MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation : Technologies de l'information et de la communication (TIC)

Forensique des mobiles

Head of telecom labs
HEPIA à Geneva
Sous la supervision Prof Tewfiq El Maliki

Fait par

Nemanja Pantic

Sous la direction de
Prof Tewfiq El Maliki
Head of telecom labs
HEPIA Geneva

Docteur Lassaad Gannoun

Lausanne, HES-SO Master, 2021

Accepté par la HES-SO Master (Suisse, Lausanne) sur proposition de

Tewfiq, El Maliki, conseiller du projet d'approfondissement
Docteur Lassaad Gannoun

Lausanne, le 04.06.2021

Prof.Tewfiq, El Maliki	Prof. Vincent Peiris
Conseiller	Responsable de la filière

Table of Contents

Remerciements

Je tiens à remercier, pour l'aide apportée lors de ce projet, ma famille et mes amis pour le soutien moral qu'ils m'ont apporté, plus particulièrement je souhaite remercier Maxime Nussbaumer pour la relecture de mon rapport, je remercie également Anastasia Charles pour son soutien, Je remercie tout autant le Prof. Tewfiq El Maliki pour ses conseils donnés tout au long du projet et Mhedhbi Maher pour son retour sur l'implémentation de ma solution et ses conseils.

Abréviations

HPO	Hyperparameter optimization
APK	Android Package
BO-GP	Bayesian optimization Gaussian Process
BO-TPE	Bayesian optimization Tree-structured Parzen Estimator
PSO	Particle swarm optimization
IOT	Internet Of Things
API	Application Programming Interface

Abstract

Dans un monde où nous sommes de plus en plus connectés que ça soit par la présence d'informatique dans les voitures ou frigo. Dans un monde où presque une personne sur deux possède un téléphone intelligent, il devient important de pouvoir se protéger d'attaque venant de l'extérieur. Trouver un outil permettant de détecter une preuve d'intrusion est une première étape vers la sécurisation de ses données et de sa vie privée. Le nombre d'attaque ne cesse d'augmenter chaque jour.

Afin de protéger les personnes les plus à mêmes de tomber dans le piège d'un pirate, développer des outils permettant de prédire une attaque lors de l'installation d'une application permettrait de protéger ces personnes.

L'objectif du travail a été de faire un état des lieux des produits de forensique existant et de lister ceux utilisant du machine learning.

Une implémentation d'anciennes études sur la détection de malwares basées sur les permissions Android a été mise sur place et obtient de bons résultats. De plus, l'outil développé permet la prise en main simple d'outil de forensique et guide l'utilisateur afin qu'il puisse créer lui-même ses jeux de données afin d'obtenir les meilleurs modèles de détection possible.

Le code est public et est disponible sur GitHub.

Key Words: Machine Learning, Digital Forensique, Malware Detection, Android Permissions

1. Introduction

Le digital forensique permet de recréer des événements du passé afin d'y effectuer des analyses permettant d'apporter une preuve d'attaque à l'aide de traces laissée par l'attaquant. Cette science est utilisée depuis plusieurs années dans le domaine des téléphones mobiles, il est par exemple possible de récupérer des données supprimées.

Du fait de la démocratisation des appareils mobiles, le sujet de la sécurité des portables doit se poser, des techniques plus développées doivent être envisagées afin de protéger au mieux l'utilisateur. Le but de ce projet est de fournir un état des lieux concernant le forensique des smartphones utilisant Android.

Plusieurs sociétés ont fait du forensique mobile leur fonds de commerce afin d'aider des entreprises ou des particuliers à protéger leur téléphone.

Ce rapport est réparti en plusieurs parties :

- Un état de l'art sera fait par rapport au nombre d'utilisateurs et les outils existants de forensique ;
- Une partie est dédiée à la détection de malware à l'aide de machine learning, des études seront citées ;
- Une description du produit crée à terme de ce projet permettant de détecter une APK malicieuse ou bénigne à l'aide de machine learning

L'utilisation du *machine learning* dans le forensique des mobiles est très peu renseignée, en termes de quantité d'études, la plupart d'entre elles se contentant de reprendre une branche du machine learning telles que la détection de malware.

1.1. But de l'étude

Le but de ce projet est d'améliorer les méthodes d'extraction de données des mobiles et de les analyser à l'aide de machine learning.

La première partie du projet consiste à rechercher des outils existants sur le marché, ces derniers seront séparés en deux catégories. La première d'entre elles sera destinée à lister les différents outils payants permettant d'effectuer du forensique sur un OS Android tandis que la seconde servira à décrire une liste d'outils gratuits, disponible par exemple sur GitHub.

