalGan [Hu+17 ; KOD19] et MalFox [Zho+20], ainsi que des techniques basées sur des problèmes d'optimisations [Dem+21b]. De nombreuses autres approches existent [Dem+21a ; Kon+21 ; PY20 ; AGA21 ; Dem+21a].

Figure 8.1 – Génération d'exemple adverse de fichier malveillant illustré par Huang et al. [Hua+19]

Anderson, Filar et Roth [AFR17] utilisent une autre branche de l'intelligence artificielle appelée « apprentissage par renforcement » (reinforcement learning, RL) pour générer des exemples adverses face à des outils de détection de malware. Ils ont publié sur GitHub un environnement RL (« Gym-Malware ») qui permet de générer de nouveaux variants de fichiers malveillants. Leur agent vise un modèle de détection de type

LightGBM entraîné sur 100000 fichiers malveillants et bénins et, après entraînement, atteint un taux d'évasion de 16%. Song et al. [Son+20] proposent aussi un environnement open-source appelé « MAB-malware », disponible sur GitHub. Leur agent atteint cette fois un 75% d'évasion sur deux modèles de détection de l'état de l'art (EMBER [AR18a] et MalConv [Raf+17]) et jusqu'à 48% face à des AVs commerciaux, dans une configuration totalement opaque. En particulier, les auteurs se concentrent sur la ré-implémentation des actions afin de ne pas casser le fonctionnement des logiciels malveillants. En effet, jusqu'à 60% des fichiers générés avec l'environnement Gym-Malware sont corrompus et donc non-fonctionnels. Fang et al. [Fan+20] proposent une approche attaque-défense avec DeepDetectNet et RLAttackNet. En entraînant RLAttackNet à outrepasser DeepDetectNet, ils gênèrent des variants de malware dont ils se servent pour améliorer ce dernier. Le taux de succès de RLAttackNet passe donc de 19.13% à 3.1%.

8.2 Contributions

Pour comprendre l'évolution des fichiers malveillants, et identifier les faiblesses des solutions de détection modernes, nous proposons un outil appelé **MERLIN**, pour Malware Evasion with Reinforcement LearnINg. Cet outil repose sur des algorithmes d'apprentissage par renforcement. Il s'agit d'un agent autonome donc le but est de générer de nouveaux variants de malware, sous contrainte qu'ils ne soient pas détectés par différents outils de détection.

Une première version de MERLIN repose sur l'algorithme Double-DQN (Double Deep Q Network), et est entraînée à outrepasser des modèles de détection ML comme Malconv [Raf+17], LightGBM-EMBER [AR18a] et Grayscale [MQC22]. Notre agent atteint un score d'évasion quasiment parfait avec, 100% de réussite face Malconv et 98% face à Grayscale. Ces performances sont moindre face à EMBER avec seulement 67% d'évasion. Nous avons donc développé une seconde version qui repose cette fois sur l'algorithme REINFORCE [Wil92]. Á notre connaissance, nous sommes les premiers à proposer un agent entraîné à l'aide de cet algorithme pour l'évasion de malware. Face à EMBER, l'agent REINFORCE montre une meilleure capacité d'évasion que l'agent DQN avec une capacité d'évasion de 74.2%. Enfin, nous testons les deux versions de MERLIN face à un AV classique, et l'agent DQN obtient 30% d'évasion, contre 70% pour l'agent REINFORCE.

Une autre partie de nos travaux consiste à créer un rapport de vulnérabilité pour un outil de détection en particulier. En effet, nos agents apprennent à modifier des fichiers malveillants afin qu'ils soient identifiés comme bénins par différentes solutions de détection. Nous pouvons donc déterminer les faiblesses d'un AV ou d'un modèle de détection de manière automatique en analysant les actions effectuées par nos différents agents pour outrepasser ces outils. Ces informations peuvent être exploitées afin de comprendre comment un outil de détection peut échouer et réagir en conséquence.

Enfin, notre outil MERLIN permet de générer de nouveaux variants de logiciels malveillants et de créer une base de données de fichiers PE inédits. Cette base de données peut être utilisée comme un atout préventif, via l'entraînement des modèles ML et DL, pour gérer de manière proactive les futurs variants de logiciels malveillants.

Une partie des travaux présentés dans cette partie ont donné lieu à la rédaction d'un

article, diffusé en ligne et en attente de soumissions :

- Benjamin Marais, Tony Quertier et Stéphane Morucci. « AI-based Malware and Ransomware Detection Models ». In : *Conference on Artificial Intelligence for Defense*. Actes de la 4ème Conference on Artificial Intelligence for Defense (CAID 2022). DGA Maîtrise de l'Information. Rennes, France, nov. 2022. URL : https://hal.science/hal-03881198

8.3 Plan

Dans le chapitre 9, nous présentons tout d'abord les notions relatives à l'apprentissage par renforcement. Nous parlons ensuite de notre cadre RL pour l'évasion de malware dans le chapitre 10, en décrivant le contexte, les agents et leur fonctionnement, les environnements pour chaque type de solutions de détection, et les actions utilisées pour tromper un AV. Dans le chapitre 11, nous présentons les résultats des phases d'entraînement et de test. Nous analysons l'efficacité des agents et nous présentons le rapport de vulnérabilité dans le chapitre 12 puis nous terminons par un résumé de nos travaux et une discussion sur les travaux futurs et les améliorations envisageables dans le chapitre 13.

Chapitre 9

Notions importantes

Dans ce chapitre, nous présentons les notions et notations importantes pour l'apprentissage par renforcement. L'ensemble des connaissances abordées dans ce chapitre sont principalement issues des travaux de Suton et Barto [SB18] ainsi que du cours sur l'apprentissage par renforcement proposé par Silver [Sil15].

9.1 Processus de décision markovien

9.1.1 Agent et environnement

Un processus de décision markovien (**Markov Decision Process, MDP**) est un modèle stochastique servant à décrire un problème de décision pour lequel un **agent** évolue dans un **environnement**. L'agent interagit avec l'environnement en exécutant un certain nombre d'**action**, ce qui va modifier l'**état** de l'environnement et procurer à l'agent une certaine **récompense**. La Figure 9.1 illustre la cadre d'un MDP avec l'interaction agent-environnement.

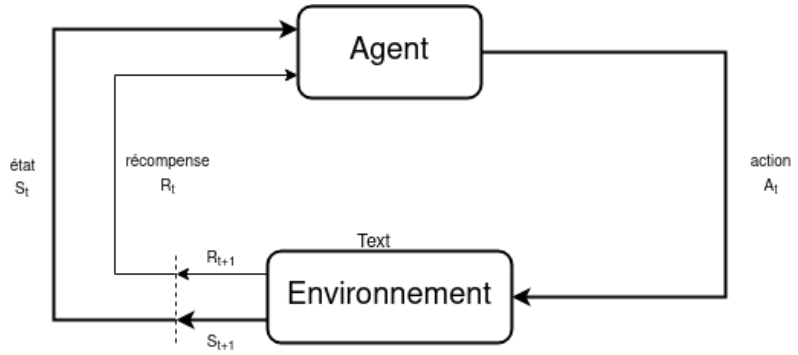


FIGURE 9.1 – Interaction agent-environnement comme décrit par Sutton et Barto [SB18]

À tout instant t, l'agent observe l'état $S_t \in \mathcal{S}$ de l'environnement et décide d'une action $A_t \in \mathcal{A}(s)$ [1]. À l'instant $t+1$, il reçoit une récompense $r_{t+1} \in \mathcal{R}$ et observe

1. On notera indifféremment $\mathcal{A}$ l'ensemble des actions ou $\mathcal{A}(s)$ l'ensemble des actions possibles dans l'état s selon l'usage

de nouveau l'état de l'environnement $S_{t+1} \in \mathcal{S}$. On peut définir une **trajectoire** (ou **épisode**) τ comme une suite de triplets (S_t, A_t, R_{t+1}) de la forme :

$$\tau = (S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...). \tag{9.1}$$

Les valeurs de la récompense et de l'état suivant dépendent directement de l'état actuel de l'environnement et de l'action choisie par l'agent. On peut donc exprimer les variables S_t et R_t par rapport au variables S_{t-1} et A_{t-1} à l'aide d'une loi de probabilité p : $\mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ définie par :

$$p(s', r|s, a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a), \tag{9.2}$$

et ce, pour tout $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$ et $r \in \mathcal{R}$. On appelle P l'opérateur de probabilité et le symbole "|" désigne une probabilité conditionnelle. La loi de probabilité p définie la **dynamique** du modèle MDP. Un triplet (s, a, s') représente une **transition** qui peut être modélisée par la probabilité de transition d'état $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$:

$$p(s'|s, a) = \sum_{r \in \mathcal{R}} p(s', r|s, a). \tag{9.3}$$

On peut aussi définir la récompense attendue par rapport à un couple $(s, a) \in \mathcal{S} \times \mathcal{A}$ par la fonction $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$:

$$r(s, a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a). \tag{9.4}$$

9.1.2 Récompense et Objectif

En formalisant l'interaction agent-environnement par un MDP, on se ramène à un problème de prise de décision où l'agent doit atteindre un certain objectif. Suton et Barto [SB18] énoncent l'hypothèse suivante : « That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward) ». Cette hypothèse traduit le fait que ce problème de prise de décision peut être considéré comme un problème de maximisation des récompenses obtenus au cours du temps (ou de maximisation du **rendement attendu**). Ils introduisent donc la notions de **rendement**, c'est-à-dire la somme des récompenses futures à partir d'un instant t. Dans le cas cas d'un épisode **fini**, le rendement est défini par :

$$G_t = \sum_{k=t+1}^{T} R_k, \tag{9.5}$$

avec T l'instant final. Dans le cas d'un épisode **continu** (**infini**), le rendement est défini par :

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{9.6}$$

où le paramètre $\gamma \in [0, 1[$ est appelé **taux de dépréciation**. Étant donné qu'un épisode est potentiellement infini, le paramètre γ garantie la convergence du rendement G_t. De plus, il détermine l'intérêt des récompenses futures pour l'agent. En effet, si $\gamma = 0$,

l'agent s'intéresse uniquement au gain de la prochaine action. Au contraire, si $\gamma \approx 1$, l'agent devient prévoyant est accordé autant d'importance à l'avenir qu'au présent. Pour avoir une notation unique, que ce soit dans le cas continue ou le cas fini, on utilisera la notation :

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{9.7}$$

où $T \in \mathbf{N}$. Par la suite, on pourra exprimer G_t en fonction de G_{t+1} :

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + ...) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{9.8}$$

9.1.3 Politique, fonction de valeur d'état et fonction de valeur d'état-action

Une politique π est une loi de probabilité qui détermine le comportement d'un agent d'un agent dans un état s. On peut définir une politique de manière déterministe où $\pi : \mathcal{S} \to \mathcal{A}$ renvoie un action a pour un état s, ou bien de manière stochastique où $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ décrit une loi de probabilité telle que :

$$\begin{aligned} &\pi(a|s) = P(A_t = a | S_t = s) \\ \text{et} \\ &\sum_{a \in \mathcal{A}} \pi(a|s) = 1, \ \forall s \in \mathcal{S}. \end{aligned} \tag{9.9}$$

Comme énoncé précédemment, l'objectif de l'agent est de maximiser les récompenses futures (ou rendement attendu). On peut donc définir la fonction de **valeur d'état** v_π qui exprime le rendement moyen attendu par un agent qui suit une politique π :

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_t + G_{t+1} | S_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in \mathcal{S}. \end{aligned} \tag{9.10}$$

On peut aussi définir la fonction de **valeur d'état-action** q_π qui renvoie le rendement attendu moyen à partir d'un couple état-action $(s, a) \in \mathcal{S} \times \mathcal{A}$, en suivant la politique π :

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \sum_{s', r} P(s', r|s, a)[r + \gamma \sum_{s', a'} q_\pi(s', a')], \\ &= \sum_{s', r} P(s', r|s, a)[r + \gamma v_\pi(s')], \ \forall (s, a) \in \mathcal{S} \times \mathcal{A} \end{aligned} \tag{9.11}$$

Comme on le voit, il est possible de définir les fonctions v_π et q_π de manière récursive par rapport aux états présent et futur. Les équations (9.10) et (9.11) sont appelées **équations de Bellman** (respectivement pour v_π et q_π).

On dit qu'une politique π' est meilleure (ou au moins aussi bonne) qu'une politique π si, pour tout $s \in \mathcal{S}$, $v_{\pi'}(s) \geq v_\pi(s)$ et on note $\pi' \geq \pi$. L'objectif de l'apprentissage par renforcement est de déterminer une politique optimale π_* tel que $\pi_* \geq \pi$ pour tout autre politique π c'est-à-dire une politique qui maximise le rendement attendu :

$$\begin{aligned} v_*(s) &= \max_\pi v_\pi(s), \\ q_*(s,a) &= \max_\pi q_\pi(s,a), \\ \pi_*(s) &= \arg\max_\pi v_\pi(s), \forall\ (s,a) \in \mathcal{S} \times \mathcal{A}. \end{aligned} \tag{9.12}$$

On peut exprimer v_* par rapport à q_* :

$$v_*(s) = \max_a q_*(s,a), \forall\ s \in \mathcal{S}. \tag{9.13}$$

Cette équation traduit que le choix de la meilleure action renvoie le meilleur rendement attendu. À partir des équations (9.10) et (9.11), on peut définir les fonctions de valeur d'état et de valeur d'état-action optimales de manière récursive :

$$\begin{aligned} v_*(s) &= \max_a q(s,a), \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')],\ \forall s \in \mathcal{S}, \end{aligned} \tag{9.14}$$

et

$$\begin{aligned} q_*(s,a) &= \max_a Q(s,a), \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')], \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')],\ \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \end{aligned} \tag{9.15}$$

En résumé, l'objectif du RL est de permettre à un agent d'apprendre la meilleure politique, c'est-à-dire le meilleur comportement à adopter afin d'atteindre son objectif. L'objectif se traduit par une récompense qui favorise les actions qui permettent à l'agent de remplir sa mission. Un problème de type MDP est donc un problème de maximisation de la récompense reçue via la recherche de la politique optimale. En théorie, la résolution de tout problème de type MDP est possible, mais en pratique, la complexité provient de l'exploration d'un espace potentiellement infini ou d'un environnement partiellement inconnu, ce qui rend compliqué la recherche d'une solution optimale. Il est néanmoins possible d'estimer une politique via des méthodes itératives qui repose principalement sur les notions abordées dans cette section, ainsi que sur les notions abordées dans la prochaine section. Une règle de mise à jour itérative couramment utilisée pour un estimateur E est la suivante :

$$E_n \leftarrow E_{n-1} + \eta[C - E_{n-1}], \tag{9.16}$$

où E_n et E_{n-1} sont les valeurs de l'estimateur aux étapes n et $n-1$, C la valeur cible (valeur observée du rendement par exemple) et le paramètre η, appelé taux d'apprentissage, indique l'importance de l'erreur $[C - E_{n-1}]$ pour la mise à jour de l'estimateur E.

9.2 Les méthodes d'estimation tabulaires

Dans cette section, nous abordons le sujet des méthodes d'estimation dites **tabulaire**. On cherche à approcher une politique π pour un problème de type MDP à partir des fonctions de valeur d'état v_π et de valeur d'état-action q_π. On parle d'estimation tabulaire puisque qu'on approche la fonction v_π par un vecteur $V \in \mathbb{R}^{|\mathcal{S}|}$, appelé estimateur V ou table des valeurs d'état (ou V-table), tel que :

$$v_\pi(s) \approx V(s), \tag{9.17}$$

et ce pour tout $s \in \mathcal{S}$. Pour ce qui est de la fonction d'état-action, on approche q_π par une matrice $Q \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$, appelée estimateur Q ou table des valeurs d'état-action (ou Q-table), telle que :

$$q_\pi(s,a) \approx Q(s,a), \tag{9.18}$$

pour tout couple $(s,a) \in \mathcal{S} \times \mathcal{A}$.

9.2.1 Programmation dynamique

La **programmation dynamique** (**Dynamic Progaming**, **DP**) est une méthode d'optimisation qui permet de résoudre certains problèmes de planification ou de prise de décision. Ces problèmes doivent avoir deux propriétés : 1) ils peuvent être décomposés en sous-problèmes et 2) les solutions de ces sous-problèmes peuvent être utilisées pour trouver une solution optimale au problème principal. Dans le cas d'un MDP, on se sert des équations de Bellman (9.10) et (9.11) pour trouver une politique optimale de façon récursive à l'aide d'un algorithme appelé **itération** de la politique (policy iteration). Cet algorithme se divise en deux parties qui sont l'**évaluation** et l'**amélioration** de la politique.

Dans le cas de la programmation dynamique, on dispose d'une connaissance complète de l'environnement et donc de la dynamique du MDP : $p(s', r|s, a)$ est connue pour toutes valeurs de $s, s' \in \mathcal{S}, a \in \mathcal{A}$ et $r \in \mathcal{R}$. De plus, on se place dans le cas d'une politique déterministe, c'est-à-dire que la politique π est une fonction qui associe une action à chaque état ($\pi(s) = a$).

Dans un premier temps, on cherche à évaluer une politique π connue en estimant la fonction v_π. Pour estimer v_π, on définit la suite de fonctions de valeur d'état $\{v_0, v_1, v_2, ...\}$ avec $v_k : S \to \mathbb{R}$ pour tout $k \in \mathbb{N}$. Dans le cas $k = 0$, $v_0(s)$ est définie arbitrairement pour tout $s \in \mathcal{S}$ (p. ex. $v_0(s) = 0$). On définit ensuite la fonction v_k par rapport à v_{k-1} pour tout $k \geq 1$ à partir de l'équation (9.10) :

$$v_{k+1}(s) = \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')], \forall s \in \mathcal{S}. \tag{9.19}$$

De par sa définition, et parce-que $\gamma < 1$, la convergence de la suite $\{v_k\}$ vers v_π est garantie.

Dans un second temps, on va chercher à définir une meilleure politique π' à partir de la politique π. Pour cela, on utilise le théorème de l'amélioration de la politique (**Policy Improvement Theorem**) qui nous dit que s'il existe une politique π' telle

que $q_\pi(s, \pi'(s)) \geq v_\pi(s)\ \forall s \in \mathcal{S}$ alors $\pi' \geq \pi$. En choisissant π' tel que pour tout $s \in \mathcal{S}$, $\pi'(s) = \arg\max_a q_\pi(s, a)$, alors π' respectent ce théorème.

Enfin, l'algorithme de l'itération de la politique consiste à alterner les phases d'évaluation et d'amélioration afin de trouver une politique optimale π_* :

Algorithme 2 Algorithme de l'itération de la politique

Paramètres :
$\theta \in \mathbb{R}_+$, un seuil pour évaluer l'estimation de v_π
Initialisation :
$V(s) \in \mathbb{R},\ \forall s \in \mathcal{S}$ (p.ex. $V(s) = 0$)
$\pi(s) \in \mathcal{A},\ \forall s \in \mathcal{S}$
$\Delta \leftarrow 0$

1 - Évaluation de la politique π
Boucle : tant que $\Delta > \theta$:
$\Delta \leftarrow 0$
Boucle : pour chaque état $s \in \mathcal{S}$:
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

2 - Amélioration de la politique π
$stable = True$
Boucle : pour chaque état $s \in \mathcal{S}$:
$a_{-1} \leftarrow \pi(s)$
$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
Si $a_{-1} \neq \pi(s)$, alors $stable \leftarrow False$
Si $stable = True$:
Renvoyer $V \approx v_*$ et $\pi \approx \pi_*$
Sinon :
Reprendre à **1 - Évaluation de la politique**

9.2.2 Les méthodes Monte-Carlo

Une des limites de la programmation dynamique est que cette méthode nécessite une connaissance totale de l'environnement. Quand l'environnement est partiellement connu, c'est-à-dire qu'il existe au moins un quadruplet $(s, a, r, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{R} \times \mathcal{S}$ tel que la probabilité $p(s', r|s, a)$ soit inconnue, on peut se tourner vers des méthodes dîtes de **Monte-Carlo** (**MC**). Ces méthodes permettent d'estimer une quantité inconnue g à partir de d'une suite de variables aléatoire (X_N) avec $N \geq 1$ tel que $g = \mathbb{E}(X)$. Dans le cas de l'apprentissage par renforcement, on cherche à estimer une politique π à l'aide des fonctions de valeur d'état et d'état-action v_π et q_π qui sont définies par :

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s], \\ q_\pi(s, a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a], \end{aligned} \tag{9.20}$$

et ce pour tout couple $(s,a) \in \mathcal{S} \times \mathcal{A}$. Ainsi, quand on dispose d'une connaissance partielle de l'environnement, on peut estimer v_π et q_π en générant plusieurs épisodes et en calculant la valeur de G_t. Dans un premier temps, on se place dans le cas d'une politique déterministe. On présente ci-dessous une première approche appelée algorithme "**MC Première-Visite**", où on génère $K \in \mathbb{N}$ épisodes de la forme :

$$\tau_k = \{S_0, A_0, R_1, S_1, A_1, ..., S_{T-1}, A_{T-1}, R_T\}_\pi, . \tag{9.21}$$

où $T \in \mathbb{N}$ correspond à l'étape finale de l'épisode. Comme le nom de l'algorithme l'indique, on s'intéresse à la première apparition d'un état s au sein d'un épisode. Pour tout $k \in \{1, 2, ..., K\}$ et $s \in \mathcal{S}$, si $s \in \tau_k$ alors on note $t_0(s,k) \in \{0, 1, ..., T\}$ l'indice de première apparition de s dans l'épisode τ_k. On note N_s le nombre d'épisodes où l'état s apparaît. On estime donc la fonction v_π comme suit :

$$\begin{aligned} v_\pi(s) &= \frac{1}{N_s} \sum_{k' \in \{k | s \in \tau_k\}} G_{t_0(s,k)} \\ &= \frac{1}{N_s} \sum_{k' \in \{k | s \in \tau_k\}} \sum_{l=t_0(s,k')}^{T} \gamma^{l-t_0(s,k')-1} R_l, \forall s \in \mathcal{S} \end{aligned} \tag{9.22}$$

Algorithme 3 Algorithme de Monte-Carlo : Première Visite

Entrée : une politique π
Initialisation :
$V(s) \in \mathbb{R}\ \forall\ s \in \mathcal{S}$ (pr. e. V(s) = 0)
$RdmCum(s) \leftarrow 0\ \forall s \in \mathcal{S}$
$N(s) \leftarrow 0\ \forall s \in \mathcal{S}$
Initialisation :
$\gamma \in [0, 1[$, le taux de dépréciation
Boucle : pour chaque épisode $k = 1, ..., K$:
Générer un épisode $\tau_k = \{S_0, A_0, R_1, S_1, A_1, ..., S_{T-1}, A_{T-1}, R_T\}_\pi$
$G \leftarrow 0$
Boucle : pour chaque étape $t = T-1, T-2, ...0$:
$G \leftarrow \gamma G + R_{t+1}$
Si $S_t \notin \{S_0, S_1, ...S_{t-1}\}$ c.a.d l'état S_t apparaît pour la première fois :
$RdmCum(s) \leftarrow RdmCum(s) + G$
$N(s) \leftarrow N(s) + 1$
Boucle : pour chaque état $s \in \mathcal{S}$:
$V(s) \leftarrow \frac{RdmCum(s)}{N(s)}$

A partir de cet algorithme, il est aussi possible d'estimer la fonction de valeur d'état-actions q_π. Cependant, certains couples (s,a) peuvent ne pas apparaître au cours de l'apprentissage. Une solution pour pallier ce problème est de se tourner vers une politique stochastique. En particulier, on peut définir les deux politiques appelées ϵ-**soft** et ϵ-**greedy**. Soit $\epsilon \in [0, 1]$, on dit que π est ϵ-soft si, pour tout couple $(s,a) \in \mathcal{S} \times \mathcal{A}(s)$:

$$\pi(a|s) \geq \frac{\epsilon}{|\mathcal{A}(s)|}, \tag{9.23}$$

et on dit que π est ϵ-greedy si, pour tout couple $(s,a) \in \mathcal{S} \times \mathcal{A}(s)$:

$$\pi(a|s) = \begin{cases} (1-\epsilon) + \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{si } a = \arg\max_{a'} q_\pi(s,a') \\ \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{sinon.} \end{cases} \tag{9.24}$$

Dans la cas de la politique ϵ-greedy, si $\epsilon = 0$, on retrouve la politique greedy définie pour le cas déterministe, et si $\epsilon = 1$, on obtient une politique uniforme où $\pi(a|s) = \frac{\epsilon}{|\mathcal{A}(s)|}$ pour tout $s \in \mathcal{S}$. On peut maintenant améliorer l'algorithme MC première visite afin qui servent aussi à évaluer une politique π en plus d'estimer la fonction de valeur d'état-action q_π.

