الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université de 8 Mai 1945 – Guelma -

Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Département d'Informatique

Mémoire de Fin d'études Master

Filière : Informatique

Option : Systèmes Informatiques

Thème :

Un Framework de sécurité pour détecter les malwares Android

Encadré Par :

Dr. Ferrag Med Amine

Présenté par :

Cherifi Abd erraouf

Septembre 2021

Remerciements

Nous tenons à la fin de ce travail de remercier avant tout notre Dieu,

Le tout puissant d'avoir donné le courage, la patience et la santé pour réaliser ce modeste travail.

Puis, Je remercie vivement mon directeur de recherche

Monsieur FERRAG MOHAMED AMINE Pour ses conseils judicieux, ses encouragements et son accompagnement.

Je Remercie également Hamouda Djallel et Boughida Adel pour leurs

Efforts et leurs dévouements, qui m'ont montré le chemin d'un travail de qualité.

Mes remerciements les plus vifs s'adressent aussi aux membres de jury

Pour l'intérêt qu'ils ont porté à ma recherche tout en acceptant d'évaluer ce travail.

Je tiens aussi à exprimer mes sincères remerciements à tout les enseignants et les étudiants du département d'informatique.

Enfin je remercie tous ceux qui ont contribué de près ou de loin

À la réalisation de ce mémoire

Dédicace

J'ai l'honneur de dédier ce modeste travail :

À ma source de joie, ceux qui ont toujours veillé sur mon bonheur,

Qui ont sacrifié pour me voir réussi et qui m'ont comblé tant d'amour et de

Tendresse, mes chers parents «RABAH et DJELAILIA FAHIMA». Ils ont été toujours

Présents à mes côtés par leurs

Sacrifices et leurs prières. Que Dieu leur procure une longue vie avec une bonne

Santé.

À vous, ma plus belle sœur «Dr. CHERIFI WEDJEDAN» qui a partagé Avec

Moi tous les moments d'émotion lors de la réalisation de ce travail, merci pour

votre support.

À vous aussi mon cher oncle «Dr. DJELAILIA KARIM» qui m'a guidé par ses

précieux conseils et ses orientations, merci KHALOU.

Je dédie aussi ma chère tante «DJELAILIA NADJETTE» merci pour votre soutien

infini et votre aide incessante.

À mon frère «MEDJELAKH MOHAMED ANIS» et mes chers amis

Avec qui j'ai partagé les meilleurs et les plus agréables moments de mon parcours

universitaire : AMINE, MONCEF, YASSER.

À tous ceux qui, par un mot, m'ont donné la force de continuer…

À tous ceux qui m'aiment et que j'aime…

Résumé

Les systèmes de détection des malwares ont fait l'objet de nombreuses recherches et jouent un rôle important dans la cybersécurité. L'objectif de cette étude est de modéliser un tel système pour aider les administrateurs et les utilisateurs du système à détecter et à identifier toute violation de la sécurité dans leur organisation afin de les prévenir avant de causer des dommages. Pour cela, nous avons étudié les performances des méthodes d'apprentissage machine (ML) appliquées à la détection des malwares pour la cybersécurité. Ensuite, nous avons appliqué une technique de détection basée sur une approche d'apprentissage profond, un réseau neuronal convolutif (CNN) pour détecter les malwares dans les connexions réseau. Nous avons évalué la méthode proposée avec l'ensemble de données CIC-AAGM2017 de référence pour la détection des malwares sur les réseaux. Nous avons également présenté une étude comparative avec des autres algorithmes d'apprentissage automatique de référence, en utilisant différentes mesures appliquées pour l'évaluation des performances d'apprentissage machine (Précision, Rappel, score F1), et deux autres indicateurs de performance importants pour la détection des malwares (taux de détection, taux de fausses alarmes). Les résultats expérimentaux ont montré que les performances de cette approche DL proposée sont supérieures à celles des algorithmes ML traditionnels en tant que modèles de détection avec une grande précision, un taux de détection idéale et un taux de fausses alarmes négligeable.

Mots clés : Cybersécurité, Système de détection des malwares, L'apprentissage profond, L'apprentissage automatique, Machine learning, CIC-AAGM2017.

Abstract

Malware detection systems have been the subject of much research and play an important role in cybersecurity. The objective of this study is to model such a system to help system administrators and users detect and identify any security violations in their organization in order to prevent them before causing damage. For this purpose, we studied the performance of machine learning (ML) methods applied to malware detection for cybersecurity. Then, we applied a detection technique based on a deep learning approach, a convolutional neural network (CNN) to detect malware in network connections. We evaluated the proposed method with the benchmark CIC-AAGM2017 dataset for network malware detection. We also presented a comparative study with other benchmark machine learning algorithms, using different metrics applied for machine learning performance evaluation (Accuracy, Recall, F1 score), and two other important performance indicators for malware detection (detection rate, false alarm rate). The experimental results showed that the performance of this proposed DL approach is superior to traditional ML algorithms as detection models with high accuracy, ideal detection rate, and negligible false alarm rate.

Key Words : Cybersecurity, Malware detection system, Deep learning, Machine learning, Machine learning, CIC-AAGM2017.

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Le développement accéléré des technologies de l'information et de la communication (TIC) a poussé les concepteurs d'applications à mettre sur le marché une quantité phénoménale d'applications. La concurrence dans ce marché est à son comble. Certains opérateurs n'excluent pas l'utilisation de techniques peu légales en vue de s'octroyer des parts de marché. Leurs méthodes sont souvent basées sur l'espionnage, l'intrusion, la déformation, etc. Ils créent pour cela des applications contenant du code malveillant et présentent souvent leur marchandise dans le cadre de campagnes promotionnelles ou in extremis une gratuité d'utilisation. Les utilisateurs peu avertis sont souvent victimes de ces manœuvres. La recherche pour pallier à ces pratiques est aujourd'hui très active. C'est la cyber-sécurité.

Beaucoup de travaux dans ce domaine sont élaborés et arrivent à endiguer efficacement le phénomène. Mais les agents mal intentionnés développent eux aussi des techniques de plus en plus élaborées en vue de rester nocifs.

La problématique :

Le problème rencontré dans le domaine de la cyber-sécurité étant qu'à chaque fois qu'il y a des applications malveillantes, les concepteurs du domaine lui appliquent des solutions sur mesure. Les efforts consentis sont de plus en plus croissants car les intrus sont de plus en plus sophistiqués. Un nouveau paradigme doit être adopté en vue de limiter ces efforts.

L'intelligence artificielle par ses techniques d'apprentissage et d'autonomisation permettrait d'arriver à bout de ces pratiques

Le but de travail:

Le but de ce travail étant de dresser un état de l'art dans le domaine de la cyber-sécurité et spécialement dans le domaine d'apprentissage automatique. Nous étaierons les techniques les plus prometteuses et appliqueront celle qui est en vogue, en l'occurrence le « deep learning » ou l'apprentissage profond. Nous démontrerons que les techniques dans ce domaine permettraient d'accroitre l'efficacité des systèmes de détection et par conséquent permettraient de meilleures contre attaques.

CHAPITRE 1 :

Malware *(logiciel malveillant)*

Les logiciels malveillants (MALWARES)

1.1 Introduction :

Avec le développement rapide d'internet, les logiciels malveillants sont devenus l'une des principales cyber-menaces actuelles qui porte un grand danger pour nos machines. Récemment les chiffres publiés concernant la détection de ces derniers sont terrifiants. Les réseaux de nos jours ainsi que les systèmes d'information connectés sont confrontés à de véritables menaces intentionnelles ou accidentelles à cause de cette quantité phénoménale de logiciels malveillants. Dans ce chapitre nous allons présenter une définition détaillée du malware, quelques types existants et leurs familles, ainsi que le fonctionnement de chacun d'entre eux, et leur pénétration dans un système.

En 1971, Bob Thomas a développé le premier virus informatique avec le nom Creeper comme une première version déplacée dans les nœuds de l'ARPANET [1]. De là, le démarrage des premiers travaux, qui ont consisté à fournir une compréhension nécessaire pour concevoir des contre-mesures efficaces et des stratégies d'atténuation contre les différents logiciels malveillants et dresser un bilan de ces types.

1.2 Définition d'un malware :

Dans les systèmes d'information, Tout logiciel effectuant des actions malveillantes, notamment le vol d'informations, l'espionnage, etc. peut être qualifié de logiciel malveillant. Kaspersky Labs définit les logiciels malveillants comme "un type de programme informatique conçu pour infecter l'ordinateur d'un utilisateur légitime et lui infliger des dommages de plusieurs façons" [2].

D'après cette définition, malware, où logiciel malveillant est un terme générique pour tous les types de logiciels informatiques à visées malveillantes. Chaque malware appartient à un échantillon de malwares. Cet échantillon malicieux correspond à un type de fichier qui contient ce malware, Par exemple un fichier APK. Une famille de malwares est l'ensemble de ces échantillons qui appartiennent au même type, avec la même façon d'attaque et de nuire à un système donné. Pour comprendre les fonctionnements et les impacts potentiels d'un malware, il faut analyser un où plusieurs de ces types, dans le but de connaitre les caractéristiques et les informations liées à ce malware.

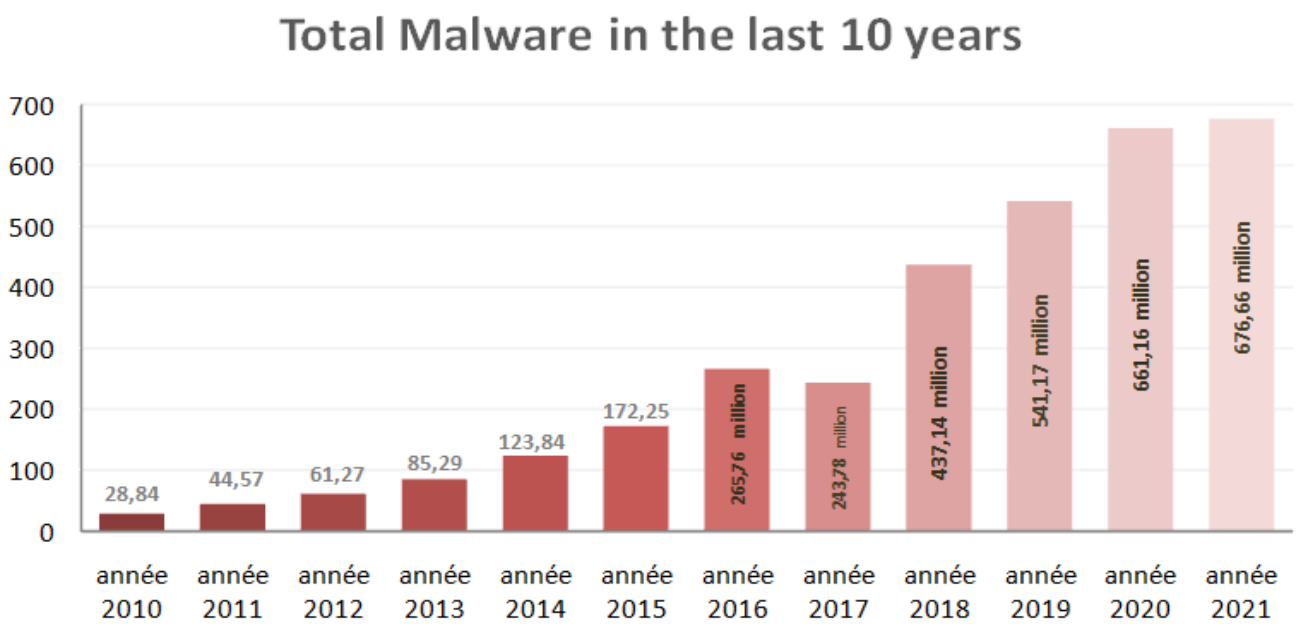

Figure 1.1: Statistique des détections des malwares 2010-2021 [W1].

1.3 Classification des malwares :

Le développement fulgurant de la technologie depuis ces dernières années a conduit à la multiplication et l'évolution des techniques de détection des malwares, Ces techniques ont permis de détecter une quantité phénoménale des types de logiciels indésirables ou malveillants. Parmi ces types :

Virus informatique

Récemment, ce terme est devenu le plus mentionné dans les malwares. Leur nom est dérivé par l'analogie des virus en biologie. Un virus informatique est un programme qui a les propriétés suivantes: infection, multiplication, possession d'une fonction nocive...etc. Et attaque un logiciel légitime, un fichier du system où un document…

Les catégories des virus informatique sont multiples, ils diffèrent par leur fonction et leur façon d'aggravation utilisée. Parmi eux :

-**Le virus classique :** aussi nommé virus parasite, ce virus est une partie du programme qui est lié à un fichier exécutable légitime.

-Les macrovirus : se sont des programmes qui utilisent le langage de programmation des macro-commandes. Ils attaquent principalement les fichiers des utilisateurs. Leur extension est due au fait qu'ils s'intègrent à des fichiers très échangés (comme les documents Word… etc.) avec l'ajout des macro-commandes dans leur code.

-Les virus de secteur d'amorçage : s'approprient au secteur de démarrage d'un support amovible (comme disque dur, CD-ROM, disquette, clef USB…etc.) Pour s'exécutent automatiquement.

Ver informatique (computer Worms)

Un ver informatique est un logiciel malveillant qui se propage sur un réseau informatique comme l'internet, pour infecter le maximum de systèmes. Il a la capacité de se dupliquer une fois qu'il a été exécuté. Contrairement au virus, Les vers informatiques peuvent exploiter les erreurs de configuration des réseaux. Le ver se dissémine sans avoir besoin de se lier à d'autres programmes exécutables. Il exploite les failles de sécurité pour se répandre d'un ordinateur à un autre. [5]

Il existe deux catégories de vers informatiques qui se distinguent par leur méthode de propagation, la première méthode est basée sur l'envoi un courrier électronique infecté et le ver se propage en utilisant le carnet d'adresses d'un client de la messagerie électronique. Pour la deuxième méthode, les vers analysent les réseaux afin d'identifier les serveurs vulnérables.

Cheval de Troie (Trojan Horse)

Tire son nom de la tactique des grecs dans leur guerre historique (l'épopée l'Iliade d'Homère). Ce type de logiciels est distingué à d'autres par sa capacité de paraître légitime et anodin, mais une fois introduit dans le système il sera malveillant. Contrairement au virus et vers, Les chevaux de Troie cherchent à encarter des programmes malveillants au système infecté, et ne cherchent pas à se propager.

Les chevaux de Troie peuvent offrir au hacker la possibilité d'effectuer différentes tâches à distance dans la machine infectée, tels que la suppression des fichiers, le téléchargement de fichiers, modifier les paramètres du registre, arrêter l'exécution de certaines applications telles que les anti-virus, etc. La figure 1.2 illustre le fonctionnement d'un cheval de Troie.

.

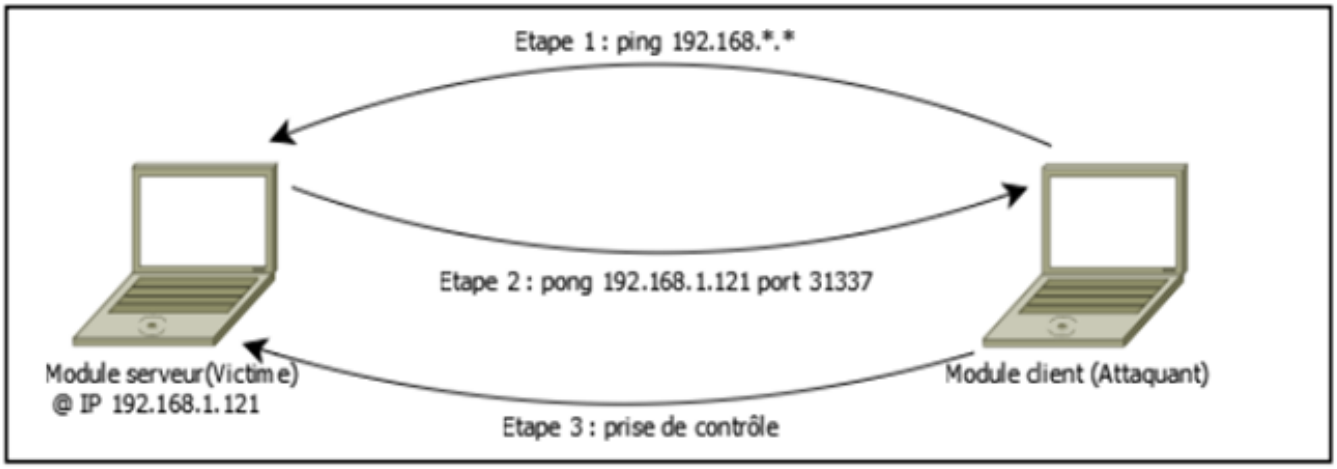

Figure 1.2: Mode opératoire d'un cheval de Troie [49].

Porte dérobée (backdoor)

La porte dérobée est une fonctionnalité inconnue de l'utilisateur légitime. Son origine est un cheval de Troie caché dans un programme où un service en ligne. Souvent installée par un développeur du logiciel où un fournisseur du service. Après sa réussite d'accès au système, la porte dérobée va espionner un utilisateur, gérer ses fichiers, installer des logiciels supplémentaires ou des malwares et surveiller l'ensemble du système du PC. Même si les failles de sécurité présentes sur le système sont corrigées, elle ne va pas être détectée.

Les keyloggers

Le rôle d'un Keylogger est de capturer les caractères saisis sur un clavier d'une machine. Les keyloggers sont généralement programmés pour s'exécuter au démarrage du système, et enregistrent dans des fichiers (journaux) tout caractère saisi que ce soit dans les courriers, les messages, les documents et même les noms d'utilisateurs et mots-passes. Cela se fait généralement à l'aide d'une fonction Hook [47]. Un keylogger peut être installé par un administrateur dans une entreprise afin de contrôler les activités des employés, comme il peut être installé par un Hacker afin d'obtenir des informations sur ses victimes. La figure 1.3 illustre le mécanisme de fonctionnement d'un Keylogger

.

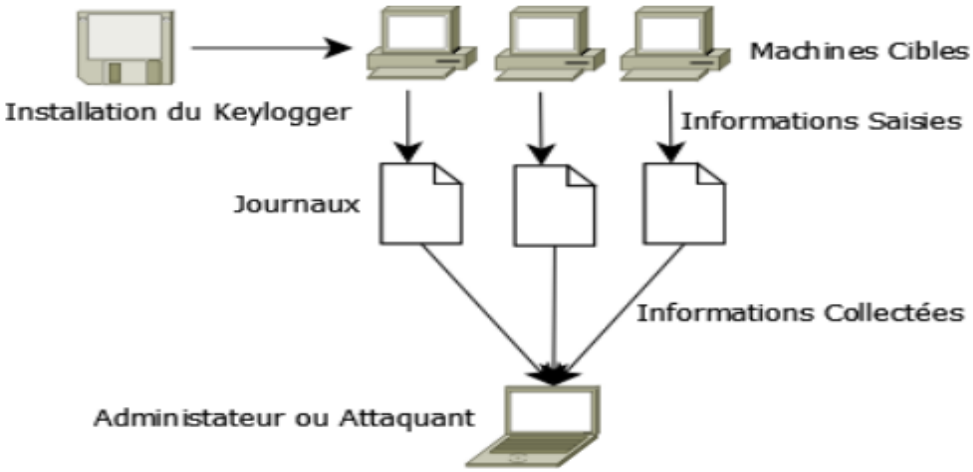

Figure 1.3: Mode opératoire d'un Keylogger. [47].