Une deuxième partie consistera à trouver et comparer des listes de jeu de données de virus Android présent sur le net. Ces derniers peuvent avoir différentes formes, une explication sur chacun de ces jeux de données sera faite afin de pouvoir choisir celle qui nous correspondrait le mieux dans un contexte de détection de malware à l'aide de machine learning.

Une troisième partie sera dédiée à donner les résultats de différentes études déjà faites sur la détection de malwares à l'aide de machine learning.

Finalement, une présentation d'une solution développée lors de ce projet sera faite. Cette solution permet d'extraire toutes les permissions présente dans une APK, de stocker ces permissions dans un fichier .csv et d'effectuer du machine learning afin de détecter si oui ou non, une application est malicieuse.

1.2. Portée et limitation du sujet

Il s'agit d'un projet académique visant à améliorer les techniques d'extraction de données et de traitement à l'aide de machine learning. Le projet ayant un temps limité, une implémentation sera faite si le temps le permet.

Les buts principaux sont :

- Établir un état de l'art et de la technique en termes de forensique des mobiles ;
- Présenter des études ayant été faites sur la détection de malware à l'aide de machine learning.

Si le temps le permet :

- Implémenter une application de machine learning basée sur un produit existant ou non et obtenir des résultats.

2. Revue de la littérature et proposition de recherche

Le but de cette section est d'obtenir un point de vue sur le marché des systèmes d'exploitation des smartphones et des risques que cette généralisation engendre. Ensuite nous parlerons de techniques de machine learning. La lecture se poursuivra sur le domaine de forensique, nous verrons un point de vue générale du forensique et décriront de quel sous ensemble fait partie le forensique des mobiles. Finalement, nous retrouverons des tableaux de comparaisons d'outils de forensique commerciaux et open source afin de pouvoir avoir un point de vue sur ce qui existe sur le marché.

2.1. Etat de l'art

Vous trouverez en annexe la spidermap du projet, cette dernière se complète également avec le schéma disponible à la figure 9.

2.1.1. Utilisation des smartphones

Selon bankmycell.com[1], 3.8 milliards de personnes possèdent un smartphone en 2021, ce qui signifie que 48.33% de la population mondiale possède aujourd'hui un smartphone. Si nous comparons ce chiffre à celui de 2016, le nombre de personnes ayant un smartphone était de 2.5 milliards, ce qui représentait alors 33.58% de la population mondiale. Selon gs.statcounter [2], en mai 2021, 72.72%% des utilisateurs de smartphone utilisent Android. Le deuxième système le plus utilisé dans le monde est iOS, avec 26.47% d'utilisateurs dans le monde.

Ces chiffres nous montrent que les appareils mobiles sont aujourd'hui un outil commun utilisé pour téléphoner, naviguer sur le Web, effectuer des virements en ligne ou pour utiliser les réseaux sociaux.