Algorithme 4 Algorithme de Monte-Carlo : Première Visite (ϵ-soft)

Paramètres :
$\epsilon \in [0,1]$
$\gamma \in [0,1[$, le taux de dépréciation
Initialisation :
une politique ϵ-soft π
$Q(s,a) \in \mathbb{R}, \forall (s,a) \in \mathcal{S} \times \mathcal{A}(s)$ (p. ex. $Q(s,a) = 0$)
$RdmCum(s,a) \leftarrow 0, \forall (s,a) \in \mathcal{S} \times \mathcal{A}(s)$
$N_{(s,a)} \leftarrow 0 \forall (s,a), \in \mathcal{S} \times \mathcal{A}(s)$
Boucle : pour chaque épisode $k = 1, \ldots, K$:
Générer un épisode $\tau_k = \{S_0, A_0, R_1, S_1, A_1, ..., S_{T-1}, A_{T-1}, R_T\}_\pi$
$G \leftarrow 0$
Boucle : pour chaque étape $t = T-1, T-2, ...0$:
$G \leftarrow \gamma G + R_{t+1}$
Si $(S_t, A_t) \notin \{(S_0, A_0), (S_1, A_1), ..., (S_{t-1}, A_{t-1})\}$:

$$\begin{aligned} RdmCum(S_t, A_t) &\leftarrow RdmCum(S_t, A_t) + G \\ N_{(S_t,A_t)} &\leftarrow N_{(S_t,A_t)} + 1 \\ Q(S_t, A_t) &\leftarrow \tfrac{RdmCum(S_t,A_t)}{N_{(S_t,A_t)}} \end{aligned}$$

Boucle : pour chaque action $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} (1-\epsilon) + \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{si } a = \arg\max_{a'} q(s,a') \\ \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{sinon} \end{cases}$$

9.2.3 Différence temporelle

La méthode dite **différence temporelle** (**Temporal Difference**, **TD**) combine les avantages de la programmation dynamique et des méthodes de Monte Carlo. En effet, avec la différence temporelle, on estime les fonctions v_π et q_π de manière itérative à partir d'épisodes générés pour l'apprentissage, donc sans connaître la dynamique de l'environnement. En reprenant la règle de mise à jour (9.16) et en l'appliquant aux méthodes de Monte Carlo pour estimer v_π, on obtient :

$$V(S_t) \leftarrow V(S_t) + \eta(G_t - V(S_t)), \tag{9.25}$$

où V est un estimateur de v_π, et le paramètre $\eta \in]0, 1]$ est le taux d'apprentissage (p. ex. $1/N_s$ dans le cas MC première visite). On observe que cette estimation nécessite de connaître G_t donc de terminer un épisode, ce qui implique certaines contraintes (durée d'un épisode, épisode qui ne se termine pas). Dans le cas des méthodes TD, on utilise seulement les n prochaines étapes pour estimer v_π. La forme générale de la formule de mise à jour pour la méthode **TD(n)** est donnée par :

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \eta[R_{t+1} + \gamma R_{t+2} + ... + \gamma^n R_{t+n+1} + \gamma^{n+1} V(S_{t+n+1}) - V(S_t)], \\ &= V(S_t) + \eta[\sum_{k=0}^{n} \gamma^k R_{t+k+1} + \gamma^{n+1} V(S_{t+n+1}) - V(S_t)] \\ &= V(S_t) + \eta\delta_t, \end{aligned} \tag{9.26}$$

où la quantité δ_t est appelée **erreur TD**.

9.2.4 La méthode TD(0)

La méthode **TD(0)** est un cas particulier de la méthode TD où l'on s'intéresse seulement à l'étape actuelle et à l'étape suivante pour mettre à jour l'estimateur V :

$$V(S_t) \leftarrow V(S_t) + \eta(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)). \tag{9.27}$$

Soit π, une politique qu'on cherche à approcher et b, une politique suivie par l'agent (politique dite **comportementale**). On cherche à estimer la fonction de valeur d'état-action q_π à l'aide de la méthode TD(0). On peut exprimer la règle (9.16) pour la mise à jour de l'estimateur Q comme suit :

$$Q(S_t, b(S_t)) \leftarrow Q(S_t, b(S_t)) + \eta[R_{t+1} + \gamma Q(S_{t+1}, \pi(S_{t+1})) - Q(S_t, b(S_t)]. \tag{9.28}$$

Dans le cas où $b = \pi$, on dit que la méthode d'apprentissage est **on-policy**, c'est-à -dire que l'agent suit la politique qu'on cherche à approcher. Au contraire, si $b \neq \pi$, alors la méthode d'apprentissage est dite **off-policy** et on cherche à approcher π à l'aide d'une politique différente. L'algorithme 5 montre l'implémentation de la méthode d'apprentissage TD(0) dans les cas on-policy et off-policy pour l'estimation de q_π.

Parmi les algorithmes d'apprentissage, il en existe deux particulièrement connus qui dérivent de la méthode TD(0) : l'algorithme **SARSA** et l'algorithme **Q-learning**. Dans les deux cas, on cherche à estimer la fonction de valeur d'état-action q_π à l'aide de l'estimateur Q et de la règle de mise à jour (9.28). Néanmoins, SARSA est une méthode on-policy alors que Q-learning est une méthode off-policy.

Q-learning

Au cours de nos travaux, nous nous sommes particulièrement intéressés à l'algorithme Q-learning 6. Il s'agit d'un algorithme de type off-policy où l'agent suit une

Algorithme 5 TD(0) on-policy/off-policy pour l'estimation de q_π

Entrée :
une politique π
Si : dans le cas on-policy :
une politique $b = \pi$
Sinon : dans le cas off-policy :
une politique $b \neq \pi$
Initialisation :
$Q(s, a) \in \mathbb{R} \; \forall \; (s, a) \in \mathcal{S} \times \mathcal{A}$ (pr. ex. $Q(s, a) = 0$)
Paramètres :
$\eta \in]0, 1]$, le taux d'apprentissage
$\gamma \in [0, 1[$, le taux de dépréciation
Boucle : pour chaque épisode $k = 1, \ldots K$:
Initialiser l'épisode : $S \leftarrow S_0$
Boucle : pour chaque étape jusqu'à $S = S_T$:
$A \leftarrow b(S)$
Exécuter l'action A, observer R, S'
$Q(S, A) \leftarrow Q(S, A) + \eta[R + \gamma Q(S', \pi(S')) - Q(S, A)]$
$S \leftarrow S'$

politique b différente de la politique π qu'on cherche à estimer. La règle de mise à jour de l'estimateur Q avec la méthode d'apprentissage Q-learning est donnée par :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)], \tag{9.29}$$

où A_t est choisie selon la politique b (par exemple, ϵ-greedy) alors que A_{t+1} est choisie selon la politique π (qui est greedy dans le cas du Q-learning).

Algorithme 6 Q-learning pour l'estimation de q_π

Initialisation :
$Q(s, a) \in \mathbb{R} \; \forall \; (s, a) \in \mathcal{S} \times \mathcal{A}$ (pr. ex. $Q(s, a) = 0$)
Paramètres :
$\eta \in]0, 1]$, le taux d'apprentissage
$\gamma \in [0, 1[$, le taux de dépréciation
$\epsilon \in]0, 1]$, pour une politique ϵ-greedy
Boucle : pour chaque épisode $k = 1, \ldots K$:
Initialiser l'épisode : $S \leftarrow S_0$
Boucle : pour chaque étape de l'épisode jusqu'à $S = S_T$:
Choisir A à partir de Q (p. ex. politique ϵ-greedy)
Prendre l'action A, observer R, S'
$Q(S, A) \leftarrow Q(S, A) + \eta[R + \gamma Q(S', A') - Q(S, A)]$
$S \leftarrow S'$

Double Q-learning

Un problème majeur lié à l'algorithme Q-learning est que l'estimateur Q possède un biais positif [TS99]. Ce biais peut entraîner une surestimation de certaines valeurs $Q(s,a)$ et donc altérer le comportement optimal de l'agent. Pour éviter ce biais, H. van Hasselt propose un nouvel algorithme d'apprentissage appelé **Double Q-learning** (alg. 7). Cet algorithme est une version améliorée de la méthode Q-learning et repose sur l'utilisation de deux estimateurs Q_1 et Q_2. Les deux estimateurs sont mis à jours comme suit :

$$\begin{aligned} &Q_1(S_t,A_t) \leftarrow Q_1(S_t,A_t) + \eta[R_t + \gamma Q_2(S_{t+1}, \arg\max_a Q_1(S_{t+1},a)) - Q_1(S_t,A_t)] \\ \text{et} \\ &Q_2(S_t,A_t) \leftarrow Q_2(S_t,A_t) + \eta[R_t + \gamma Q_1(S_{t+1}, \arg\max_a Q_2(S_{t+1},a)) - Q_2(S_t,A_t)]. \end{aligned} \tag{9.30}$$

On remarque que l'estimateur Q_1 (respectivement Q_2) est mis à jour en utilisant une politique greedy dérivé de Q_2 (respectivement Q_1). C'est cette modification de la valeur cible C qui permet d'obtenir un estimateur non biaisé positivement, sous réserve que Q_1 et Q_2 soit mis à jour au cours d'épisodes différents. Cet algorithme à aussi l'avantage de converger plus vite que l'algorithme Q-learning.

Algorithme 7 Double Q-learning pour l'estimation de q_π

Initialisation :
  $Q_1(s,a), Q_2(s,a) \in \mathbb{R}\ \forall\ (s,a) \in \mathcal{S} \times \mathcal{A}$ (par exemple $Q_i(s,a) = 0$)
Paramètres :
  $\eta \in]0,1]$, le taux d'apprentissage
  $\gamma \in [0,1[$, le taux de dépréciation
  $\epsilon \in]0,1]$, pour une politique ϵ-greedy
Boucle : pour chaque épisode $k = 1, \dots K$:
  Initialiser l'épisode : $S \leftarrow S_0$
  Boucle : pour chaque étape de l'épisode jusqu'à $S = S_T$:
    Choisir A à partir de Q_1 et Q_2 (p. ex. politique ϵ-greedy)
    Prendre l'action A, observer R, S'
    Choisir l'estimateur à mettre à jour entre Q_1 et Q_2 (p. ex. de manière aléatoire)
    Si Q_1 alors :
      $Q_1(S,A) \leftarrow Q_1(S,A) + \eta[R + \gamma Q_2(S, \arg\max_a Q_1(S',a)) - Q_1(S,A)]$
    Sinon :
      $Q_2(S,A) \leftarrow Q_2(S,A) + \eta[R + \gamma Q_1(S, \arg\max_a Q_2(S',a)) - Q_2(S,A)]$
    $S \leftarrow S'$

9.3 Les méthodes du gradient de la politique

Après les méthodes tabulaires, nous abordons les méthodes dites du **gradient de la politique** (**policy gradient**). Avec ces méthodes, on cherche à estimer directement une politique π à partir d'une fonction paramétrée f telle que :

$$f(s, a; \theta) = P(A_t = a | S_t = s, \theta_t = \theta). \tag{9.31}$$

Ainsi, la fonction f décrit la probabilité de prendre l'action $a \in \mathcal{A}$ dans l'état $s \in \mathcal{S}$ à l'instant t. Le vecteur $\theta = (\theta_1, \theta_2, ..., \theta_d)^t \in \mathbb{R}^d$ est un vecteur de paramètres pour la fonction f. Dans la suite, on notera $\pi_\theta(a|s) = f(s, a; \theta)$. Comme dans le cas tabulaire, le but est de de maximiser les performances de l'agent donc de trouver une meilleure politique π_{θ_*} telle que $\pi_{\theta_*} \geq \pi_\theta$ pour tout vecteur de paramètre $\theta \in \mathbf{R}^d$. Pour ce faire, on introduit la fonction de performance (ou fonction de score) $J(\theta)$ qui détermine l'efficacité d'une politique π_θ. Par exemple, la performance $J(\theta)$ peut être définie comme le rendement attendu à partir de l'état s_0 en suivant la politique π_θ :

$$J(\theta) = v_{\pi_\theta}(s_0). \tag{9.32}$$

Ainsi, déterminer la meilleure politique revient à maximiser la fonction de performance J :

$$J(\theta_*) = \max_\theta J(\theta). \tag{9.33}$$

Pour ce faire, on utilise l'algorithme de la monté de gradient 8. Cet algorithme permet de trouver le maximum d'une fonction de manière itérative. Dans notre cas, on cherche le paramètre θ qui maximise J en appliquant la règle de mise à jour suivante :

$$\theta_{t+1} = \theta + \eta \nabla J(\theta), \tag{9.34}$$

$$\nabla J(\theta) = \left[\frac{\partial J(\theta)}{\partial \theta_i} \right]_{i \in \{1,...,d\}}. \tag{9.35}$$

Algorithme 8 Montée de gradient

Entré
$J(\theta)$ une fonction de performance différentiable
Initialisation :
$\theta \in \mathbb{R}^d$ (p. ex. 0 ou aléatoire)
$\Delta \leftarrow +\infty$
Paramètres :
$\eta \in]0, 1]$, le taux d'apprentissage
ϵ très petit
Boucle : tant que $\Delta \geq \epsilon$:
$J_{-1} \leftarrow J(\theta)$
$\theta \leftarrow \theta + \eta \nabla J(\theta)$
$\Delta \leftarrow |J(\theta) - J_{-1}|$

Chapitre 10

Cadre de l'apprentissage

Un des objectifs de nos travaux est de créer un agent autonome qui peut modifier des fichiers PE malveillants afin qu'ils ne soient plus reconnus comme tel par des solutions de détection. Dans ce chapitre, nous présentons le cadre (framework) utilisé pour l'évasion de malware à l'aide de l'apprentissage par renforcement. Pour rappel, un problème RL peut être décrit par un MDP, c'est-à-dire un problème de prise de décisions où un agent interagit avec un environnement afin d'atteindre un objectif particulier. La figure 10.1 illustre notre problématique, et comme on peut le voir, le cadre de l'apprentissage doit répondre à certains critères. Tout d'abord, l'agent doit pouvoir interagir avec l'environnement en effectuant plusieurs actions (a_t). Ensuite, l'environnement doit renvoyer deux informations essentielles qui sont une récompense (r_{t+1}) et l'état de l'environnement (s_{t+1}). L'environnement est constitué d'une solution de détection, que l'agent doit contourner, et d'une base de données de fichiers malveillants, utilisée pour l'entraînement et l'évaluation de nos différents agents. Dans la suite du chapitre, nous détaillons les différents agents utilisés, les actions qu'ils peuvent effectuer ainsi que la façon dont nous avons construit l'environnement.

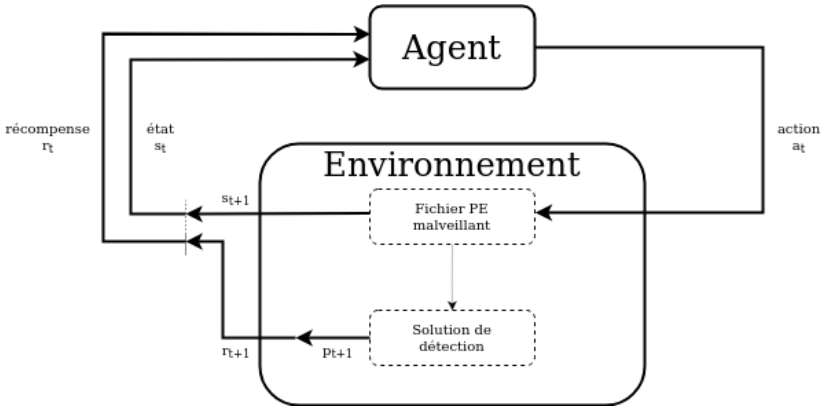


FIGURE 10.1 – Cadre de notre problématique

10.1 Environnements

Comme on peut le voir dans la figure 10.1, l'environnement est la partie du framework avec laquelle l'agent interagit et sur laquelle il a une influence. L'environnement

est constitué d'une solution de détection, que nous cherchons à tromper, et d'une base de données de fichiers malveillants, qui serviront pour l'entraînement de l'agent. Les fichiers utilisés pour entraîner et tester nos modèles d'évasions sont issus de la base de données BODMAS [Yan+21] qui contient 57293 instances. Ces fichiers ont été récoltés entre le mois d'août 2019 et le mois de septembre 2020, ce qui en fait une base de données relativement récente. Pour ce qui est des solutions de détection, nous utilisons trois modèles d'intelligence artificielle qui sont EMBER [AR18a], Malconv [Raf+17] et Grayscale [MQC22]. Ces modèles ne sont pas considérés comme "boîte blanche" [Loy19] car leurs prédictions sont difficilement interprétables et explicables. Néanmoins, ces solutions permettent d'avoir accès à certaines informations utiles pour construire un environnement adapté à notre problématique. Nous avons aussi eu l'occasion de travailler avec un antivirus (AV) classique totalement opaque. Nous avons dû revoir notre environnement pour s'adapter à cette solution de détection.

10.1.1 Solutions de détection basées sur l'IA

Dans cette section, nous décrivons comment nous avons construit notre environnement pour y intégrer des solutions de détection basées sur de l'IA. Comme abordé dans la partie I, un modèle de machine learning peut être vu comme une fonction $f : \mathcal{X} \to \mathcal{Y}$, où $\mathcal{X}$ est l'espace des variables et $\mathcal{Y}$ est l'espace des labels. De plus, il est souvent nécessaire d'avoir recours à une fonction de pré-traitement ϕ qui transforme les données brutes dans une forme plus usuelle. Par exemple, dans notre cas, nous cherchons une fonction de pré-traitement $\phi : U \to \mathcal{X}$ où U est l'ensemble des fichiers PE. Par la suite, nous noterons $f(x) = f(\phi(u))$ avec $u \in U$ et $x \in \mathcal{X}$.

Au cours de nos travaux, nous avons ciblé trois solutions de détection qui sont EMBER, Malconv et Grayscale. Avec ces solutions, nous avons accès à certaines informations relativement utiles. Pour commencer, quand un fichier u est analysé, le modèle de détection renvoie une prédiction (ou score de détection) $p = f(x) \in [0, 1]$ ainsi qu'un label $l \in \{0, 1\}$. Si $l = 1$, le fichier est malveillant, sinon il est bénin. En général, le label est défini à partir du score de détection tel que :

$$l = \begin{cases} 1 & \text{si } p \geq \psi \\ 0 & \text{sinon,} \end{cases} \tag{10.1}$$

où $\psi \in]0, 1[$ est un seuil qui dépend du modèle. Comme énoncé plus haut, l'environnement doit renvoyer une récompense et un état à l'agent pour qu'il puisse apprendre. Pour commencer, nous pouvons définir la fonction de récompense comme suit :

$$r_t = r(p_t, p_{t-1}) = \begin{cases} R \in \mathbb{R} & \text{si } p_t \leq \psi \\ p_{t-1} - p_t & \text{sinon,} \end{cases} \tag{10.2}$$

où p_t et p_{t-1} sont les prédictions du modèle ciblé aux instant t et $t - 1$, et où R est un hyperparamètre de notre cadre d'apprentissage. Il s'agit de la récompense en cas de succès de l'agent (c'est-à-dire, si le modèle ne détecte plus le fichier comme malveillant). Ensuite, pour chaque solution, nous connaissons la fonction ϕ associée, donc nous pouvons définir l'état s renvoyé par l'environnement de la manière suivante :

$$s_t = \phi(u_t), \tag{10.3}$$

où u_t désigne le fichier u après avoir été modifié t fois pas l'agent ($u_0 = u$). En faisant ce choix, nous pouvons facilement adapter notre environnement à de nouvelles solutions de détection tant que la fonction ϕ est connue. Nous résumons les informations connues et utiles dans la table 10.1.

TABLE 10.1 – Information sur les solutions de détection basées sur l'IA

	EMBER [AR18b]	Malconv [Raf+17]	Grayscale [MQC22]
Type de modèle	LGBM	DNN	CNN
Type de données	Variables extraites du fichier PE	Séquences d'octets	Fichier PE transformé en image
Taille de l'état s [1]	2381	≈1M	64×64
Seuil ψ	0.8336	0.50	0.50

10.1.2 Solution de détection classique

Au cours de nos travaux, nous avons pu travailler avec un antivirus classique. À la différence des modèles basées sur l'IA utilisés précédemment, les informations renvoyées par l'AV sont très limitées. La première contrainte a été que l'AV ne renvoie pas de score de détection p, mais seulement un label $l \in \{0, 1\}$. Cela nous a obligé à modifier la fonction de récompense (10.2) afin d'adapter l'environnement :

$$r_t = r(l_t) = \begin{cases} R \in \mathbb{R} & \text{si } l_t = 0 \\ 0 & \text{sinon,} \end{cases} \tag{10.4}$$

où l_t est le label du fichier modifié renvoyé par l'AV à l'instant t. La deuxième contrainte est que nous ne savons pas quelles informations sont extraites des fichiers soumis au logiciel antivirus, ni quel type d'analyse il effectue (signature, comportementale, heuristique). Par rapport aux solutions ML, nous ne disposons pas d'une fonction ϕ associée à cet outil, en supposant qu'il en existe une, l'agent ne peut donc pas observer l'environnement. Pour pallier ce manque, on choisit d'utiliser la fonction ϕ de EMBER. Ainsi, nous disposons d'une représentation (partielle) du fichier qu'on cherche à modifier et notre agent peut apprendre à tromper l'AV malgré ces limites techniques.

10.2 Actions

Dans cette partie, nous listons les actions que notre agent peut choisir pour modifier un fichier PE, sous contrainte de ne pas corrompre le fichier et de ne pas changer son fonctionnement. Nous utilisons, entre autres, les actions introduites par Anderson et al. [AFR17] dans leurs travaux. Toutes les actions sont listées dans la table 10.2. En particulier, notre agent doit pouvoir :

– compresser et décompresser un fichier PE,

1. correspond à la taille du vecteur extrait du fichier

– ajouter des chaînes de caractères (issues de fichiers bénins) à la fin d'une section,
– ajouter des octets de manière aléatoire dans les espaces non utilisés,
– ajouter des fonctions à la table des imports,
– modifier la date de création du fichier ("timestamp").

Les modifications effectuées sur un fichier PE doivent être cohérentes et légitimes, nous avons donc extrait des informations pertinentes à partir de 5000 fichiers bénins (nom des sections, imports, chaînes de caractères, etc.). Nous pouvons ensuite utiliser ces informations quand certaines actions le nécessitent et, à terme, nous gardons les modifications les plus efficaces.

TABLE 10.2 – Table des actions (indexées à partir de 0)

Actions	Index
modify machine type	0
pad overlay	1
append benign data overlay	2
append benign binary overlay	3
add bytes to section cave	4
add section strings	5
add section benign data	6
add strings to overlay	7
add imports	8
rename section	9
remove debug	10
modify optional header	11
modify timestamp	12
break optional header checksum	13
upx unpack	14
upx pack	15

La librairie python LIEF [Tho17] est une librairie utilisée pour l'analyse de fichiers exécutables tels que les fichiers PE. Cette librairie permet aussi de modifier un fichier PE, mais avec quelques limitations. Comme mentionné par SONG et al. [Son+20], il peut arriver que l'utilisation de la librairie LIEF corrompe l'exécutable en affectant ou empêchant son fonctionnement. Ils proposent comme solution d'analyser le fichier dans une sandbox (Cuckoo) afin de déterminer si les modifications faites avec LIEF n'ont pas corrompu le fichier. Pendant nos expérimentations, nous utilisons une méthode similaire. Après avoir généré un malware non détecté par une des solutions testées, nous vérifions que le logiciel est toujours fonctionnel en l'examinant dans une solution interne qui nous permet de voir s'il a gardé son comportement original. Nous utilisons aussi la plateforme ANY.RUN qui génère des graphes comportementaux et permet donc de comparer le nouveau fichier et l'original. Nous présentons ce genre de graphe dans la figure 10.2 pour une version de Wannacry modifiée avec notre agent. Si le nouveau fichier est corrompu ou ne s'exécute pas, nous considérons l'entraînement comme un échec, ce qui ne modifie pas significativement notre agent.