Rançongiciel (ransomware)

Le rôle de ce type est d'extorquer des données personnelles aux victimes. Il est nommé aussi un logiciel rançonneur ou un logiciel d'extorsion. Le Rançongiciel est créé pour prendre en otages les données personnelles des victimes et fait un chantage pour payer une rançon pour l'attaquant, sinon l'utilisateur infecté ne pourra pas utiliser son système d'information et ses documents.

Logiciel de sécurité falsifié (rogueware)

On peut l'appeler aussi rogue. Ce type de logiciel s'annonce comme un logiciel de sécurité, un anti-virus où un anti-spyware, mais en réalité c'est un faux logiciel où un malware qui sera vendu par des développeurs des programmes où de fausses sociétés. Ce programme commence par convaincre les utilisateurs que leur machine est infectée par des malwares, mais quand ils l'achètent, c'est lui-même qui sera un malveillant.

Rootkit

Comme un fantôme, les Rootkit sont des programmes malveillants, autonomes et très minutieux. Leur méthode est d'effacer les traces de l'attaquant sur la machine infectée. Leur existence est cachée au niveau du registre du système. Le but étant d'obtenir un accès à un ordinateur le plus furtivement possible [7]. L'utilité d'un rootkit est : espionner et accéder aux données stockées ou en transit sur la

machine cible. Avec ces rootkits, le pirate peut aussi intercepter le trafic Internet, lire les e-mails et suivre les saisies clavier de la machine infectée.

Logiciel espion (Spyware)

Ce type de logiciel est localisé dans des logiciels gratuits, des pages internet et des sites web proposant des contenus illégaux. L'espiogiciel a plusieurs méthodes d'espionnage et de propagation. On peut citer : l'affichage des offres publicitaires pour le téléchargement d'autres logiciels infectés, la capture des mots de passe en enregistrant les touches sélectionnées au clavier, l'espionnage des programmes exécutés à telle où telle heure ainsi que l'espionnage des sites Internet visités [8].

Malwares / propriété	virus	Trojan	Worm	Spyware	Adware	Rootkit
Auto réplication	+	-	+	-	-	+
Espionnage	+	+	+	+	-	+
Composants cachés	-	-	+	-	-	+
Déni de service	+	+	+	-	-	+
Auto-propagation	+	+	+	-	-	+
Furtivité	-	+	+	+	-	-
Exécutable	-	-	+	-	-	+
Insérer une charge dans la cible	+	-	+	-	+	+
Publicité	-	-	-	-	+	-

Tableau 1.1 : Comparaison de fonctionnement entre quelques types des malwares [W7].

+ : à cette propriété

- : n'as pas cette propriété

1.4 Actions effectuées par les malwares:

Tous les types des logiciels malveillants partagent une ou plusieurs actions, dont le but est de nuire un système donné. En peut classifier ses actions en quatre parties :

- Violation des privilèges et l'écrasé
- Contrôle à distance
- Vol de données
- Espionnage sur des informations privées.

1.5 Les techniques d'obscurcissement:

Afin de rendre leurs codes malveillants difficiles à détecter, les attaquants utilisent des méthodes d'obscurcissement, qui se divisent en trois catégories et qui sont : la compression (en anglais Packing), le polymorphisme et le métamorphisme. Dans ce qui suit, nous allons présenter ces différentes techniques.

1.5.1 La compression (Packing) :

Le Packing signifie la compression d'un fichier exécutable, qui une fois lancé, va procéder à sa décompression avant son exécution effective. Cette technique a été développée dans un but légitime qui est la protection des logiciels commerciaux des risques de piratage, cependant elle a été utilisée par les créateurs de malwares afin de rendre leurs programmes difficiles à détecter, ainsi que pour faciliter leur propagation dans les réseaux en réduisant leurs tailles [35]. La figure 1.4 présente la structure d'un fichier PE avant et après sa compression.

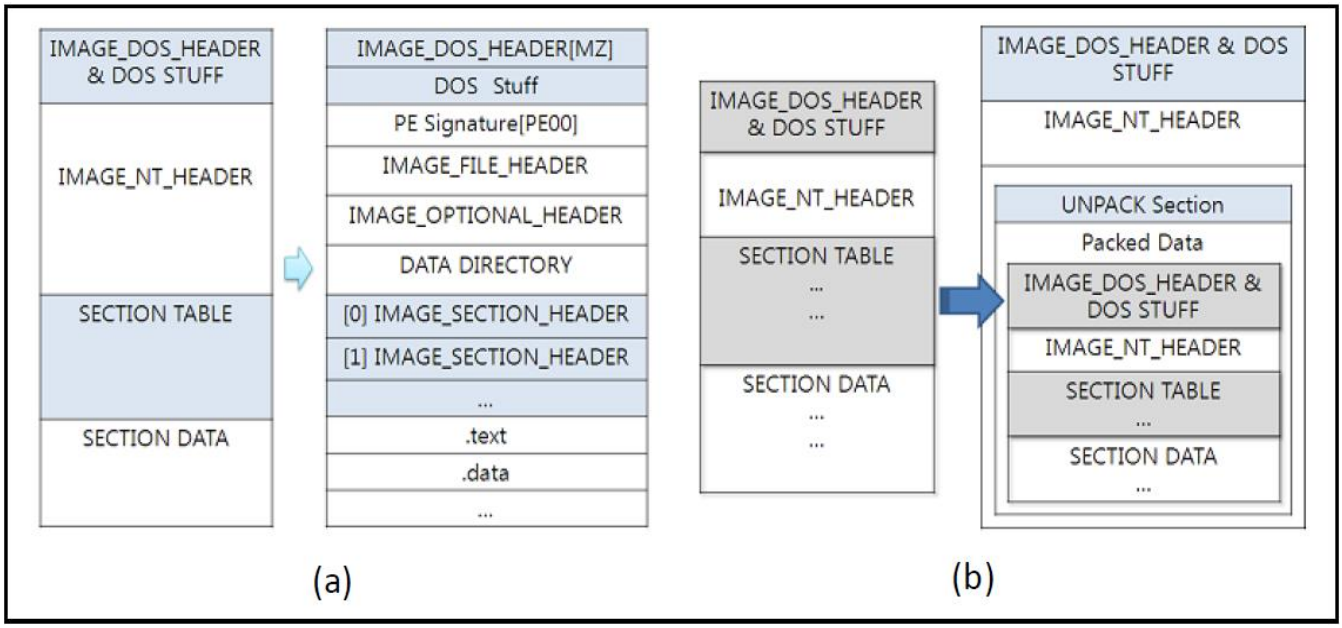

Figure 1.4: Comparaison entre la structure de deux programmes PE, l'un non-compressé (a) et l'autre compressé (b) [35].

La compression permet donc de changer la structure du code exécutable tout en conservant la même fonctionnalité, ce qui le rend difficile à détecter par l'antivirus classique.

1.5.2 Le polymorphisme:

Les malwares dits polymorphes consistent principalement à crypter le programme cible en utilisant différentes clés cryptographiques et fournir en même temps le mécanisme de déchiffrement, dans le but de rendre certaines portions de leur code illisible [37].
O'Kane et al. [45] ont présenté un exemple simplifié du fonctionnement d'un malware polymorphe. Selon cet exemple, le malware, après avoir été exécuté commence par décrypter le code du malware à l'aide de la clé de chiffrement qui est stockée au niveau du fichier malicieux. Le code est ensuite chargé en mémoire, ce qui engendre l'exécution du code malicieux (la charge). Une fois cette phase terminée, une nouvelle clé de chiffrement est générée. Elle est utilisée afin de crypter une nouvelle fois le code du malware. Le fait de générer une nouvelle clé, permet d'obtenir une nouvelle version du malware à chaque exécution.

1.5.3 Le métamorphisme:

Un malware métamorphique, est capable de changer sa structure interne, en modifiant son code machine lors de son chargement en mémoire, avant de le réécrire à nouveau dans le fichier hôte, tout en conservant le même comportement (fonctionnalité) [38].

1.6 La pénétration du malware dans un système:

Pour Comprendre le fonctionnement d'un malware dans un système il faut répondre à plusieurs questions :

Quel est l'objectif du malware ?
Quelles sont ses actions ?
Comment se propage-t-il ?
Quelle est son origine ?
Selon les travaux de Zhu et al [28]. La méthode d'infection principale utilisée par les développeurs de malware est d'ajouter leur code malicieux à des applications existantes et proposer les versions infectées de ces applications en téléchargement sur les plateformes de téléchargement tel que Google Play.

Les attaquants ont créé leurs virus, leurs vers et leurs chevaux de Troie pour le diffuser sur autant d'ordinateurs ou de téléphones mobiles que possible. Leur objectif est rendu possible par deux manières :

- La 1ere est nommée **l'ingénierie sociale**, Cette manière est basée sur l'encouragement de l'utilisateur à ouvrir un fichier ou une page infectée d'une manière sournoise. Le rôle des attaquants consiste à attirer l'attention de l'utilisateur sur le lien ou le fichier infecté, ou avec de faux courriers électroniques provocants. Une fois le fichier ouvert, les vers cachés commençaient à créer des copies d'eux-mêmes sur le disque et se lancent d'une façon cachée au système et sans l'intervention de l'utilisateur [11].
- La 2e manière est basée sur les vulnérabilités dans la logique opérationnelle de plusieurs programmes. Les systèmes d'exploitation actuels disposent d'une structure complexe ainsi que de vastes fonctionnalités qui rendent presque impossible de développer un programme sans erreur [25]. Cette vulnérabilité est devenue un moyen pour infiltrer dans un programme de la part de l'attaquant, avec des techniques qui introduisent de manière dissimulée un code malveillant ou l'ajout des macro-commandes dans le code d'un programme légitime.

1.7 Conclusion:

Nous avons présenté dans ce chapitre les malwares en général, leurs types et leurs fonctions. Nous avons introduit une base de connaissance sur ces derniers utile pour notre étude. La quantité phénoménale des malware a été détectée par un système de détection qui nous a permis de connaitre ces types et ces familles.

Dans le 2eme chapitre nous allons présenter les méthodes de détection des malwares, ainsi que le fonctionnement de chacun d'eux et finir par une comparaison entre ces derniers.

CHAPITRE 2 :

Système de détection des malwares

Système de détection des malwares

2.1 Introduction :

Afin de lutter contre certains types de menaces informatiques, il y a nécessité d'un système de détection. Ce dernier fournit la compréhension nécessaire pour concevoir des contre-mesures efficaces et des stratégies d'atténuation contre les différents logiciels malveillants. Dans ce 2eme chapitre nous allons présenter des systèmes de détection des malwares, ainsi que le fonctionnement de chacun entre eux, et la déférence entre leurs principes de détection.

2.2 Définition:

Le système de détection des malwares est basé sur les signatures qui sont un ensemble de propriétés communes pour les fichiers de logiciels malveillants qui sont dans le cas des applications Android des fichiers apks. Pour révéler les fichiers potentiellement malveillants, La détection a principalement deux méthodes: l'analyse statique et l'analyse dynamique. Chacune entre les deux à son propre principe de détection.
L'analyse, statique ou dynamique, sert à construire ces signatures et profils comportementaux d'un malware ou en d'autres termes : l'ensemble de propriétés communes à l'exécution des échantillons du malware. Les méthodes d'apprentissage sont actuellement utilisées pour classifier et détecter des anomalies.

L'analyse d'une application est faite dans le but de l'extraction des informations liées à l'application telle que les ressources qu'elle a besoins d'utiliser, son comportement et tout ce qui est attendu pour exécuter une partie de son code [42].

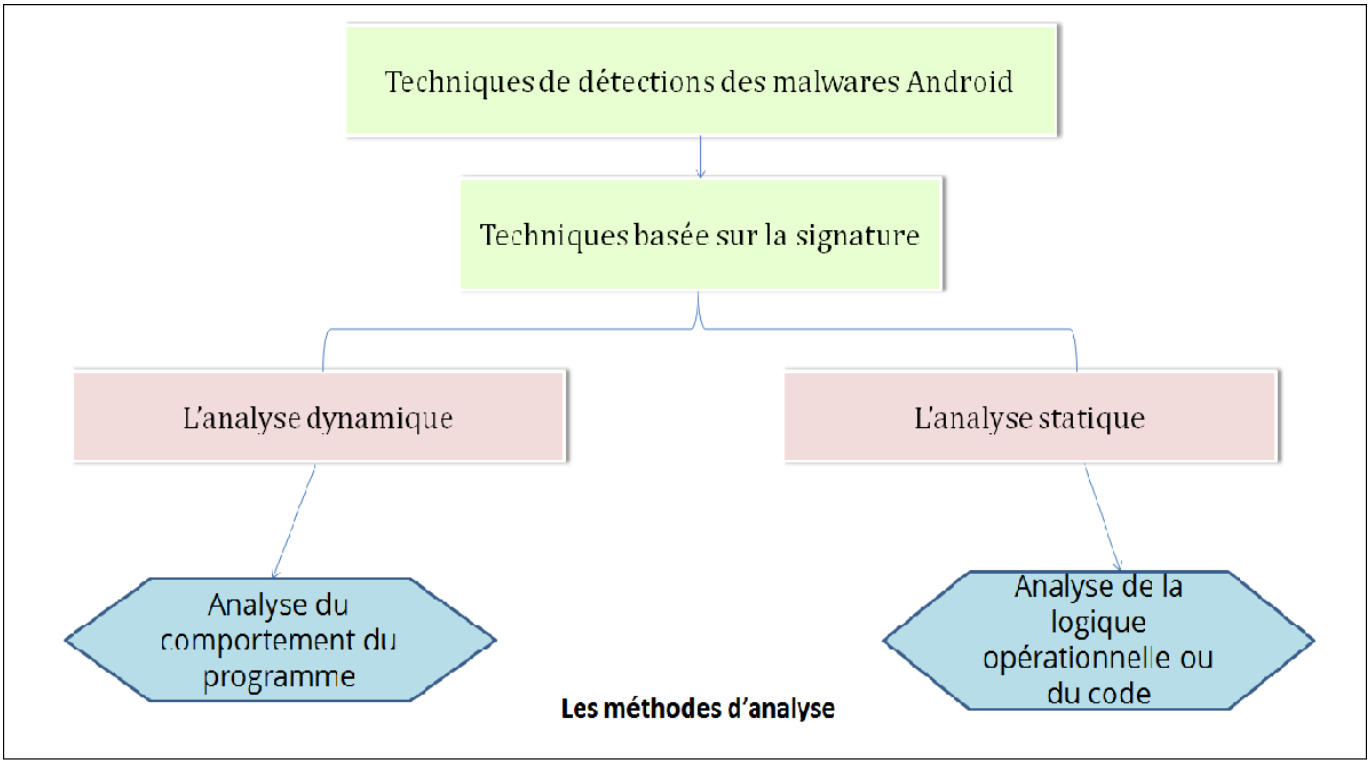

Figure 2.1: Taxonomie du système de détection des malwares.

2.3 Analyse statique :

L'analyse statique où l'analyse du code, est une technique basée sur la décompilation d'application afin d'en étudier le code. Cette méthode appelée aussi **reverse-engineering**, n'est pas efficace à 100% à cause du temps d'analyse qui peut être considéré comme du temps en moins pour tester la sécurité de l'application. La détection des vulnérabilités du côté client est effectuée sans avoir à lancer le code, mais ceci est trop compliqué dans la pratique. Donc l'analyse du code a pour but d'avoir une idée sur la sécurité globale de l'application par la détection de troubles de sécurité qui sont plus où moins évidents. Pour effectuer ce type d'analyse il vaut mieux d'utiliser soit le Framework Androguard où bien d'utiliser ARE [41]. Le premier facilite le travail d'analyse (permissions, instructions dangereuses, similarité entre 2 applications, etc.) et la deuxième est une machine virtuelle qui contient une ancienne version du premier.

L'analyse de programmes incluant la recherche d'erreurs possibles à l'exécution, est donc indécidable : il n'existe aucune méthode qui peut toujours répondre sans se tromper qu'un programme puisse ou non produire des erreurs à l'exécution.

Bien qu'il soit impossible de détecter toutes les erreurs présentes dans tous les programmes, ce n'est pas une raison pour abandonner complètement les méthodes d'analyse statique. Il est possible dans la majorité des cas de détecter une grande quantité des problèmes présents. Il faut utiliser des méthodes

qui marchent raisonnablement bien sur la majorité des programmes réels. Il s'agit donc de méthodes approximatives. Ces méthodes approximatives ne sont pas un manque de rigueur [40]. Les techniques appliquées nécessitent d'être fiables mais pas forcément optimal. Comme il est possible que certaines erreurs restent présentes dans le système. Les outils d'analyse statistique ne vont pas pouvoir détecter exactement si il y'a une erreur où non. Donc il y'a toujours un risque que les outils signalent des incidents qui n'existent pas où bien de ne pas découvrir des erreurs qui existent dans le code. Il y a deux grandes familles d'analyses statiques: la vérification de modèle et l'interprétation abstraite.

2.3.1 Vérification de modèle (model checking):

Les méthodes formelles à une importance principalement associée à l'analyse statique des spécifications et du code mais, récemment, ces dernières prennent également de l'envergure dans le domaine du test, plus particulièrement en utilisant les model-checkers lors de la validation des systèmes. Les méthodes formelles, typiquement utilisées dans les phases de spécification et d'analyse lors du développement de systèmes logiciels, offrent à la fois l'opportunité de réduire les coûts du test et la possibilité d'augmenter la confiance dans le système sous développement.

La vérification de modèle consiste à vérifier algorithmiquement si un modèle donné satisfait à une spécification [16]. Dans ce contexte, le modèle est le système lui-même ou une abstraction du système. La spécification utilisée est souvent formulée en termes de logique temporelle.

Ce type d'analyse sert à créer un modèle qui ne reflète pas seulement le même comportement du système original mais aussi il consiste à l'émission des aspects afin de rendre le modèle plus simple. Cette famille d'analyse est souvent appliquée pour vérifier les systèmes concurrents.

Dans le cas ou le système ne peut être vérifié d'une manière complète, la méthode de vérification de modèle exige la construction d'un nouveau modèle qui est plus simplifié mais qui conserve les caractéristiques essentielles du système [17]. La conception manuelle du modèle est obligatoire et cette étape est essentielle parce qu'une faute lors de la création du modèle peut rendre les résultats d'analyse invalides. Lorsque la construction du modèle est terminée on peut alors le vérifier par la simulation de tout les cas possibles même si cette méthode n'était pas possible à partir du programme lui-même.

2.3.2 Interprétation abstraite :

Selon Cousot, l'interprétation abstraite peut être définie comme une exécution partielle d'un programme pour obtenir des informations sur sa sémantique, par exemple sur sa structure de contrôle où sur son flot de données, sans avoir à en faire le traitement complet [14]. Un programme dénote une série de traitements dans un univers donné de valeurs où d'objets. A travers cette définition, L'interprétation abstraite consiste à utiliser cette dénotation pour décrire les traitements dans un autre univers d'objets abstraits de façon que les résultats de l'interprétation abstraite puissent fournir de l'information sur les calculs concrets du programme.

L'exemple suivant démontre que l'application d'une abstraction permet de simplifier l'analyse d'un programme :

Le texte « -2424 * 12 » peut dénoter les calculs dans un univers abstrait {(+), (-), (±)} où la sémantique des opérateurs arithmétiques est définie par la règle des signes.