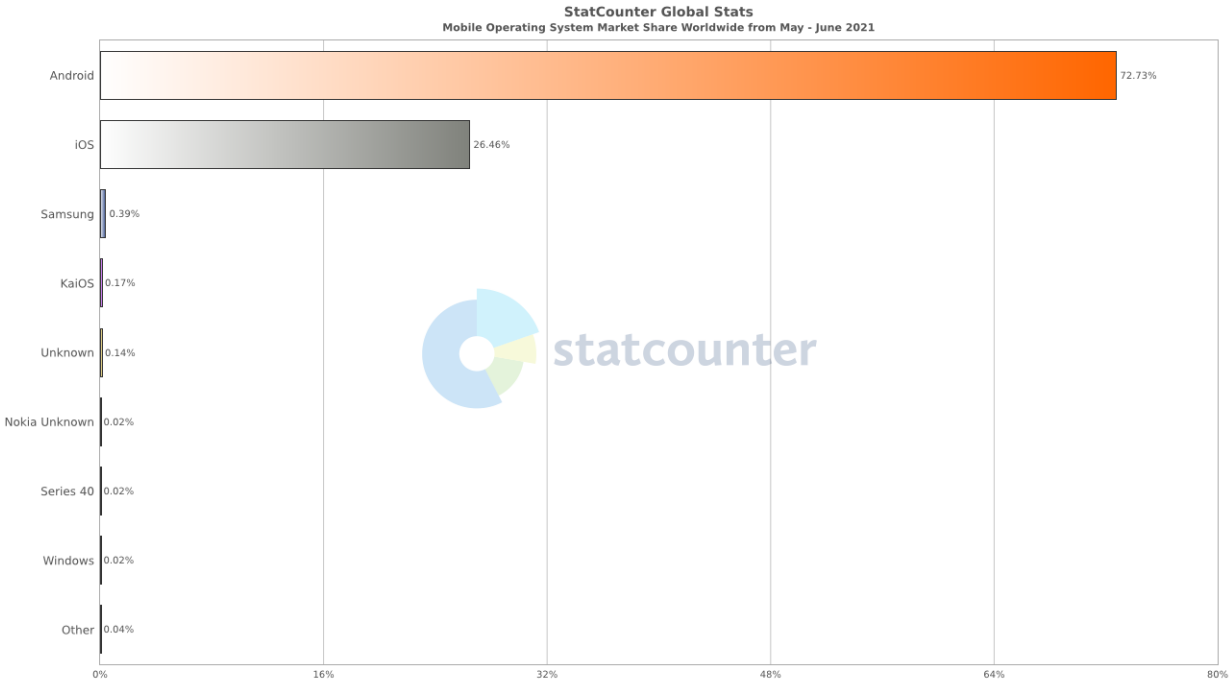


Figure 1 Système d'exploitation utilisé dans le monde (en % de la population) - gs.statcounter,2021

[1] https://www.bankmycell.com/blog/how-many-phones-are-in-the-world

[2] https://gs.statcounter.com/os-market-share/mobile/worldwide

Ces téléphones sont remplis de données personnelles (SMS, conversation privées, mots de passe et de photos). Toutes ces informations sont prisées par les pirates informatiques car elles permettent de gagner de l'argent en revendant les données par exemple.

Un point rendant le forensique des appareils Android difficile réside dans sa grande diversité d'hardware et de versions. Ce système d'exploitation est le plus utilisé car il est libre de droits et est par conséquent utilisable par tout le monde.

Il existe plusieurs attaques, en voici une liste non-exhaustive accompagnée d'une petite définition :

1. *Ransomware* : l'attaquant va chiffrer l'intégralité de vos données et va menacer de les supprimer après X temps si vous ne payez pas la somme qu'il vous demande.
2. *Spyware* : Il s'agit d'un logiciel espion existant dans le but de collecter et de transférer des données sur votre téléphone à un serveur distant.
3. Keylogger : Enregistre vos saisies et les envoie à un serveur distant.

Les attaques sur ciblant les Android sont de plus en plus nombreuses [3] et leurs variété ne cesse d'évoluer.

Ce système est basé sur Linux et était de base destinée pour les téléphones et tablettes mais au fil du temps, Android s'est retrouvé dans nos télévisions, voitures ou montre connectées. Ce qui rend difficile l'extraction des données dans la majorité des cas.

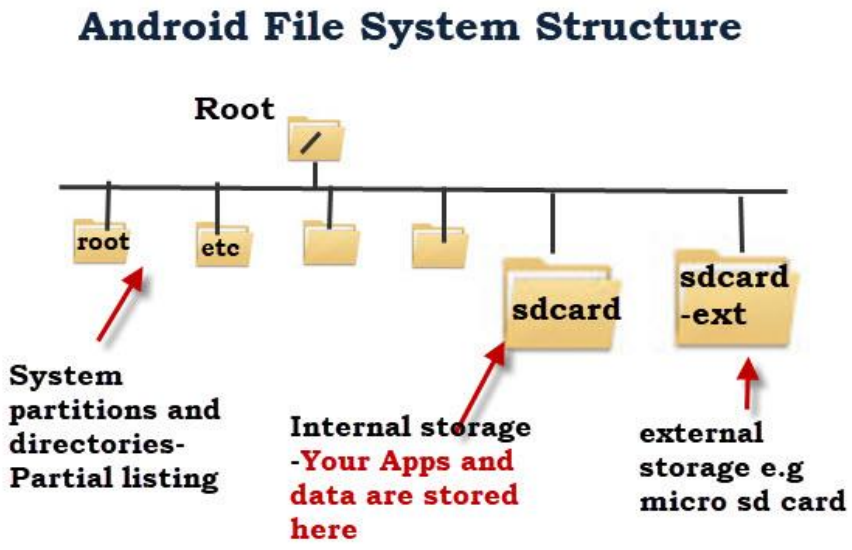


Figure 2 Hiérarchie simplifiée des fichiers Android, swipetips.com[4]

Lorsque vous parcourez vos photos, applications, vous allez lire les éléments présents dans votre carte SD.