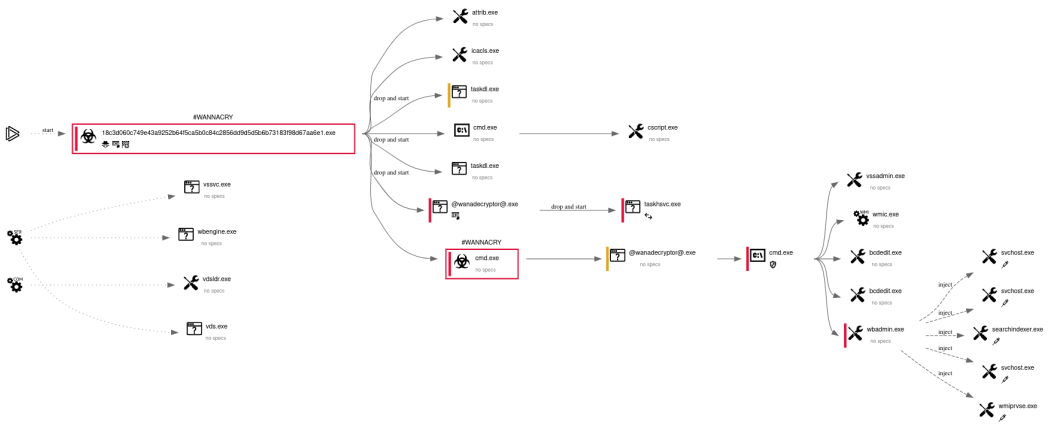


FIGURE 10.2 – Graphe comportemental d'une version modifiée de du malware Wannacry, issu de ANY.RUN

10.3 Agents

Maintenant que l'environnement a été mis en place et que les actions ont été définies, nous devons créer un agent capable de modifier des fichiers PE afin de les rendre indétectables par différentes solutions de détection. D'un point de vue technique, un agent est simplement un programme optimisé pour prendre les meilleures décisions selon la situation (ou l'état de l'environnement). Dans le chapitre 9, nous avons présenté plusieurs algorithmes d'apprentissage par renforcement qui permettent d'optimiser un agent RL. Ces algorithmes sont adaptés pour des espaces d'états et d'actions de petite taille. Dans notre cas, l'espace des actions a une taille convenable ($|\mathcal{A}| = 16$), par contre, l'espace des états peut être considéré de taille infinie. En effet, en prenant l'exemple du modèle EMBER, l'espace des états est $\mathcal{S} = \mathbb{R}^{2381}$, donc contient un très grand nombre d'états différents. Dans ce type de cas, les algorithmes RL classiques ne suffisent plus et nous devons nous tourner vers des méthodes dîtes "profondes". Avec ces méthodes, nous cherchons à modéliser une politique ou une fonction de valeurs d'état-action à l'aide d'un réseau de neurones. Dans nos travaux, nous utilisons deux algorithmes de RL profonds qui sont **Double Q-learning profond** et **REINFORCE profond**.

10.3.1 Double Q-learning profond

Dans le chapitre précédent, nous avons présenté l'algorithme Double Q-learning comme une méthode tabulaire pour estimer la fonction de valeur d'état-action q_π à l'aide de deux estimateurs Q_1 et Q_2. Dans le cas profond, introduit par Hasselt et al. [HGS16], ces deux estimateurs sont remplacés par deux réseaux de neurones profonds notés Q_θ et $Q_{\theta'}$, où θ, $\theta' \in \mathbb{R}^d$ correspondent aux poids des réseaux de neurones. Q_θ est appelé "réseau principal" et $Q_{\theta'}$ est appelé "réseau cible".

Dans le cas profond, ce sont les poids du modèle Q_θ qui sont mis à jour de manière itérative à l'aide de l'algorithme de la descente de gradient. Soit e_i un épisode de la forme (s, a, s', r). On définit la valeur cible y_i par :

$$y_i = r + \gamma Q_{\theta'}(s', \arg\max_{a'} Q_\theta(s', a')), \tag{10.5}$$

et la valeur prédite q_i par :

$$q_i = Q_\theta(s, a). \tag{10.6}$$

On note $Y = \{y_i\}_{\{i=1,\dots,N\}}$, l'ensemble des valeurs cibles et $Q = \{q_i\}_{\{i=1,\dots,N\}}$, l'ensemble des valeurs prédites. Pour optimiser le modèle Q_θ, on définit la fonction de perte $L(\theta)$ qu'on cherche à minimiser. Ici, il s'agit de l'erreur quadratique moyenne (MSE) :

$$\begin{aligned} L(\theta) &= MSE(Y, Q; \theta) \\ &= \mathbb{E}[(Y - Q)^2] \\ &= \frac{1}{N}\sum_{i=1}^{N}(y_i - q_i)^2 \end{aligned} \tag{10.7}$$

Ainsi, dans le cas profond, la règle de mise à jour pour les poids θ est données par :

$$\begin{aligned} \theta &\leftarrow \theta - \eta\nabla L(\theta) \\ &= \theta - \eta\nabla\frac{1}{N}\sum_{i=1}^{N}(y_i - q_i)^2 \end{aligned} \tag{10.8}$$

Les poids du modèle cibles $Q_{\theta'}$ sont régulièrement mis à jour à partir des poids du modèle principale Q_θ de la façon suivante :

$$\theta' \leftarrow \alpha\theta + (1 - \alpha)\theta', \tag{10.9}$$

où $\alpha \in]0, 1[$ est un paramètre. L'algorithme 9 décrit l'entraînement complet d'un agent en utilisant l'algorithme Double Q-learning profond. On note la présence d'un buffer $\mathcal{D}$ qui permet de stocker les épisodes afin d'entraîner le réseau de neurones sur un sous-ensemble de taille N plutôt qu'à épisode par épisode afin d'éviter le sur-apprentissage.

10.3.2 REINFORCE et REINFORCE profond

REINFORCE

L'algorithme **REINFORCE**, aussi connu sous le nom **Monte-Carlo policy gradient**, est un algorithme d'apprentissage, de la famille policy gradient, qui permet d'estimer une politique optimale à partir d'une fonction paramétrée $\pi_\theta(a|s)$. En générale, on définit π_θ à l'aide de la fonction softmax :

$$\pi_\theta(a|s) = \frac{\mathrm{e}^{h_\theta(s,a)}}{\sum_{b\in\mathcal{A}}\mathrm{e}^{h_\theta(s,b)}}, \tag{10.10}$$

où h est une fonction paramétrée qui caractérise la **préférence** de l'agent. Si $h_\theta(s, a) \geq h_\theta(s, a')$, alors, dans l'état s, l'agent préféra prendre l'action a plutôt que l'action a'. Le choix de la fonction softmax garantie que π_θ soit une loi de probabilité, c'est-à-dire que :

Algorithme 9 Deep Double Q-learning pour l'estimation de q_π (avec buffer)

Initialisation :
θ, $\theta' \in \mathbb{R}$
$\mathcal{D}$ pour stocker les épisodes (s, a, s', r)
Paramètres :
$\eta \in]0, 1]$, le taux d'apprentissage
$\gamma \in [0, 1[$, le taux de dépréciation
$\epsilon \in]0, 1]$, pour une politique ϵ-greedy
$\alpha \in]0, 1[$,
Boucle : pour chaque épisode $k = 1, \ldots, K$:
Initialiser l'épisode : $S \leftarrow S_0$
Boucle : pour chaque étapes de l'épisode jusqu'à $S = S_T$:
Observer S, choisir A selon Q_θ (p. ex. politique ϵ-greedy)
Prendre l'action A, observer R, S'
Stocker (S, A, S', R) dans $\mathcal{D}$
$S \leftarrow S'$
Boucle : pour $i = 1, \ldots, N$:
Choisir un épisode (S, A, S', R) dans $\mathcal{D}$
$y_i \leftarrow r + \gamma Q_\theta(S', \arg\max_{A'} Q_\theta(S', A'))$
$q_i \leftarrow Q_\theta(S, A)$
$Y \leftarrow \{y_i\}_{i \in \{1,\ldots,N\}}$
$Q \leftarrow \{q_i\}_{i \in \{1,\ldots,N\}}$
Mise à jour de Q_θ par descente de gradient :
$\theta \leftarrow \theta + \nabla[\frac{1}{N}(Y - Q)^2]$
Mise à jour de $Q_{\theta'}$:
$\theta' \leftarrow \alpha\theta + (1 - \alpha)\theta'$

$$\begin{aligned} &\pi_\theta(a|s) \in [0, 1], \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A} \\ \text{et} \\ &\sum_{a \in \mathcal{A}} \pi_\theta(a|s) = 1, \qquad \forall s \in \mathcal{S}. \end{aligned} \tag{10.11}$$

Comme vu précédemment, dans le cas des algorithmes de type gradient de la politique, on cherche à trouver la meilleure politique en maximisant une fonction de score J. Pour ce faire, on utilise l'algorithme de la montée de gradient 8 qui nécessite de calculer le gradient de J par rapport au vecteur de paramètre θ. D'après Suton et Barto [SB18, p. 326], il est possible d'exprimer le gradient $\nabla J(\theta)$ comme suit :

$$
\begin{aligned}
\nabla J(\theta) &\propto \mathbb{E}\Big[\sum_{a\in\mathcal{A}} q_\pi(S_t,a)\nabla\pi_\theta(a|S_t,)\Big] \\
&= \mathbb{E}\Big[\sum_{a\in\mathcal{A}} q_\pi(S_t,a)\pi_\theta(a|S_t)\frac{\nabla\pi_\theta(a|S_t)}{\pi_\theta(a|S_t)}\Big] \\
&= \mathbb{E}\Big[q_\pi(S_t,A_t)\frac{\nabla\pi_\theta(A_t|S_t}{\pi_\theta(A_t|S_t)}\Big] \\
&= \mathbb{E}\Big[G_t\frac{\nabla\pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}\Big] \\
&= \mathbb{E}\Big[G_t\nabla\log\pi_\theta(A_t|S_t)\Big],
\end{aligned}
\tag{10.12}
$$

où le symbole $\propto$ indique que $\nabla J(\theta)$ est proportionnel à la quantité $\mathbb{E}\Big[G_t\nabla\log\pi_\theta(A_t|S_t)\Big]$. On peut donc réécrire la formule (9.35) de la mise à jour de θ en exprimant le gradient de $J(\theta)$ selon π :

$$
\theta_{t+1} = \theta + \eta G_t\nabla\log\pi_\theta(A_t|S_t). \tag{10.13}
$$

A partir de cette formule de mise à jour du paramètre θ, on peut implémenter l'algorithme REINFORCE 10 qui permet à un agent d'apprendre une politique optimale à partir de l'expérience. Comme π_θ est stochastique, si l'on choisit d'initier le vecteur de paramètre θ de façon à ce que π_θ soit uniforme :

$$
\pi_\theta(s|a) = \frac{1}{|\mathcal{A}(s)|},\ \forall(s,a)\in\mathcal{S}\times\mathcal{A}(s). \tag{10.14}
$$

Cela garantit que l'agent parcourt une large partie de l'environnement avant de converger vers une solution optimale.

Algorithme 10 REINFORCE (Monte-Carlo policy gradient) pour l'estimation de π_*

Entré
$\pi_\theta(a|s)$ une politique différentiable
Initialisation :
$\theta\in\mathbb{R}^d$ (p. ex. 0 ou aléatoire)
Paramètres :
$\eta\in]0,1]$, le taux d'apprentissage
$\gamma\in[0,1[$, le taux de dépréciation
Boucle : pour chaque épisode $k=1,\ldots,K$:
Générer un épisode $\tau_k = \{S_0,A_0,R_1,S_1,A_1,\ldots S_{T-1},A_{T-1},R_T\}_{\pi_\theta}$
Boucle : pour chaque étapes de l'épisode $t=0,1,\ldots,T-1$:
$G\leftarrow\sum_{k=t+1}^{T}\gamma^{k-t-1}R_k$
$\theta\leftarrow\theta+\eta\gamma^t G\nabla\log\pi_\theta(A_t|S_t)$

REINFORCE profond

REINFORCE profond est une extension de l'algorithme REINFORCE pour lequel la fonction π_θ est modélisé par un réseau de neurones (figure 10.3) dont les poids

correspondent au vecteur de paramètres $\theta \in \mathbb{R}^{n_\theta}$. Pour optimiser π_θ, on définit la fonction de perte $L(\theta)$, qu'on cherche à minimiser, par :

$$L(\theta) = -G \log \pi_\theta(a|s), \ \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{10.15}$$

Les poids du réseau π_θ sont mis à jours de manière itérative à l'aide de l'algorithme de descente de gradient :

$$\begin{aligned} \theta &\leftarrow \theta - \eta \nabla L(\theta) \\ &= \theta - \eta \nabla (-G \log \pi_\theta(a|s)) \\ &= \theta + \eta \nabla G \log \pi_\theta(a|s)) \\ &= \theta + \eta \nabla J(\theta). \end{aligned} \tag{10.16}$$

On remarque que minimiser la fonction de perte L revient à maximiser la fonction de performance J. Pour l'implémentation, on se référera à l'algorithme 10 qui reste identique dans le cas profond, en considérant la politique π_θ modélisée par un réseau de neurones.

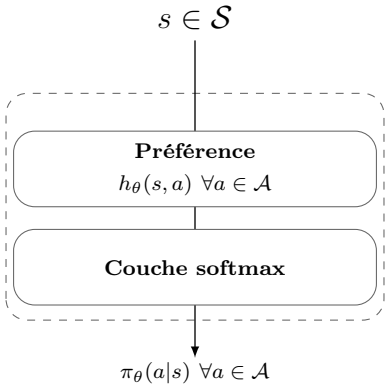


FIGURE 10.3 – REINFORCE profond

Chapitre 11

Expérimentations

Pendant nos expérimentations, nous avons remarqué que des fichiers malveillants issus de sources de données différentes n'étaient pas équivalents d'un point de vue de l'évasion, certains sont relativement faciles à évader, alors que d'autres sont plus compliqués. En particulier, les fichiers venant de BODMAS nous ont confronté à certaines difficultés pour l'évasion des solutions de détection. Nous supposons que ces difficultés peuvent être liées à l'âge des fichiers : le jeu de données étant relativement récent puisqu'il contient 57 293 instances de logiciels malveillants collectés entre août 2019 et septembre 2020. Dans le présent document, nous avons décidé de nous appuyer uniquement sur ce jeu de données pour nos expérimentations.

Comme précédemment, les informations fournies par les solutions de ML et un AV commercial sont différentes (un score de prédiction contre un label). De plus, l'antivirus commercial avec lequel nous travaillons à des temps de réponse significativement plus élevés que les solutions de ML. Par conséquent, nous séparons nos expériences : pour les solutions de ML (Malconv, Grayscale et Ember), nous utilisons 1 000 fichiers malveillants pour l'entraînement et 500 fichiers malveillants pour le test. Pour l'expérience de l'antivirus commercial, et principalement en raison du temps de réponse, nous n'utilisons que 100 logiciels malveillants pour l'entraînement et 50 logiciels malveillants pour les tests.

11.1 Contournement de Malconv

Nous commençons nos expérimentations par l'entraînement d'un agent pour outrepasser Malconv. Notre agent apprend à partir d'un réseau de neurones profond et de l'algorithme Q-learning (DQN). La figure 11.1 montre les résultats de l'entraînement de notre agent face à Malconv avec le score cumulé pour une certaine action (gauche), et le nombre de fois qu'elle a été utilisée au cours de l'entraînement (droite). Comme nous utilisons cette représentation tout au long de nos expérimentations, nous allons donner quelques exemples d'interprétations possible pour expliciter les informations fournies par cette figure :

Pour commencer, nous entraînons l'agent DQN à outrepasser le modèle Malconv. La figure 11.1 décrit le comportement de l'agent, au cours de l'entraînement. Pour chaque action, nous pouvons observer le score cumulé et le nombre de fois qu'elle a été utilisée. Par exemple, pour l'action indexé par 0 (modify machine type), l'agent l'utilise

une centaine de fois au cours de l'entraînement sans effet notable sur le score cumulé. Nous pouvons en déduire que cette action est inutile pour contourner Malconv et la variable "machine type" des fichiers PE n'a aucune incidence sur sa prise de décision. L'action numéro 5 (add section strings) est la plus utilisée au cours de l'entraînement, et renvoie un score cumulé relativement important (≈ 490). Cela signifie que cette action est très efficace pour modifier un fichier PE et tromper Malconv. Nous pouvons aussi voir que l'action numéro 6 (add section benign data) a un score cumulé positif, bien que moins utilisée. Malconv est un modèle de traitement du langage naturel (NLP), donc peu robuste à l'injection de chaînes de caractères. Notre agent a réussi à trouver cette faiblesse en utilisant les actions 5 et 6, qui ajoutent de l'information en ciblant les sections du fichier PE. Cela permet à l'agent DQN d'atteindre 96% d'évasion au cours de l'entraînement.

Pendant la phase d'évaluation, l'agent utilise seulement l'action 5 et atteint un taux d'évasion de 100%. Nous ne parlerons pas davantage de l'outil de détection Malconv dans la suite, ces outils repose sur l'analyse de séquences d'octets et les résultats montrent qu'il est facile à contourner en ajoutant des chaînes de caractères à un fichier PE. De plus, plusieurs travaux de recherche ont déjà discuté sur les forces et les faiblesses de Malconv [Fle+18 ; Kol+18 ; Dem+19].

11.2 Contournement de Grayscale

L'algorithme de détection ; que nous appelons Grayscale [MQC22] ; est un modèle de détection de malware basé sur la transformation de fichiers binaires en images. Il s'agit d'un réseau de neurones convolutionnels puisque ce type de réseau est adapté à la classification d'images. Face à Grayscale, notre agent atteint un score d'évasion de 90% au cours de l'entraînement. Face à Malconv, notre agent avait sur-appris en effectuant toujours la même action. Ici, l'agent entraîné face à Grayscale effectue une combinaison d'une dizaine d'action, en moyenne, pour outrepasser le modèle. Comme nous pouvons le voir dans la figure 11.2, plusieurs actions différentes ont un score cumulé positif (actions {2, 3, 6, 7}), ce qui signifie que l'agent diversifie ses choix au cours du processus d'évasion. Cet agent atteint 98% sur l'ensemble de tests avec les mêmes actions identifiées au cours de l'entraînement. Comme pour Malconv, le modèle Grayscale est facile à outrepasser. Ici, le problème vient de l'utilisation d'images qui peuvent être facilement modifiées en effectuant de grosses modifications dans le fichier binaire associé.

11.3 Contournement de EMBER

Le modèle appelé EMBER [AFR17] est une solution de détection de fichiers malveillants, de type LightGBM, entraînée sur un dataset contenant un million d'instances. EMBER est connu pour être particulièrement robuste face aux attaques adverses. Nous entraînons donc deux agents pour tenter de contourner EMBER, un agent DQN et un agent REINFORCE.

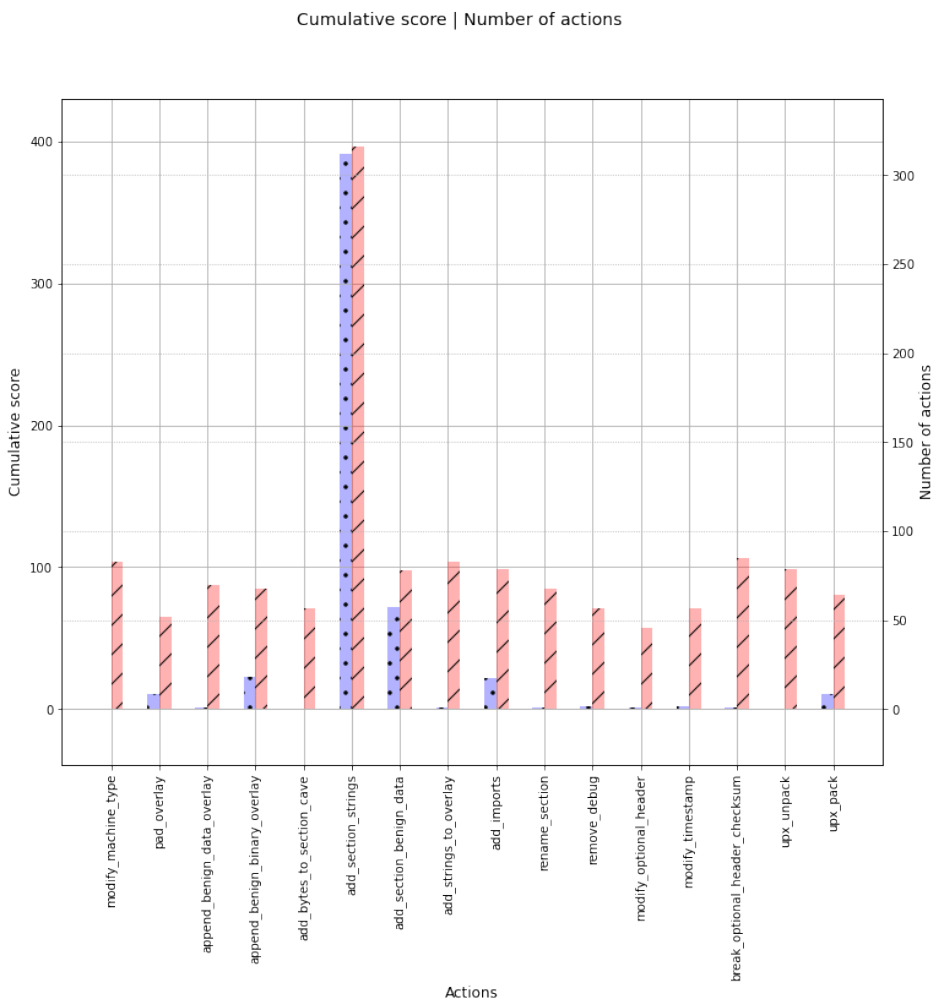


FIGURE 11.1 – Score cumulé (gauche) et nombre d'action (droite)
Entraînement de l'agent DQN face à Malconv

Agent DQN

L'agent DQN atteint un score d'évasion de 40% pendant l'entraînement. Comme évoqué précédemment, EMBER semble plus robuste que Malconv et Grayscale face aux attaques adverses. De plus, l'agent DQN a besoin de plus d'étape pour rendre un fichier malveillant avec 30 actions en moyenne, contre 10 pour Malconv ou Grayscale. L'agent DQN semble avoir plus de difficultés à apprendre comment outrepasser la solution de détection EMBER. La figure 11.3 présente les scores cumulés de chaque action au cours de l'entraînement. L'utilisation de l'action 5 (add section strings) est prépondérante et cette action renvoie un score cumulé relativement important. D'autres actions, comme l'action 15 (upx pack), renvoie un score cumulé positif, mais beaucoup moins important.

Pendant l'entraînement, l'agent DQN montre une forme de sur-apprentissage puisqu'il n'utilise que l'action 5 (add section strings) jusqu'à qu'un malware ne soit plus détecté par EMBER. L'agent atteint néanmoins un taux d'évasions de 67% en 8 étapes en moyenne. Même si l'agent n'utilise que l'action 5, cette approche est efficace pour outrepasser EMBER et il semblerait que l'ajout de chaîne de caractères soit une des faiblesses de EMBER et perturbe fortement le modèle. Néanmoins, l'agent DQN n'est

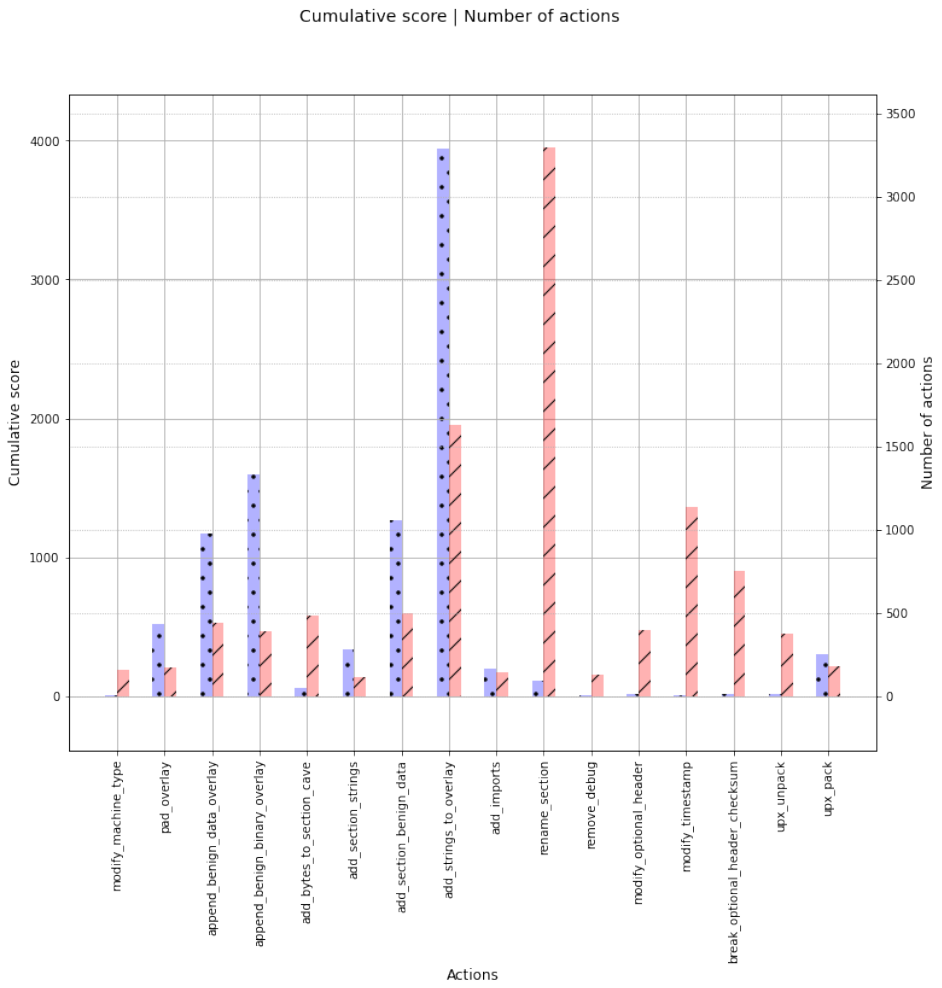


FIGURE 11.2 – Score cumulé (gauche) et nombre d'action (droite)
Entraînement face à Grayscale avec l'agent DQN

pas aussi efficace face à EMBER que face à Malconv ou Grayscale.