L'exécution abstraite de -2424 * 12 ⇒ -(+) * (+) ⇒ (-) * (+) ⇒ (-) prouve que « -2424 * 12 » est un nombre négatif. L'interprétation abstraite s'intéresse à une propriété spécifique de l'univers des calculs (le signe du résultat dans l'exemple présenté). L'interprétation abstraite donne un sommaire d'une facette de l'exécution d'un programme. En général, ce sommaire est facile à obtenir, mais imprécis. Ainsi, -2424 + 12 ⇒ -(+) + (+) ⇒ (-) + (+) ⇒ (±), ce qui signifie qu'il est impossible de déterminer le signe du résultat uniquement à partir du signe des opérandes.

Le rôle d'interprétation abstraite se diffère selon le langage de programmation, il permet de raisonner rigoureusement sur les programmes en permettant de faire des approximations. Il permet aussi de déterminer des sémantiques liées par des relations d'abstraction [5].

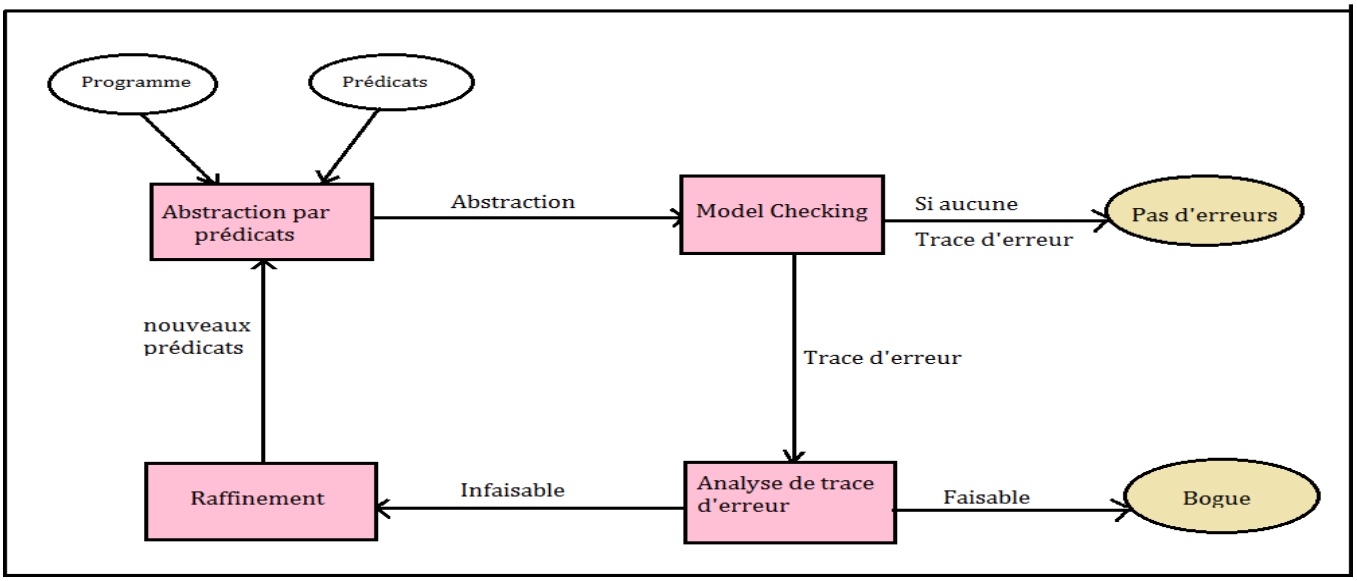

Figure 2.2: Raffinement d'abstraction guidée par les contre-exemples proposé Par [18].

2.3.3 Outils d'analyse statique :

Voici quelques outils pour aider à l'analyse statique :

- Findbugs: Permet de trouver des bugs dans le code d'un programme Java.

- Android Lint: Permet de scanner un projet Android afin d'y trouver des bugs potentiels.

- Dex2jar: Outil permettant de convertir une application Android au format dex en fichier de class Java

- Jd-gui: Application graphique permettant de lire les codes-sources des fichiers .class (java)

- Intellij Idea: Editeur Java.

- PMD: Analyseur de code-source

- Androguard: Androguard est un outil d'analyse statique populaire pour les applications Android. Il peut démonter et décompiler Dalvik bytecode. Il peut aussi calculer une mesure de similarité pour détecter les applications reconditionnés ou logiciels malveillants connus

2.4 Analyse dynamique :

L'analyse dynamique consiste à exécuter et à surveiller les actions exécutées par une application. Au contraire de l'analyse statique et de l'étude du code-source de l'application, l'analyse dynamique consiste à étudier le comportement de l'application : appel de fonctions, chaines stockées en mémoire, trafic généré, etc.

On utilise l'analyse dynamique pour l'étude de malwares (dans ce cas-là, le recours à un environnement sécurisé), et quand il n'est pas possible d'accéder au code-source de l'application (légalité du reverse engineering) [16].

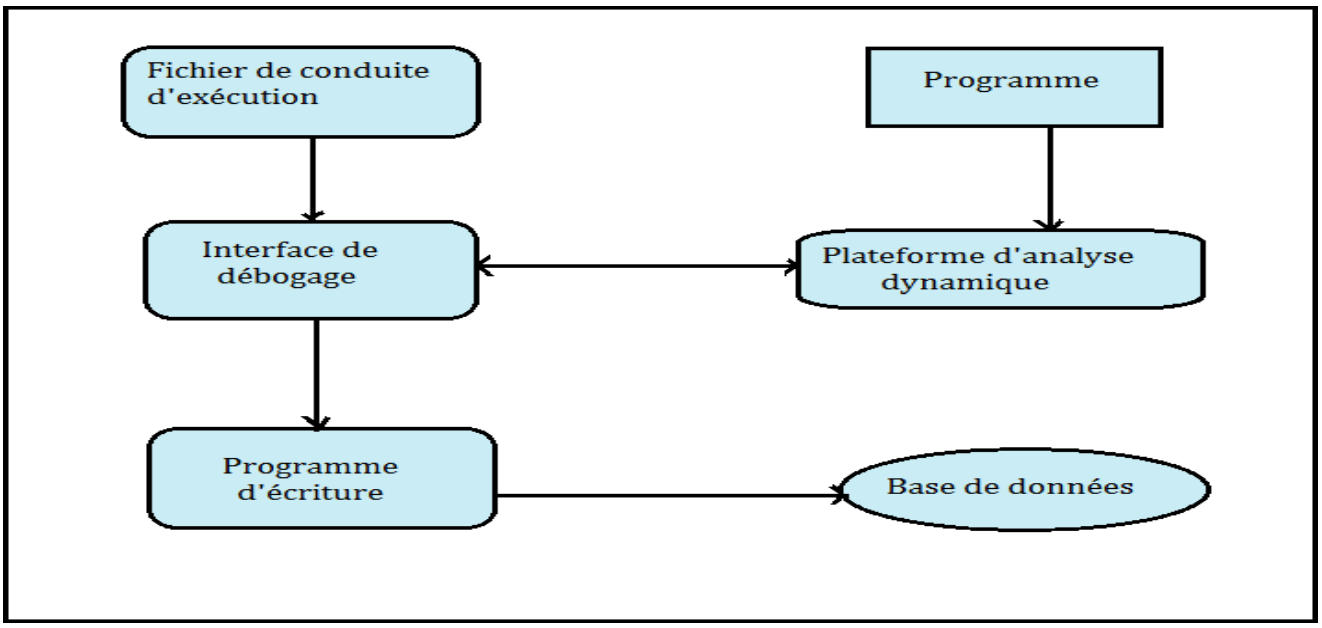

Figure 2.3: Fonctionnement de la plate-forme d'analyse dynamique proposé par [43].

2.4.1 Analyse formelle :

Le rôle d'analyse dynamique formelle de programme est d'évaluer les propriétés formalisées sur des exécutions du programme. Récemment cette analyse considérée comme une branche en plein essor, également connue sous le nom de runtime vérification [17]. Il existe deux écoles dans cette méthode d'analyse :

- Soit la propriété est vérifiée conjointement à l'exécution,
- Soit elle est vérifiée a postériori sur une trace d'exécution, à savoir une séquence chronologique des différents états du programme.

Le runtime vérification est un interpréteur responsable de l'exécution des programmes informatiques écrits dans un langage de programmation donné. Il offre des services tels que le traitement des erreurs de calcul, l'utilisation de services offerts dans un autre langage de programmation, le débogage et le profilage [43]. Il permet aussi d'interpréter le code source, manipule les variables, réserve de la mémoire et prend en charge les erreurs d'exécution.

L'exécution de programme peut se faire dans un environnement de test classique ou dans un environnement qui simule le matériel du logiciel.

2.4.2 Analyse non formelle :

La méthode de vérification la plus répandue est le test. Il comporte trois étapes successives, à savoir :

1. définition d'un jeu d'entrées et des résultats attendus
2. exécution du programme avec le jeu d'entrées défini,

3. comparaison du résultat obtenu avec le résultat attendu appelé oracle, pour s'assurer que le résultat est correct ou non [19].

On peut utiliser un environnement de test pour automatiser la vérification du logiciel.

L'environnement de test est typiquement composé d'un langage qui définit un jeu de tests (jeu des entrées, programme ou partie de programme à vérifier, oracle pour un point donné du programme uniquement. «A. Ferlin, 2013 » [20]. À partir de cette définition, avant de lancer un programme, un exécutable de test est généré automatiquement et il est lié au programme afin de commencer à le vérifier. Quand la vérification est terminée, Les résultats sont collectés et comparés automatiquement. Le rôle de l'opérateur est de pointer efficacement les cas de tests qui ont échoué.

2.4.3 Outils d'analyse dynamique :

Voici quelques outils pour aider à l'analyse dynamique :

Droidbox: Outil de type Sandbox pour les applications Android. Permet l'analyse dynamique (monitoring d'API, détection des fuites de données, analyse préliminaire statique, etc.)

Mobile Sandbox : Sandbox pour applications mobile disponible en ligne.

AndroidAuditTools: Outils pour analyse dynamique d'applications Android.

On utilise ces outils pour, en général, tester des applications malveillantes dans un environnement protégé.

2.5 Comparaison des méthodes d'analyse :

La base de comparaison des méthodes d'analyse est le comportement avant et après l'exécution du programme et les résultats de chaque méthode. Voici un tableau de comparaison entre analyse statique et analyse dynamique à la base de : Le coût, la couverture du relevé, la consommation du temps, la découverte et l'exécution.

Base de comparaison	Analyse statique	Analyse dynamique
De base d'analyse	N'exécute pas le logiciel	L'exécution du logiciel est nécessaire
Le coût d'analyse	Fiable	Haute
Couverture du relevé	100%	50%
Consommation du temps	Moins	Plus
Découvrissions	Grande variété de bugs	Types de bugs limités
Exécution	Avant la compilation	Uniquement lorsque les exécutables sont disponibles

Tableau 2.1 : comparaison entre analyse statique et analyse dynamique [25].

2.6 Conclusion:

Les outils d'analyse statique ou dynamique cités plus haut sont plutôt basés sur des méthodes formelles. Actuellement de nouveaux outils basés sur des méthodes d'apprentissage voient le jour. Ils sont utilisés pour classifier et détecter des malwares. Ces méthodes se fondent sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d'apprendre à partir des données. Dans le 3eme chapitre nous allons détailler l'apprentissage automatique dans le domaine de la cyber-sécurité et nous expliquerons le fonctionnement de chacune de ces approches.

CHAPITRE 3 :

La détection des malwares basée sur l'apprentissage automatique:

La détection des malwares basée sur l'apprentissage automatique:

3.1 Introduction :

Ces dernières années, les possibilités de l'intelligence artificielle (IA) semblent s'accroître de manière exponentielle, dans le but de réaliser des machines capables de simuler l'intelligence humaine. Dans le langage des ordinateurs on peut appliquer l'IA à travers l'apprentissage automatique pour donner aux machines la capacité d'apprendre à partir des données. Donc à travers cette définition on peut dire que l'apprentissage automatique est un domaine de l'intelligence artificielle [46].

Dans ce domaine, les chercheurs tentent d'imiter au maximum le fonctionnement du cerveau humain et les modes de traitement de l'information et de communication observé dans le système nerveux biologique, pour donner aux machines la capacité d'apprendre depuis les données, les interpréter et prendre des décisions éclairées.

Les méthodes d'apprentissage automatique ont été appliquées avec succès dans plusieurs produits des TIC (la reconnaissance d'image, traduction automatique, diagnostic médical, . . . etc.), ainsi que d'autres différents secteurs technologiques, ces dernières années (voiture autonome, robots intelligents, . . . etc.). Cependant, les performances de ces dernières reposent implicitement sur la qualité des données d'apprentissage, elles exigent une étape critique appelée l'ingénierie des caractéristiques. **Feature Engineering**, qui se définit comme une méthode dictée par les experts du domaine pour sélectionner les caractéristiques ou les propriétés importantes des données de chaque problème. Pour cela, et avec la disponibilité du Big-data, un nouveau procédé de l'apprentissage automatique appelé le Deep learning (l'apprentissage profond) a été utilisé pour apprendre la représentation et abstraire implicitement les caractéristiques [39, 40]. Ce dernier est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données. Dans le domaine de la sécurité informatique ce dernier obtient un grand succès dans de nombreuses tâches de la cybersécurité.

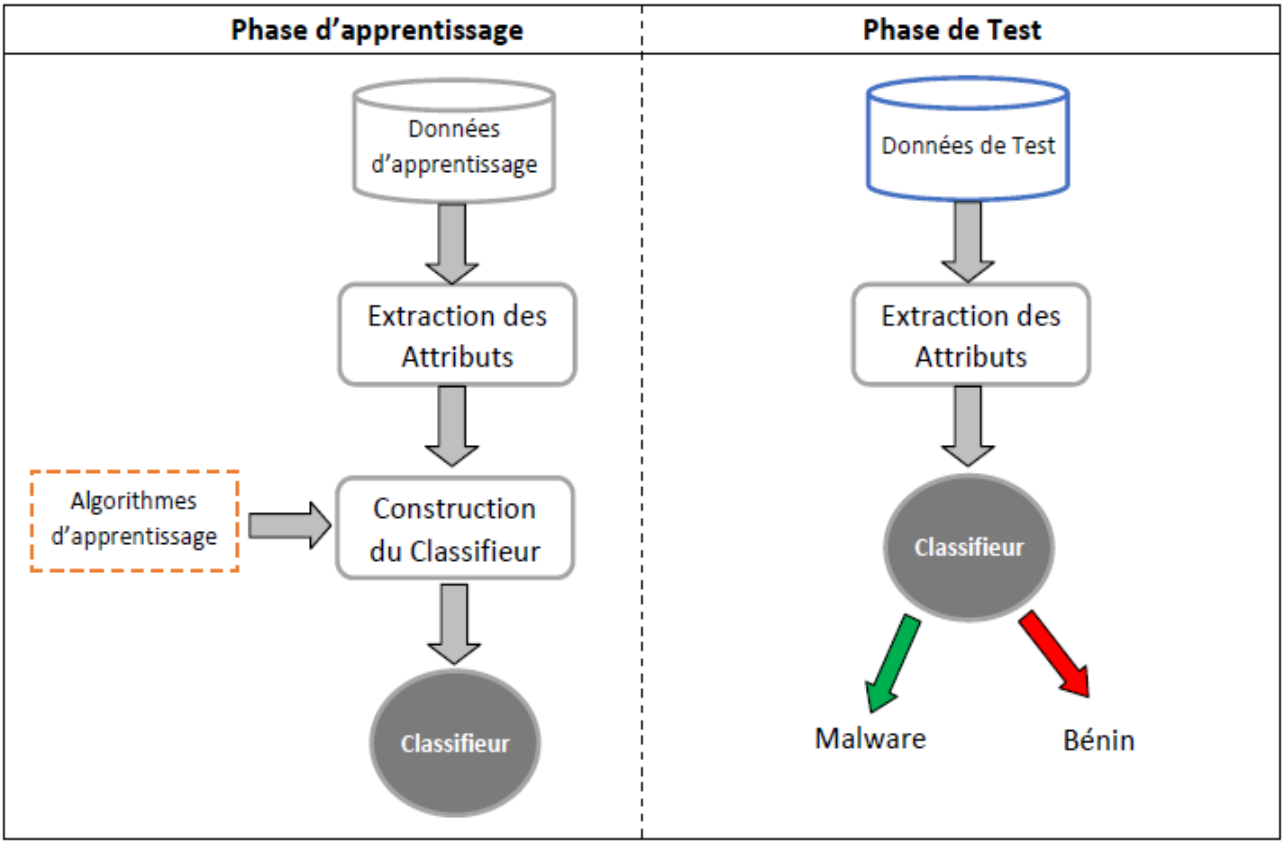

Figure 3.1: Phases d'apprentissage, et de test d'un classifieur basé sur l'apprentissage supervise [48].

3.2 L'apprentissage profond pour la cybersécurité :

Avec la disponibilité de grandes quantités de données de la cyber infrastructure des réseaux, des systèmes d'exploitation ou des systèmes d'informations et pour relever les défis de la cybersécurité, des méthodes et des techniques comme l'apprentissage automatique (machine learning), data mining, les statistiques et d'autres capacités interdisciplinaires ont été exploitées.

L'apprentissage profond qui fait partie de l'apprentissage automatique pourrait être utilisé pour la détection des logiciels malveillants basés sur la signature. Ces méthodes de classification et de prédiction peuvent être utilisées pour détecter des motifs et des comportements inhabituels des diverses cyberattaques qui permettent une cyber-réponse en temps réel. Ils ont la capacité de détecter les malwares lorsqu'ils se manifestent [50]. Les méthodes basées sur l'apprentissage approfondi peuvent aider à surmonter les défis liés au développement d'un système de détection des malwares efficace [51].

D'un autre coté, la collection des données et des trafics réseau ont conduit à un problème de big-data, les experts en sécurité souhaitent toujours de meilleures performances d'un système de détection qui ont un taux de détection le plus élevé et un taux de fausses alarmes le plus bas. Par conséquent, les approches de Deep learning s'adaptent bien à une très grande quantité de données. Ces dernières ont été introduites pour la détection des anomalies de réseau dans le but de différencier les comportements normaux et des comportements anormaux afin de détecter des activités malveillantes ou suspectes d'être malveillants [52].

3.3 Définition de l'apprentissage profond :

L'apprentissage profond ou Deep learning, appartient à une classe de techniques d'apprentissage automatique (machine learning ou ML). Par rapport aux algorithmes de ML classiques, le Deep learning obtient un grand succès dans de nombreuses tâches de l'intelligence artificielle (IA). Les modèles profonds ont plusieurs architectures relativement récentes où de nombreuses étapes de traitement non linéaire de l'information sont exploitées, dans lesquelles les informations sont traitées en couches hiérarchiques, chacune recevant et interprétant les informations de la couche précédente dans le but d'apprentissage des représentations de données [14].

3.3.1 Fonctionnement :

Le Deep Learning s'appuie sur un réseau de neurones artificiels s'inspirant du cerveau humain. Ce réseau est composé de dizaines voire de centaines de « couches » de neurones, chacune recevant et interprétant les informations de la couche précédente.

Généralement, l'architecture des réseaux profonds est organisée en couches de neurones pour n'importe quel type de ces réseaux :

- une Couche d'entrée (Input Layer), une ou plusieurs Couches cachées (Hidden Layers).
- une Couche de sortie (Output Layer).