Afin d'installer des applications sur notre smartphone Android, nous allons utiliser un fichier d'installation nommé APK qui contient les différents fichiers nécessaires à l'installation de l'application.

L'APK utilisée précédemment est stockée dans le dossier /data/ , ce dernier contient toutes les APK installées sur le téléphone.

[3] https://www.statista.com/statistics/680705/global-android-malware-volume/

[4] https://www.swipetips.com/android-files/

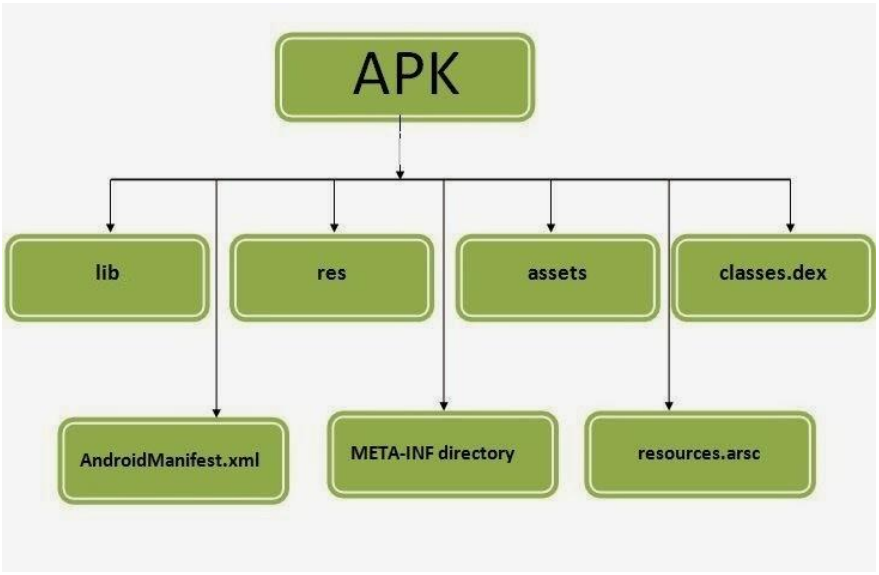


Figure 3 Architecture d'une APK, Eugen Minibaev[5]

- Lib : contient le code compilé selon le type de processeur (x86, x86_64, arm64-v8a, etc.)
- Res : contient les ressources non compilées
- Assets : contient les assets de l'application
- AndroidManifest.xml : fichier décrivant le nom, la version, les droits d'accès, les librairies utilisées par l'application.
- Classes.dex : les classes compilées au format dex, compréhensible par Dalvik Virtual machine et Android Runtime.
- Resources.arsc : un fichier contenant des ressources précompilées.

2.1.2. Machine Learning

Ce chapitre a été rédigé pour donner suite à la lecture de [1]. Ce papier donne une définition claire des concepts de machine learning et de l'état de la technique.

À proprement parler, le machine learning est une branche dérivée de l'apprentissage automatique aussi connu sous le nom d'intelligence artificielle. L'apprentissage automatique désigne « un mode d'apprentissage par lequel un agent évalue et améliore ses performances et son efficacité sans que son programme soit modifié en acquérant de nouvelles connaissances et aptitudes à partir de données et/ou en réorganisant celles qu'il possède déjà ».[6]

En d'autres termes, il s'agit d'un domaine informatique qui se concentre sur le fait de créer des machines ayant des comportements qu'un être humain pour juger comme intelligent, une intelligence artificielle n'a pas de « pensée propre à elle », elle permet à un ordinateur d'effectuer des tâches qu'un humain pourrait faire. Un exemple d'intelligence artificielle serait par exemple un petit capteur qui va fournir des informations à un agent qui va se charger d'effectuer la décision pour laquelle il a été programmé.

Les premières études sur l'intelligence artificielle remontent à la seconde moitié du XXème siècle, comme en atteste les recherches menées par Claude Shannon dans son article intitulé « Programming a Computer for Playing Chess »[7], dans lequel il relate le développement de son algorithme « Minamax ». Ce dernier permet de calculer un score pour chaque coup jouable et de choisir celui permettant d'aboutir, dans la majorité des cas, à une victoire.