Agent REINFORCE

Au cours de l'entraînement, l'agent REINFORCE atteint $54,7\%$ d'évasion en 8 étapes en moyenne face à EMBER. Á première vue, l'agent REINFORCE est plus performant que l'agent DQN (40% d'évasion). Comme le montre la figure 11.4, les actions {5, 7} sont prépondérantes en termes d'utilisations et de scores cumulés, mais le nombre d'actions efficaces est plus diversifié avec un score cumulé meilleur que pendant l'entraînement de l'agent DQN. Comme pour Malconv, l'ajout de chaînes de caractères au fichier binaire semble être une modification efficace pour outrepasser EMBER. Ce manque de robustesse est cohérent avec les conclusions de OYAMA, MIYASHITA et KOKUBO [OMK19] qui explique que seules quelques variables contribuent vraiment à la décision du modèle EMBER.

Les résultats sur l'ensemble de tests sont meilleurs avec l'agent REINFORCE que l'agent DQN, avec une augmentation du taux d'évasion de 67% (DQN) à 80% (REINFORCE) avec en moyenne $7,5$ étapes. En plus de l'action 5, les actions {0, 7, 12}

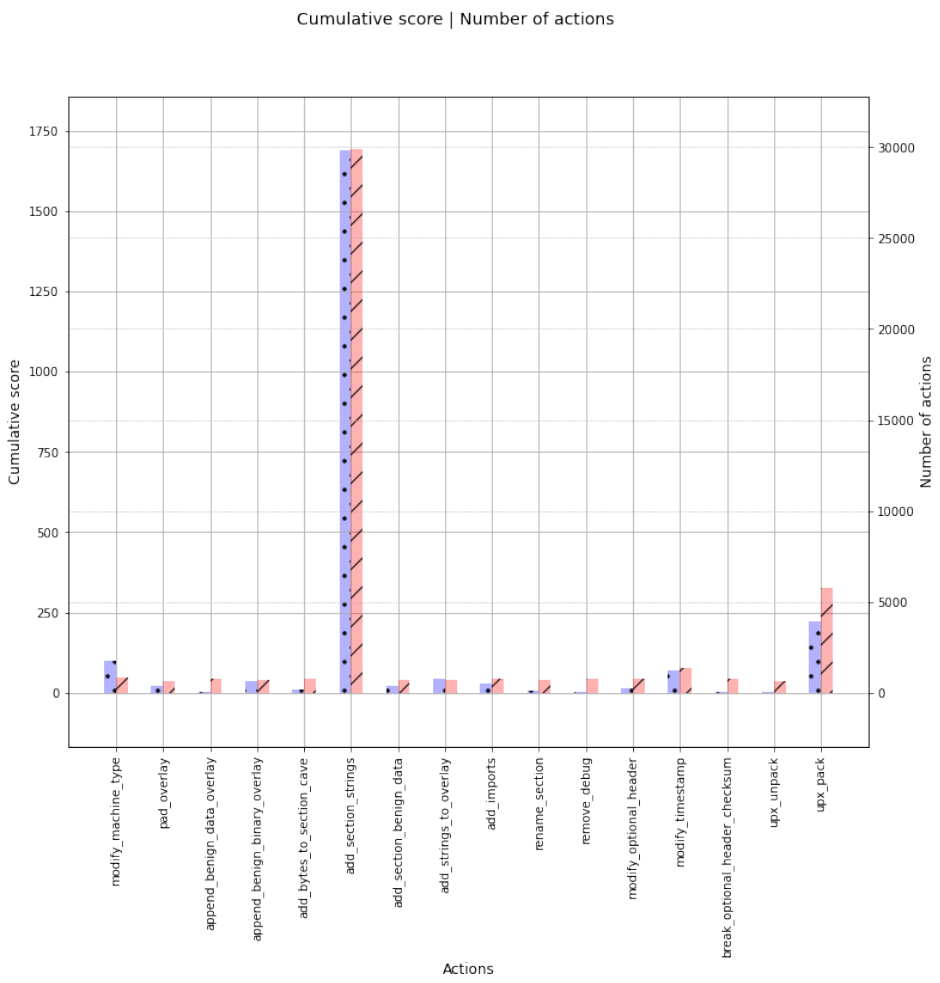


FIGURE 11.3 – Score cumulé (gauche) et nombre d'action (droite)
Entraînement face à EMBER avec l'agent DQN

contribuent fortement à tromper EMBER comme le montre la figure 11.5

11.4 Antivirus commercial

Comme nous l'avons expliqué dans la section 10.1, notre framework est limité face à une solution de détection commerciale, en particulier face à l'AV sur lequel nous testons notre méthode. En effet, l'AV sur lequel nous travaillons ne renvoie pas de score de détection, mais un label qui vaut 0 (fichier bénin) ou 1 (fichier malveillant) quand un fichier est analysé. Ainsi, quand notre agent apprend à outrepasser cet AV, l'impact réel de chaque action ne peut pas être explicitement déterminé, seule la dernière action peut être considérée. Dans les deux prochaines sections, nous présentons les résultats des agents DQN et REINFORCE entraînés à tromper l'AV commercial.

Agent DQN

L'agent DQN atteint 61% d'évasion avec en moyenne 9 étapes par fichier au cours de l'entraînement. Néanmoins, dans la figure 11.6, nous pouvons observer que l'action

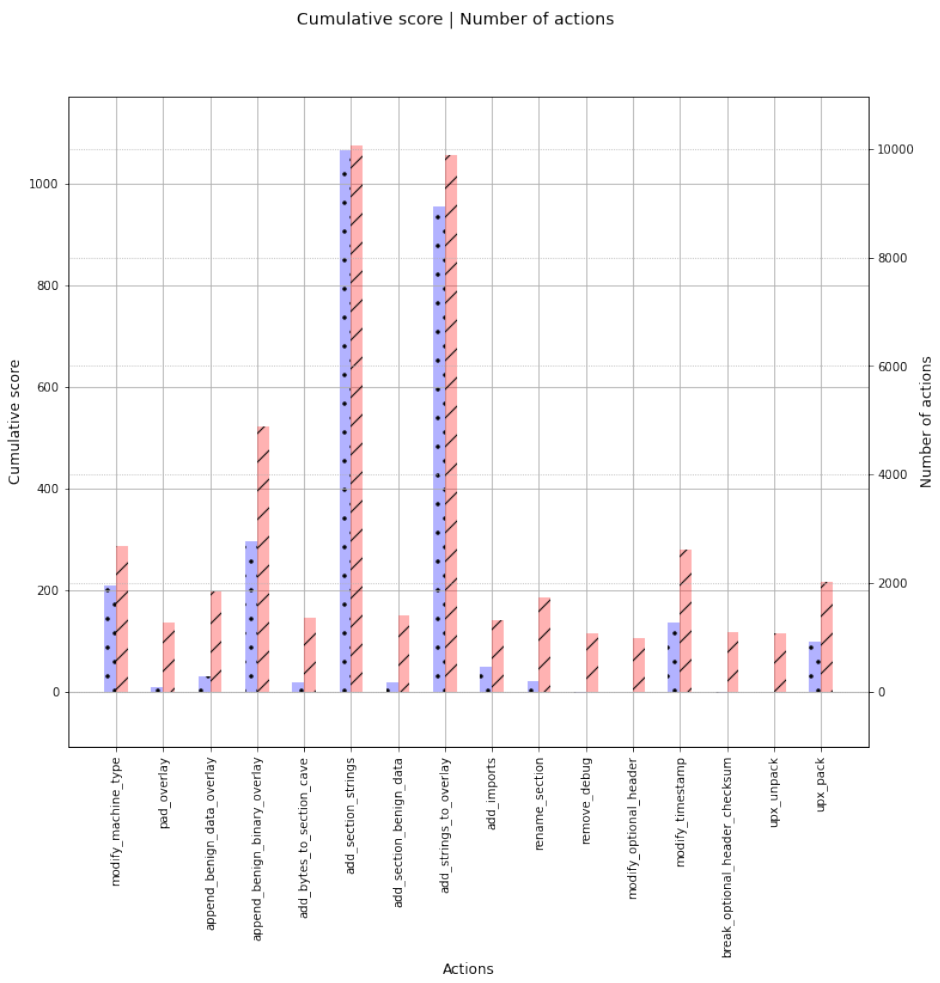


FIGURE 11.4 – Score cumulé (gauche) et nombre d'action (droite)
Entraînement face à EMBER avec l'agent REINFORCE

8 est particulièrement utilisée et retourne le meilleur score cumulé, même si d'autres actions ont un score cumulé non nul. Pour rappel, dans le cas de l'AV commercial, seule la dernière action d'une séquence d'évasion est récompensée, car la configuration de l'AV ne permet pas d'identifier les actions qui ont pleinement contribué à outrepasser.

Comme pour le cas de l'agent DQN face à EMBER, sur l'ensemble de tests, l'agent DQN face à l'AV commercial utilise seulement l'action qui a donné les meilleurs résultats pendant l'étape d'entraînement. Néanmoins, le nombre d'évasion atteint seulement 30%. Ces résultats sont dus au fait que l'apprentissage de l'agent DQN est plus compliqué dans ce cas précis à cause de la configuration contraignante de l'AV commercial. Au cours de l'entraînement, l'agent identifie l'action 8 comme la meilleure action car il ne prend pas en considération une séquence d'action, mais seulement la dernière action qui conduit à l'évasion. Cependant, au cours d'une évasion, les actions précédentes peuvent contribuer autant, si ce n'est plus, que la dernière action. Ici, l'agent DQN montre une forme de sur-apprentissage.

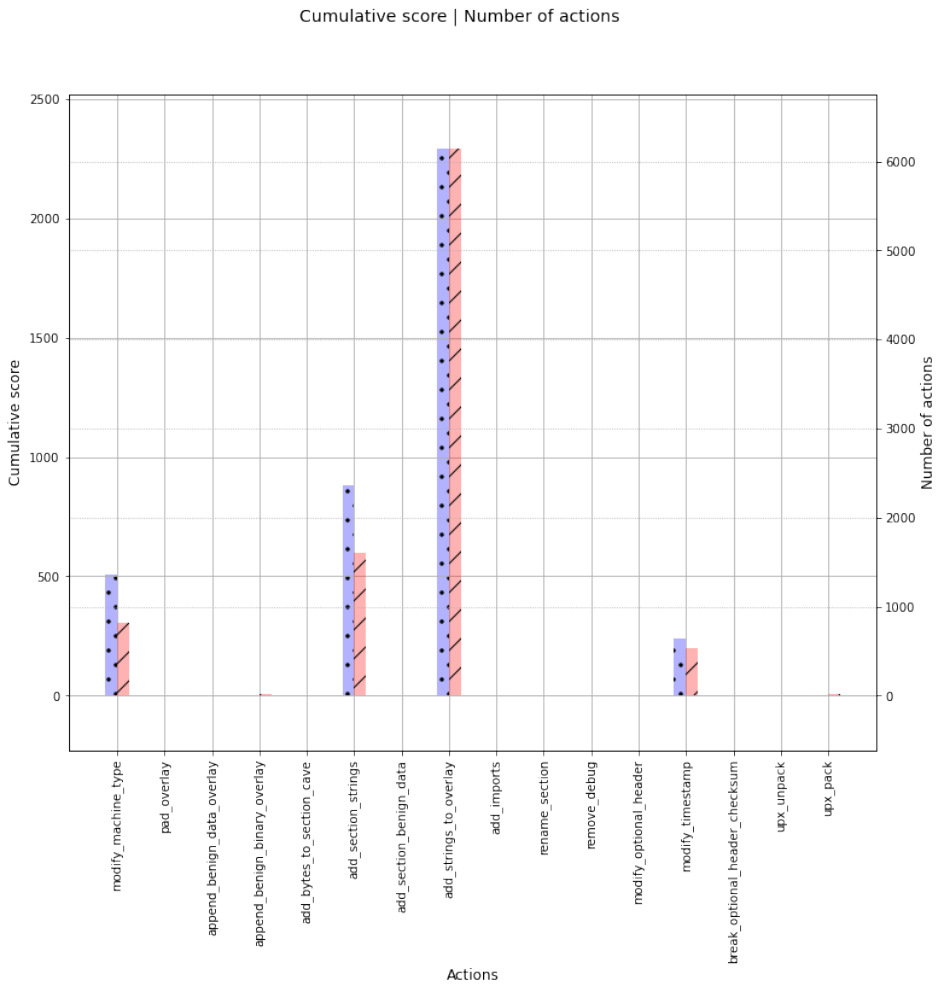


FIGURE 11.5 – Score cumulé (gauche) et nombre d'action (droite)
Évaluation de l'agent REINFORCE face à EMBER

Agent REINFORCE

Dans la section 11.3, nous pouvons observer que l'agent REINFORCE présente de bonnes performances en ayant un comportement plus diversifié dans le choix des actions. Au contraire de l'agent DQN, il ne se contente pas d'utiliser une seule action. Face à un antivirus commercial, l'agent REINFORCE arrive à évader 63% des fichiers malveillants au cours de l'entraînement. L'action la plus efficace est l'action 8, ce qui montre qu'une faiblesse de cet AV commercial est l'ajout de DLL et de fonctions à la table des importations d'un fichier binaire.

Au cours des tests, l'agent REINFORCE n'utilise pas seulement l'action 8, mais aussi d'autres actions qui renvoient un score cumulé positif au cours de l'entraînement. De plus, à l'issue de l'entraînement, l'agent REINFORCE parvient à outrepasser l'AV commercial dans 70% des cas en 8 étapes, en moyenne. Ici aussi, l'agent REINFORCE est meilleur que l'agent DQN face à l'AV commercial (en termes de pourcentage de succès).

Nous soutenons que l'algorithme REINFORCE est plus performant que l'algorithme DQN et que l'utilisation de différentes actions est un avantage supplémentaire et im-

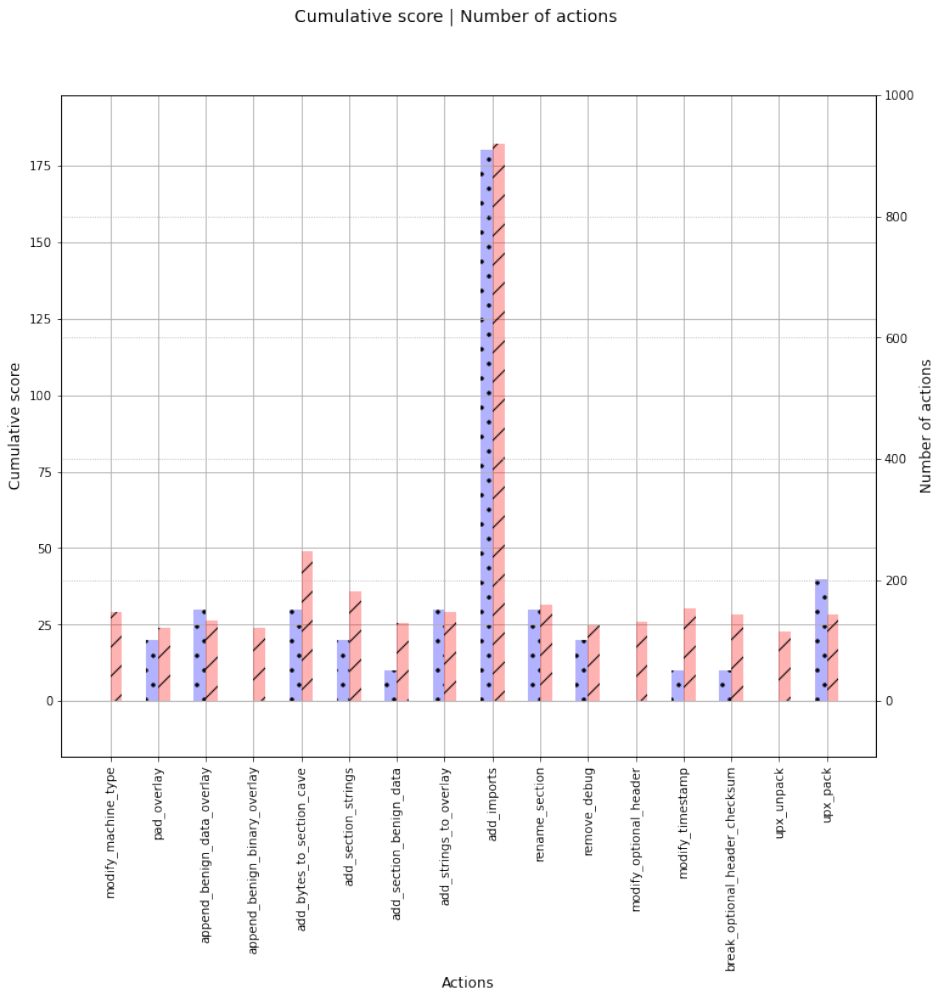


FIGURE 11.6 – Score cumulé (gauche) et nombre d'action (droite)
Entraînement face à l'AV commercial avec l'agent DQN

portant de cet agent.

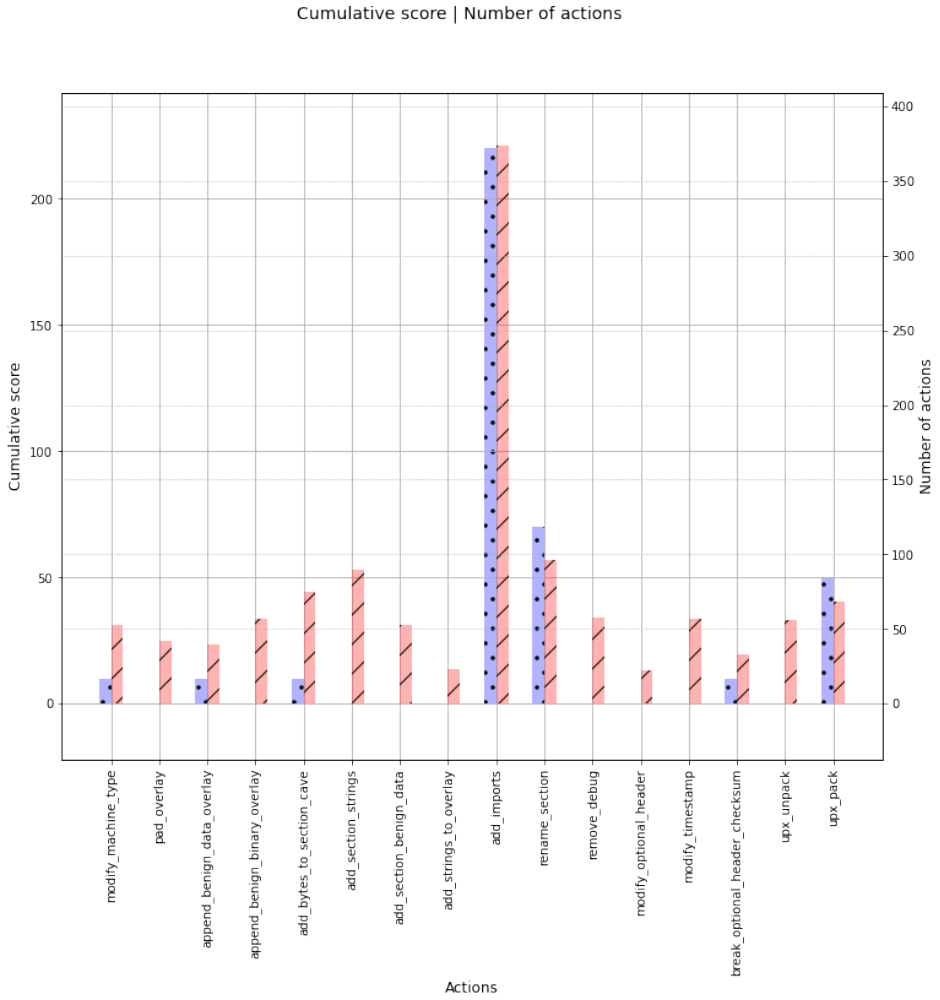


FIGURE 11.7 – Score cumulé (gauche) et nombre d'action (droite)
Entraînement face à l'AV commercial avec l'agent REINFORCE

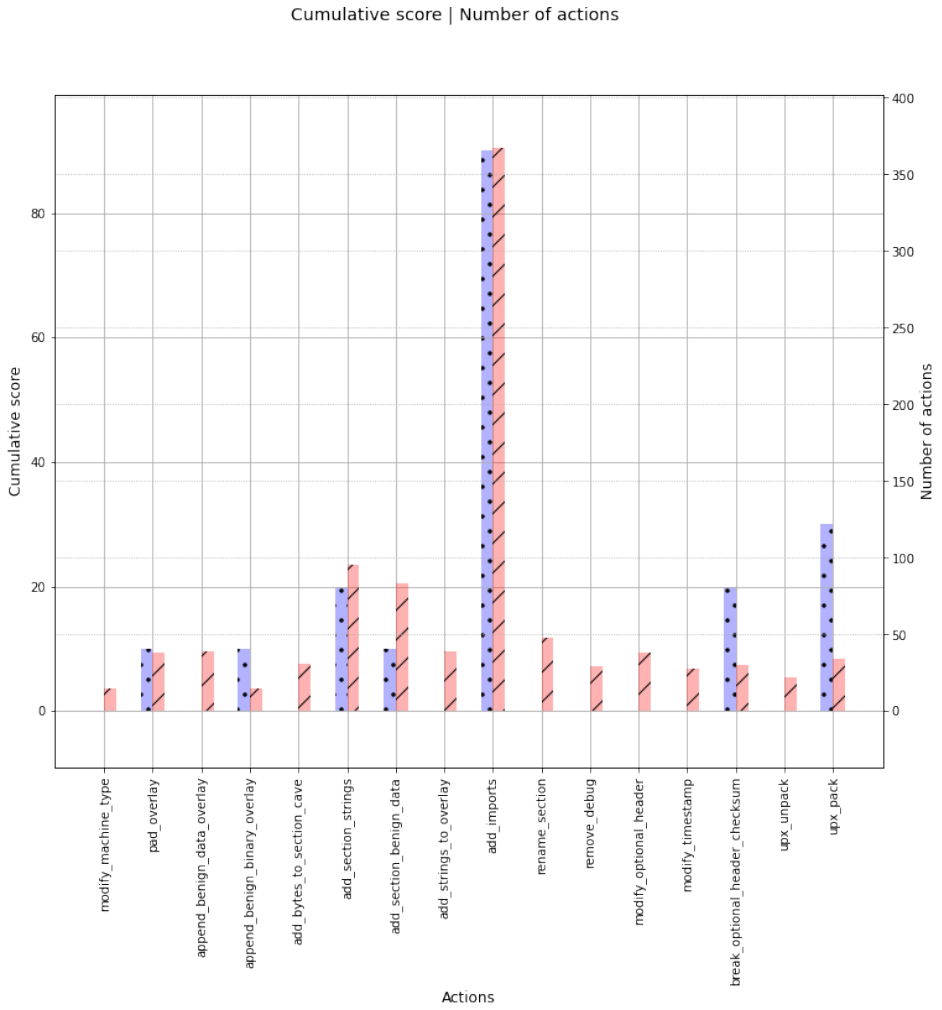


FIGURE 11.8 – Score cumulé (gauche) et nombre d'action (droite)
Évaluation de l'agent REINFORCE face à l'AV commercial

Chapitre 12

Analyse et Explicabilité des Résultats

Dans ce chapitre, nous discutons des résultats des expérimentations et de l'efficacité des deux agents DQN et REINFORCE face aux différentes solutions de détection. Nous présentons également un outil intégré dans MERLIN qui permet d'identifier les faiblesses des antivirus et des modèles de détection ML. Pour terminer, nous illustrons l'intérêt d'un tel outil avec quelques exemples d'évasion de malware produits par nos agents.