Chaque paire de couches voisines est connectée. Les connexions entre elles sont appelées poids (Weights). Les "neurones" d'une même couche généralement appelés "nœuds" n'ont aucune association, la figure 3.1 illustré une architecture standard d'un modèle de réseau de neurones profond.

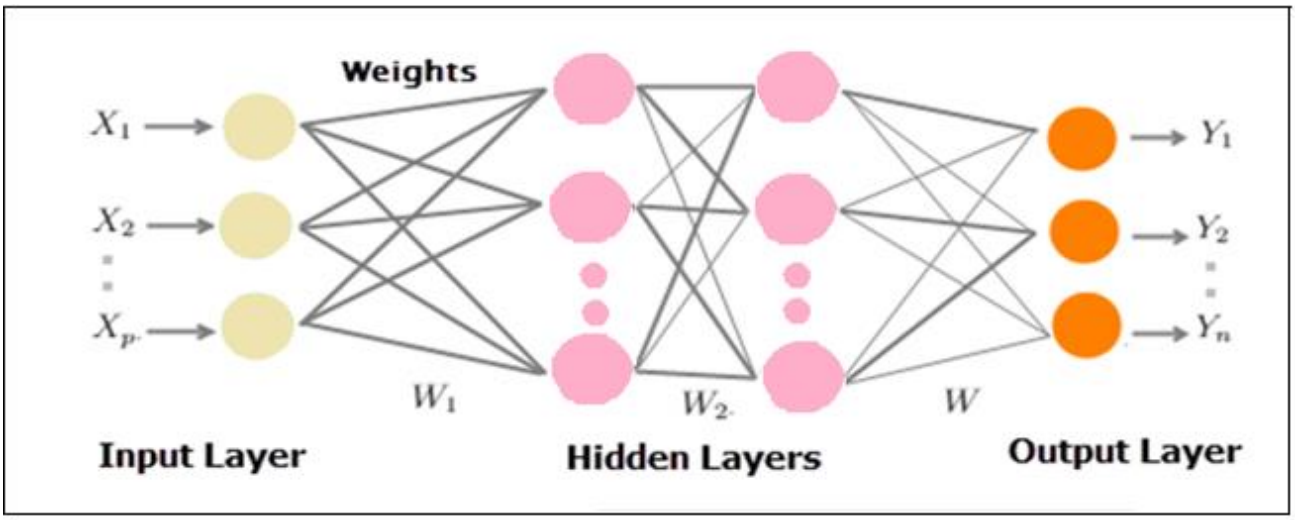

Figure 3.2: L'architecture d'un modèle Deep Learning [9].

L'apprentissage profond est un système de calcul avancé, il est constitué d'une variété de techniques issues du domaine de l'apprentissage automatique qui utilisent un déluge de neurones (nœuds) non linéaires disposés en plusieurs couches de traitement qui extraient et convertissent des valeurs de variables d'entité à partir du vecteur d'entrée pour créer plusieurs niveaux d'abstraction afin de représenter les données [36].

3.3.2 Classification des méthodes Deep learning :

Les approches d'apprentissage profond sont des réseaux de neurones (Neural networks), qui partagent certaines propriétés de base communes. Ils sont tous constitués de neurones inter-connectés, ils sont organisés en couches. Ces derniers ce déférent par leur architecture du réseau, et parfois la manière dont ils sont formés [22]. Ces approches peuvent être classées en trois modèles, en fonction de la manière dont elles sont formées et destinées à être utilisées.

- **Deep learning pour l'apprentissage supervisé :** On utilise le Deep learning pour l'apprentissage supervisé quand il y a la disponibilité des données d'étiquettes cible, il s'agit des modèles profonds discriminatoires à savoir le Deep neural networks (DNNs), Recurrent neural networks (RNNs), convolutional neural networks (CNNs).
- **Deep learning pour l'apprentissage non-supervisé :** Quand les données d'entrée ne sont pas étiquetés on utilise le Deep learning pour l'apprentissage non supervisé. le but des modèles génératifs est de classifier les données d'après certaines caractéristiques de ressemblances ou de synthèse de

modèles, à savoir le Deep belief networks (DBN), Deep autoencoders (DA), Restricted Boltzmann machine (RBM) et Deep Boltzmann machines (DBM).

- **Deep learning pour l'apprentissage hybride :** Dans cette méthode, les modèles mentionnés ci dessus sont combinés. L'apprentissage supervisé pourrait être examiné pour une excellente initialisation fournie par les réseaux profonds non supervisés.

3.4 Quelques méthodes d'apprentissage profond :

Les réseaux neuronaux profonds sont un ensemble de neurones organisés en une séquence de couches inter-connectés. Ce qui les différencie, c'est l'architecture du réseau (la manière dont les neurones sont organisés dans le réseau et la manière dont ils fonctionnent). Parmi de nombreuses implémentions de modèles d'apprentissage profond, on trouve :

3.4.1 Deep Neural Network (DNN) :

Les Deep Neural Networks (DNN) sont un ensemble de neurones organisés en une séquence de couches multiples appelée Multi-layer Perceptrons (MLP). Le DNN est un réseau neuronal artificiel (ANN) comportant plusieurs couches entre les couches d'entrée et de sortie. Lorsqu'un ANN possède deux couches cachées ou plus, il est connu sous le nom de réseau neuronal profond. Il existe différents types de réseaux neuronaux, mais ils se composent toujours des mêmes éléments : neurones, synapses, poids, biais et fonctions. Ces derniers tentent à modéliser des données contenant des architectures complexes en combinant différentes transformations non linéaires [34].

Le concept de base de la perception a été introduit par Rosenblatt en 1958. La perception calcule une sortie unique à partir de multiples entrées à valeurs réelles (xi) en formant une combinaison linéaire en fonction de ses poids (w) d'entrée, puis en plaçant la sortie via une fonction d'activation non linéaire. Mathématiquement, cela peut être écrit comme suit :

$$Y = \delta\left(\sum_{n=1}^{n} W_i x_i + b\right) = \delta\left(W^T X + b\right)$$

Avec :

- W : est le vecteur des poids.
- X : est le vecteur des entrées.

- b : désigne le biais.
- δ : représente la fonction d'activation.

Un réseau typique de la perception multi-couches (MLP) comprend un ensemble de nœuds sources formant la couche d'entrée, une ou plusieurs couches cachées de nœuds de calcul et une couche de sortie de nœuds. Le signal d'entrée se propage couche par couche sur le réseau. Les réseaux DNNs sont généralement utilisés dans les problèmes d'apprentissage supervisé. La formation de modèle (l'apprentissage) signifie l'adaptation de tous les poids et les biais à leurs valeurs optimales.

3.4.2 Convolutional neural networks (CNNs) :

Réseau convolutionnel ou réseau de neurones convolutionnel ou encore CNN est un complément au réseau classique feed-forward (FFN), principalement utilisé dans le domaine du traitement des images [23]. Il est illustré à la figure 3.2, où toutes les connexions et couches cachées et ses unités ne sont pas représentées. Les CNNs surpassent tous les autres algorithmes ML classiques et fait un grand succès dans les tâches de traitement de vision par ordinateur (Computer Vision Tasks), ils ont de larges applications dans le traitement d'image et vidéo, le traitement du langage naturel (NLP), les systèmes de recommandation . . . etc.

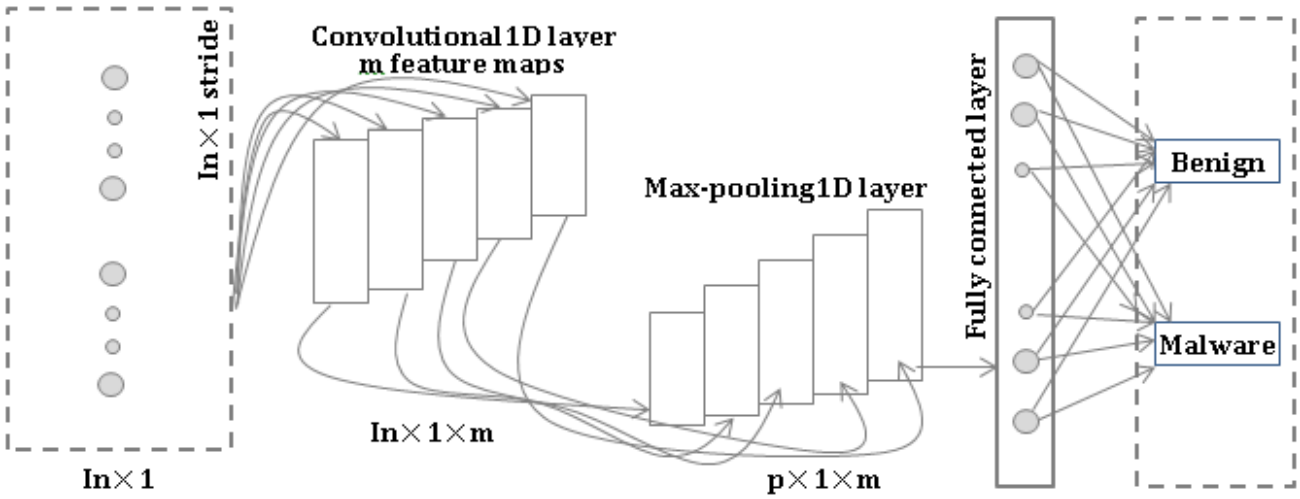

Figure 3.3: Architecture du CNN pour la détection des logiciels malveillants [W4].

Avec :

- M : le nombre de filtres
- Ln : le nombre de caractéristique d'entrée

- P : la dimension réduite des dimensions réduites des caractéristiques

Les réseaux convolutifs sont particulièrement efficaces grâce à plusieurs types de couches spéciales : des couches de convolution, des couches groupement (Pooling) et de couches entièrement connectées.

Convolution Layers :

L'objectif de la convolution est d'extraire les caractéristiques de haut niveau. Il est constitué d'un ensemble de filtres (ou noyaux) apprenants, chacun représente une certaine fonctionnalité indépendante avec le volume d'entrée. Ces filtres sont constitué d'une couche de poids de connexion, ils ont un petit champ de réception (la taille du noyau), mais lors de la passe en avant (feed forward), chaque filtre est convolé sur la largeur et la hauteur du volume d'entrée, calculant le produit des points entre les entrées et les valeurs du filtre produisant une nouvelle carte de caractéristiques qui représente mieux l'information. En conséquence, le réseau apprend les filtres qui s'activent lorsqu'il détecte un type de caractéristique importante et spécifique à une certaine position spatiale dans l'entrée. La figure 3.3 présente une opération de convolution 1D avec une entré de dimension 1.

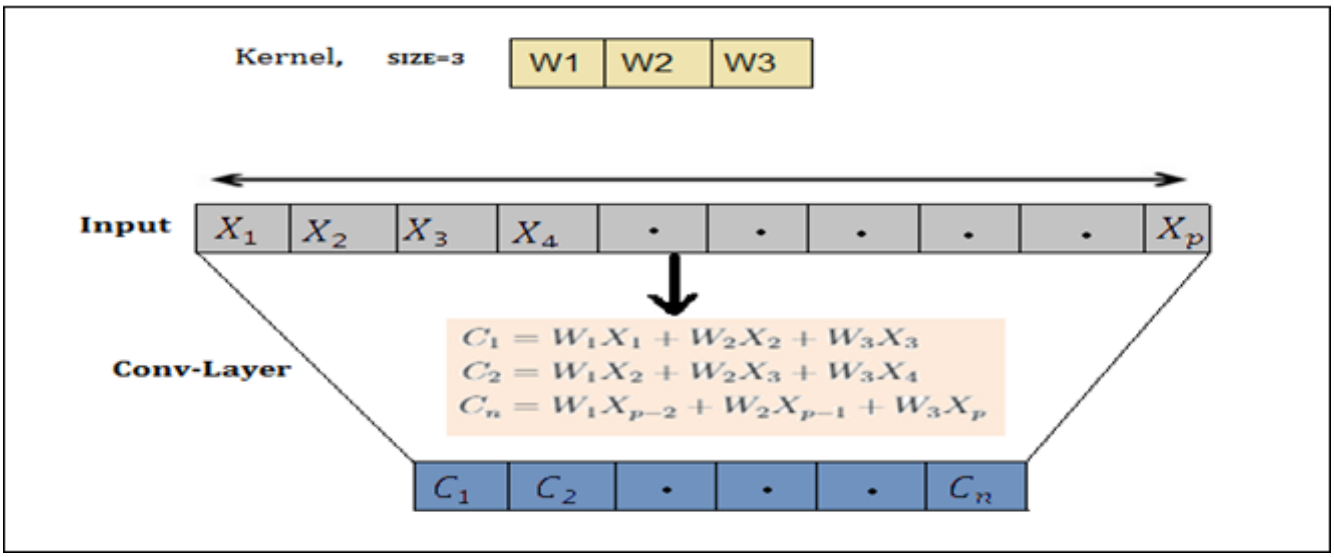

Figure 3.4: Convolution Layers [9].

Une couche convolutionnelle partage le même noyau de convolution, ce qui réduit considérablement le nombre de paramètres nécessaires pour l'opération de convolution. Une fonction d'activation non linéaire sera appliquée immédiatement après chaque couche convolutionnelle. Les CNN profonds avec la fonction d'activation "**Rectified Linear Units ReLU**"

$$[F(x) = \max(0; x)]$$

Renvoie x pour toutes les valeurs de x > 0 et renvoie 0 pour toutes les valeurs de x <= 0. S'entraînent plusieurs fois plus vite que leurs équivalents avec unités "Tanh Units" [44].

Pooling Layers :

Après la transformation ReLU, l'opération de mise en commun (Pooling) regroupe l'activation des neurones d'une couche en un seul neurone de la couche suivante. La couche de Pooling fonctionne indépendamment sur chaque entité d'entrée, elle permet de réduire progressivement la taille des représentations afin de réduire le nombre de paramètres ou de poids, ce qui diminue le coût de calcul dans le réseau, tout en préservant les informations les plus critiques. Elle permet aussi de contrôler le sur-apprentissage.

Il peut utiliser deux méthodes de mise en commun différentes :

- La mise en commun maximale (Max-Pooling) : utilise la valeur maximale de chaque groupe de neurones de la couche précédente.
- La mise en commun moyenne (Average-Pooling) : utilise la valeur moyenne de chaque groupe de neurones de la couche précédente

Le Pooling est une forme de sous-échantillonnage non linéaire fonctionne de manière similaire à la convolution. Le noyau de Pooling se convole sur le volume d'entrée et le divise en un ensemble de région qui ne se chevauchent pas, et chaque sous-région produit une seule valeur en sortie qui est la valeur maximale pour Max-Pooling ou la valeur moyenne pour Average-Pooling, la figure 3.4 décrit l'opération de Max-Pooling avec une entrée 1D et un noyau de taille 2.

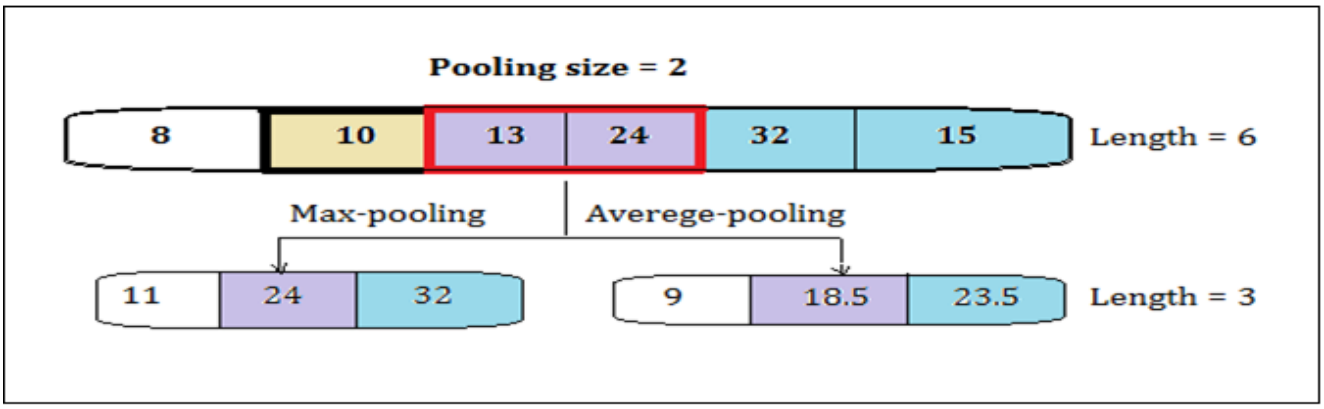

Figure 3.5: Pooling Layers

La couche de Pooling n'a aucun paramètre pouvant être appris. De ce fait, ces couches ne sont généralement pas incluses dans le nombre total de couches de réseaux de convolution.

Fully Connected Layers :

À la fin d'un réseau CNN, il y a une ou plusieurs couches entièrement connectées (chaque nœud de la première couche est connecté à chaque nœud de la couche suivante). Elles consistent à effectuer une classification basée sur les caractéristiques extraites des convolutions. La couche finale contient une fonction d'activation Softmax, qui génère une valeur de probabilité de 0 à 1 pour chacune des étiquettes de classes que le modèle tente de prédire. Dans certaines architectures de réseaux CNNs récentes, les couches entièrement connectées peuvent se remplacer par plusieurs couches de mise en commun moyennes (average-pooling). Cela permet à ces réseaux de réduire considérablement le nombre total des paramètres et qui permet une meilleure prévention de sur-apprentissage [10].

3.4.3 Recurrent neural networks (RNNs) :

Les réseaux neuronaux s'inspirent du fonctionnement des neurones biologiques du cerveau humain, ces neurones sont considérés comme le centre de réflexion, et parfois ils doivent mémoriser certains évènements pour les utiliser ultérieurement avant de prendre la décision. Les réseaux neuronaux traditionnels n'ont pas cette propriété, alors le fonctionnement d'un réseau de neurones récurrents (RNN) est motivé par le fait qu'un être humain raisonne en s'appuyant sur les connaissances qu'il a acquises et qui qu'il a mémorisées précédemment [4].

Les réseaux RNNs sont des réseaux de type Feed-Forward ayant un état interne (ou mémoire) qui prennent en compte tout ou partie des données vues précédemment (déjà fournies au réseau), en plus de la donnée vue actuellement pour adapter leur décision. L'idée clé de base de ces réseaux est le déploiement d'un calcul récurrent grâce aux boucles dans l'architecture du réseau. La sortie de réseau est une combinaison de son état interne (mémoire d'entrées) et la dernière entrée. Au même temps, l'état interne change pour intégrer cette nouvelle donnée saisie. Cela permet aux informations de persister en mémoire, comme le montre la figure 3.5.

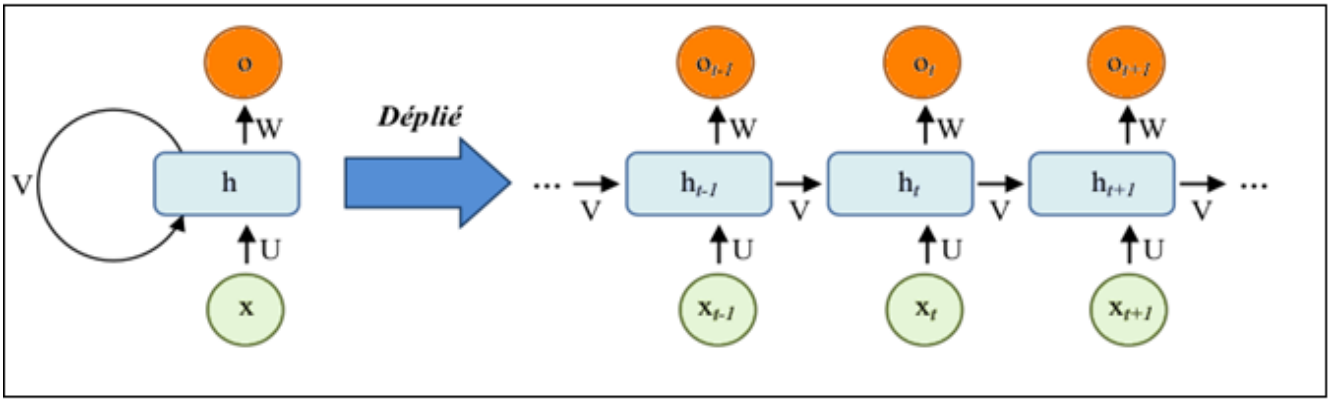

Figure 3.6: L'architecture d'un modèle RNN [9].