---

[5] https://www.researchgate.net/publication/317741139_Static_Dalvik_VM_bytecode_instrumentation

[6] http://gdt.oqlf.gouv.qc.ca/ficheOqlf.aspx?Id_Fiche=8395061

[7] https://vision.unipv.it/IA1/ProgrammingaComputerforPlayingChess.pdf

Le machine learning quant à lui peut être vu comme une façon d'offrir la possibilité à un ordinateur de programmer par lui-même. Le fait d'écrire soi-même son code peut être vu comme un goulot d'étranglement en termes de résolution de problèmes car cette dernière est limitée par nos connaissances et notre imagination.

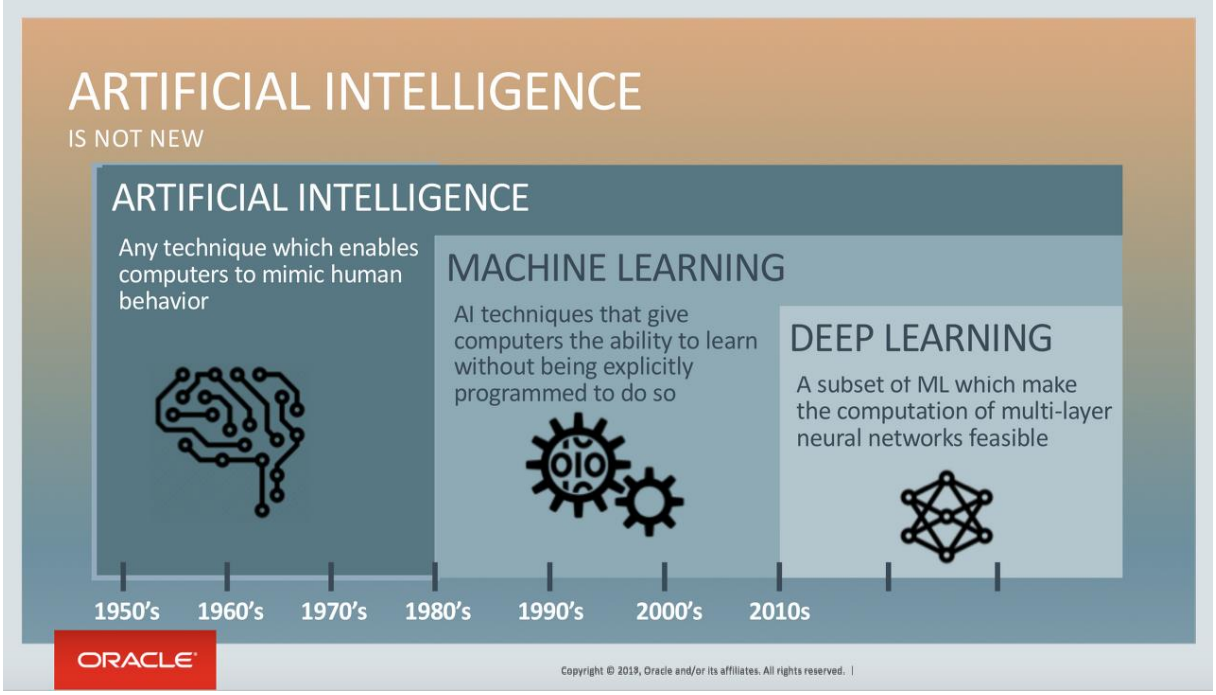


Figure 4 Evolution de l'intelligence artificielle selon le temps et définition de ses sous-ensemble - oracle[8]

Le machine learning permet d'utiliser les données comme moyen d'écrire du code, il permettrait même de trouver « des solutions que l'être humain ne pourrait pas décrire »[9]. Il faut voir le machine learning comme un système permettant d'apprendre à partir d'expériences et d'exemples. Les décisions seront prises en fonction de l'apprentissage fait par le système.

Un bref point décrivant le deep learning me semble intéressant à apporter : cette technique est un sous-ensemble de machine learning où nous allons utiliser un réseau de neurones artificiels qui va apprendre et s'améliorer avec le temps à l'aide d'une grande quantité de données. Ce projet ne traitera pas de deep learning.

Les données ayant aujourd'hui un domaine bien à elle (Data Science) et des enjeux économiques (la vente de celles-ci), ces dernières se retrouvent très nombreuses et les utiliser en les fournissant à un système de machine learning ou de deep learning permet beaucoup d'applications. La data science est aujourd'hui indissociable de l'intelligence artificielle[10].