12.1 Synthèse des résultats

Dans la table 12.1, nous présentons le taux de succès (taux d'évasion) des agents DQN et REINFORCE face à chacune des solutions de détections au cours de la phase d'évaluation des modèles, tandis que la table 12.2 montre le nombre d'actions qu'il faut, en moyenne, pour rendre un fichier indétectable. Face à EMBER, l'agent REINFORCE est plus efficace que l'agent DQN avec 13% de succès en plus, pour le même nombre d'action en moyenne. L'agent REINFORCE a besoin de plus de temps pour tromper l'AV classique, par rapport à l'agent DQN, mais il reste plus efficace avec une augmentation du taux d'évasion de 30% à 70%. La principale différence entre REINFORCE et DQN est l'expression du choix de l'agent. En effet, dans le cas de REINFORCE, l'agent choisit la meilleure action à partir d'une politique, donc nous sommes dans un cas probabiliste. Pour l'algorithme DQN, l'agent a un comportement déterministe, et comme nous avons pu l'observer, si l'agent a sur-apprit à l'issue de l'entraînement, il ne choisira qu'une action au cours de la phase d'évaluation, et cela, indifféremment de l'état de l'environnement. Ainsi, l'agent REINFORCE est plus efficace car il a appris une politique qui repose davantage sur la diversité des actions. Par exemple, dans le cas de l'AV classique, REINFORCE nécessite en moyenne plus de temps que DQN pour tromper l'outil de détection, néanmoins le taux d'évasion est plus élevé, nous pouvons en déduire que l'agent choisit mieux les actions à effectuer, même si cela doit prendre plus de temps.

12.2 Rapport de vulnérabilités

Une des notions clé en machine learning est la notion d'explicabilité des résultats, et c'est d'autant plus important dans un domaine comme la cyber-sécurité où les décisions

Table 12.1 – Taux d'évasion des deux agents DQN et REINFORCE

	Solutions ML			AV classique
	Ember	Malconv	Grayscale	
DQN	67%	100%	98%	30%
REINFORCE	80%	100%	100%	70%

Table 12.2 – Nombre moyen d'étapes pour contourner les outils de détection (max 100)

	Solutions ML			AV classique
	Ember	Malconv	Grayscale	
DQN	8	9.5	10	5
REINFORCE	7.5	8	9	8

de modèles ML doivent être justifiées et compréhensibles pour les analystes. Ainsi, nous avons introduit une composante supplémentaire à notre framework RL dont l'objectif est d'enregistrer toutes les modifications appliquées aux fichiers binaires au cours de l'entraînement ou de l'évaluation des agents. L'objectif de cette composante, nommée "rapport de vulnérabilités", est d'aider à comprendre comment les actions transforment les fichiers binaires, et pourquoi ces changements modifient la décision de la solution de détection.

Parmi toutes les actions utilisables par les agents, certaines ($\{5, 6, 7, 8\}$) vont ajouter de l'information aux fichiers binaires et d'autres ($\{9, 11, 12\}$) vont modifier des zones spécifiques. Par exemple, l'action 8, **add imports**, ajoute de manière aléatoire une fonction à la table des imports. En analysant l'impact de ce genre de modifications sur la solution de détection, via le score de détection, il est possible d'isoler les modifications les plus efficaces et donc d'identifier des failles potentielles dans les modèles de détection basés sur l'IA ou dans les antivirus plus classiques.

La figure 12.1 illustre comment le rapport de vulnérabilités s'intègre au framework RL de MERLIN. Les informations importantes, c'est-à-dire les actions, récompenses et états, sont utilisées afin de générer ce rapport. Si l'agent ajoute de l'information au fichier ou modifie une certaine partie du binaire, cette information est aussi ajoutée au rapport.

Ce rapport de vulnérabilités nous permet de garder une trace de tous les changements effectués par un agent sur un fichier malveillant qui permettent de le rendre indétectable aux yeux d'une solution de détection. Les outils de visualisation présentés plus tôt et le rapport de vulnérabilités pourraient fournir à des analystes et experts en cybersécurité des informations utiles afin de comprendre les faiblesses des solutions de détection pour les améliorer et les rendre plus efficaces face à de nouveaux variants de logiciels malveillants, ou plus robustes face aux techniques d'évasion utilisées par les éditeurs de malware.

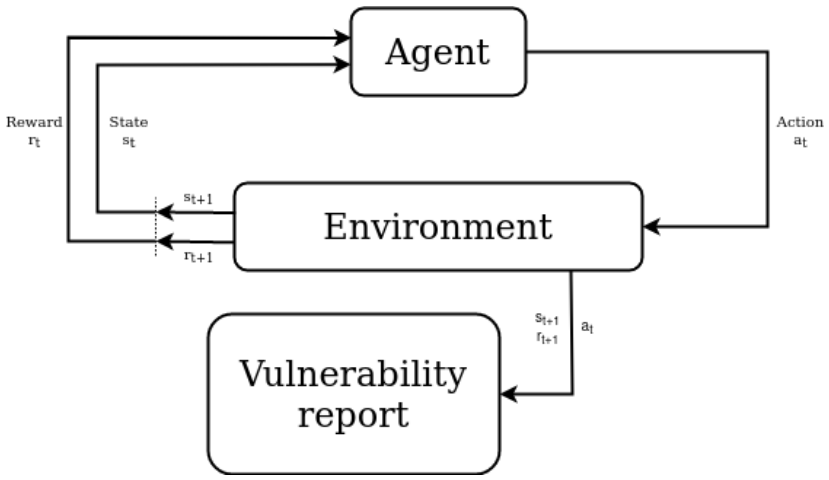


FIGURE 12.1 – Framework de MERLIN avec l'outil "rapport de vulnérabilités"

12.3 Visualisation du processus d'évasion

Grâce au rapport de vulnérabilités, nous pouvons identifier les actions effectuées sur un fichier pour tromper un outil de détection, mais nous pouvons aussi connaître les modifications ou ajouts faits au fichier. Par exemple, pour l'action 8 (add import), l'agent ajoute une fonction à la librairie correspondante dans la table des imports. Si la librairie n'existe pas, il ajoute aussi la librairie. Le rapport de vulnérabilités associe à cette action un score (récompense) qui dépend de son effet sur le score de prédiction renvoyé par l'outil de détection. Dans cette section, nous présentons des exemples d'évasions réussies face à EMBER et face à l'AV classique. Á toute fin utile, nous rappelons la table des correspondances (table 12.3) action-index pour aider à la compréhension des différents graphiques qui vont suivre.

TABLE 12.3 – Table des actions (indexées à partir de 0)

Actions	Index
modify machine type	0
pad overlay	1
append benign data overlay	2
append benign binary overlay	3
add bytes to section cave	4
add section strings	5
add section benign data	6
add strings to overlay	7
add imports	8
rename section	9
remove debug	10
modify optional header	11
modify timestamp	12
break optional header checksum	13
upx unpack	14
upx pack	15

EMBER

À l'aide du rapport de vulnérabilités évoqué précédemment, nous pouvons analyser les modifications apportées par l'agent à un fichier malveillant. C'est particulièrement intéressant avec les logiciels qui ont réussi à tromper un outil de détection. Nous présentons ici des exemples de fichiers dont l'évasion est un succès. Le premier exemple est une évasion réalisée avec l'agent REINFORCE face à l'outil EMBER. La figure 12.2 est une représentation simplifiée du processus d'évasion où chaque nœud représente un état, et l'état final est représenter par le nœud blanc. Une flèche entre deux nœuds indique qu'il y a eu un changement d'état, ainsi que l'action effectuée par l'agent et des informations complémentaires. Dans notre cas, il s'agit d'un logiciel ayant subi deux modifications. La première modifie le nom d'une section du fichier PE, et la deuxième ajoute du contenu au fichier à partir d'un fichier bénin.

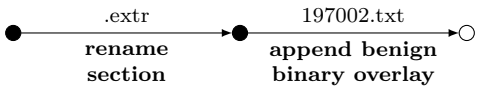


FIGURE 12.2 – Description étapes par étapes des modifications apportées à un fichier

Comme EMBER est un modèle qui renvoie un score de détection, nous pouvons observer son évolution au cours du temps grâce à la figure 12.3. Pour rappel, EMBER utilise un seuil de 0.8336 en dessous duquel un fichier est considéré comme bénin. Ici, seulement la deuxième action provoque une baisse du score de détection et contribue à l'évasion.

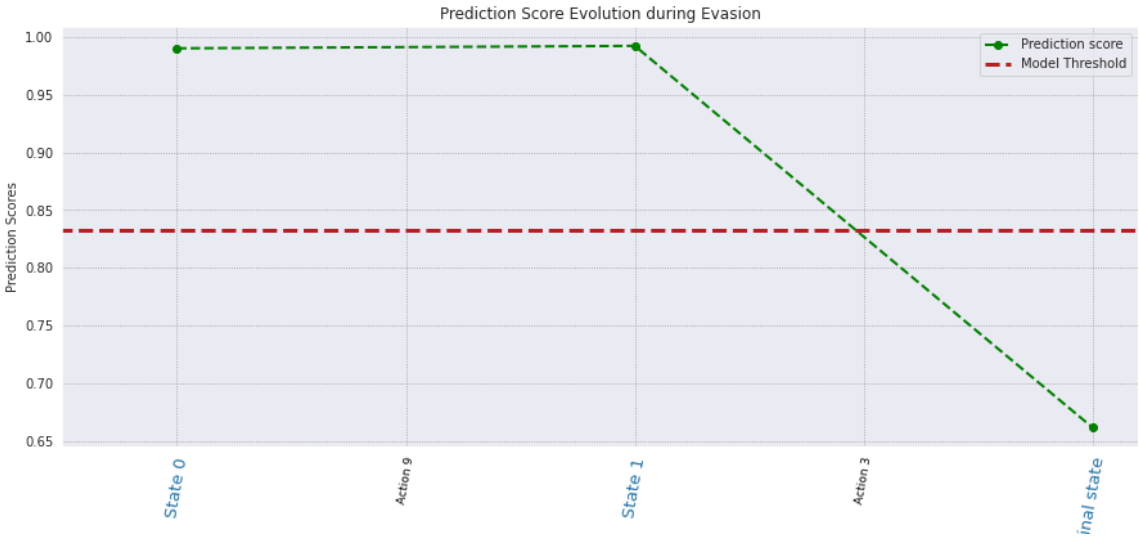


FIGURE 12.3 – Représentation graphique de l'évolution du score de détection au cours d'une évasion en 2 étapes

Nous présentons un second exemple d'évasion face à EMBER pour montrer que le score de détection peut varier au cours du processus. La figure 12.4 nous montre une évasion en dix étapes. Nous pouvons observer trois tendances. Soit l'action ne contribue pas et le score reste inchangé, soit elle contribue positivement et le score diminue légèrement (entre l'état 3 et 4 par exemple) ou beaucoup (entre l'état 9 et l'état

final). Enfin, les modifications peuvent provoquer une légère augmentation du score comme c'est le cas entre les états 7 à 9. Même si en générale, la dernière action contribue le plus à l'évasion d'un fichier malveillant, il peut s'agir aussi d'une accumulation de petites modifications qui permettent de tromper un outil de détection.

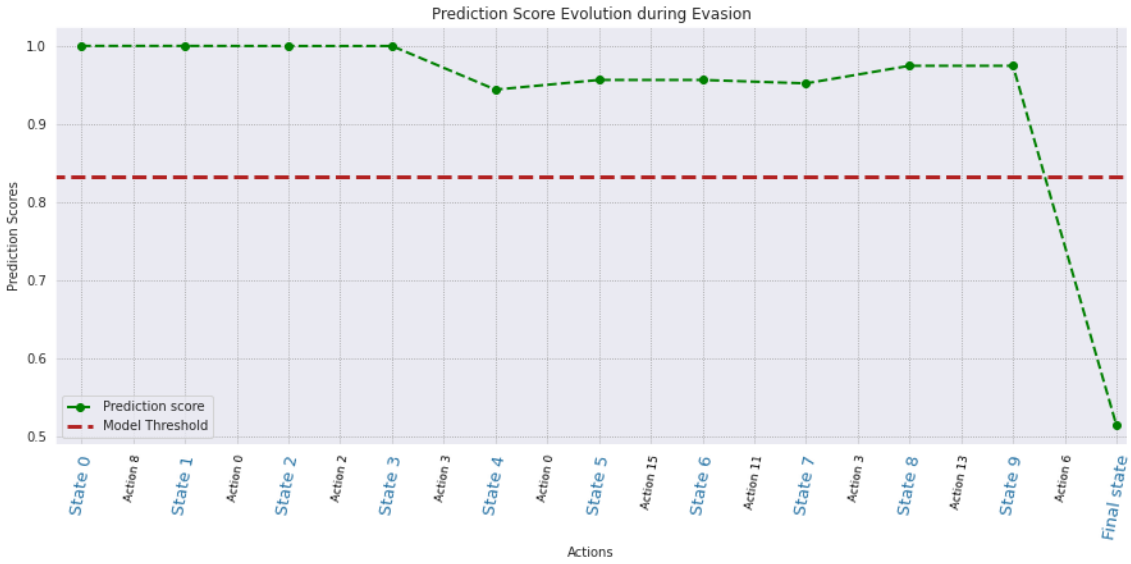


FIGURE 12.4 – Représentation graphique de l'évolution du score de détection au cours d'une évasion en 10 étapes

AV classique

Nous présentons deux exemples d'évasions de malware réussies face à l'AV classique. Comme cet outil ne renvoie pas de score de détection, contrairement aux modèles ML, nous avons adopté la représentation simplifiée présentée précédemment. Le premier exemple est décrit dans la figure 12.5. Dans cet exemple, l'agent REINFORCE parvient à rendre le fichier indétectable en quatre étapes grâce aux actions {8, 8, 0, 6}. L'agent ajoute les deux librairies {msvcrl20.dll, gdi32.dll} à la table des imports, modifie la variable MACHINE_TYPE du fichier PE pour qu'elle prenne la valeur AMD64 et, enfin, l'agent extrait des chaînes de caractère contenue dans un fichier nommé vsiXinstaller.txt pour l'ajouter à une section du malware.

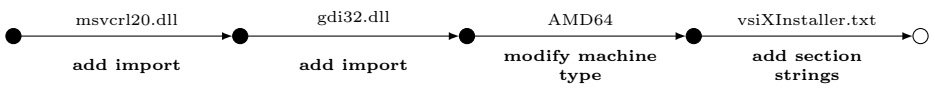


FIGURE 12.5 – Exemple de modifications d'un malware étapes par étapes jusqu'à tromper l'AV classique (4 étapes)

La figure 12.6 montre en détail les étapes exécutées par l'agent pour tromper l'AV classique avec un second malware. Le fichier n'est plus détecté après seulement trois étapes avec les actions {12, 0, 8} avec une modification du timestamps, de la variable MACHINE_TYPE et l'ajout d'une fonction à la table des importations.

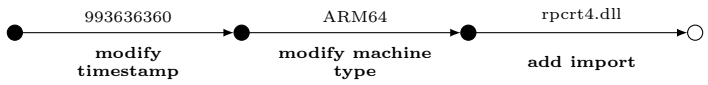


FIGURE 12.6 – Exemple de modifications d'un malware étapes par étapes jusqu'à tromper l'AV classique (3 étapes)

Par la construction de notre fonction de récompense, face à un AV classique, nous attribuons la réussite de l'évasion à la dernière action utilisée. Mais comme nous l'avons vu avec les modèles de détection ML, le changement de décision d'un outil peut être dû à une suite de différentes modifications et pas seulement à la dernière action choisie par l'agent. Nous pourrons donc parler de contribution potentielle d'une action quand elle participe à la réussite de l'évasion.

Chapitre 13

Conclusion et travaux futurs

13.1 Conclusion

Dans cette partie, nous avons présenté un outil appelé MERLIN dont le but est de contourner des outils de détections classiques ou basés sur l'intelligence artificielle, à l'aide de l'apprentissage par renforcement, et sans connaissance préalable sur ces outils. Notre première version, basée sur l'algorithme Deep Q Learning, montre une certaine efficacité face à des modèles comme Malconv ou Grayscale qui repose sur l'analyse de séquences ou de patterns. L'agent comprend, à l'issu de l'entraînement, que l'ajout d'information permet de briser les liens entres les variables et donc de fausser la prédiction de ces outils. Néanmoins, face à un modèle plus robuste comme EMBER, qui repose sur des règles de décision, ou face à un antivirus classique qui retourne peu d'information, l'agent DQN est beaucoup moins efficace, à cause de sa politique déterministe et d'un effet de sur-apprentissage qui conduisent l'agent à utiliser toujours la même action. Pour aller plus loin, nous avons proposé une deuxième version de MERLIN qui repose cette fois sur l'algorithme REINFORCE. Cette version montre de meilleurs résultats face à EMBER et face à l'antivirus, entre autres, car l'agent utilise une politique probabiliste, ce qui le rend plus versatile.

En plus d'être entraîné pour contourner des outils de détection, MERLIN permet de générer un rapport de vulnérabilité qui permet de suivre l'évolution d'un fichier au cours du processus d'évasion. De plus, ce rapport identifie les failles d'un outil de détection, à partir des actions les plus efficaces pour le contourner. Nous pensons que cet outil permettrait d'aider les analystes de logiciels malveillants pour comprendre les évolutions de nouveaux variants et améliorer les solutions de détection.

13.2 Travaux futurs

Même si notre outil MERLIN montre une certaine efficacité face à différentes solutions de détection, nous pensons qu'il peut être amélioré, en optimisant davantage les agents DQN et REINFORCE par exemple. Il pourrait aussi s'agir de travailler spécifiquement avec certaines catégories de malware comme les trojan ou les ransomware, particulièrement actifs et dangereux en ce moment.

Pour ce qui est du rapport de vulnérabilités, en travaillant avec des analystes et experts en cybersécurité, il serait possible d'identifier leurs besoins pour développer un

outil plus précis pour l'analyse des failles des solutions de détection.

Enfin, MERLIN peut permettre de générer des nouveaux variants de fichiers malveillants. Ces variants ont la particularité de ne pas être détectés et en les utilisant pour entraîner ou ré-entraîner des modèles basés sur l'apprentissage supervisé, cela pourrait permettre d'avoir des solutions de détection plus robustes.

Bibliographie

[AFR17] H. S Anderson, Bobby Filar et Phil Roth. « Evading Machine Learning Malware Detection ». In : *BlackHat DC* (2017), p. 6.

[AGA21] Kshitiz Aryal, Maanak Gupta et Mahmoud Abdelsalam. « A Survey on Adversarial Attacks for Malware Analysis ». In : (2021). arXiv : 2111.08223.

[And+13] Ross Anderson et al. « Measuring the cost of cybercrime ». In : *The Economics of Information Security and Privacy*. 2013, p. 265-300. isbn : 9783642394980. doi : 10.1007/978-3-642-39498-0_12.

[AR18a] Hyrum S Anderson et Phil Roth. *EMBER : An open dataset for training static pe malware machine learning models*. 2018. arXiv : 1804.04637.

[AR18b] Hyrum S. Anderson et Phil Roth. *EMBER : An Open Dataset for Training Static PE Malware Machine Learning Models*. 2018. arXiv : 1804.04637 [cs.CR].

[AV-Test] *AV-Test, The Independant IT-Security Institute*. Disponible en ligne : https://www.av-test.org/. En ligne, consulté le 23/05/2021.

[Dem+19] Luca Demetrio et al. « Explaining vulnerabilities of deep learning to adversarial malware binaries ». In : *CEUR Workshop Proceedings* 2315 (2019). issn : 16130073. eprint : 1901.03583.

[Dem+21a] Luca Demetrio et al. « Adversarial EXEmples : A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection ». In : *ACM Transactions on Privacy and Security* 24.4 (2021). issn : 24712574. doi : 10.1145/3473039. arXiv : 2008.07125.

[Dem+21b] Luca Demetrio et al. « Functionality-Preserving Black-Box Optimization of Adversarial Windows Malware ». In : *IEEE Transactions on Information Forensics and Security* 16 (2021), p. 3469-3478. doi : 10.1109/TIFS.2021.3082330.

[Fan+20] Yong Fang et al. *DeepDetectNet vs RLAttackNet : An adversarial method to improve deep learningbased static malware detection model*. T. 15. 4. 2020, p. 1-32. isbn : 1111111111. doi : 10.1371/journal.pone.0231626.

[Fle+18] William Fleshman et al. « Non-negative networks against adversarial attacks ». In : *arXiv preprint arXiv :1806.06108* (2018).

[Goo+14] Ian Goodfellow et al. « Generative adversarial nets ». In : *Advances in neural information processing systems* 27 (2014).

[HGS16] Hado van Hasselt, Arthur Guez et David Silver. « Deep Reinforcement Learning with Double Q-Learning ». In : *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona : AAAI Press, 2016, p. 2094-2100.

[HR20] Richard Harang et Ethan M. Rudd. « SOREL-20M : A Large Scale Benchmark Dataset for Malicious PE Detection ». In : (2020). arXiv : 2012.07634.

[Hu+17] Donghui Hu et al. *The Concept Drift Problem in Android Malware Detection and Its Solution*. Sept. 2017. doi : 10.1155/2017/4956386.

[Hua+19] Yonghong Huang et al. « Malware evasion attack and defense ». In : *Proceedings - 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop, DSN-W 2019*. 2019, p. 34-38. isbn : 9781728130309. doi : 10.1109/DSN-W.2019.00014. arXiv : 1904.05747.

[KOD19] Masataka Kawai, Kaoru Ota et Mianxing Dong. « Improved MalGAN : Avoiding Malware Detector by Leaning Cleanware Features ». In : *1st International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2019* (2019), p. 40-45. doi : 10.1109/ICAIIC.2019.8669079.

[Kol+18] Bojan Kolosnjaji et al. « Adversarial malware binaries : Evading deep learning for malware detection in executables ». In : *European Signal Processing Conference* 2018-Septe (2018), p. 533-537. issn : 22195491. doi : 10.23919/EUSIPCO.2018.8553214. arXiv : 1803.04173.

[Kon+21] Zixiao Kong et al. « A Survey on Adversarial Attack in the Age of Artificial Intelligence ». In : (2021). doi : 10.1155/2021/4907754.

[Loy19] Octavio Loyola-Gonzalez. « Black-Box vs. White-Box : Understanding Their Advantages and Weaknesses From a Practical Point of View ». In : *IEEE Access* 7 (2019), p. 154096-154113. doi : 10.1109/access.2019.2949286.

[MQC22] Benjamin Marais, Tony Quertier et Christophe Chesneau. « Malware Analysis with Artificial Intelligence and a Particular Attention on Results Interpretability ». In : *Distributed Computing and Artificial Intelligence, Volume 1 : 18th International Conference*. Sous la dir. de Kenji Matsui et al. Cham : Springer International Publishing, 2022, p. 43-55. isbn : 978-3-030-86261-9.

[MQM22] Benjamin Marais, Tony Quertier et Stéphane Morucci. « AI-based Malware and Ransomware Detection Models ». In : *Conference on Artificial Intelligence for Defense*. Actes de la 4ème Conference on Artificial Intelligence for Defense (CAID 2022). DGA Maîtrise de l'Information. Rennes, France, nov. 2022. url : https://hal.science/hal-03881198.

[OMK19] Yoshihiro Oyama, Takumi Miyashita et Hirotaka Kokubo. « Identifying useful features for malware detection in the ember dataset ». In : *2019 seventh international symposium on computing and networking workshops (CANDARW)*. IEEE. 2019, p. 360-366.

[PY20] Daniel Park et Bülent Yener. *A survey on practical adversarial examples for malware classifiers*. T. 1. 1. Association for Computing Machinery, 2020. doi : 10.1145/3433667.3433670. arXiv : 2011.05973.

[Raf+17] Edward Raff et al. « Malware detection by eating a whole EXE ». In : *arXiv* (2017). doi : 10.13016/m2rt7w-bkok. arXiv : 1710.09435.

[RN20] Edward Raff et Charles Nicholas. « A Survey of Machine Learning Methods and Challenges for Windows Malware Classification ». In : (2020). arXiv : 2006.09271.

[SB18] Richard S. Sutton et Andrew G. Barto. *Reinforcement Learning : An Introduction*. Cambridge, MA, USA : A Bradford Book, 2018. isbn : 0262039249.