En raison de ces propriétés, les réseaux récurrents sont adaptés aux cas où la présence d'une forme n'est pas la seule information discriminante mais également un ordre d'apparition par exemple. Ils sont de bons candidats pour les tâches qui traitent des données séquentielles, telles que les données textuelles ou des données avec des caractéristiques temporelles. La description mathématique du processus de transfert de mémoire est comme suit :

$$h_t = \delta(\mathrm{U}x_t + \mathrm{V}\, h_{t-1} + b_h)$$

$$O_t = \delta(\mathrm{W}h_t + b_y)$$

Ou :

- h_t : est l'état caché au temps t.
- x_t : est l'entrée au même temps t.
- U, V, W : sont les matrice de pondération, Input-to-Hidden, Hidden-to-Hidden et Hidden-to-Output respectivement (connue comme des matrices de transition).
- b_h : est la valeur du biais de l'état caché.
- b_y : est la valeur du biais de sortie.
- O_t : est la valeur de sortie au temps t.
- δ : est une fonction de non-linéarité appelées fonctions d'activation. (soit une fonction sigmoïde logistique ou Tanh) qui est un outil standard de changement d'échelle pour condenser des valeurs très grandes ou très petites dans un espace logistique, ainsi que pour rendre les gradients exploitables pour la rétro-propagation.

Un bloc de réseau neuronal, examine une entrée x_t et génère une valeur O_t. Une boucle de rétroaction se produit à chaque pas de temps, chaque état caché ht contient des traces non seulement de l'état masqué précédent, mais également de tous ceux qui ont précédé h_{t-1} aussi longtemps que la mémoire peut persister.

3.5 Les Data-sets d'évaluation de détection des malwares basé sur Deep learning :

Les Data-sets utilisés dans les travaux publiés pour l'application de l'apprentissage approfondi dans la cybersécurité jouent un rôle essentiel pour la validation de toutes approches DL proposés.

Dataset public	Type	Étiqueté	Année	**Réf**
MalGenome	trafic du réseau	Oui	2000	[14]
Drebin	trafic du réseau	Oui	2010	[41]
MalDozer	trafic du réseau	Oui	2018	[21]
Malimg	trafic internet	Oui	2017	[24]
AndroidAdware	trafic du réseau	Oui	2017	[W5]
CICMalDroid	trafic du réseau	Oui	2020	[22]

Tableau 3.1 : Ensembles de données public relatives à la détection des malwares

3.6 Comparaison entre la machine learning et le Deep learning :

On se demande bien souvent comment les ordinateurs prennent et apprennent des décisions intelligentes. Et on confond les deux méthodes les plus constituantes qui peuvent rendre l'intelligence artificielle possible.

La machine learning est basée sur un algorithme qui adapte le système fait par l'humain. Elle est alimenté par des données structurées, contrôlable modifiées. Cette technologie est la plus simple et la plus traditionnelle. Tandis que la technologie nécessaire pour le deep learning est plus sophistiquée, plus couteuse que la machine learning, le système doit disposer de plusieurs données pour la fiabilité des résultats. Avec le deep learning, le système est autonome, il n'a pas besoin d'être entrainé par un développeur.

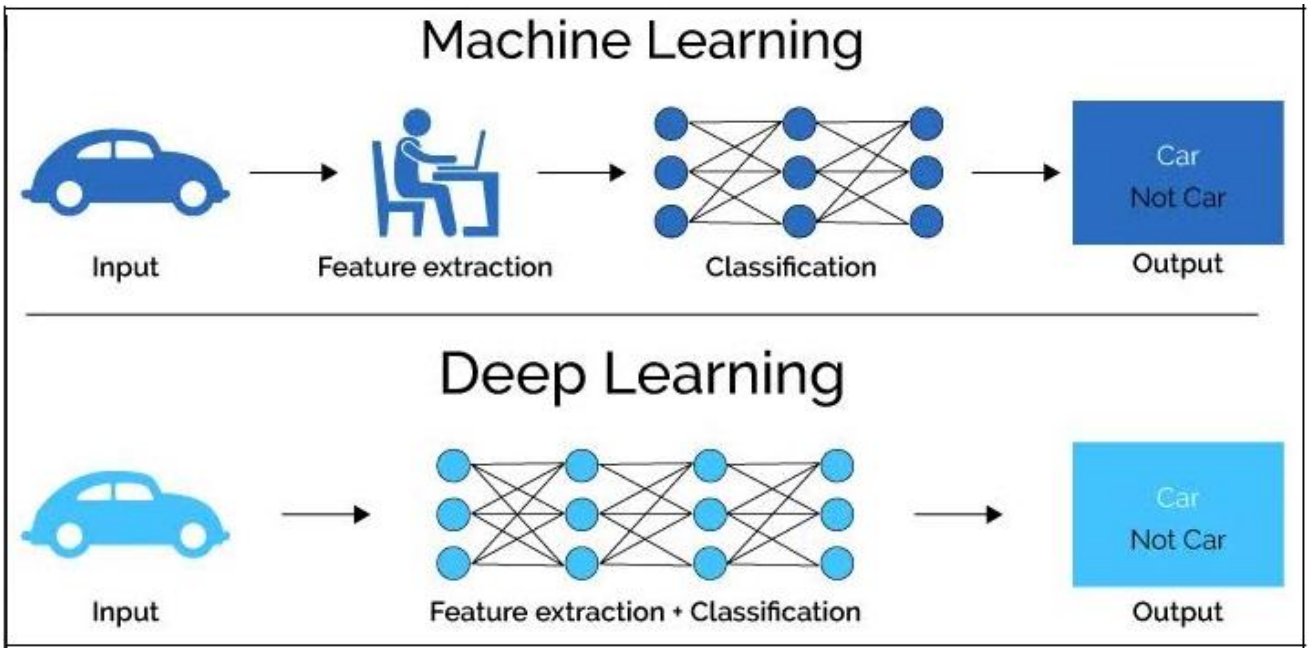

Figure 3-7: Comparaison entre Machine learning et deep learning [W3].

Il existe de nombreuses différences entre Machine Learning et Deep Learning. Selon [W8], les cinq plus importantes différences sont :

1. Intervention humaine: L'apprentissage automatique nécessite une intervention humaine plus continue pour obtenir des résultats. Le deep learning est plus complexe à mettre en place mais ne nécessite qu'une intervention minimale par la suite.

2. Matériel: Les programmes d'apprentissage automatique ont tendance à être moins complexes que les algorithmes de Deep Learning et peuvent souvent fonctionner sur des ordinateurs classiques, mais les systèmes d'apprentissage profond nécessitent du matériel et des ressources beaucoup plus puissants. Cette demande de puissance a entraîné une utilisation accrue des GPUs.

3. Le temps: Les systèmes d'apprentissage automatique peuvent être mis en place et fonctionner rapidement, mais peuvent être limités dans la puissance de leurs résultats. Les systèmes d'apprentissage profond prennent plus de temps à mettre en place mais peuvent générer des résultats instantanément.

4. Approche: L'apprentissage automatique a tendance à nécessiter des données structurées et utilise des algorithmes traditionnels comme la régression linéaire. Le Deep Learning utilise des réseaux neuronaux et est conçu pour traiter de grands volumes de données non structurées.

5. Applications: L'apprentissage automatique est déjà utilisé dans votre boîte aux lettres électronique, votre banque et votre cabinet médical. La technologie de Deep Learning permet de réaliser des programmes plus complexes et autonomes, comme les voitures à conduite autonome ou les robots qui effectuent des opérations chirurgicales avancées.

3.7 Conclusion

Le domaine du Deep learning est très vaste. Il a une croissance rapide, des nouveaux algorithmes, architectures ou variantes apparaissent toutes les semaines. L'application des nouvelles méthodes de DL et l'évaluation des performances de diverses architectures DL existantes sont restés un objet et une orientation importante de la recherche pour les chercheurs en sécurité.

CHAPITRE 4 :

La détection des malwares basée sur le Deep learning

La détection des malwares basée sur le Deep learning

4.1 Introduction :

Plusieurs approches de Deep learning ont été étudiées récemment pour la détection des malwares. Dans les systèmes de détection basée sur la signature, les malwares sont détectées en comparant les comportements surveillés avec des motifs du malwares prédéfinis, tandis que les systèmes basés sur les anomalies se concentrent sur la connaissance du comportement normal afin d'identifier toute déviation et toutes activités suspectes.

Les méthodes de Deep learning sont applicables pour les 2 types de détection grâce a ses capacités qui permettent d'extraire des niveaux plus élevés de relations non linéaires entre les données, afin d'identifier toute déviation d'une activité bénigne. Cependant, ces méthodes exigent une quantité énorme des données dans le but de détecter et identifier les motifs de différentes classes. Nous allons présenter ici quelques approches de DL qui ont été appliqué au domaine de la détection des malwares pour la cybersécurité.

4.2 Travaux connexes pour la détection des malwares basé sur le DL :

Des travaux antérieurs ont montré que les méthodes DL surpassent d'autres algorithmes d'apprentissage machine tels que la machine à vecteur de support (SVM) et les réseaux artificiels neuronaux traditionnels (ANN) dans la détection des anomalies. La détection des malwares basé sur l'apprentissage profond a commencé en 2014, lorsque Yuan et al. [26] présentent une approche hybride combinant le réseau de croyances profondes (DBN) et machines à vecteurs supports (SVM) afin de classifier les données du réseau en deux catégories : BENIGN ou MALWARE. Le réseau DBN est composé de multiples couches de machines Boltzman restreinte (RBM) est utilisé comme une méthode de réduction de dimension pour obtenir de meilleures caractéristiques d'apprentissage suivies par un classeur SVM. Yuan et al. [26] ont testé minutieusement DroidDetector et effectué une analyse approfondie des caractéristiques que l'apprentissage profond exploite essentiellement pour caractériser les logiciels malveillants. Les résultats montrent que l'apprentissage profond convient à la caractérisation des logiciels malveillants Android et qu'il est particulièrement efficace avec la disponibilité d'un plus grand nombre de données d'entraînement.

➢ Dans cette étude, Yuan et al. [26] ont proposé d'associer les caractéristiques de l'analyse statique à celles de l'analyse dynamique des applications Android et de caractériser les logiciels malveillants à l'aide de techniques d'apprentissage profond. Ils mettent en œuvre un moteur de détection de logiciels malveillants Android en ligne basé sur l'apprentissage profond (DroidDetector) qui peut détecter automatiquement si une application est un logiciel malveillant ou non. Avec des milliers d'applications Android, Yuan et al. [26] ont testé minutieusement DroidDetector et effectué une analyse approfondie des caractéristiques que l'apprentissage profond exploite essentiellement pour caractériser les logiciels malveillants. Les résultats montrent que l'apprentissage profond convient à la caractérisation des logiciels malveillants Android et qu'il est particulièrement efficace avec la disponibilité d'un plus grand nombre de données d'entraînement. DroidDetector peut atteindre une précision de détection de 96,76 %, ce qui surpasse les techniques traditionnelles d'apprentissage automatique.

➢ Zhu et al. [28] extraient un total de 323 caractéristiques en utilisant FlowDroid pour extraire les flux de données des applications Android (y compris 3 000 applications bénignes extraites de Google Play Store et 8 000 applications malveillantes extraites d'Android Malware Genome Project, VirusShare, etc. Ensuite, ils ont implémenté l'architecture de détection de logiciels malveillants DeepFlow basée sur un modèle d'apprentissage profond DBN. Il a évalué DeepFlow et effectué une analyse approfondie des caractéristiques de flux de données qu'il a exploitées pour caractériser les logiciels malveillants. Les résultats montrent que DeepFlow surpasse considérablement les approches traditionnelles d'apprentissage automatique et obtient un score F1 élevé dans des paramètres appropriés.

➢ Dans cette étude, pour évaluer la performance des classificateurs, Vinayakumar et al. [27] ont considéré l'exactitude (Accuracy 2 [0 ; 1]), la précision (Précision 2 [0 ; 1]), le rappel (Recall 2 [0 ; 1]) et les valeurs de F1-plus. (Precision 2 [0; 1]), Recall (Recall 2 [0 ; 1]) et F1-(score F1 2 [0 ; 1]). Ces métriques sont estimées sur la base de True Positive (TP), True Négative (TN), les falses positifs (FP) et les falses négatifs (FN). TP représente le nombre d'échantillons d'applications malveillantes correctement identifiés comme application malveillante, TN représente le nombre d'échantillons d'applications bénignes correctement identifiés en tant que d'applications bénignes, FP représente le nombre d'échantillons d'applications bénignes mal classées en tant qu'échantillons d'applications malveillante, FN représente le nombre d'échantillons d'application malveillants classés à tort comme des échantillons d'applications bénignes.

➢ Karbab et al. [30] ont présenté MalDozer. C'est un système de détection et d'attribution automatique, efficace et performant pour Android.

Mal-Dozer s'appuie sur des techniques d'apprentissage profond et sur des séquences brutes d'appels de méthodes API afin d'identifier les logiciels malveillants Android.

Afin d'identifier les logiciels malveillants Android, Karbab et al. [30] évaluent MalDozer sur plusieurs petits et grands ensembles de données, y compris Malgenome, Drebin et le jeu de données MalDozer, ainsi qu'un jeu de données d'applications bénignes.

L'ensemble de données d'applications bénignes téléchargées depuis Google Play. Les résultats de l'évaluation montrent que MalDozer est très précis en termes de détection des logiciels malveillants ainsi que de leur attribution à des familles correspondantes. De plus, MalDozer peut s'exécuter efficacement sous de multiples architectures de déploiement, allant des serveurs aux petits systèmes d'information, architectures de déploiement, allant des serveurs aux petits appareils internet of things (IoT) de petite taille.

➢ Giacomo et al. [31] ont proposé dans leur article une approche visant à aider les analystes de sécurité à comprendre les résultats des méthodes basées sur l'apprentissage profond et destinées à distinguer les logiciels malveillants Android des échantillons fiables. Leur travail présente une méthodologie pour évaluer les modèles d'apprentissage profond pour la classification des logiciels malveillants basée sur l'image. Elle utilise les informations fournies par un Grad-CAM et tente d'évaluer la phase de formation des modèles analysés. Dans les approches classiques, les cartes thermiques générées par le Grad-CAM peuvent être utilisées par les analystes de sécurité ou les chercheurs pour garantir la fiabilité de l'inférence du modèle. Cette étape n'est pas simple lorsque l'image d'entrée ne contient pas un motif spécifique et connu (comme la forme d'un animal ou d'un objet), l'analyste de sécurité ne peut donc pas utiliser directement les informations fournies par le Grad-CAM. Leurs méthodologie aide l'analyste de sécurité à évaluer les modèles en deux étapes principales : (1) elle maintient toujours l'utilisation de la vérification manuelle effectuée par l'analyste, en fournissant une "carte thermique cumulative" qui fournit des informations préalables sur l'analyse d'inférence, qui peut être utilisée avec des connaissances préalables sur le problème lui-même ou simplement pour interpréter et comprendre les résultats du modèle, et (2) elle fournit une comparaison entre différents modèles formés sur le même ensemble de données, ils exploitent la distance euclidienne moyenne entre les cartes thermiques produites, et ils donnent ainsi des informations utiles sur le modèle le plus fiable dans la phase d'inférence.

➢ Dans cet article, Taeguen Kim et al. [29] ont proposé un nouveau cadre de détection des logiciels malveillants Android qui utilisent de nombreuses caractéristiques statiques pour refléter les propriétés des applications sous divers aspects. Au total, sept types de caractéristiques sont extraits en analysant des fichiers tels que le fichier manifeste, le fichier dex et le fichier .so d'un fichier APK, et ces caractéristiques enrichissent les informations extraites pour exprimer les caractéristiques des applications. En outre, ils ont suggéré une méthode efficace de génération de vecteurs de caractéristiques qui est appropriée pour détecter les logiciels malveillants qui sont similaires aux applications bénignes. Grâce à la représentation des caractéristiques que nous proposons, il est possible d'empêcher le vecteur de caractéristiques des logiciels malveillants de contenir les propriétés communes qui apparaissent dans les applications bénignes. Enfin, ils utilisent la méthode d'apprentissage profond multimodal, qui est conçue pour traiter différents types de caractéristiques. Différents types de caractéristiques sont exclusivement utilisés pour former les réseaux initiaux, et les résultats des réseaux initiaux sont ensuite utilisés pour former le réseau final. Cette architecture du modèle est adaptée à leur cadre pour améliorer la précision de la détection des logiciels malveillants. À leur connaissance, cette recherche est la première application de l'apprentissage profond multimodal à la détection des logiciels malveillants Android.

Lors de l'évaluation, Taeguen Kim et al. Ont Réalisé de nombreuses expériences. Ils ont comparé la précision de détection de plusieurs modèles de détection différents. Et, ils réalisent une expérience pour démontrer que leur modèle de détection peut être efficacement mis à jour. En outre, ils ont Réalisé des expériences pour confirmer l'utilité de la caractéristique et de la méthode de génération de vecteur de caractéristique qu'ils proposent. Ils ont également mené des expériences sur l'applicabilité de la classification basée sur l'apprentissage non supervisé et la résistance à l'obscurcissement. En conséquence, leur cadre est suffisamment efficace pour être utilisé dans la détection des logiciels malveillants Android.

➢ Dans le cadre de cette étude, Yuan et al. [32] extraient un total de 192 caractéristiques à partir d'analyses statiques et dynamiques statiques et dynamiques d'applications Android et caractérisé les modèles d'apprentissage profond basé sur DBN. Ils ont conçu

DroidDetector et Ils ont évalué avec 20 000 applications bénignes extraites du Google Play Store et 1760 malwares 1760 logiciels malveillants collectés à partir du projet bien connu Contagio Community and Genome Project. Les résultats montrent que l'utilisation de DroidDetector avec un modèle d'apprentissage profond peut atteindre une précision supérieure dans différentes conditions, dépassant de manière significative les techniques traditionnelles d'apprentissage automatique.

Actuellement, DroidDetector a été déployé en ligne pour être testé par les utilisateurs. En outre, ils approfondi les caractéristiques que l'apprentissage profond exploite pour caractériser les logiciels malveillants Android en utilisant des techniques d'exploration de règles d'association. L'évaluation de dix logiciels antivirus populaires indique qu'il est urgent d'apporter des changements à la détection des malwares Android.

Beaucoup de travail est nécessaire. Premièrement, des caractéristiques plus fines pour caractériser les applications Android.