Il existe plusieurs techniques générales de machine learning. Ces différentes méthodes sont décrites ci-dessous :

[8] https://blogs.oracle.com/bigdata/post/whatx27s-the-difference-between-ai-machine-learning-and-deep-learning

[9] https://www.youtube.com/watch?v=B8J4uefCQMc

[10] https://ai.plainenglish.io/data-science-vs-artificial-intelligence-vs-machine-learning-vs-deep-learning-50d3718d51e5

	Intelligence artificielle	Machine learning	Deep learning
Définition	Toutes techniques permettant à une machine d'imiter un comportement humain.	Sous-ensemble d'intelligence artificielle utilisant des méthodes mathématiques et statistiques afin de s'améliorer	Sous-ensemble de machine learning utilisant plusieurs niveaux de neurones artificiels afin d'apprendre.
Technique utilisée	Data Science	Utilise l'intelligence artificielle ainsi que la data science	Utilise le machine learning et la data science
Ensemble de données	N'en a pas forcément besoin.	Peut fonctionner avec un petit ou moyen jeu de données	A besoin de beaucoup de données
Précision[11]	Dépend de l'implémentation du programmeur	Modérée, avec de faux positifs autour de 5%	Très haute avec un pourcentage de faux positifs proches de 0
Date	1950	1980	2010

Machine Learning Supervisé

Dans ce type d'apprentissage, nous fournissons un ensemble de données dont nous connaissons la sortie souhaitée, ces données sont donc souvent représentées à l'aide de labels. Nous savons donc à l'avance ce que doit donner l'information à l'aide d'un traitement sur cet ensemble.

Nous laissons le système trouver une fonction qui va faire correspondre une observation (une entrée) à une sortie désirée. Nous aurons donc en général une base de données d'entraînement contenant :

(X1, Y1)

X1 étant l'entrée et Y1 la sortie désirée.

Le modèle qui en sortira dépendra donc de ce que nous souhaitons obtenir. En prenant l'exemple de détection de pièce de monnaie, nous pourrions les détecter à l'aide de plusieurs éléments les caractérisant, leur taille, couleur, masse.

Certaines caractéristiques seront plus constantes que d'autres, nous pouvons par exemple imaginer que la couleur d'une pièce se détériore avec le temps ce qui empêcherait de correctement les différencier. De plus certaines pièces ont la même couleur.

Le modèle qui en sortira ne sera donc pas assez précis, tandis que si nous utilisons la taille ou la masse d'une pièce, la classification sera de meilleure qualité car ces caractéristiques sont souvent propres à chaque type de pièce. Notons toutefois qu'utiliser uniquement une de ces caractéristiques n'est pas recommandé, il est préférable d'en utiliser plusieurs afin d'obtenir une précision accrue. Cet exemple est tiré du cours en ligne [12]donné par le Professeur Yaser Abu-Mostafa.

Prenons le cas des malwares, en donnant des paramètres qui sont propres à ces derniers à un système de machine learning supervisé, nous devrions avoir de bons résultats d'analyse. Des détails seront donnés plus loin sur les caractéristiques qui nous semblent adéquates.

Il faut donc faire attention au jeu de données et un grand travail doit se faire sur ce dernier. Nous allons donc souvent utiliser cette méthode d'apprentissage lorsque nous souhaitons faire de la classification, c'est-à-dire, définir selon ses connaissances, quel type d'objet nous avons en entrée (), nous faisons donc une séparation selon différentes observations.

[11] https://www.deepinstinct.com/2018/06/24/from-machine-learning-to-deep-learning-and-what-it-means-for-cybersecurity-part-1/

[12] https://www.youtube.com/watch?v=mbyG85GZ0PI

Nous utilisons également ce type d'apprentissage lorsque nous souhaitons faire de la régression, cette dernière permet de faire des prédictions sur une valeur discrète selon les données fournies. Par exemple, en ayant une liste des années et la population mondiale, nous pourrons prédire, selon la fonction obtenue, la population pour une année future ou l'année pour une taille de population donnée.

Il est également possible de détecter des anomalies à l'aide de la supervision. Dans un contexte de sécurité, il est plus pertinent d'utiliser la supervision lorsque nous souhaitons détecter des attaques connues. Nous utiliserons souvent cette méthode afin de détecter des spams ou des fraudes.