[Sil15] David Silver. *Lectures on Reinforcement Learning*. url : https://www.davidsilver.uk/teaching/. 2015.

[Son+20] Wei Song et al. « MAB-Malware : A Reinforcement Learning Framework for Attacking Static Malware Classifiers ». In : *arXiv preprint arXiv :2003.03100* (2020).

[Sop2020] Sophos Ltd. « The State of Ransomware : Results of an independent study of 5,000 IT managers across 26 countries ». In : *White Paper May 2020* Abingdon, England (2020), p. 1-19.

[Sze+13] Christian Szegedy et al. « Intriguing properties of neural networks ». In : *arXiv preprint arXiv :1312.6199* (2013).

[Tho17] Romain Thomas. *LIEF - Library to Instrument Executable Formats*. https ://lief.quarkslab.com/. Avr. 2017.

[TS99] Sebastian Thrun et Anton Schwartz. « Issues in Using Function Approximation for Reinforcement Learning ». In : 1999.

[UAB19] Daniele Ucci, Leonardo Aniello et Roberto Baldoni. « Survey of machine learning techniques for malware analysis ». In : *Computers and Security* 81 (2019), p. 123-147. issn : 01674048. doi : 10.1016/j.cose.2018.11.001. arXiv : 1710.08189.

[Wil92] Ronald J. Williams. « Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning ». In : *Machine Learning* 8.3 (1992), p. 229-256. issn : 15730565. doi : 10.1023/A:1022672621406.

[Yan+21] Limin Yang et al. « BODMAS : An Open Dataset for Learning based Temporal Analysis of PE Malware ». In : *Proceedings - 2021 IEEE Symposium on Security and Privacy Workshops, SPW 2021*. 2021, p. 78-84. isbn : 9781728189345. doi : 10.1109/SPW53761.2021.00020.

[Zho+20] Fangtian Zhong et al. « MalFox : Camouflaged Adversarial Malware Example Generation Based on C-GANs Against Black-Box Detectors ». In : (2020), p. 1-14. arXiv : 2011.01509.

Troisième partie

Travaux connexes

Chapitre 14

Amélioration des Modèles de Détection de Fichiers Malveillants face à la Dérive des Données

14.1 Introduction

Généralement, les solutions de détection de fichiers malveillants reposent sur des méthodes d'analyse par signatures, d'analyse comportementale ou heuristique [Abo+22 ; Baz+13]. Bien que ces solutions soient fonctionnelles, elles sont confrontées à l'évolution des fichiers malveillants, qui rendent ces différentes approches obsolètes si elles ne sont pas mises à jour régulièrement. Une solution prometteuse est l'utilisation de l'intelligence artificielle, et en particulier de modèles d'apprentissage supervisé. En effet, ces technologies ont fait leurs preuves dans des domaines tels que l'analyse d'images, le traitement automatique du langage ou bien la détection d'anomalies. L'utilisation de machine learning pour la détection de fichier malveillants a montré des résultats encourageants [UAB19], en particulier en utilisant des réseaux de neurones [AF22] ou des arbres de décision de type LightGBM [AAA20]. Ces modèles utilisent des variables extraites des fichiers binaires (fonctions, chaînes de caractères...) ou observés lors de leur exécution (activité réseaux, modification des registres...) pour identifier un logiciel malveillant. Néanmoins, les solutions basées sur le ML sont aussi confrontées à l'évolution des logiciels malveillants. En 2022, SonicWall déclare avoir découvert plus de 465 000 nouveaux variants de malware [Son23]. Ces nouveaux fichiers peuvent engendrer un phénomène appelé "dérive conceptuelle" (ou concept drift) [Lu+19] qui se produit lorsque la distribution des données change au cours du temps. Lorsqu'un modèle est affecté par le problème du drift, cela peut avoir un effet sur son efficacité et entraîner une chute des performances. Pour pallier le drift, une solution courante est de ré-entraîner les modèles ML régulièrement avec de nouvelles données. Cette méthode, bien qu'efficace, peut être une perte de temps pour les entreprises. De plus, dans des domaines comme la cyber-sécurité et l'analyse de fichiers malveillants, le drift peut entraîner des failles de sécurité et des vulnérabilités.

Contributions

Dans ce chapitre, nous proposons un protocole pour réduire l'impact de la dérive conceptuelle sur les performances des modèles ML de détection de fichiers malveillants. Notre protocole repose sur trois approches agnostiques du modèle ciblé, donc utilisable par de nombreux algorithmes d'apprentissage supervisé. Tout d'abord, nous étudions l'effet de l'ensemble de validation sur la généralisation du modèle. Nous nous intéressons aussi au choix des variables utilisées au cours de l'entraînement. Enfin, nous proposons une amélioration de la fonction de perte Binary Cross-Entropy (BCE), dans le but d'aider le modèle à mieux généraliser et appréhender les nouveaux fichiers malveillants et leurs variants. Pour estimer l'impact de nos contributions, nous entraînons différents modèles sur la base de données EMBER [AR18] (2018) et nous les évaluons sur BODMAS (2019-2020) et un ensemble de fichiers collectés par nos soins (2020-2023).

Organisation

Ce chapitre est organisé de la façon suivante : la section 14.2 définit la notion de "dérive" (ou de "drift") et présente les solutions proposées dans différents travaux. Dans la section 14.3, nous décrivons le protocole que nous proposons pour entraîner une solution de détection, tout en minimisant la baisse de performance liée au problème du drift. Dans la section 14.4, nous présentons le cadre expérimental et les résultats que nous avons obtenus. Enfin, dans la section 14.5, nous discutons de nos travaux, de nos résultats et des possibles pistes d'amélioration.

Contexte

Ce chapitre a été réalisé à l'issue du stage de William MAILLET, étudiant en dernière année à l'INSA Centre Val de Loire. Ce stage, que j'ai eu le plaisir d'encadrer, portait sur l'analyse de fichiers malveillants à l'aide de l'intelligence artificielle. Nous nous sommes particulièrement intéressés à l'effet de la dérive conceptuelle sur les solutions de détection de malware, et comment minimiser la baisse de performances au cours du temps. Nos travaux ont fait l'objet d'une pré-publication :

> William Maillet et Benjamin Marais. *Neural Networks Optimizations Against Concept and Data Drift in Malware Detection.* 2023. arXiv : `2308.10821 [cs.CR]`

14.2 Définitions et états de l'art

Dans cette section, nous commençons par définir la notion de "dérive" ("drift"), avant de nous intéresser à la façon dont ce phénomène se produit dans le contexte de l'analyse de fichiers malveillants. Nous faisons ensuite un résumé des méthodes qui existent pour détecter le drift, et des solutions proposées dans le cadre de la détection de malware.

14.2.1 Définitions

Un modèle de ML est sujet au drift lorsque ses performances diminuent au cours du temps. Cela peut arriver lorsque les données observées en entrée évoluent dans le temps, ou quand la relation entre les données observées et le caractère de l'instance change. Quoi qu'il en soit, il s'agit d'une évolution des données que le modèle ne peut pas anticiper. Ces changements peuvent arriver de différentes manières, et peuvent être soudains, progressifs, graduels, ou récurrents comme le montre la figure 14.1.

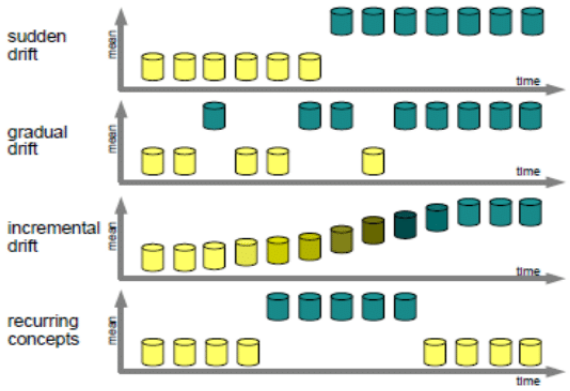


FIGURE 14.1 – Différentes formes de dérives ([LSB15])

Plusieurs travaux de recherches approfondies ont été menés sur le phénomène de la dérive, et il apparaît qu'il existe plusieurs changements possibles dans la distribution des données qui mènent à différentes formes de dérive [Lu+19]. Au cours de nos travaux, nous nous sommes principalement concentrés sur l'étude de la perte de performance des modèles de ML au cours du temps.

Soit f_{θ^*}, un modèle prédictif entraîné sur un ensemble $S_{[t_0,t_1]} = (X_{[t_0,t_1]}, y_{[t_0,t_1]})$ d'observations collectées entre un deux instants t_0 et t_1. L'erreur de prédiction du modèle f_θ peut être mesuré à l'aide de la fonction d'erreur $E : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, et nous considérons que le modèle est fiable tant que l'erreur de prédiction est inférieur à un seuil $\epsilon \in \mathbb{R}$. Le modèle est soumis au drift si ses performances diminuent à partir d'un certain temps, autrement dit, s'il existe $t_2, t_3 > t$ tels que :

$$E(f_{\theta^*}(X_{[t_1,t_2[}); Y_{[t_1,t_2[}) < \epsilon \quad \text{et} \quad E(f_{\theta^*}(X_{[t_2,t_3[}); y_{[t_2,t_3[}) \geq \epsilon. \tag{14.1}$$

Cette définition implique que le drift peut être détecté en analysant la chute de performance du modèle, ce qui n'est pas toujours possible. En général, le but d'un modèle est de prédire la nature d'une nouvelle instance, et il est souvent impossible d'évaluer l'exactitude du résultat sans une intervention humaine pour valider les prédictions du modèle. Parmi toutes les études sur le drift [Gon+14 ; Gem+20], nous nous intéressons aux travaux dont le but est de réduire l'impact du drift sur les modèles de machine learning et leurs performances.

14.2.2 Sources de dérive dans le cas des fichiers malveillants

Plusieurs sources de drift peuvent affecter les modèles ML entraînés pour l'analyse ou la détection de fichiers malveillants. La source la plus courante est l'émergence

d'une nouvelle famille ou d'un nouveau type de malware [Yan+21b], considérée comme une dérive soudaine, puisque ces données n'ont jamais été observées auparavant. Une autre source de drift est liée à l'utilisation particulière de certains fichiers à un moment donné. En effet, les attaquants travaillent souvent en groupes ou en organisations, et sont susceptibles de suivre des tendances en fonction des logiciels malveillants les plus utilisés et les plus efficaces. Ce changement dans la prédominance de certains fichiers [Kan+21] peut être considéré comme progressif quand l'utilisation d'une famille de fichier en particulier s'intensifie, ou redondant quand il s'agit de réutiliser d'anciens types de malware. Un phénomène de dérive soudaine peut aussi apparaître quand une mise à jour importante des API call ou des bibliothèques utilisées par les fichiers bénins ou malveillants intervient. Les différentes sources de drift affectent principalement les fichiers, donc les observations (ou les entrées du modèle) mais rarement la nature du fichier. Ce genre de dérive est appelé dérive virtuelle (virtual drift ou covariate shift) [Huy22 ; Lu+19].

14.2.3 Solutions face au drift

Pour traiter les problèmes liés au drift, il faut tout d'abord identifier s'il y a une dégradation des performances du modèle liée à un changement dans les données. Il faut ensuite trouver l'origine de ce changement. Enfin, il faut améliorer et ré-entraîner le modèle pour empêcher la baisse des performances.

Cependant, la détection d'un phénomène de dérive constitue un véritable défi puisqu'il faut pouvoir détecter un changement dans la distribution des données [Yan+21b ; SCS20] ou bien identifier une baisse de l'efficacité d'un modèle face à de nouvelles instances [Jor+17 ; Bar+22 ; Hu+17].

Certains travaux se concentrent sur la réduction de l'impact de la dérive sur les modèles de machine learning à travers l'apprentissage et l'optimisation. L'apprentissage incrémental (online-learning) est une solution populaire [Nar+16 ; HNA17 ; Xu+19] car cette méthode donne de bons résultats et s'intègre facilement dans un environnement de travail. L'apprentissage auto-supervisé (contrastive learning) [Yan+21b ; CDW23] est aussi une solution envisagée, car il permet d'apprendre à un modèle à identifier des différences ou des similitudes entre deux instances. Une autre approche, proposée par Tumpach, Krcál et Holeňa [TKH19], repose sur l'utilisation d'instances générées à partir d'un auto-encodeur variationnel (VAE) afin de rendre le modèle prédictif plus robuste face aux nouvelles observations. Enfin, parmi les méthodes ayant un effet positif sur les performances du modèle, nous retrouvons l'utilisation du spectral decoupling de Pezeshki et al. [Pez+21], un terme de pénalisation ajouté à la fonction de perte. Le spectral decoupling aide les réseaux de neurones à être plus robustes face à la dérive conceptuelle [CS22] et face aux changements dans la distributions des données étudiées [Poh+22].

14.3 Méthodologie

Dans cette section, nous présentons le protocole que nous avons mis en place afin de réduire l'impact de la dérive conceptuelle sur des solutions ML de détection de fichiers malveillants. Nous décrivons les données utilisées et les améliorations effectuées pour

entraîner différents modèles.

14.3.1 Données et méthodologie

Données

Au cours de nos expérimentations, nous utilisons deux ensembles de données publiques et un ensemble de données dont nous avons collecté nous-mêmes les instances. Le premier ensemble utilisé est EMBER [AR18]. C'est un jeu de données bien connu dans le domaine de l'IA appliqué à la détection de fichiers malveillants. Les instances de EMBER correspondent à des fichiers PE bénins et malveillants collectés entre 2017 et 2018. Les créateurs de EMBER utilisent une méthode qui extrait différentes informations et caractéristiques d'un fichier PE pour le transformer en vecteur. Le deuxième ensemble utilisé est BODMAS [Yan+21a]. Il contient des données plus récentes et collectées entre 2019 et 2020. Les instances sont au même format que EMBER puisque les auteurs de BODMAS utilisent la même méthode d'extraction et de transformation. Comme nous souhaitons évaluer les performances de solutions ML au cours du temps, nous séparons l'ensemble des données de BODMAS selon leurs mois de collecte, pour un total de 14 sous-ensembles couvrant la période d'août 2019 à juillet 2020 (14.2).

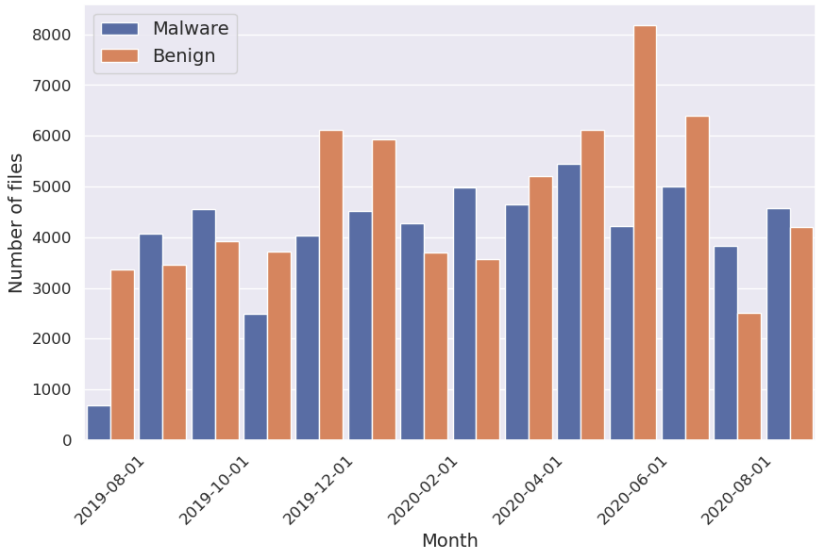


FIGURE 14.2 – Distribution mensuelle du jeu de données BODMAS

Le troisième jeu de données a été créé par nos soins. Il contient des fichiers malveillants collectés sur le site MalwareBazaar [Mal] entre 2020 et 2023. Nous transformons ensuite ces données au même format que EMBER et BODMAS. Tout comme nous l'avons fait pour BODMAS, nous divisons ce jeu de données en plusieurs sous-ensembles selon les mois de collecte. La figure 14.3 montre la distribution des données de MalwareBazaar au cours du temps.

Les données des ensembles EMBER et BODMAS ont été collectées auprès d'entreprises spécialisées dans l'analyse de malware, en plus d'être relativement proches, d'un point de vue temporel. MalwareBazaar est un site qui permet de soumettre des fichiers malveillants dans une base de données collaborative. Même si la politique du site est

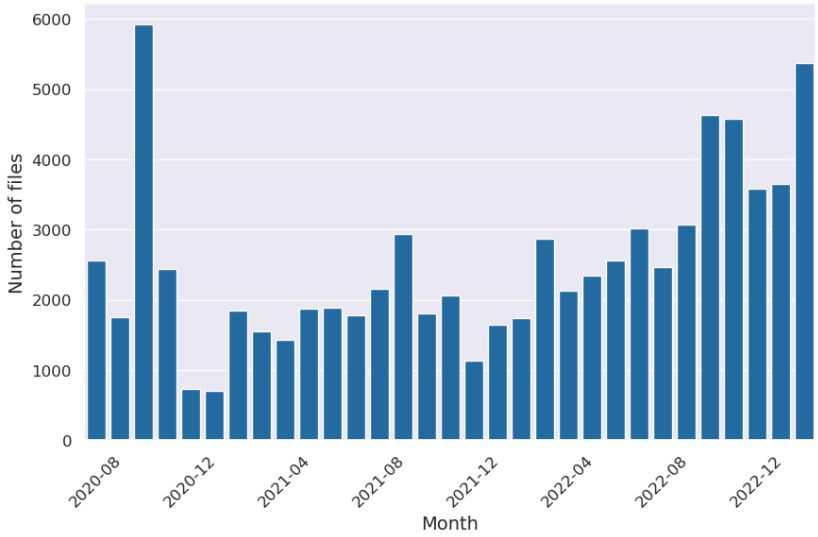


FIGURE 14.3 – Distribution mensuelle du jeu de données MalwareBazaar

relativement rigoureuse, les fichiers ne sont pas constamment vérifiés par des pairs ou une liste d'antivirus (comme c'est le cas de Virustotal [VT]), ce qui fait que la base de données MalwareBazaar est beaucoup plus atypique, en plus de ne contenir que des fichiers malveillants.

Entraînement, test et validation

L'objectif de notre étude est d'améliorer des modèles ML afin qu'ils soient plus résilients face au problème du drift. En général, pour construire des modèles de ML, nous séparons nos données en trois ensembles pour l'entraînement, la validation et l'évaluation. Pour l'ensemble d'entraînement, nous utilisons 700 000 instances choisies aléatoirement dans EMBER. Nous évaluons les performances des différents modèles proposés sur BODMAS (130 000 instances bénignes et malveillantes) et sur MalwareBazaar (78 000 instances malveillantes). Notre intuition est que les solutions de détection basées sur le ML peuvent être soumises au drift. Ainsi, en entraînant des solutions de détection sur d'anciennes données, et en les évaluant sur des données récentes, nous devrions observer une baisse de performance. Nous devrions aussi pouvoir estimer leurs capacités de généralisation.

Notre intuition est qu'en entraînant des solutions de détection sur les données les plus vieilles, nous devrions être en capacité d'estimer leurs capacités de généralisation et leur résilience au drift en évaluant sur des données plus récentes.

Au cours de l'entraînement, nous utilisons deux ensembles de validation différents provenant de EMBER et contenant chacun 120000 instances. Le premier contient des fichiers choisis aléatoirement au sein de la base de données (hors fichiers du jeu d'entraînement) et le second est constitué des derniers fichiers collectés. En utilisant un sous-ensemble de validation plus récent et "disjoint" du jeu d'entraînement, nous espérons réduire le sur-apprentissage et l'impact du drift, ainsi qu'améliorer les performances générales du modèle. La figure 14.4 résume les périodes de collecte de chaque jeu de données ainsi que leurs utilisations pour le développement de solutions ML de détection de fichiers malveillants.

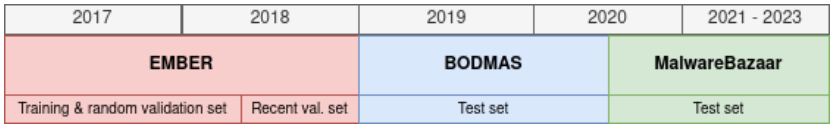


FIGURE 14.4 – Période de collecte et utilisation des ensembles de données EMBER, BODMAS et MalwareBazaar

14.3.2 Sélection et réduction de variables

Il a été montré que la réduction du nombre de variables permettait d'améliorer les capacités de généralisation des modèles ML [Kum14 ; VA19] et de réduire le coût de calcul et d'entraînement, ce qui est souvent contraignant pour les entreprises. De plus, comme énoncé par les auteurs de EMBER, les variables extraites des fichiers mélangent toutes sortes de connaissances qui peuvent ne pas être pertinentes dans le cas de la détection de logiciels malveillants. La méthode utilisée par les auteurs de EMBER permet d'extraire de l'information d'un fichier PE et de le transformer en un vecteur contenant 2381 caractéristiques. Il s'agit, entre autres, de variables statistiques comme la distribution des octets du fichier ou bien la valeur de l'entropie, mais aussi des informations liées à l'en-tête PE (header) ou aux fonctions contenues dans la table des imports (IAT). Nous avons choisi d'effectuer toutes nos expériences sans la variable "timestamp" (qui définit la date de création d'un fichier PE) parce qu'elle est connue pour être peu fiable et facile à modifier pour un attaquant [Che18].

Nous utilisons un algorithme qui permet d'estimer l'importance d'une variable à la prédiction d'un modèle, c'est-à-dire si la variable est utile ou non. Cet algorithme est appelé **Permutation Feature Importance** (PFI) est a été introduit par BREIMAN [Bre01] et amélioré par FISHER, RUDIN et DOMINICI [FRD19]. Pour calculer l'importance d'une variable, la meilleure façon de faire est de comparer la performance entre un modèle entraîné avec et un modèle entraîné sans la variable. Dans le cas où $\mathcal{X} = \mathbb{R}^d$ (l'ensemble des caractéristiques), il faut créer un modèle qui a appris sur toutes les variables et un modèle pour chaque variable qu'on souhaite évaluer, soit $d+1$ modèles, ce qui est gourmand en capacité de calcul et en temps. Pour estimer l'importance d'une variable sans entraîner autant de modèle, il suffit de mélanger aléatoirement toutes ces valeurs. Cette méthode permet de briser la dépendance entre la variable et la prédiction [Mol22, Ch. 8]. Á l'aide de l'algorithme PFI, l'importance de la variable X_j, $j \in \{1, \ldots, d\}$, est définie par :

$$I_j = M(f(X), Y) - M(f(X'_j), Y), \tag{14.2}$$

où f est un modèle entraîné tel que pour tout $(x, y) \in \mathcal{X} \times \mathcal{Y}, f(x) \approx y$ et M est une mesure de performance (le score F1 par exemple). L'ensemble $(X, Y) \in \mathcal{X}^n \times \mathcal{Y}^n$ est un ensemble de données utilisées pour l'entraînement ou l'évaluation du modèle. La variable X'_j correspond à la variable X où la j-ième variable a été mélangé aléatoirement. Plus la valeur I_j est grande, plus la variable j est importante pour le modèle f. De plus, si I_j est positif, nous gardons la variables j à l'issue de l'algorithme de sélection de variables, sinon nous pouvons l'exclure de l'ensemble d'entraînement. L'algorithme 11 décrit le processus de sélection de variables, à l'aide de l'algorithme PFI.

Notre intuition est qu'en sélectionnant les variables qui ont le plus d'importance, cela nous permettra d'identifier les caractéristiques qui sont importantes malgré l'évolution

Algorithme 11 Sélection de variables à l'aide de l'algorithme PFI

Entrée :
f un modèle prédictif
X l'ensemble des observations, Y l'ensemble des étiquettes
M une mesure de score
Sortie :
L une liste des variables dont l'importance est positive

$M_{base} \leftarrow M(f(X), y)$
$L \leftarrow []$
Boucle : pour chaque variable $j = 1, \ldots, d$:
$X_{prm} \leftarrow X'_j$ (mélange aléatoirement la j-ième variable)
$M_{prm} \leftarrow M(f(X_{prm}), Y)$
Si $M_{base} - M_{prm} > 0$ **alors :**
$L \leftarrow L + [j]$

des fichiers malveillants.

14.3.3 Fonction de perte DRBCE

Au cours de nos travaux, nous utilisons une nouvelle fonction de perte nommée DRBCE pour Drift Resilient Binary Cross-Entropy. Il s'agit d'une amélioration de la fonction de perte Binary Cross-Entropy :

$$L_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)] \tag{14.3}$$

où y_i est la classe de l'instance $i \in \{1, \ldots, n\}$ et $p_i = f(x_i)$ est la prédiction de l'instance i telle que $p_i = P(y = 1|x = x_i)$.