Les applications Android est un ensemble plus complet et plus fin de caractéristiques plus complet et plus fin peut couvrir plus d'aspects d'Android, ce qui permet de mieux caractériser et détecter des logiciels malveillants. En plus du total de 192 caractéristiques de cette étude, Yuan et al. Également ajouter les caractéristiques introduites dans les Réf et les types de caractéristiques présentées dans l'ensemble des caractéristiques.

En outre, des caractéristiques discrètes plus riches que les caractéristiques binaires peuvent être utilisées pour établir les caractéristiques profondes binaires peuvent être utilisées pour établir le modèle d'apprentissage profond. Par exemple, si une fonction sensible de l'API est appelée deux fois ou qu'un comportement dynamique se produit deux fois, ils ont définir les valeurs des caractéristiques correspondantes comme 2 (c'est-à-dire des valeurs discrètes) au lieu de 1 (valeurs binaires), (valeurs discrètes) au lieu de 1 (c'est-à-dire des valeurs binaires). Et Deuxièmement, il faut collecter davantage de données sur les applications (c'est-à-dire plus de types d'échantillons malveillants et bénins), pour l'apprentissage du modèle d'apprentissage profond. Plus de types d'échantillons d'entraînement pourraient conduire à une meilleure optimisation du modèle d'apprentissage profond, et ainsi obtenir une précision supérieure dans la détection des logiciels malveillants Android dans le monde réel.

4.3 Conclusion :

Différentes approches et architectures Deep learning ont été appliquées pour la détection des malwares. Ces algorithmes d'apprentissage profond proposés ont des performances différentes selon les ensembles de données sélectionnés et les caractéristiques d'entrée.

Cependant, l'utilisation des mêmes approches et des mêmes techniques d'apprentissage ne garantissent pas toujours les mêmes résultats pour une variété de classes différentes d'attaques possibles.

Approche DL	Dataset	performances	Année	Références	Cité*
DBN	Network	ACC = 96.5%	2014	Yuan et al. [26]	355
CNN-LSTM	Malimg	ACC = 96.3% PR = 94.0% RC = 98.4% F1 = 96.2 %	2019	Vinayakumar et al. [27]	105
DBN	MalGenome	ACC = 95.05%	2017	Zhu et al. [28]	59
MNN	MalGenome	ACC = 85% TNR = 96% FNR = 14% TPR = 85% FPR = 15%	2017	Kim et al. [29]	157
ANN	MalDozer	F1 = 98.18% FPR = 1.15%	2018	Karbab et al. [30]	146
CNN	Drebin MalGenome	ACC = 98.0% AUC = 99.5% PR = 97.2% RC = 97.0% F-M = 97.1%	2020	Iadarola et al. [31]	2
DBN	Network	ACC = 96.76%	2016	Yuan et al. [32]	287

Tableau 4.1 : Travaux antérieurs connexes pour la détection du malwares basé sur le Deep learning

CHAPITRE 5 :

Conception de la méthode proposée

Conception de la méthode proposée

5.1 Introduction :

Alors que la diversité des logiciels malveillants ne cesse de croître, les scanners antivirus ne peuvent pas répondre aux besoins de protection, ce qui fait que des millions d'hôtes sont attaqués. Par conséquent, la protection des systèmes informatiques contre les logiciels malveillants est l'une des tâches de cybersécurité les plus importantes pour les utilisateurs individuels et les entreprises, car même une seule attaque peut entraîner une compromission des données et des pertes suffisantes. Pour cette raison, des techniques basées sur l'apprentissage automatique peuvent être utilisées. L'objectif de ce projet est de développer la preuve de concept pour la classification des logiciels malveillants basée sur l'apprentissage automatique en utilisant des classificateurs forts. Les meilleures caractéristiques sont déterminées parmi toutes les caractéristiques données et sont extraites. L'algorithme le plus précis peut distinguer le fichier malveillant avec le plus faible taux d'erreur. Les performances des approches de détection proposées ont été évaluées en prenant en compte les différentes mesures d'évaluation des algorithmes de l'apprentissage profond à savoir, la précision, le rappelle, le score F1, le taux de détection et le taux de fausse alarmes.

5.2 Environnement d'exécution :

L'exigence de la disponibilité de certain ressources matérielles dans le Deep learning est obligatoire pour faire des calcules intenses. Au premier temps, On a commencé par la configuration d'un environnement local de développement Python utilisant la plate-forme PyCharm.

PyCharm : Est un logiciel multi-plateforme qui fonctionne sous Windows, Linux et Mac OS X. Ce dernier est développé par l'entreprise tchèque JetBrains, Il permet l'analyse de code et contient un débogueur graphique. Il permet également la gestion des tests unitaires, l'intégration de logiciel de gestion de versions, et supporte le développement web avec Django. La préparation des données aussi a été faite localement en plusieurs étapes et à l'aide des techniques qui n'exigent pas beaucoup de calculs.

Dans notre travail, on avait besoin d'aller vers le Cloud pour aller vers le Deep learning. Le Cloud fournit pour nous des ressources de calculs et de mémoires importantes qui dépassent nos ordinateurs.

Selon le besoin, le Cloud peut fournir l'accès à un processeur graphique GPU totalement gratuit. Google Collab parmi les outils Cloud les plus utilisés dans le domaine de la machine learning.

Google Colaboratory : Colaboratory, souvent raccourci en "Collab", C'est un environnement particulièrement adapté à la machine learning, à l'analyse de données et à l'éducation. Ce dernier permet à n'importe qui d'écrire et d'exécuter le code Python de son choix par le biais du navigateur. En termes plus techniques, Collab est un service hébergé de notebooks Jupyter qui ne nécessite aucune configuration et permet d'accéder gratuitement à des ressources informatiques, dont des GPU. Pour l'accès dans ce service il nous suffit simplement d'avoir qu'un compte Google. Il nous offre un processeur GPU gratuit, 12 Go de RAM et plus de 100 Go de stockage.

On a choisi Python comme un langage de développement. Ce dernier est un langage de programmation interprété multi-paradigme. Il favorise la programmation impérative structurée, et orientée objet. Python est un langage qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation comme l'apprentissage automatique et l'intelligence artificielle grâce à ses bibliothèques open source spécialisées à chaque traitement. Dans notre projet nous avons utilisé Pandas, Numpy, Scikit-learn …etc. Pandas et Numpy sont utilisés pour la manipulation des données (le chargement, la réorganisation et le traitement des données), Scikit-Learn nous permet d'expérimenter différentes techniques et algorithmes d'apprentissage automatique et d'analyse de données prédéfinis rapidement et facile a utilisé.

Pour l'implémentation des méthodes DL nous avons utilisé les frameworks TensorFlow et Keras.

La 1ére est une bibliothèque open source créée par Google, permettant de développer et d'exécuter des applications de Machine Learning et de Deep Learning pour effectuer des opérations numériques complexes sur plusieurs plateformes comme Les GPU et Les CPU.

La 2eme est une bibliothèque open source écrite en Python dans le but de permettre la constitution rapide de réseaux de neurones. Keras ne fonctionne pas comme un Framework propre mais comme une interface de programmation applicative (API) pour l'accès et la programmation de différents frameworks d'apprentissage automatique.

Avec le plus faible délai possible, Collab capable d'aller au résultat. Et aussi cet environnement supporte à la fois les (CNN) et les (RNN), ainsi que la combinaison des deux items Pas de fichiers séparés de configuration des modèles, tout est déclaré dans le code.

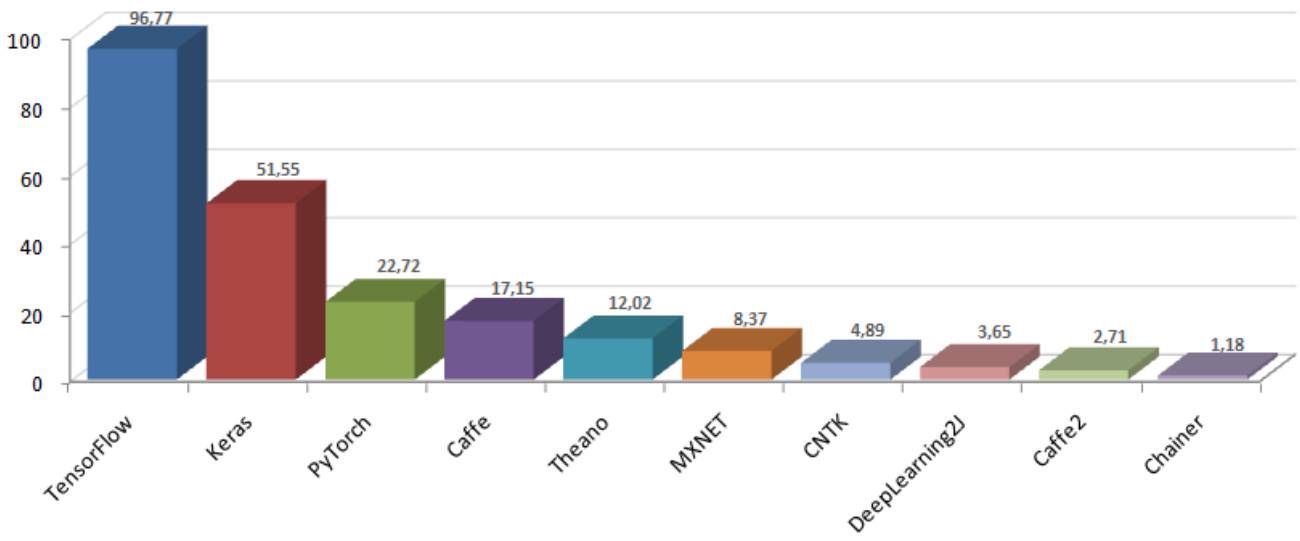

Figure 5.1: Les 10 frameworks Deep learning les plus populaires [W6]

Afin de bénéficier les avantages des deux nous avons utilisé TensorFlow comme back-end avec Keras. L'environnement d'expérimentation s'est Google Collab avec un processeur GPU et une RAM de 15 Go.

5.3 Dataset :

La data-set canadienne CIC-AAGM2017 est l'ensemble de données choisi pour cette étude. Elle est fournie par l'Institut canadien de la cybersécurité (CIC) [W5]. CIC-AAGM2017 est constituée de plusieurs types des malwares. Ces types sont des données des flux réseau réel. Les données de CIC-AAGM2017 sont des formats plus condensés qui contiennent principalement des métas-informations sur des connexions réseau. Chaque donnée est une ensemble des paquettes qui partagent certaines propriétés dans une fenêtre temporelle. L'ensemble de données comporte une version unique, sont des données Comma-separated values (CSVs).

Pour l'extraction et la sélection des caractéristiques, Les auteurs ont capturé les caractéristiques du trafic réseau (fichiers .pcap), et extrait plus de 80 caractéristiques à travers l'analyseur du trafic CICFlowMeter-V3 pendant les trois états (installation, avant le redémarrage, et après le redémarrage) et les résultats ont été sauvegardé dans des fichiers CSVs.

CIC-AAGM2017 Comprend également 4 types des malwares les plus récemment a jour. L'ensemble complet des données contient 1240998 instances (1101998 MALWARE, 123778 BENIGN). Ce data-set

contient également 78 caractéristiques (Feature) qui sont étiquetés et caractérisés sur le flux trafic du réseau.
Les données de cette data-set ont l'désavantage que les données BENIGN représentent seulement 15% de l'ensemble de données. En plus, ses données ne sont pas étiquetées par la même étiquette comme illustrées dans le tableau 5.1, ce qui impose de modifier l'étiquetage avant de l'utiliser.

5.4 La préparation des données :

La quantité des données dans le deep learning joue un rôle essentiel pour la bonne performance d'apprentissage (plus de donnée avec qualité égale plus précision et bon résultat). Dans notre travail nous avons besoin de deux grandes étapes pour résoudre le problème du déséquilibrage des classes de données et les différentes étiquettes.

5.4.1 La réduction des données :

La quantité des données dans la data-set CIC-AAGM2017 est phénoménale et plus volumineuse d'espace et du temps. C'est pour ça que nous avons été obligé de faire une réduction de ces données.

Dans notre cas, la quantité des données est constitue de deux dossiers (BENIGN, MALWARE), Le premier est constitue de 1 698 fichiers CSVs de taille de 2 Go pour l'apprentissage.

Le 2eme est constitue de 104 fichiers CSVs de taille de 1 Go avec différentes étiquettes. La réduction de ces fichiers a été définie avec les étapes suivantes :

- Nous avons gardé toutes les instances de classes minoritaires, Ces dernier sont collectés depuis plusieurs fichiers.

- Nous avons aussi Choisi aléatoirement un nombre prédéfini d'instances pour chaque classe majoritaire dans l'ensemble des instances de cette classe.
- Nous avons organisé, ensuite, les données d'apprentissage et les données de test en 2 fichiers séparés "Train.csv" et "Test.csv".

Et à la fin nous avons deux fichiers CSVs : un fichier pour l'entrainement et un fichier pour le test. Ces fichiers sont collectés aléatoirement par des données BENIGN et des donnée MALWARE. Le tableau 5.1 montre la répartition de ces ensembles de données.

Dataset CIC-AAGM2017	Les Classes concernés	Nb d'instances pour l'apprentissage	Nb d'instances pour le Test
Classification binaire	BENIGN MALWARE	56101 922598	17146 314716

Tableau 5.1 : répartition des données de notre Dataset (classification binaire)

Dataset CIC-AAGM2017	Les Classes concernés	Nb d'instances pour l'apprentissage	Nb d'instances pour le Test
Classification multiple	BENIGN ADWARE RANSOMWARE SCAREWARE SMS MALWARE ROOTKIT	56101 803362 619446 619251 618696 790662	17146 138602 157076 150701 150706 150416

Tableau 5.2: répartition des données de notre Dataset (multi-classification)

Pour la classification binaire :

Les fichiers CSVs de cette Dataset comportent plusieurs classes, ces derniers sont des MALWARES, et une seule classe BENIGN. Pour Binary classes, la modification d'étiquettes se fait comme suit :

L'étiquette de la classe Adware a été modifie en MALWARE classe.

L'étiquette de la classe Ransomware a été modifie en MALWARE classe.

L'étiquette de la classe Scareware a été modifie en MALWARE classe.

L'étiquette de la classe SMS MALWARE a été modifie en MALWARE classe.

L'étiquette de la classe Rootkit a été modifie en MALWARE classe.

L'étiquette de la classe BENIGN ne change pas malgré la présence de plusieurs dossiers BENIGN : BENIGN2015, BENIGN2016, BENIGN2017 avec les mêmes étiquettes

5.4.2 Pré-traitements des données :

L'importance de cette étape est de construire un modèle très précis, Le pré-traitement de l'ensemble de données est effectué avant d'être appliqué au réseau neuronal profond. Parmi ses étapes :

- Nous avons supprimé toutes les lignes redondantes qui représentent les instances des classes.
- Pour les valeurs des colonnes, Nous avons supprimé toutes les colonnes vides (leurs valeurs est 0) Par la fonction 'Attribut.DROP', ces colonnes sont : **'Bwd PSH Flags' , 'Fwd URG Flags' , 'Bwd URG Flags' , FIN Flag Count' , 'PSH Flag Count' , 'ECE Flag Count' , 'Fwd Avg Bytes/Bulk' , 'Fwd Avg Packets/Bulk' , 'Fwd Avg Bulk Rate' , 'Bwd Avg Bytes/Bulk' , 'Bwd Avg Packets/Bulk' , 'Bwd Avg Bulk Rate'.**
- Ensuite on a analysé toutes les données qui ont des valeurs NAN (NOT A NUMBER) ou INF (INFINITE VALUE).
- Nous avons supprimé aussi les colonnes qui n'ont pas des valeurs numériques par exemple : la date, l'IP, l'heure... etc. et aussi les adresse IP car les malwares peut être produit à tout moment par n'importe quelle machine contre n'importe quelle machine victime. Ces colonnes sont: **'source IP' , 'Destination IP' , 'Source Port' , 'Destination Port' , 'Timestamp'.**
- Après le pré-traitement des données, Nous avons normalisé toutes ces dernières (Les données d'entrées) : cette action est effectuée dans le but d'augmenter le taux d'apprentissage et aussi pour diminuer le taux d'erreurs.
- Les données d'entrainement sont divisées en 2, des données pour l'apprentissage et pour la validation de modèle. Ensuite, le modèle sera testé uniquement sur l'ensemble de test du Dataset CIC-AAGM2017.
- Ensuite Nous avons divisée toutes les données en 2 phases, des données d'entrainement et des données de test. La division est 80% pour l'entrainement et 20% pour le test.
- Nous avons utilisé la fonction OverSimpling (SMOTE) pour augmenter la taille de la classe BENIGN parce que cette classe est plus petite que l'autre. (umbalenced data-set).

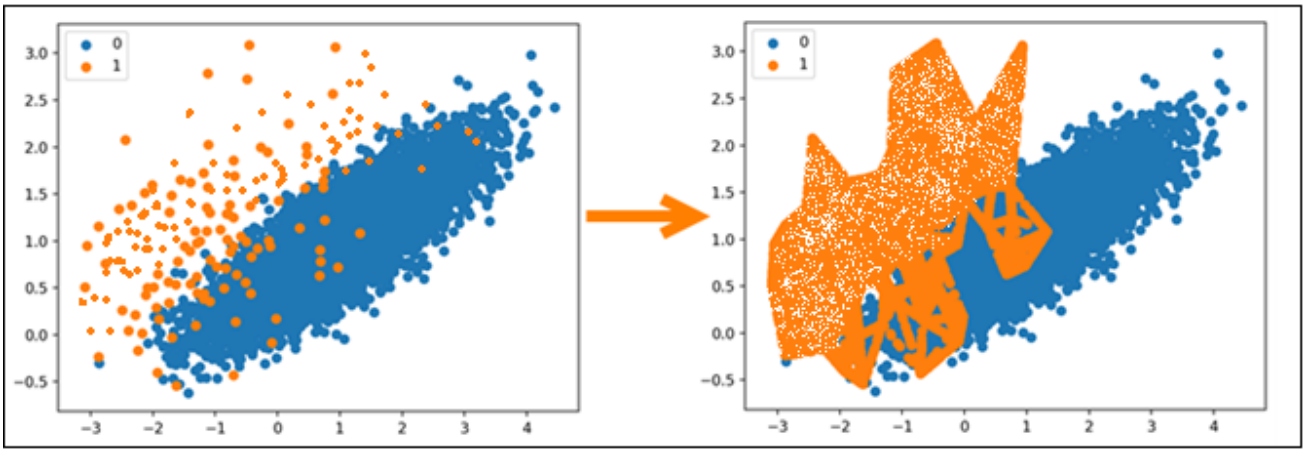

Figure 5.2: Le sur-échantillonage via SMOTE (Oversampling Technique) [W2]

- Dans ce travail on a utilisé aussi le paramètre class_weight pour gérer l'ensemble des données déséquilibrées.