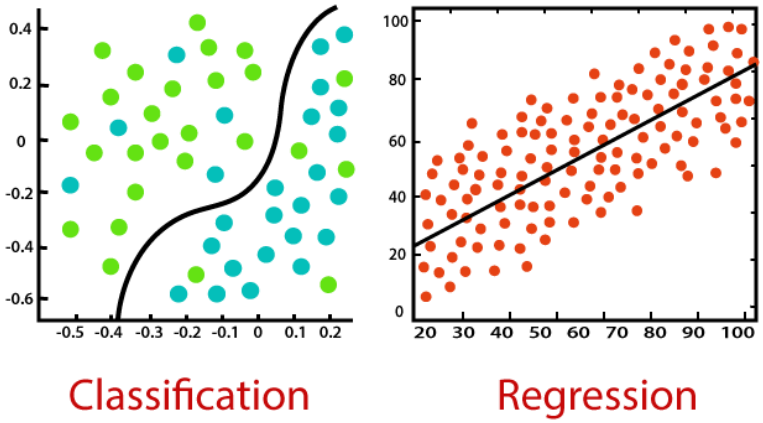


Figure 5 Différences entre classification et régression[13]

Machine Learning Non-supervisé

Cette méthode d'apprentissage diffère dans ses données, cette dernière n'aura aucune information sur ce que le système est censé comprendre.

Il doit trouver de lui-même un lien entre les données. L'objectif est donc de découvrir une structure ou une similarité entre les données fournies, ces dernières n'ayant aucun label, ce travail se fera lors de la phase d'apprentissage.

Nous aurons donc en général une base de données d'entraînement contenant :

(X1)

X1 étant la donnée à traiter

Il est donc possible de découvrir des classes ou de grouper des données selon leur similarité. La classification se fera sur la différence et la similarité que ces données auront selon d'autres points. Par exemple notre algorithme va être capable, selon l'implémentation, de détecter la différence entre un humaine, une voiture et des étoiles. C'est cette technique qu'utilise Netflix [14]afin de recommander des séries similaires à celle que regarde un utilisateur.

La majorité des données dans le monde ne sont pas labélisées, la non-supervision est plus compliquée à utiliser afin d'obtenir un résultat correct, cette dernière reste moins utilisée que la supervision mais rester quand même à prendre en compte car cette dernière permettrait de découvrir plus facilement une attaque inconnue sur un téléphone.

[13] https://www.javatpoint.com/regression-vs-classification-in-machine-learning

[14] https://blog.bismart.com/en/classification-vs.-clustering-a-practical-explanation

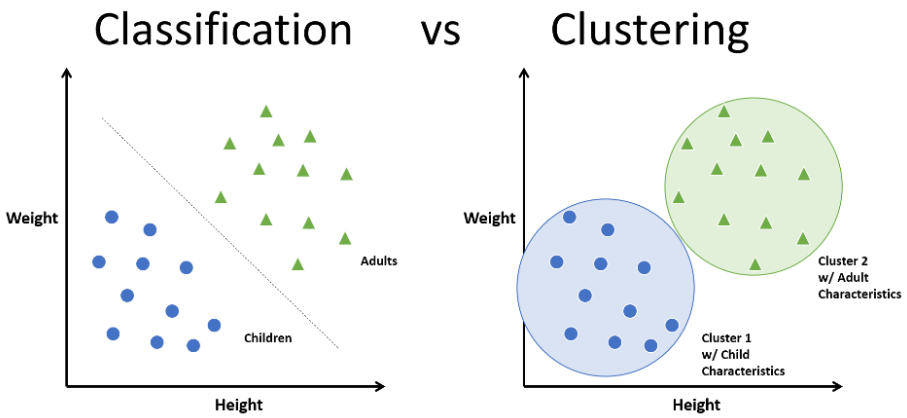


Figure 6 Différences entre clustering et classification

Comme les données ne sont pas libellées, il n'y a pas de phase d'apprentissage, le clustering est considéré comme moins complexe et beaucoup plus adapté à un grand nombre de données.

Dans le clustering nous utilisons la similarité et la différence afin de créer un ensemble commun à tous les membres de l'ensemble. Le nombre de cluster est inconnu tandis que le nombre de classes est connu.

Machine learning d'apprentissage par renforcement

Cette méthode d'apprentissage va utiliser un principe de récompense afin de faire apprendre quelque chose à la machine. Par exemple, je vais envoyer une réponse positive à chaque fois que mon algorithme trouve quelque chose de correct et une réponse négative dans le cas contraire.