La DRBCE est donc une amélioration de la fonction BCE à laquelle s'ajoutent trois composantes. La première composante est un terme de pénalisation appelé "spectral decoupling" [Pez+21] connue pour améliorer la généralisation et la résilience du modèle face au drift. Le spectral decoupling est définie par :

$$L_{\text{SD}} = \frac{1}{n}\sum_{i=1}^{n}\frac{\lambda}{2}||z_i||_2^2, \tag{14.4}$$

où $\lambda \in \mathbb{R}$ est un coefficient de pénalisation, $z_i = h(x_i) \in \mathbb{R}$ (voir 1.2.1, dans le cas de la classification binaire) et $|| \cdot ||$ est la norme euclidienne.

Nous utilisons aussi des termes de pénalisation, sur les faux positifs et les faux négatifs, qui favorisent la diminution de prédiction de faux négatifs ou de faux positifs. Cette méthode à montrer son efficacité, dans le cas de la détection de fichiers malveillants [Gao+22b], en améliorant les performances d'un modèle de type LightGBM. Enfin, nous utilisons deux termes de pondération de classes qui améliore l'optimisation et l'apprentissage du modèle dans le cas de jeux de données avec des classes déséquilibrées :

$$L_{\text{BCE}^+} = -\frac{1}{n}\sum_{i=1}^{n}[w_1 \cdot P_{FN} \cdot y_i \cdot \log(p_i) + w_0 \cdot P_{FP} \cdot (1 - y_i) \cdot \log(1 - p_i)], \tag{14.5}$$

où $w_0 = \frac{n_0}{n}$ (respectivement $w_1 = \frac{n_1}{n}$) est le coefficient de la classe $y = 0$ (resp. $y = 1$) avec n_0 (resp. n_1) le nombre d'instances pour lesquelles $y = 0$ (resp. $y = 1$). Les valeurs $P_{FN} \in \mathbb{R}$ et $P_{FP} \in \mathbb{R}$ sont les coefficients de pénalisation pour les faux négatifs et les faux positifs. La fonction DRBCE est donc définie par :

$$L_{\text{DRBCE}} = L_{\text{BCE}^+} + L_{\text{SD}}. \tag{14.6}$$

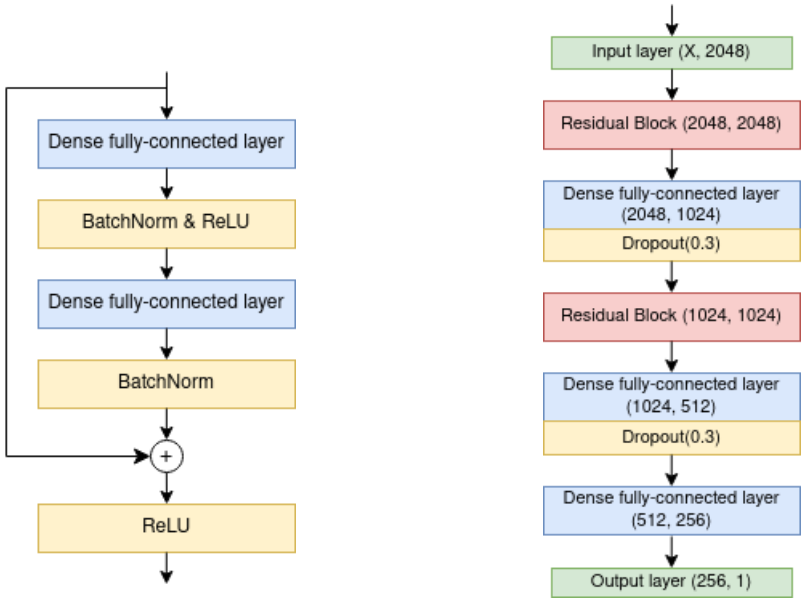
Les hyper-paramètres tels que P_{FP}, P_{FN} et λ sont choisies à l'issue des expérimentations au cours d'une phase d'optimisation.

14.3.4 Modèles

Pour estimer l'effet de ses solutions, nous entraînons un modèle de référence, et nous le comparons à d'autres modèles auxquels nous avons appliqué des modifications comme la réduction de variables, l'utilisation de la fonction DRBCE ou l'utilisation d'un ensemble de validation différent.

Modèle de référence

Notre modèle de référence est un réseau de neurones résiduel [He+15]. L'architecture du modèle est décrite dans la figure 14.5 et est donc composée de couches résiduelles (residual block) et de couches entièrement connectées (Dense FC layer). Comme les solutions proposées sont agnostiques du modèle, nous avons choisi de travailler avec un modèle à l'architecture relativement simple



(a) Couche résiduelle

(b) Architecture du modèle de référence

FIGURE 14.5 – Architecture du modèle de référence

Ce modèle est entraîné à l'aide de la fonction de perte BCE, avec l'algorithme d'optimisation AdamW [LH19], un taux d'apprentissage de 1×10^{-4} et un coefficient de réduction des poids de 1×10^{-4}. Tous les autres modèles utilisent la même architecture, où la taille de la couche d'entrée varie selon le nombre de variables utilisées.

Modifications appliquées au modèle de référence

Au cours des expérimentations, nous appliquons plusieurs modifications au modèle de référence pour étudier l'effet des solutions proposées sur les performances. Les principales modifications sont

- l'utilisation d'un ensemble de validation plus récent que l'ensemble d'entraînement, pour éviter le sur-apprentissage,
- l'entraînement à l'aide de la fonction de perte DRBCE, pour favoriser les capacités d'apprentissage du modèle et minimiser les prédictions de faux négatifs,
- la réduction de l'espace des caractéristiques, pour améliorer la généralisation du modèle et exclure les variables potentiellement inutiles.

Nous entraînons donc plusieurs modèles pour déterminer quelles modifications ont le plus d'impact sur les performances.

14.3.5 Évaluation et métriques

Pour comparer les performances des modèles proposés, nous utilisons quatre mesures différentes, définie dans le chapitre 1, qui sont l'accuracy (ACC), le score F1, le taux de faux positifs (TFP) et le taux de faux négatifs (TFN). L'accuracy permet de mesurer la performance du modèle par rapport au nombre d'instances correctement classifiées. Le score F1 est une métrique qui combine les valeurs de précision et de rappel, et

est particulièrement adapté pour étudier les performances d'un modèle entraîné sur des données non équilibrées. Les deux mesures TFP et TFN sont particulièrement utiles dans le cas de la classification binaire pour estimer la performance du modèle. De plus, selon le contexte, il est possible de chercher à réduire l'une ou l'autre des mesures. En effet, dans le contexte de la détection de fichiers malveillants, les éditeurs de solutions voudront minimiser le TFN pour empêcher les logiciels les plus discrets d'infecter un système ou de se propager, en pénalisant néanmoins le TFP. Cette pratique peut conduire à des phénomènes comme la fatigue d'alerte [Ban21] où l'analyse de fichiers bénins, détectée comme malveillante, fait perdre du temps aux entreprises. Dans nos travaux, nous nous concentrons particulièrement dans la maximisation des performances (ACC et F1) et la minimisation des prédictions faussement négatives (TFN).

14.4 Expérimentation et résultats

14.4.1 Fonction de perte DRBCE

Dans cette section, nous décrivons les expérimentations et les résultats liés à l'utilisation de la fonction de perte DRBCE ainsi qu'à l'optimisation des paramètres P_{FN}, P_{FP} et λ. Pour rappel, la fonction DRBCE est définie par :

$$L_{\text{DRBCE}} = -\frac{1}{n}\sum_{i=1}^{n}[w_1 \cdot P_{FN} \cdot y_i \cdot \log(p_i) + w_0 \cdot P_{FP} \cdot (1-y_i) \cdot \log(1-p_i)] + \frac{1}{n}\sum_{i=1}^{n}\frac{\lambda}{2}||z_i||_2^2 \quad (14.7)$$

Pour tester les différentes valeurs de nos paramètres, nous nous plaçons dans les conditions initiales du modèle de référence où l'espace des caractéristiques n'est pas réduit ($\mathcal{X} = \mathbb{R}^{2380}$) et l'ensemble de validation utilisé est aléatoire (cf. figure 14.4).

Choix du coefficient de spectral decoupling λ

Pour choisir la valeur du coefficient λ, nous entraînons plusieurs modèles à l'aide de la fonction de perte DRBCE en faisant varier la valeur de λ, avec une pénalisation des faux positifs et des faux négatifs neutre ($P_{FN} = 1, P_{FP} = 1$). Dans la table 14.1, nous pouvons voir les performances des différents modèles selon les valeurs de λ. En prenant $\lambda = 0.05$, nous avons de meilleurs résultats sur l'ensemble de données BODMAS. Néanmoins, avec la valeur $\lambda = 0.01$, le modèle a une meilleure capacité de prédiction sur l'ensemble MalwareBazaar, donc sur les fichiers récents, tout en gardant de bonnes performances sur les fichiers de BODMAS. Comme nous privilégions le gain de performance sur les fichiers récents, nous fixons $\lambda = 0.01$ dans la suite de nos travaux.

(P_{FN}, P_{FP})	BODMAS			MalwareBazaar	
	ACC	**F1**	**FNR**	**ACC**	**F1**
$\lambda = 0.5$	0.8806	0.8469	0.1606	0.3191	0.4694
$\lambda = 0.1$	0.9370	0.9274	0.0550	**0.3715**	**0.5410**
$\lambda = 0.05$	**0.9422**	**0.9337**	**0.0468**	0.3621	0.5314
$\lambda = 0.01$	0.9379	0.9272	0.0708	0.3517	0.5202
$\lambda = 0.001$	0.9255	0.9137	0.0744	0.3642	0.5337

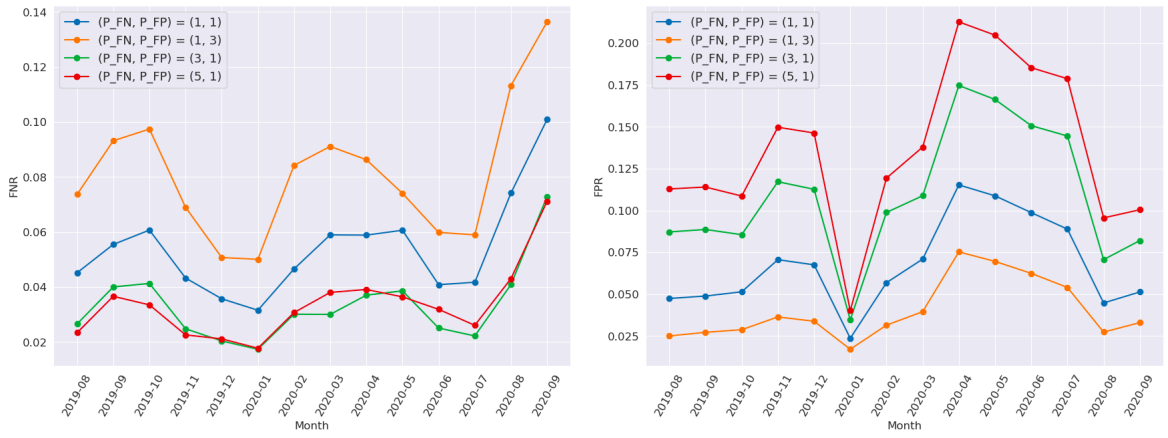
TABLE 14.1 – Performances des modèles pour différentes valeurs de λ

Choix des coefficient de pénalisation P_{FN} et P_{FP}

Nous nous intéressons maintenant aux coefficients de pénalisation des faux positifs et des faux négatifs. Notre intuition est que la pénalisation des mauvaises prédictions permet d'aider le modèle à intégrer ces notions, et pas seulement la notion de performance absolue. Nous testons plusieurs couples (P_{FN}, P_{FP}) en entraînant différents modèles, toujours dans les conditions initiales, et avec $\lambda = 0.01$. Nous pouvons observer les performances de différents modèles pour différentes valeurs de P_{FN} et P_{FP} dans la table 14.2. Nous pouvons aussi observer l'effet des coefficients de pénalisation sur BODMAS dans la figure 14.6.

(P_{FN}, P_{FP})	BODMAS				MalwareBazaar		
	ACC	**F1**	**FNR**	**FPR**	**ACC**	**F1**	**FNR**
$P_{FN} = 1, P_{FP} = 1$	0.9370	0.9274	0.0550	0.0690	0.3715	0.5410	0.6285
$P_{FN} = 1, P_{FP} = 3$	**0.9406**	**0.9294**	0.0825	**0.0465**	0.3043	0.4659	0.6765
$P_{FN} = 3, P_{FP} = 1$	0.9227	0.9142	**0.0342**	0.1094	0.4605	0.6300	0.5395
$P_{FN} = 5, P_{FP} = 1$	0.9079	0.8995	0.0348	0.1346	**0.4877**	**0.6957**	**0.5123**

TABLE 14.2 – Performances des solutions de détection pour différentes pairs (P_{FN}, P_{FP})



(a) FNR (b) FPR

FIGURE 14.6 – Comparaisons des performances pour différentes valeurs de pénalisation

En plus d'influencer la performance générale des modèles, nous pouvons remarquer qu'en augmentant la pénalisation sur les faux positifs, c'est-à-dire en augmentant P_{FN}, cela réduit le TFN à défaut du TFP, et inversement en augmentant P_{FP}. Au vu des résultats, nous fixons $(P_{FN}, P_{FP}) = (5, 1)$. En choisissant ces valeurs pour les coefficients de pénalisation, nous assumons la recherche de performance à long terme avec une augmentation du score F1 de plus 0.15 par rapport au modèle de référence ($(P_{FN}, P_{FP}) = (1, 1)$).

Choix des paramètre et comparaison avec le modèle de référence

Nous choisissons donc de fixer le coefficient de spectral decoupling $\lambda = 0.1$, et les coefficients de pénalisation $(P_{FN}, P_{FP}) = (5, 1)$. Nous comparons l'efficacité des fonctions BCE et DRBCE à l'aide de la table 14.3. Le modèle entraîné à l'aide de la fonction DRBCE ($\lambda = 0.1$, $(P_{FN}, P_{FP}) = (5, 1)$) est plus efficace pour la détection de fichier malveillant récent, malgré une chute de performance sur l'ensemble de fichiers de BODMAS.

Fonction de perte	**BODMAS**			**MalwareBazaar**		
	ACC	**F1**	**FNR**	**ACC**	**F1**	**FNR**
BCE	**0.9247**	**0.9127**	0.0769	0.3640	0.5333	0.6360
DRBCE	0.9079	0.8995	**0.0348**	**0.4877**	**0.6552**	**0.5123**

TABLE 14.3 – Comparaison des performances pour des modèles entraînés avec les fonction BCE et DRBCE

Dans la suite des travaux, nous fixons donc les paramètres $\lambda = 0.1$, $(P_{FN}, P_{FP}) = (5, 1)$.

14.4.2 Ensemble de validation

L'entraînement de modèles d'apprentissage profond passe aussi par la validation des performances sur un ensemble disjoint de l'ensemble d'entraînement. Cette phase de validation permet d'éviter le sur-apprentissage et permet d'améliorer la capacité de généralisation du modèle. Généralement, les ensembles d'entraînement et de validation sont choisis aléatoirement parmi un ensemble de données plus grand, donc les données sont sémantiquement proches. Dans cette partie, nous proposons d'utiliser un ensemble de fichiers collectés plus récemment, par rapport aux données d'entraînement, pour construire notre ensemble de validation. Nous comparons l'effet de la validation avec deux ensemble de 120 000 fichiers. Le premier ensemble couvre toute la période de collecte de EMBER et le deuxième ensemble contient seulement des fichiers collectés dans les trois derniers mois. Nous testons notre approche sur des modèles entraînés avec les fonctions BCE et DRBCE.

Fonction de perte et ensemble de validation	BODMAS			MalwareBazaar		
	ACC	**F1-Score**	**FNR**	**ACC**	**F1**	**FNR**
BCE, val. aléatoire	0.9247	0.9127	0.0769	0.3640	0.5333	0.6360
BCE, val. récente	**0.9338**	**0.9228**	0.0696	0.3551	0.5238	0.6449
DRBCE, val. aléatoire	0.9079	0.8995	0.0348	0.4877	0.6552	0.5123
DRBCE, val. récente	0.9128	0.9056	0.0193	**0.4971**	**0.6634**	**0.5029**

TABLE 14.4 – Comparaisons des performances des solutions de détection selon la fonction de perte et l'ensemble de validation

Nous pouvons voir dans la table 14.4 que l'utilisation d'un ensemble de validation plus récent que le jeu de données d'entraînement à un léger effet sur la capacité de détection des différents modèles. En effet, pour ce qui des modèles entraînés avec la BCE, on observe une légère augmentation des performances sur BODMAS (0.01 pour le score) malgré une diminution sur l'ensemble MalwareBazaar. Pour ce qui est des modèles entraînés avec la DRBCE, on observe une augmentation des performances dans les deux cas, couplée à une baisse du nombre de prédictions faussement négatives. L'utilisation d'un ensemble de validation plus récent que l'ensemble des données d'entraînement permet donc de favoriser la généralisation du modèle à court et long terme, et ce, indépendamment de la fonction de perte utilisée. Il est néanmoins nécessaire de rappeler que la collecte et l'utilisation de données récentes est toujours une étape compliquée dans le cas de l'analyse de malware car il peut s'écouler un certain temps entre le moment où un fichier malveillant est mis sur le marché et le moment où il est détecté comme tel.

14.4.3 Sélection et réduction des variables

Dans cette partie, nous voulons évaluer l'efficacité de la réduction de l'espace des caractéristiques sur la capacité de prédiction de nos modèles de détection. Pour cela, nous utilisons l'algorithme Permuation Feature Importance (décrit dans la section 14.3.2). En utilisant cet algorithme, nous voulons filtrer les variables qui ne sont pas nécessaires pour la tâche de détection. Pour évaluer l'importance des caractéristiques, nous utilisons le score F1 qui traduit l'efficacité d'un modèle et est plus approprié dans le cas d'ensembles de données déséquilibrés. Nous utilisons l'algorithme PFI sur BODMAS car il s'agit de l'ensemble de données le plus récent à notre disposition contenant à la fois des fichiers bénins et malveillants. Nous avons appliqué le PFI sur le modèle de référence et sur le modèle entraîné avec la fonction DRBCE à l'aide d'un ensemble de validation aléatoire. En effet, comme le spectral decoupling (donc la DRBCE) modifie le rapport qu'à le modèle avec certaines variables, nous pensons qu'il est judicieux d'utiliser deux sous-ensembles de variables différents selon la fonction de perte utilisée. Il nous reste donc 1529 variables pour le modèle entraîné avec la BCE et 1478 pour le modèle entraîné avec la DRBCE.

Fonction de perte et sélection de variables	BODMAS			MalwareBazaar		
	ACC	**F1-Score**	**FNR**	**ACC**	**F1**	**FNR**
BCE, sans sélection	0.9247	0.9127	0.0769	0.3640	0.5333	0.6360
BCE, avec sélection	**0.9510**	**0.9436**	0.0389	0.3911	0.5616	0.6089
DRBCE, sans sélection	0.9079	0.8995	**0.0348**	0.4877	0.6552	0.5123
DRBCE, avec sélection	0.9262	0.9167	0.0468	**0.5031**	**0.6691**	**0.5083**

TABLE 14.5 – Comparaisons des performances des solutions de détection selon la fonction de perte et la sélection de variables

Comme le montre la table 14.5, l'utilisation d'un sous-espace de caractéristique permet d'améliorer la capacité de détection à court et long terme, indifféremment de la fonction de perte utilisée pour l'entraînement.

14.4.4 Modèle final et résultats

Notre modèle amélioré est un réseau de neurones, avec la même architecture que le modèle de référence (figure 14.5). Il est entraîné sur l'ensemble EMBER, réduit à l'aide de l'algorithme PFI, avec la fonction DRBCE. Pour éviter le sur-apprentissage, nous utilisons l'ensemble de validation qui contient les fichiers de EMBER les plus récents. Les tables 14.6 et 14.7 résument les résultats des différents modèles selon les diverses améliorations, avec la solution de référence (première ligne) et notre solution améliorée (dernière ligne).

Fonction de perte		Ensemble de val.		Réduction de var.		BODMAS	
BCE	**DRBCE**	**Aléatoire**	**Récent**	**Sans**	**Avec**	**ACC**	**F1-Score**
✓		✓		✓		0.9247	0.9127
✓			✓		✓	**0.9557**	**0.9487**
	✓	✓		✓		0.9079	0.8995
	✓		✓	✓		0.9128	0.9056
	✓		✓		✓	0.9258	0.9175

TABLE 14.6 – Performances des différents modèles évalués sur BODMAS

Fonction de perte		Ensemble de val.		Réduction de var.		MalwareBazaar	
BCE	**DRBCE**	**Aléatoire**	**Récent**	**Sans**	**Avec**	**ACC**	**F1-Score**
✓		✓		✓		0.3640	0.5333
✓			✓		✓	0.3679	0.5372
	✓	✓		✓		0.4877	0.6552
	✓		✓	✓		0.4971	0.6634
	✓		✓		✓	**0.5155**	**0.6798**

TABLE 14.7 – Performances des différents modèles évalués sur MalwareBazaar

Chacune des contributions améliore graduellement les performances de la solution de référence. En évaluant les différents modèles sur MalwareBazaar, nous pouvons observer une augmentation du taux de détection de 15.2%. Indépendamment de la fonction de perte choisie, l'utilisation d'une ensemble de validation récent et d'un espace de variables réduit permet d'améliorer la capacité de généralisation du modèle. C'est le cas sur l'ensemble BODMAS où l'on observe une amélioration des performances malgré l'utilisation de la fonction de perte BCE. La figure 14.7 montre que l'amélioration des capacités de détection s'applique à tous les sous-ensembles de fichiers.

(a) Modèle de référence et modèle BCE-val.(b) Modèles de référence et amélioré sur Mal-aléatoire-sélection de variables sur BODMAS warBaazar

FIGURE 14.7 – Comparaisons des performances pour différents modèles

14.4.5 Test du protocole sur un modèle LightGBM

Nos contributions sont agnostiques du modèle. En effet, il est possible d'appliquer nos modifications à d'autres algorithmes de ML. Nous avons donc testé notre protocole sur un modèle de type LightGBM. Nous avons utilisé le modèle entraîné sur EMBER par ANDERSON et ROTH [AR18] en tant que point de comparaison. Cette solution est connue pour être relativement robuste et efficace dans la détection de fichiers malveillants, comme point de comparaison. La table 14.8 résume les performances du modèle LightGBM EMBER et de notre modèle LightGBM amélioré. Le modèle EMBER est très efficace sur BODMAS, avec une capacité de détection de 99%. Néanmoins, nous pouvons observer une diminution des performances sur l'ensemble MalwareBazaar. Notre solution de détection améliorée est plus efficace que la solution de EMBER avec une capacité de détection supérieur de 12.5%. Cela montre que notre protocole est efficace pour améliorer la résistance des modèles d'apprentissage supervisé face à la dérive conceptuelle.

Modèle	BODMAS			MalwareBazaar	
	Acc.	**F1**	**TFN**	**Acc.**	**F1**
LightGBM EMBER	**0.9902**	**0.9885**	0.0137	0.6366	0.7780
LightGBM amélioré	0.9499	0.9441	**0.0053**	**0.7603**	**0.8639**

Table 14.8 – Comparaison des deux fonctions d'erreurs

14.5 Conclusion and discussions

Dans ce chapitre, nous cherchons des solutions pour contrer l'effet de la dérive des données sur des solutions de détection de logiciels malveillants. Nous proposons trois solutions qui sont utilisables lors de l'entraînement d'un modèle de ML. La première solution consiste à utiliser un jeu de données plus récent que le jeu d'entraînement lors de la phase de validation du modèle. Cela permet d'éviter le sur-apprentissage du modèle sur des données plus vieilles. Ensuite, nous utilisons un algorithme de sélection de variables pour réduire l'espace des caractéristiques dans le but de garder seulement celles qui ont du sens pour la tâche de détection de fichiers malveillants. Enfin, notre contribution principale est la création et l'utilisation d'une fonction de perte appelée Drift Resilient Binary Cross-Entropy, qui est une fonction élaborée dans le but de limiter l'effet du drift en pénalisant, au cours de l'entraînement, les prédictions erronées.

De plus, nous avons utilisé ces améliorations pour l'entraînement d'un modèle de type LightGBM. Malgré une dégradation des performances sur le jeu de donnés BODMAS, nous avons pu observer une nette amélioration sur l'ensemble MalwareBazaar, ce qui montre que nos contributions fonctionnent pour différents types de modèle, et aident à généraliser pour la détection de nouveaux fichiers malveillants.