Features	Type	Features	Type
Protocol	int64	Bwd IAT Max	float64
Flow Duration	int64	Fwd PSH Flags	int64
Total Fwd Packets	int64	Bwd PSH Flags	int64
Total Backward Packets	int64	Fwd URG Flags	int64
Total Len of Fwd Packets	float64	Bwd URG Flags	int64
Total Len of Bwd Packets	float64	Fwd Header Len	int64
Fwd Packet Length Max	float64	Bwd Header Len	int64
Fwd Packet Length Min	float64	Fwd Packets/s	float64
Fwd Packet Length Mean	float64	Bwd Packets/s	float64
Fwd Packet Length Std	float64	Min Packet Length	float64
Bwd Packet Length Max	float64	Max Packet Length	float64
Bwd Packet Length Min	float64	Packet Len Mean	float64
Bwd Packet Length Mean	float64	Packet Len Std	float64
Bwd IAT Min	float64	Fwd IAT Max	float64
Bwd IAT Total	float64	Fwd IAT Min	float64
Bwd IAT Mean	float64	Fwd Avg Packets/Bulk	int64
Bwd IAT Std	float64	Fwd Header Length.1	int64
Fwd Avg Bytes/Bulk	int64	FIN Flag Count	int64
Flow Packets/s	Object	SYN Flag Count	int64
Flow IAT Mean	float64	RST Flag Count	int64
Flow IAT Std	float64	PSH Flag Count	int64
Flow IAT Max	float64	ACK Flag Count	int64
Flow IAT Min	float64	URG Flag Count	int64
Fwd IAT Total	float64	CWE Flag Count	int64
Fwd IAT Mean	float64	ECE Flag Count	int64
Fwd IAT Std	float64	Down/Up Ratio	float64
Average Packet Size	float64	Avg Fwd Segment Size	float64
Avg Bwd Segment Size	float64	Active Std	float64
Fwd Avg Bulk Rate	int64	Bwd Avg Bytes/Bulk	int64
Bwd Avg Bytes/Bulk	int64	Bwd Avg Bulk Rate	int64
Subflow Fwd Packets	int64	Subflow Fwd Bytes	int64
Subflow Bwd Packets	int64	Subflow Bwd Bytes	int64
Init_Win_bytes_forward	int64	Init_Win_bytes_backward	int64
act_data_pkt_fwd	int64	min_seg_size_forward	int64
Active Mean	float64	Idle Max	float64

Tableau 5.3: L'ensemble des caractéristiques utilisées pour la détection des malwares

Cette étude est basée sur les caractéristiques de trafic extraites par CICFlowMeter-V3. Nous avons évalué le taux de détection et le taux de fausse alarme ainsi que d'autre métrique de classification pour l'approche proposée pour les deux types de classifications.

5.5 L'architecture du modèle :

Avec la multiplication des techniques de deep learning, On a implémenté notre modèle avec Le CNN 1D (Le réseaux de neurone convolutif d'une seule dimension). On a choisi cette approche a cause de ses performances et leur taux d'apprentissage élevée dans les Travaux existants dans ce domaine. La majorité de ces derniers ont obtenu les meilleurs résultats Avec cette architecture et aussi avec le Faster R-CNN. On a aussi utilisé cette technique car nos données sont des séries chronologiques avec une forme d'une seule dimension (forme textuel numérique). Notre Modèle est implémenté comme suite :

- Nous avons choisi la fonction ReLU Comme une fonction d'activation. Comme il ya aussi plusieurs fonctions (Tanh, sigmoid …etc.) mais les meilleurs résultats sont toujours obtenus avec «**ReLU**».
- Nous avons constaté qu'on a un problème du sur-apprentissage (Notre modèle est entrainé beaucoup avec les données d'entrainement. Donc le 'Accuracy' des données de test est réduit) ce problème est nommé OVERFITTING. C'est pour ça que nous avons utilisé la technique 'Dropout' pour combiner le pourcentage de l'Accuracy des données d'entrainement et les données de la validation.
- Nous avons utilisé la fonction 'Binary-cross-entropy' comme une fonction de perte (Loss).
- Nous avons utilisé aussi l'optimiseur 'Adam' avec un taux d'apprentissage 0.001 (Learning Rate = 0.001) au lieu de l'algorithme stochastique gradient descendant SGD (dans notre cas le SGD est donné un mauvais résultat). Le rôle de 'Adam' est de mettre à jour les poids d'un réseau de neurone afin de réduire la perte avec le minimum d'erreur.
- Notre modèle est implémenté avec plusieurs couches organisées comme suit : Des Couches Conv1D, Des filtres (64, 34 et 16) avec une longueur de taille 3, Une Couche global average pooling 1D et des couches Fully Connected Layers (la fonction Dense)

Voici un exemple de notre modèle dans Google Collab :

```
Model = Sequential ()
model.add (Conv1D (input_shape=(None, train.shape[2]), filters=64, kernel_size=3,
activation='relu', padding='same'))
model.add (Conv1D (filters=34, kernel_size=3, activation='relu', padding='same'))
model.add (Conv1D (filters=16, kernel_size=3, activation='relu', padding='same'))
model.add (GlobalAveragePooling1D())
model.add (Dense (130, activation='relu'))
model.add (Dropout (0.2))
model.add (Dense (1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

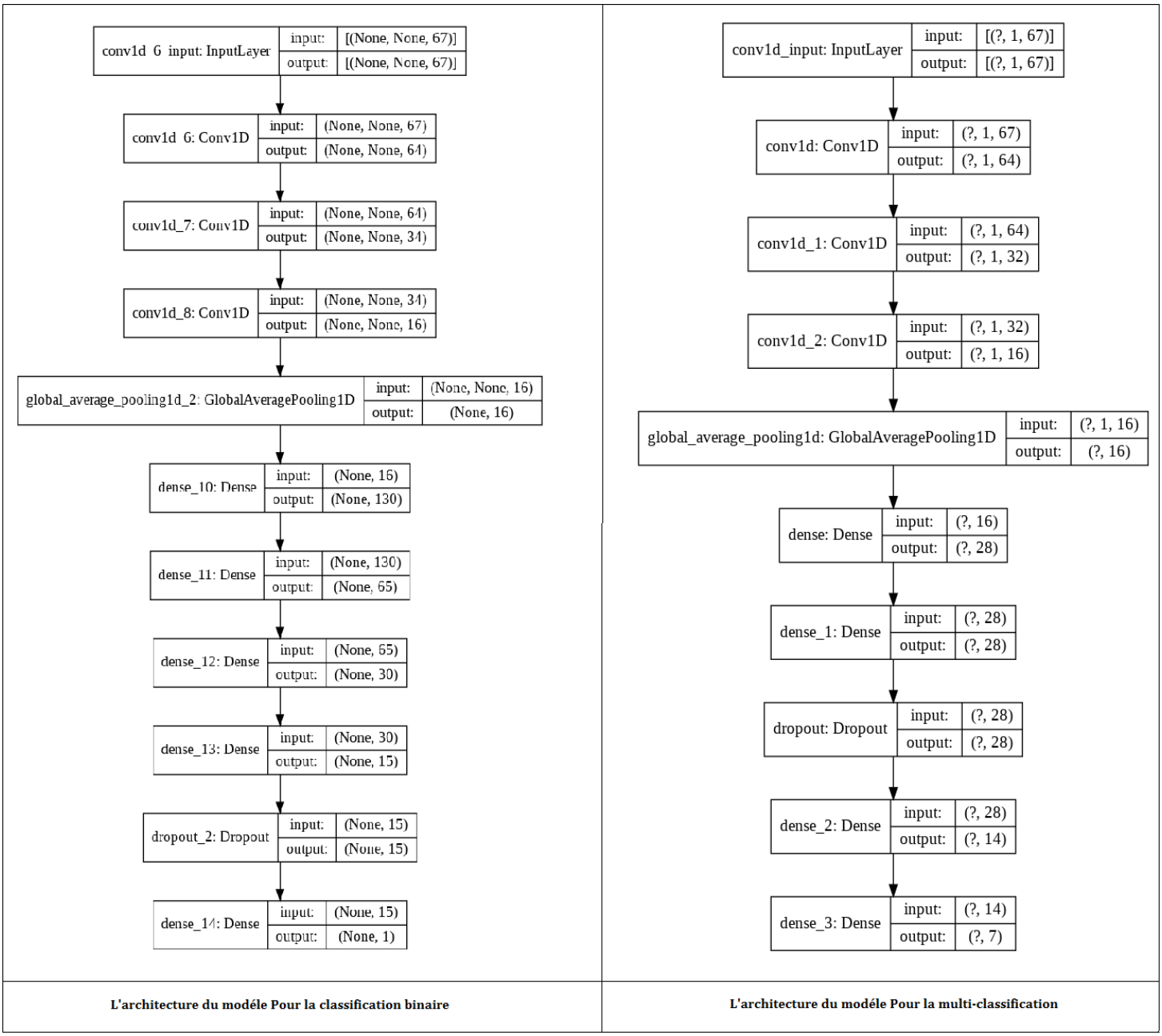

Figure 5.3: L'architecture de notre modèle pour les deux types de classification

5.6 Modèle de détection des malwares basé sur Le CNN :

Dans notre étude, les événements du trafic réseau sont représentés comme des données d'une série chronologique sous forme 1D.

Les CNNs 1D ont été initialement étudiées pour le traitement du langage naturel utilisant des couches de convolution 1D [6]. La couche convolutive 1D est composée de plusieurs filtres pour détecter plusieurs types de pattern (Les caractéristiques spéciaux) à la fois. Aussi à comme paramètre le filtre size, le

padding et aussi le stride qui contrôle l'opération de convolutif. Il nous permet aussi de réduire le coût de calcul et le nombre de paramètre du model pour augmenter l'efficacité de la détection.

Pour la couche de pooling layer, permet de réduire la complexité du calcul et de combat le overffiting

Le point faible des malwares c'est qui ils génèrent des comportements réseaux différents à l'habitude. Mais avec l'utilisation du CNN, nous permet facilement différencier entre le comportement normal et le comportement malicieux grâce à sa puissance d'identifier des caractéristique spéciaux à plusieurs niveau afin de discriminer le malware de la normal.

A chaque couche CNN, un ensemble de n noyaux (kernel) W= $\{w_1, w_2, \ldots, w_n\}$ et leurs biais B= $\{b_1, b_2, \ldots, b_n\}$, est convolutionné avec les données d'entrée. La convolution entre les données et chaque noyau produit une nouvelle carte de fonctionnalités X_k. Pour chaque couche convolutive, la transformation est définie par: [3]

$$X_k^l = \delta\left(W_k^{l-1} * X^{l-1} + b_k^{l-1}\right)$$

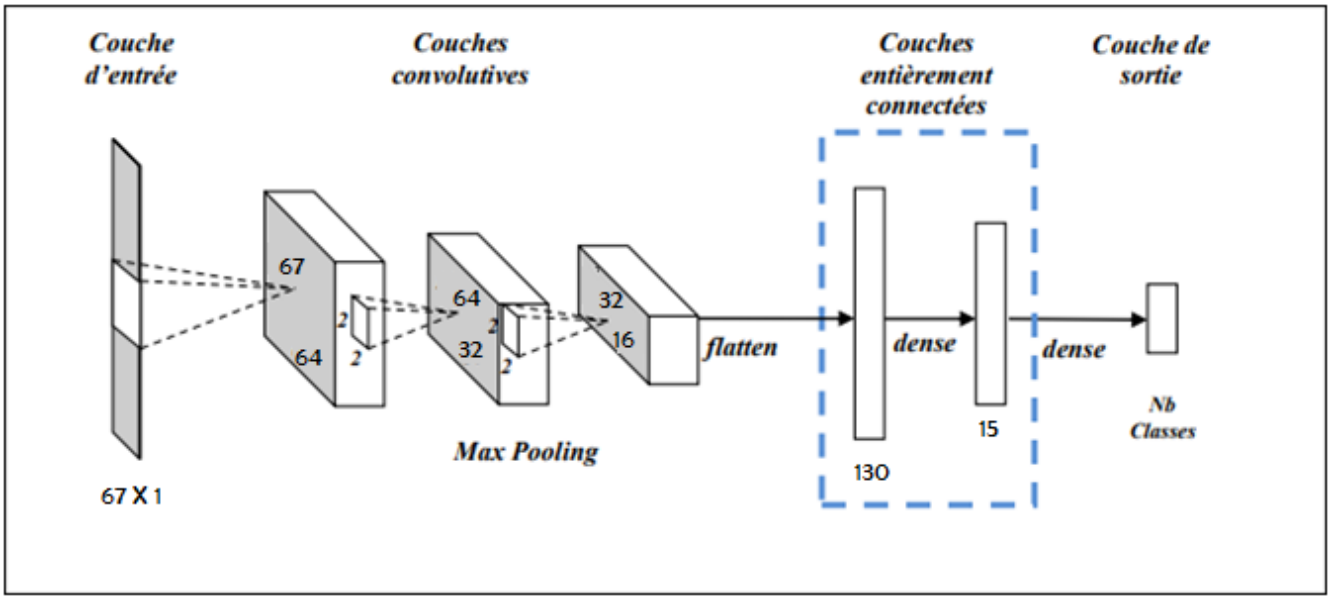

Figure 5.4: La détection des malwares basé sur le CNN 1D [12]

Au cours du processus d'apprentissage de CNN, une petite fenêtre est glissée sur les entrées et les valeurs de biais et de pondération via cette fenêtre peuvent être optimisées à partir de diverses caractéristiques des données d'entrée sans leur position dans les données d'entrée.

5.7 Implémentation et Résultats :

Cette étude est créée dans le but de développer un framework efficace pour la détection des logiciels malveillants. Notre but est de minimiser le taux d'erreurs et des fausses alarmes et aussi de maximiser les exactitudes possibles. Tout d'abord, plusieurs tests ont été faits afin d'obtenir les bons paramètres pour ce modèle. Ces paramètres (**Hyter-paramètres**) ne peuvent pas être ajustés durant la phase de l'apprentissage et pourtant, ils ont un grand impact sur les performances de modèle durant l'apprentissage. Ils comprennent les variables qui déterminent la structure du réseau (Nbr de neurones, Nbr de couches, fonction d'activation, . . .), le lot d'échantillons (Batch Size) et le nombre d'itérations ...etc.

Lorsqu'on arrive à un bon modèle avec le minimum de taux d'erreur et le maximum d'exactitude, nous avons ensuite testé ce modèle sur le sous-ensemble de test. Les résultats sont présentés dans les figures 5.4 et 5.5. On a obtenu un bon résultat comme une première expérience (Accuracy = 99%, Loss = 0.2%). Ces résultats montrent que l'apprentissage profond convient à la caractérisation des logiciels malveillants et qu'il est particulièrement efficace avec la disponibilité d'un plus grand nombre de données d'entraînement. Ce qui surpasse les techniques traditionnelles d'apprentissage automatique ML.

La classification binaire qui a été faite dans ce travail indiquée au tableau 5.1. Le modèle a été évalué directement sur l'ensemble de test. Ce dernier a été formé sur 30 itérations et nous notons que l'exactitude d'apprentissage et de validation augmente constamment du début à la fin, elle a atteint une valeur maximale tendant vers 1. Nous notons aussi que la valeur de perte diminue fortement durant l'entraînement et l'évaluation et atteint une valeur minimale tendant vers 0. Cela signifie que ce modèle apprend mieux et effectue des meilleures prédictions après chaque époque d'optimisation.

La classification Multi-classes qui a été faite dans ce travail est indiquée au tableau 5.2. Les données d'entraînement ont été divisées en 02 ensembles: 80% pour l'apprentissage et 20% pour l'évaluation. L'apprentissage ne prend pas beaucoup de temps, Il a obtenu une exactitude très bonne 97.96% avec l'architecture CNN. Nous notons ici que le modèle converge vers une valeur de perte minimale. Il a presque la même valeur de perte lors de l'apprentissage et l'évaluation. Ce qui indique que ce modèle sera généralisé bien au-delà de l'ensemble d'apprentissage. Ensuite, nous avons testé ce modèle sur l'ensemble de tests.

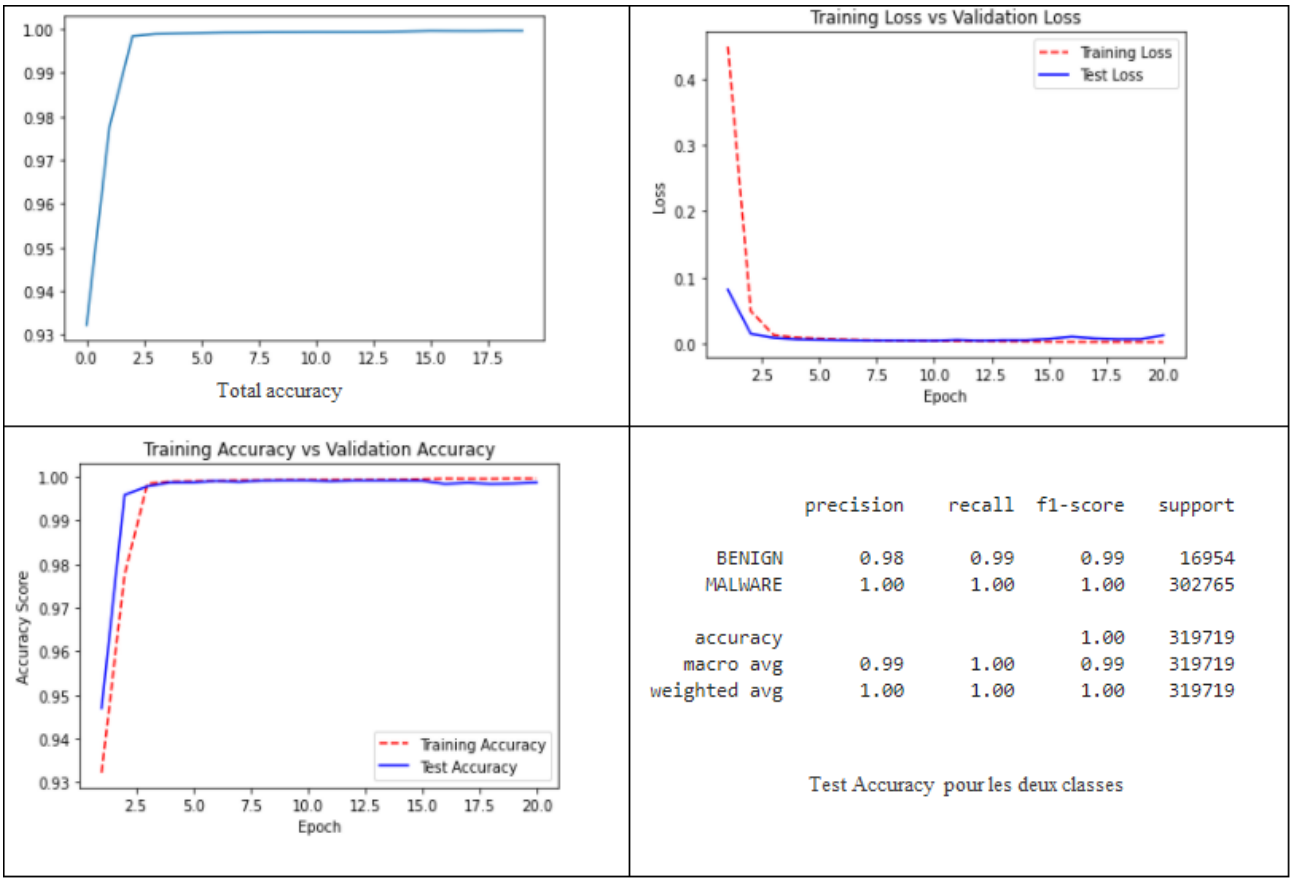

Figure 5.5: L'évaluation et les résultats du modèle pour la classification binaire

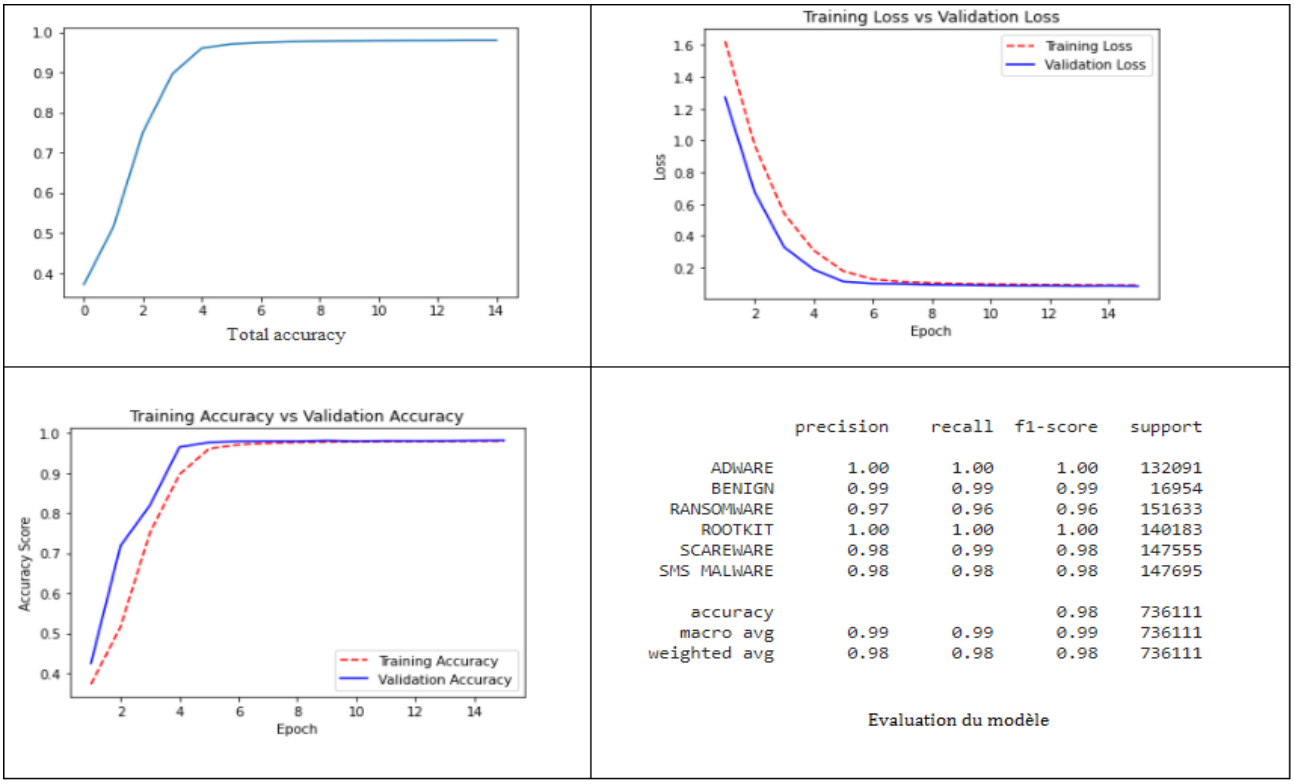

Figure 5.6: L'évaluation et les résultats du modèle pour la classification multiple

La courbe Receiver Operating Characteristic (ROC) est une mesure de performance souvent utilisé pour le classifieur binaire et surtout dans le cas où les classes sont déséquilibrées. Elle trace le taux de vrai positif en fonction du taux de faux positif à différents seuils de classification, afin de montrer le compromis entre la sensibilité (TPR) et la spécificité (FPR). Mathématiquement, il est calculé par la surface sous la courbe appelée Area Under Curve (AUC), Dans le cas idéal, nous aimerions avoir un Auc = 1, de sorte que la courbe ROC passe par le coin supérieur gauche du carré au point (FPR = 0, TPR = 1). Les courbes ROCs nous permettent de bien évaluer la précision de notre classifieur avec des classes très déséquilibrées ou la classe BENIGN ne représente qu'une 5.3% de l'ensemble de données. Nous pouvons constater que nos méthodes DL sont très performantes dans la discrimination entre la classe BENIGN et la classe MALWARE.

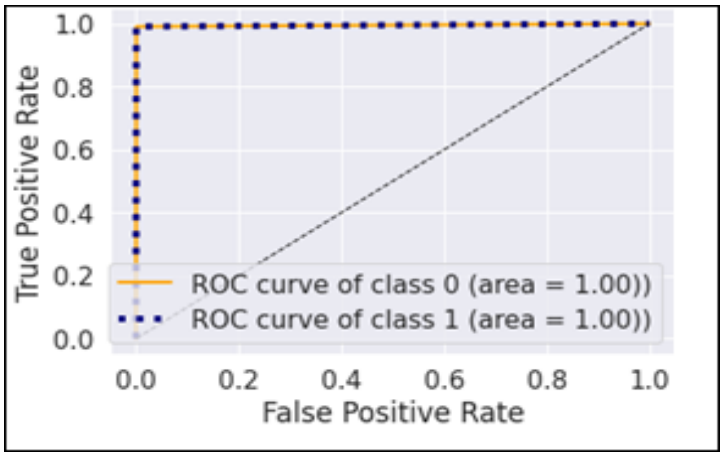

Figure 5.7: Les résultats du ROC courbe pour les classes (BENIGN /MALWARE)

5.8 Les mesures d'évaluation d'un modèle :

- Précision (Pr) : le pourcentage des Malwares identifiées comme des malwares TP parmi tous les exemples pré dictés comme malwares, il est donné par :

$$\text{Précision} = \frac{TP_{malware}}{TP_{malware} * FP_{BENIGN}}$$

- Recall (Rc) : le pourcentage des Malwares identifiées comme des malwares TP parmi tous les malwares dans l'ensemble de données :

$$\text{Recall} = \frac{TP_{malware}}{TP_{malware} * FN_{malwre}}$$

- F1-score (F1) : la moyenne harmonique pondérée de précision et de rappel (Recall), il est donné par :

$$\text{F1-Score} = \frac{2 * (Pr * Rc)}{Pr+Rc}$$

- True Positive Rate (TPR) : le nombre des malwares identifiées comme des malwares divisées par le nombre des malwares TP dans l'ensemble de données, il est donné par :

$$\text{TPR} = \frac{\sum TP_{malware}}{\sum malwares}$$

- False Positive Rate (FPR) : le nombre de cas bénins incorrectement classés comme des Malwares divisés par le nombre total de cas bénins dans l'ensemble de données, il est donné par :

$$\text{FPR} = \frac{\sum FP_{BENIGN}}{\sum Total\ BENIGN}$$

	Precision	Recall	F1-score	Support
BENIGN	0.99	0.99	0.99	16954
MALWARE	1.00	0.85	0.92	273946

Tableau 5.4 : Le rapport de classification binaire avec le CNN

	Precision	Recall	F1-score	Support
BENIGN	0.99	0.99	0.99	16954
ADWARE	1.00	1.00	1.00	132091
RANSOMWARE	0.97	0.96	0.96	151633
ROOTKIT	1.00	1.00	1.00	140183
SCAREWARE	0.98	0.99	0.98	147555
SMS MALWARE	0.98	0.98	0.98	147695

Tableau 5.5: Le rapport de la multi-classification (6_Classes) avec le CNN

	Approche utilisée	Precision	Recall	F1-score
Yujie et al. [13]	CNN 1D	97.90%	98.0%	97.80%
Chen et al. [15]	RandomForest	95.0%	95.0%	95.0%
Notre étude	CNN 1D ML traditionnelle	99.93% 99.83%	99.98% /	99.92% /

Tableau 5.6 : Comparaison des résultats des détections des malwares avec la dataset CIC-AAGM2017

5.9 Conclusion :

Nous avons implémenté un modèle du Deep learning avec l'architecture CNN, en utilisant l'ensemble des données CIC-AAGM2017 Dataset pour la détection des malwares. Le développement avait de nombreux problèmes qui nous ont fait perdre beaucoup de temps pour les résoudre. La masse des données du data-set, le déséquilibre et l'étiquetage des données et ainsi que les limites des outils matériels disponibles (processeur, mémoire). Afin de surmonter ces problèmes, nous avons combiné plus que 1 million de données dans 2 fichiers CSV. 80% de ces données dans un fichier Train.csv et 20% de ces données dans un fichier Test.csv. Avant l'implémentation du modèle, on a fait un pré-traitement pour nos données, La suppression des colonnes vides et les colonnes de Type NAN et INF, et aussi les colonnes de type Object. Ces caractéristiques ne contiennent aucune information discriminatoire permettant de différencier les classes Malwares, par contre ils peuvent donner des mauvais résultats. On a obtenu des résultats très satisfaisants avec une précision de détection très élevée comme une première expérience (Accuracy = 99%, Loss = 0.2%).

Conclusion Générale

Au terme de ce travail, il nous a été possible d'avoir une vue d'ensemble sur les techniques utilisées dans le domaine de la cyber-sécurité.

Nous avons démontré que l'intelligence artificielle et spécialement l'apprentissage automatique sont les méthodes les plus prometteuses pour arriver à éradiquer les malwares. Cependant notre travail reste à parfaire sur plusieurs fronts à savoir

- L'utilisation d'autres méthodes de deep learning en vue de comparer les méthodes entre elles
- L'affinage des procédés de prétraitement en vue de l'obtention de meilleurs ensembles de données prétraitées qui, en général, sont une condition sine qua non à l'obtention de meilleurs résultats.

Nous espérons continuer dans ce domaine et avoir un cadre de recherche pour y parvenir.

Bibliographie

[1]: Thomas, R. H. Core War: Creeper & Reaper

[2]: Kaspersky, E. (2008). Défis de la cybercriminalité. Securite globale, (4), 19-28

[3]: Mohammadpour, L., Ling, T. C., Liew, C. S., & Chong, C. Y. (2018). A convolutional neural network for network intrusion detection system. Proceedings of the Asia-Pacific Advanced Network, 46, 50-55

[4]: Hartmann, M., & Søgaard, A. (2021, April). Multilingual Negation Scope Resolution for Clinical Text. In Proceedings of the 12th International Workshop on Health Text Mining and Information Analysis (pp. 7-18)

[5]: Auray, N. (2015). L'invisible et le clandestin. Une ethnographie du virus informatique Storm. Terrain. Anthropologie & sciences humaines, (64), 32-49

[6]: Zhang, Y., & Wallace, B. (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv preprint arXiv:1510.03820

[7]: Chiang, K., & Lloyd, L. (2007). A Case Study of the Rustock Rootkit and Spam Bot. HotBots, 7(10-10), 7

[8]: TRUDEL, P., ABRAN, F., & DUPUIS, G. (2007). Analyse du cadre réglementaire québécois et étranger à l'égard du pourriel, de l'hameçonnage et des logiciels espions

[9] : HAMOUDA, D. (2020). Un système de détection d'intrusion pour la cybersécurité

[10]: Kim, K., Aminanto, M. E., & Tanuwidjaja, H. C. (2018). Network Intrusion Detection Using Deep Learning: A Feature Learning Approach. Springer

[11]: Prieur, B. (2020). Fondamentaux de la sécurité informatique utilisateur (Doctoral dissertation, IT-Akademy)

[12]: Baligh, M. B., & Hamza, M. L. Approche basée sur l'apprentissage profond pour la détection d'intrusion réseau

[13]: Yujie, P., Weina, N., Xiaosong, Z., Jie, Z., Wu, H., & Ruidong, C. (2020, December). End-To-End Android Malware Classification Based on Pure Traffic Images. In 2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP) (pp. 240-245). IEEE

[14]: Zhou, Y., & Jiang, X. (2012, May). Dissecting android malware: Characterization and evolution. In 2012 IEEE symposium on security and privacy (pp. 95-109). IEEE

[15]: Noorbehbahani, F., & Saberi, M. (2020, October). Ransomware Detection with Semi-Supervised Learning. In 2020 10th International Conference on Computer and Knowledge Engineering (ICCKE) (pp. 024-029). IEEE

[16]: Cédric BERTRAND. Les malwares sous android, Panorama des malwares sous Android, Analyser une application Android, juin 2012

[17]: Bartocci, E., & Falcone, Y. (Eds.). (2018). Lectures on Runtime Verification: Introductory and Advanced Topics (Vol. 10457). Springer

[18]: Chebaro, O. (2011). Classification de menaces d'erreurs par analyse statique, simplification syntaxique et test structurel de programmes (Doctoral dissertation, Université de Franche-Comté)

[19]: BELAOUED, M. (1955). Approches Collectives et Coopératives en Sécurité des Systèmes Informatiques

[20]: Ferlin, A. (2013). Vérification de propriétés temporelles sur des logiciels avioniques par analyse dynamique formelle (Doctoral dissertation, Toulouse, ISAE)

[21]: Mahdavifar, S., & Ghorbani, A. A. (2020). DeNNeS: deep embedded neural network expert system for detecting cyber attacks. Neural Computing and Applications, 32(18), 14753-14780

[22]: Leke, C. A., & Marwala, T. (2019). Deep learning and missing data in engineering systems. Springer International Publishing

[23]: Yann LeCun et al. Generalization and network design strategies. Connectionism in perspective, 19 :143–155, 1989

[24]: Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D., Wang, Y., & Iqbal, F. (2018, February). Malware classification with deep convolutional neural networks. In 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS) (pp. 1-5). IEEE

[25]: Mat, S.R.T., Ab Razak, M.F., Kahar, M.N.M. et al. Towards a systematic description of the field using bibliometric analysis: malware evolution. Scientometrics 126, 2013–2055 (2021)

[26] : Yuan, Z., Lu, Y., Wang, Z., & Xue, Y. (2014, August). Droid-sec: deep learning in android malware detection. In Proceedings of the 2014 ACM conference on SIGCOMM (pp. 371-372)

[27] : Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., & Venkatraman, S. (2019). Robust intelligent malware detection using deep learning. IEEE Access, 7, 46717-46738

[28] : Zhu, D., Jin, H., Yang, Y., Wu, D., & Chen, W. (2017, July). DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data. In 2017 IEEE symposium on computers and communications (ISCC) (pp. 438-443). IEEE

[29] : Kim, T., Kang, B., Rho, M., Sezer, S., & Im, E. G. (2018). A multimodal deep learning method for android malware detection using various features. IEEE Transactions on Information Forensics and Security, 14(3), 773-788

[30] : Karbab, E. B., Debbabi, M., Derhab, A., & Mouheb, D. (2018). MalDozer: Automatic framework for android malware detection using deep learning. Digital Investigation, 24, S48-S59

[31] : Iadarola, G., Martinelli, F., Mercaldo, F., & Santone, A. (2021). Towards an interpretable deep learning model for mobile malware detection and family identification. Computers & Security, 105, 102198

[32] : Yuan, Z., Lu, Y., & Xue, Y. (2016). Droiddetector: android malware characterization and detection using deep learning. Tsinghua Science and Technology, 21(1), 114-123

[33] : Deng, L. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. APSIPA Transactions on Signal and Information Processing, 3

[34] : Al-Maksousy, H. H., Weigle, M. C., & Wang, C. (2018, October). NIDS: Neural network based intrusion detection system. In 2018 IEEE International Symposium on Technologies for Homeland Security (HST) (pp. 1-6). IEEE

[35] : Han, S. W., & Lee, S. J. (2009). Packed PE file detection for malware forensics. The KIPS Transactions: PartC, 16(5), 555-562

[36] : Deng, L., & Yu, D. (2014). Deep learning: methods and applications. Foundations and trends in signal processing, 7(3–4), 197-387

[37] : Li, X., Loh, P. K., and Tan, F. (2011). Mechanisms of polymorphic and metamorphic viruses. In Intelligence and Security Informatics Conference (EISIC), 2011 European, pages 149–154. IEEE

[38] : Toderici, A. H. and Stamp, M. (2013). Chi-squared distance and metamorphic virus detection. Journal of Computer Virology and Hacking Techniques, 9(1) :1–14

[39] : Carlini, N., & Wagner, D. (2017, May). Towards evaluating the robustness of neural networks. In 2017 ieee symposium on security and privacy (sp) (pp. 39-57). IEEE

[40] : KP, S., & Alazab, M. (2020). A Comprehensive Tutorial and Survey of Applications of Deep Learning for Cyber Security

[41] : Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. E. R. T. (2014, February). Drebin: Effective and explainable detection of android malware in your pocket. In Ndss (Vol. 14, pp. 23-26)

[42] : Andriatsimandefitra Ratsisahanana, R. (2014). Caractérisation et détection de malware Android basées sur les flux d'information (Doctoral dissertation, Supélec)

[43] : Ferlin, A., Bon, P., Wiels, V., & Collart-Dutilleul, S. (2015, June). Vérification parallélisée de propriétés temporelles sur des traces d'exécution, par analyse dynamique formelle. In Approches Formelles dans l'Assistance au Développement Logiciel (pp. pp-1)

[44] : Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1097-1105

[45] : O'Kane, P., Sezer, S., & McLaughlin, K. (2011). Obfuscation: The hidden malware. IEEE Security & Privacy, 9(5), 41-47

[46] : LeCun, Y. (2016). L'apprentissage profond, une révolution en intelligence artificielle. La lettre du Collège de France, (41), 13

[47] : Erbschloe, M. (2004). Trojans, worms, and spyware : a computer security professional's guide to malicious code. Butterworth-Heinemann

[48]: Shabtai, A., Moskovitch, R., Elovici, Y., & Glezer, C. (2009). Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. information security technical report, 14(1), 16-29

[49] : Filiol, E. (2009). ´ Les virus informatiques : th´eorie, pratique et applications. Springer Science & Business Media

[50] : Sommer, R., & Paxson, V. (2010, May). Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE symposium on security and privacy (pp. 305-316). IEEE

[51] : Masud, M., Khan, L., & Thuraisingham, B. (2011). Data mining tools for malware detection. CRC Press

[52] : Sommer, R., & Paxson, V. (2010, May). Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE symposium on security and privacy (pp. 305-316). IEEE

WEBOGRAPHIE

[W1] : Faits et analyses sur la cybercriminalité, https://www.av-test.org/fr/nouvelles/faits-et-analyses-sur-la-cybercriminalite-le-rapport-de-securite-2019-2020-dav-test/. [En ligne ; Consulté le 03/05/2021]

[W2] : umbalenced learning, https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html. [En ligne ; Consulté le 06/07/2021]

[W3] : Sébastian Raschka, difference between deep learning and usual machine learning, https://www.quora.com/What-is-the-difference-between-deep-learning-and-usual-machine-learning. [En ligne ; Consulté le 18/07/2021].

[W4] : Mohamed brahimi, detection des virus basee sur l'apprentissage incremental actif, https://www.researchgate.net/profile/MedBrahimi/publication/266150553 Nouveau_systemede_detection_de_virus_base_sur_l%27apprentissage_incremental_actif/links / /542912f60cf26120b7b584b7/Nouveau-systeme-de-detection-de-virus-base-sur-lapprentissage-incremental-actif.pdf7. [Consulté le 21/07/2021].

[W5] : Canadian institute for Cybersecurity, Android Adware and General malware Dataset 2017 CIC-AAGM2017 https://www.unb.ca/cic/datasets/android-adware.html. [Consulté le 28/07/2021].

[W6] : Jeff Hale. Deep learning framework. https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a. [En ligne ; Consulté le 09/08/2021].

[W7] : A Flow Based Approach to Detect Advanced Persistent Threats in Communication Systems - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Malware-types-and-their-properties_tbl1_327813496 [En ligne ; Consulté le 21/05/2021].

[W8] : Michael Middleton. Deep learning vs. Machine learning — what's the difference ? https://flatironschool.com/blog/deep-learning-vs-machine-learning, [Consulté le 18/07/2021].