Il y a néanmoins quelques points à préciser. Même si nos contributions aident les modèles ML à mieux généraliser et à mieux gérer le problème du drift, les résultats à long terme restent faibles. Les modèles utilisés ne sont pas optimisés pour une tâche de détection, avec une architecture basique. De plus, une étude quantitative de l'effet du drift serait nécessaire pour comprendre comment nos contributions aident à la détection de nouveaux fichiers. Enfin, notre jeu de données récent, issu de MalwareBazaar, ne contient pas de fichiers bénins, en raison de la difficulté de s'en procurer, souvent dû à des problèmes de copyright et de licences. Le choix des paramètres de la fonction DRBCE a donc été fait pour minimiser le TFN, au risque d'augmenter le TFP. L'avantage de la fonction DRBCE est qu'elle peut être optimisée pour limiter les fausses prédictions positives, négatives ou un compromis entre les deux.

Notre étude montre que le problème du drift touche particulièrement le domaine de la détection de fichiers malveillants, souvent dû aux techniques employées par les créateurs de malware pour tromper les antivirus et solutions de détection.

Chapitre 15

Détection de Fichiers Binaires Modifiés

15.1 Introduction

Dans ce chapitre, nous nous intéressons à la problématique de l'obfuscation de fichiers malveillants, afin de les rendre indétectables. En particulier, nous souhaitons savoir s'il est possible d'identifier certains fichiers obfusqués à l'aide de réseaux de neurones. Nous proposons pour cela une approche unique avec un outil composé de plusieurs sous-modèles entraînés pour différentes tâches de détection.

Problématique

Les éditeurs de logiciels malveillants ont l'habitude d'utiliser des techniques d'obfuscation pour empêcher les antivirus de détecter leurs produits, et pour compliquer la tâche des analystes de malware. Certaines techniques comme la compression (packing) ou le chiffrement (encryption) [Asg+22] sont des méthodes utilisées traditionnellement par les éditeurs de logiciels bénins pour réduire la taille d'un fichier ou protéger des données sensibles et la propriété intellectuelle des entreprises. Ces méthodes ont été détournées et peuvent être utilisées pour modifier des fichiers malveillants afin de les rendre indétectables. De plus, Aghakhani et al. [Agh+20] ont montré que les outils de détection de malware, basés sur l'apprentissage supervisé, avait tendance à produire plus de faux positif dans le cas où les fichiers analysés étaient compressés, ce qui peut ralentir la détection de vraies menaces, mais aussi impacter la productivité des analyste à cause de la fatigue d'alerte [Ban21]. Muralidharan et al. [Mur+22] montrent la nécessité d'intégrer la détection de packing dans le domaine de la détection de fichiers malveillants car cela pourrait permettre de faciliter la découverte de variants inconnues dans le futur. Beaucoup de travaux se sont intéressés à l'identification de fichiers compressés [Gao+22a ; Li+19 ; Jia+23] mais, bien que les outils développés soient efficaces, ils se heurtent souvent à des problématiques connus comme les malware polymorphiques ou les exemples adverses, dans le cas des modèles d'IA.

Plan et contributions

Dans les sections 15.2 et 15.3 nous présentons les données et la méthode de prétraitement utilisées. Ensuite, nous décrivons les modèles développés dans la section 15.4, avec une attention particulière sur une approche unique de modèle multi-couches. Nous résumons rapidement les résultats dans la section 15.5 avant de discuter des points faibles et des points forts de notre proche dans la section 15.6.

15.2 Données

Dans ce chapitre, nous utilisons un jeu de données, contenant 22 835 instances, construit à partir de sources internes à l'entreprise Orange. Les fichiers bénins sont principalement issus de différentes versions de l'OS Windows, tandis que les fichiers malveillants ont été récupérés via des sandboxes et des outils d'analyse de malware. Ce jeu de données possède deux particularités intéressantes. Tout d'abord, les fichiers sont labellisés selon leur nature (bénin ou malware), mais aussi selon qu'ils aient été modifiés ou non, c'est-à-dire s'ils ont été chiffrés ou compressés. De plus, les fichiers malveillants contenus dans ce dataset ont été particulièrement difficiles à détecter, ils ont réussi à outrepasser certains logiciels AV. La figure 15.1 présente la distribution du jeu de données.

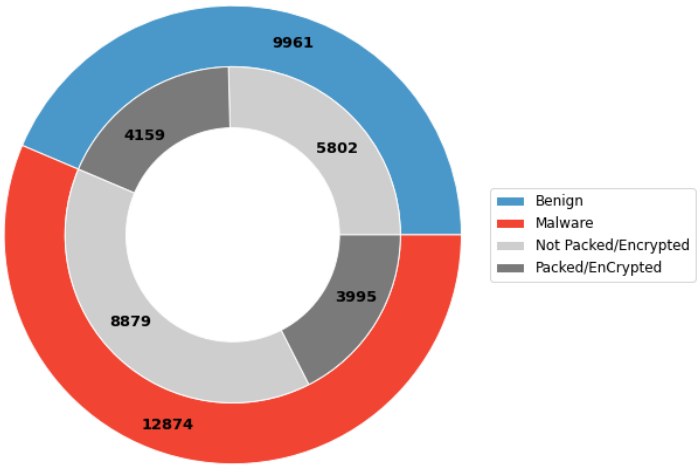


FIGURE 15.1 – Distribution du jeu de données

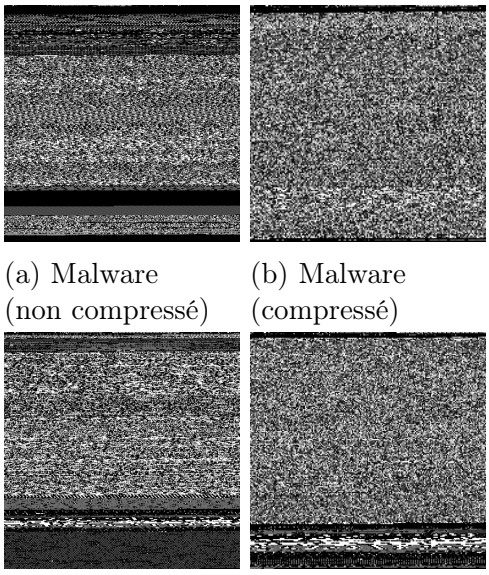
Par la suite, pour l'entraînement des différents modèles, nous utilisons 80% de cette base pour l'entraînement, 10% pour la validation et 10% pour l'évaluation des modèles. Au vue du peu de données, cette répartition est la plus optimisée pour garder un jeu d'entraînement suffisamment grand, et un jeu de test suffisamment complexe.

15.3 Prétraitement

Dans ce chapitre, nous utilisons la technique de prétraitement Grayscale (chapitre 4) qui permet de convertir un fichier binaire en image en niveau de gris. Cette transformation est définie par la fonction ϕ telle que :

$$\phi \colon U \longrightarrow [0, 255]^{h \times w}$$
$$u \longmapsto x,$$

où $u \in U$ est un fichier binaire, x est la représentation de u en matrice (image) de taille $h \times w$.



(a) Malware (non compressé) (b) Malware (compressé)

(c) Bénin (non compressé) (d) Bénin (compressé)

FIGURE 15.2 – Transformation Grayscale de fichiers bénins et malveillants, compressé et non compressé

Cette transformation permet d'identifier des comportements suspects dans un fichier. Par exemple, avec la figure 15.2, nous pouvons observer la même instance d'un fichier malveillant compressé et non compressé. Nous pouvons observer, dans la version compressée, une distribution plus aléatoire, contrairement à la version non compressée où l'on distingue les différentes sections du fichier.

15.4 Modèles

Comme les données sont sous forme d'image, nous utilisons des modèles de type réseaux de neurones convolutifs pour créer des outils de détection. Pour détecter les fichiers malveillants et les fichiers obfusqués, nous utilisons deux approches différentes. Avec la première approche, nous développons un modèle multi-tâches composé de trois sous-modèles ayant chacun un rôle particulier. Comme nous pouvons le voir avec la figure 15.3, le premier modèle a pour objectif de séparer les fichiers obfusqués et non-obfusqués. Si le fichier est obfusqué, il est envoyé vers un deuxième modèle, entraîné spécialement sur des fichiers chiffrés ou compressés, dont la tâche est la détection de malware. Sinon, il est envoyé vers un modèle entraîné uniquement sur des fichiers non obfusqués. Les trois modèles sont optimisés spécialement pour leur tâche.

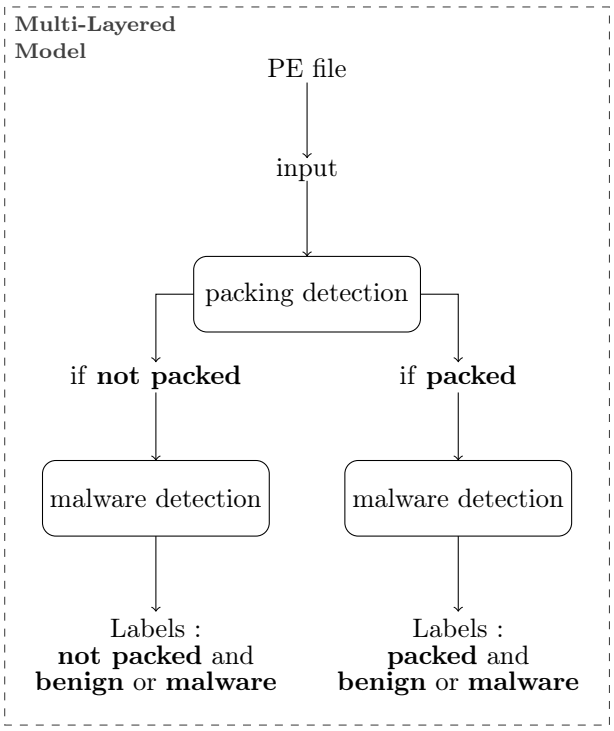


FIGURE 15.3 – Modèles multi-tâches pour la détection de fichiers compressés et de fichiers malveillants

Pour comparer notre approche multi-couches à une approche plus classique, nous entraînons aussi un modèle de classification à quatre labels qui sont $\{BP, BnP, MP, MnP\}$, appelé modèle benchmark. Ainsi, un fichier bénin non obfusqué sera étiqueté BnP (table 15.1). Avec un seul modèle, nous avons donc une double connaissance sur la nature d'un fichier. Nous retrouvons l'architecture de cet outil dans la figure 15.4.

TABLE 15.1 – Labels pour le modèle de classification

	Bénin	Malware
obfusqué	BP	MP
Non obfusqué	BnP	MnP

15.5 Expérimentation

Dans cette section, nous présentons les performances, avec la mesure d'accuracy, des modèles multis-couches et benchmark, résumés dans la table 15.2. Le modèle benchmark est plus efficace pour la détection de fichiers malveillants, mais légèrement moins performant pour identifier les fichiers compressés ou chiffrés. Globalement, les performances sont correctes, mais perfectible, avec un score d'accuracy inférieur à 90%, sûrement à cause du peu de données disponibles, et à leur complexité. Nous supposons aussi que le modèle multi-couches pourrait être meilleur avec plus de données. En effet, les deux sous-modèles pour la détection sont entraînés sur peu d'instances avec 8154 pour les données obfusquées et 14681 pour les données non-obfusquées.

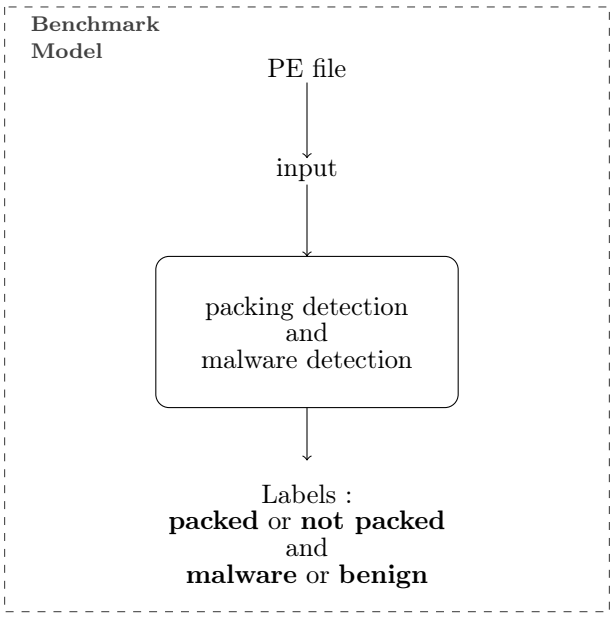


FIGURE 15.4 – Modèles benchmark pour la détection de fichiers compressés et de fichiers malveillants

TABLE 15.2 – Performances des modèles (ACC) pour la détection de malware et d'obfuscation

	Détection de malware	Détection d'obfuscation
Modèle multi-couches	0.8699	0.85
Modèle benchmark	0.8924	0.84

15.6 Conclusion

Dans ce chapitre, nous travaillons en particulier sur la détection de fichiers malveillants ayant subi de l'obfuscation (chiffrement, compression). Pour cela, nous utilisons une approche unique qui consiste à superposer plusieurs modèles ayant été entraînés pour des tâches simples et spécifiques. Malgré des résultats corrects, les modèles ne sont pas adaptés à une utilisation industrielle car leurs performances restent inférieures à 90% de précision, ce qui est peu, surtout pour des domaines liés à la cybersécurité. Malgré tout, la détection d'obfuscation peut renforcer la vigilance des analystes de malware. En effet, un fichier détecté comme bénin, mais fortement obfusqué, mériterait une analyse plus approfondie car il pourrait dissimuler une charge malveillante. Ces résultats montrent les limites de réseaux de neurones pour ce genre de tâche de détection (malware, obfuscation) avec peu de données, et des données complexes. Néanmoins, nous pensons que la méthode multi-modèle est une approche intéressante puisqu'elle fournit plusieurs informations à un analyste, et chaque modèle est indépendant donc optimisé pour une tâche particulière. De plus, si nous disposons à l'avenir de plus de données, par exemple de plus de données obfusquées, il est possible de ré-entraîner ou d'actualiser un des modèles sans modifier les autres, ce qui est un gain de temps et de ressources. Il est aussi possible d'ajouter des modèles spécifiques pour d'autres tâches, comme la

détection d'un type de logiciel en particulier (ransomware, trojan, etc.) (section 5.4).

Bibliographie

[AAA20] Mouhammd Alkasassbeh, Mohammad Abbadi et Ahmed Al-Bustanji. « LightGBM Algorithm for Malware Detection ». In : juill. 2020, p. 391-403. isbn : 978-3-030-52242-1. doi : 10.1007/978-3-030-52243-8_28.

[Abo+22] Faitouri A. Aboaoja et al. *Malware Detection Issues, Challenges, and Future Directions : A Survey*. 2022. doi : 10.3390/app12178482.

[AF22] Muhammad Shoaib Akhtar et Tao Feng. *Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time*. 2022. doi : 10.3390/sym14112308.

[Agh+20] Hojjat Aghakhani et al. « When Malware is Packin'Heat ; Limits of Machine Learning Classifiers Based on Static Analysis Features ». In : *Network and Distributed Systems Security (NDSS) Symposium 2020*. 2020.

[AR18] Hyrum S Anderson et Phil Roth. *EMBER : An open dataset for training static pe malware machine learning models*. 2018. arXiv : 1804.04637.

[Asg+22] Hassan Asghar et al. *SoK : Use of Cryptography in Malware Obfuscation*. Cryptology ePrint Archive, Paper 2022/1699. https://eprint.iacr.org/2022/1699. 2022. url : https://eprint.iacr.org/2022/1699.

[Ban21] Tao Ban. « Combat Security Alert Fatigue with AI-Assisted Techniques ». In : 2021. url : https://api.semanticscholar.org/CorpusID:236986309.

[Bar+22] Federico Barbero et al. *Transcending TRANSCEND : Revisiting Malware Classification in the Presence of Concept Drift*. 2022. doi : 10.1109/SP46214.2022.9833659.

[Baz+13] Zahra Bazrafshan et al. « A survey on heuristic malware detection techniques ». In : *The 5th Conference on Information and Knowledge Technology*. 2013, p. 113-120. doi : 10.1109/IKT.2013.6620049.

[Bre01] L Breiman. *Random Forests*. Oct. 2001. doi : 10.1023/A:1010950718922.

[CDW23] Yizheng Chen, Zhoujie Ding et David Wagner. *Continuous Learning for Android Malware Detection*. 2023. doi : 10.48550/ARXIV.2302.04332.

[Che18] Raymond Chen. *Why are the module timestamps in Windows 10 so nonsensical ?* https://devblogs.microsoft.com/oldnewthing/20180103-00/?p=97705. 2018.

[CS22] Ilias Chalkidis et Anders Søgaard. *Improved Multi-label Classification under Temporal Concept Drift : Rethinking Group-Robust Algorithms in a Label-Wise Setting*. 2022. arXiv : 2203.07856 [cs.CL].

[FRD19] Aaron Fisher, Cynthia Rudin et Francesca Dominici. *All Models are Wrong, but Many are Useful : Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously*. 2019. arXiv : 1801.01489 [stat.ME].

[Gao+22a] Xianwei Gao et al. « MaliCage : A packed malware family classification framework based on DNN and GAN ». In : *Journal of Information Security and Applications* 68 (2022), p. 103267. issn : 2214-2126. doi : https://doi.org/10.1016/j.jisa.2022.103267.

[Gao+22b] Yun Gao et al. *Malware Detection Using LightGBM With a Custom Logistic Loss Function*. 2022. doi : 10.1109/ACCESS.2022.3171912.

[Gem+20] Rosana Noronha Gemaque et al. *An overview of unsupervised drift detection methods*. 2020. doi : https://doi.org/10.1002/widm.1381. eprint : https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1381.

[Gon+14] Paulo Gonçalves Jr et al. *A Comparative Study on Concept Drift Detectors*. Déc. 2014. doi : 10.1016/j.eswa.2014.07.019.

[He+15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. doi : 10.48550/arxiv.1512.03385.

[HNA17] Andy Huynh, Wee Keong Ng et Kanishka Ariyapala. « A New Adaptive Learning Algorithm and Its Application to Online Malware Detection ». In : sept. 2017, p. 18-32. isbn : 978-3-319-67785-9. doi : 10.1007/978-3-319-67786-6_2.

[Hu+17] Donghui Hu et al. *The Concept Drift Problem in Android Malware Detection and Its Solution*. Sept. 2017. doi : 10.1155/2017/4956386.

[Huy22] C. Huyen. *Designing Machine Learning Systems : An Iterative Process for Production-ready Applications*. O'Reilly Media, Incorporated, 2022. isbn : 9781098107963.

[Jia+23] Lichen Jia et al. « ERMDS : A obfuscation dataset for evaluating robustness of learning-based malware detection system ». In : *BenchCouncil Transactions on Benchmarks, Standards and Evaluations* 3.1 (2023), p. 100106. issn : 2772-4859. doi : https://doi.org/10.1016/j.tbench.2023.100106. url : https://www.sciencedirect.com/science/article/pii/S2772485923000236.

[Jor+17] Roberto Jordaney et al. *Transcend : Detecting Concept Drift in Malware Classification Models*. Vancouver, BC, août 2017.

[Kan+21] Zeliang Kan et al. *Investigating Labelless Drift Adaptation for Malware Detection*. Nov. 2021. doi : 10.1145/3474369.3486873.

[Kum14] Vipin Kumar. *Feature Selection : A literature Review*. Juin 2014. doi : 10.6029/smartcr.2014.03.007.

[LH19] Ilya Loshchilov et Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv : 1711.05101 [cs.LG].

[Li+19] Xingwei Li et al. « A Consistently-Executing Graph-Based Approach for Malware Packer Identification ». In : *IEEE Access* 7 (2019), p. 51620-51629. doi : 10.1109/ACCESS.2019.2910268.

[LSB15] Vincent Lemaire, Christophe Salperwyck et Alexis Bondu. *A Survey on Supervised Classification on Data Streams*. Mars 2015. doi : 10.1007/978-3-319-17551-5_4.

[Lu+19] Jie Lu et al. *Learning under Concept Drift : A Review*. 2019. doi : 10.1109/TKDE.2018.2876857.

[Mal] MalwareBazaar. *MalwareBazaar Website*. https://bazaar.abuse.ch/. En ligne ; consulté le 24/05/2023.

[MM23] William Maillet et Benjamin Marais. *Neural Networks Optimizations Against Concept and Data Drift in Malware Detection*. 2023. arXiv : 2308.10821 [cs.CR].

[Mol22] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2e éd. 2022. url : https://christophm.github.io/interpretable-ml-book.

[Mur+22] Trivikram Muralidharan et al. « File Packing from the Malware Perspective : Techniques, Analysis Approaches, and Directions for Enhancements ». In : *ACM Comput. Surv.* 55.5 (déc. 2022). issn : 0360-0300. doi : 10.1145/3530810.

[Nar+16] Annamalai Narayanan et al. *Adaptive and scalable Android malware detection through online learning*. 2016. doi : 10.1109/IJCNN.2016.7727508.

[Pez+21] Mohammad Pezeshki et al. *Gradient Starvation : A Learning Proclivity in Neural Networks*. 2021. arXiv : 2011.09468 [cs.LG].

[Poh+22] Joona Pohjonen et al. *Spectral decoupling for training transferable neural networks in medical imaging*. Fév. 2022. doi : 10.1016/j.isci.2022.103767.

[SCS20] Siddharth Singhal, Utkarsh Chawla et Rajeev Shorey. *Machine Learning & Concept Drift based Approach for Malicious Website Detection*. 2020. doi : 10.1109/COMSNETS48256.2020.9027485.

[Son23] SonicWall. *SonicWall 2023 Cyber Threat Report*. https://www.sonicwall.com/2023-cyber-threat-report/. Accessed : 2023-05-06. 2023.

[TKH19] Jirí Tumpach, Marek Krcál et Martin Holeňa. *Deep Networks in Online Malware Detection*. 2019.

[UAB19] Daniele Ucci, Leonardo Aniello et Roberto Baldoni. « Survey of machine learning techniques for malware analysis ». In : *Computers and Security* 81 (2019), p. 123-147. issn : 01674048. doi : 10.1016/j.cose.2018.11.001. arXiv : 1710.08189.

[VA19] B. Venkatesh et J. Anuradha. *A Review of Feature Selection and Its Methods*. 2019. doi : doi:10.2478/cait-2019-0001.

[VT] Virustotal. *Virustotal Website*. https://www.virustotal.com/. En ligne, consulté le 20/04/2023.

[Xu+19] Ke Xu et al. *DroidEvolver : Self-Evolving Android Malware Detection System*. 2019. doi : 10.1109/EuroSP.2019.00014.

[Yan+21a] Limin Yang et al. « BODMAS : An Open Dataset for Learning based Temporal Analysis of PE Malware ». In : *Proceedings - 2021 IEEE Symposium on Security and Privacy Workshops, SPW 2021*. 2021, p. 78-84. isbn : 9781728189345. doi : 10.1109/SPW53761.2021.00020.

[Yan+21b] Limin Yang et al. *CADE : Detecting and Explaining Concept Drift Samples for Security Applications*. Août 2021.

Résumé

Titre : Améliorations des outils de détection de malware par analyse statique grâce à des mécanismes d'intelligence artificielle

Mot clés : Analyse de logiciels malveillants, Intelligence artificielle, Apprentissage profond, Apprentissage par renforcement

Résumé : Cette thèse porte sur l'analyse de fichiers malveillants. Nous nous intéressons particulièrement à l'utilisation de l'intelligence artificielle pour développer et améliorer des outils d'analyse de malware. Dans un premier temps, nous abordons le problème d'un point de vue défensif en proposant des outils d'analyse, et en élaborant de nouveaux modèles de détection de logiciels malveillants basés sur l'apprentissage supervisé et l'apprentissage profond. Nous proposons aussi une approche offensive, basée sur des méthodes d'apprentissage par renforcement, dans le but de contourner différentes solutions de détection. Cette méthode permet d'étudier leurs failles et leurs faiblesses afin d'évaluer leur efficacité.

Abstract

Title : Static analysis malware detection tools enhanced by artificial intelligence mechanisms

Keywords : Malware analysis, Artificial intelligence, Deep learning, Reinforcement Learning

Abstract : This thesis focuses on the analysis of malicious files. We are particularly interested in using artificial intelligence to develop and improve malware analysis tools. Firstly, we propose a defensive approach, by providing analysis tools and by developing new malware detection models based on supervised and deep learning. We also propose an offensive approach, based in reinforcement learning methods, with the purpose of bypassing various detection solutions. This method allows us to examine their vulnerabilities and weaknesses in order to assess their effectiveness.