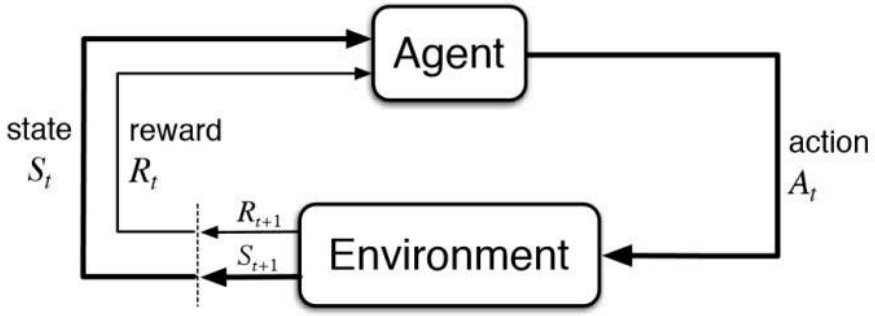


Figure 7 Fonctionnement de machine learning d'apprentissage par renforcement, kdnuggets[15]

À la réception de ces informations, le système sera capable de modifier et d'ajuster sa capacité à effectuer une tâche souhaitée.

Il s'agit donc d'un modèle qui va apprendre par l'erreur et qui va s'affiner au fur et à mesure du temps.

Ce système d'apprentissage a besoin d'interagir dans un environnement qui va lui permettre d'apprendre, s'il devait apprendre à jouer aux échecs, il deviendrait meilleur au fur et à mesure du temps en apprenant de ses erreurs et de ses bons coups joués. Nous aurons tendance à utiliser cette méthode dans un environnement où nous interagissons beaucoup avec un utilisateur. Cette méthode ne sera pas du tout utilisée dans ce projet.

[15] https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

Implémentations

Plusieurs solutions existent afin de faciliter l'implémentation de ces méthodes et une grosse communauté de machine learning existe sur internet, les outils revenants presque systématiquement sont :

- Scipy [16]: cette librairie est destinée à fournir un écosystème libre permettant d'effectuer des mathématiques, de la science et de l'ingénierie ;
- Pandas [17]: utilisé afin de structurer des données et les analyser ;
- Matplotlib [18]: permet de créer des graphiques ;
- Numpy [19]: facilite les calculs et opérations sur des tableaux ;
- Keras [20]: librairie destinée à faire du deep learning ;
- Sickity-learn [21]: permet d'effectuer en toute simplicité du machine learning ;
- TensorFlow [22]: plate-forme de machine learning.

Toute ces librairies d'apprentissage sont très bien documentées, à noter que Keras utilise TensorFlow afin d'implémenter son système de neurone. Ces différentes libraires seront utilisées dans ce projet.

Figure 8 Représentation des outils permettant de faire du machine learning en Python,towardsdatascience.com[23]

À noter que Google a mis en place un système de *notebook* en ligne nommé Colaboratory[24], un outil permettant de facilement développer du code Python en y important une bonne quantité de librairies dont celle citées-précédemment. Il est également possible d'installer d'autres librairies qui ne seraient pas déjà présente. L'outil permet également d'avoir de la puissance de calcul en utilisant des GPU. De plus, le partage de fichier est possible car ces documents sont stockés sur votre compte Google Drive.

La documentation est claire et l'outil est très facile à prendre en main et permet à des personnes ayant une machine peu performante pour l'analyse de données, d'augmenter sa puissance de calcul et de traitement.

---

[16] https://www.scipy.org/

[17] https://pandas.pydata.org/pandas-docs/stable/index.html

[18] https://matplotlib.org/

[19] https://numpy.org/

[20] https://keras.io/

[21] https://scikit-learn.org/stable/

[22] https://www.tensorflow.org/

[23] https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c

[24] https://colab.research.google.com/notebooks/welcome.ipynb?hl=fr

2.1.3. Forensique digital

Il faut d'abord parler de la science de forensique avant de parler de la forensique digitale. Cette branche scientifique est utilisée dans énormément de disciplines telles que l'archéologie, la botanique, la chimie, etc. Dans notre perspective, la forensique digitale est le domaine qui nous intéresse et peut être subdivisée en plusieurs catégories : la *computer forensique*, *l'IOT forensique*, etc.

Notre choix se tournera vers le forensique mobile, plus précisément, celle qui concerne les appareils tournant sous le système d'exploitation Android. Ce type de terminal peut être analysé de différentes façons : par exemple, on peut parler d'accès direct au téléphone, d'extraction physique de données ou encore d'extraction logique de données.

Dans le cadre de ce projet, la détection de malware dans une application sera ce qui nous intéressera ce qui correspond à l'extraction logique de données.

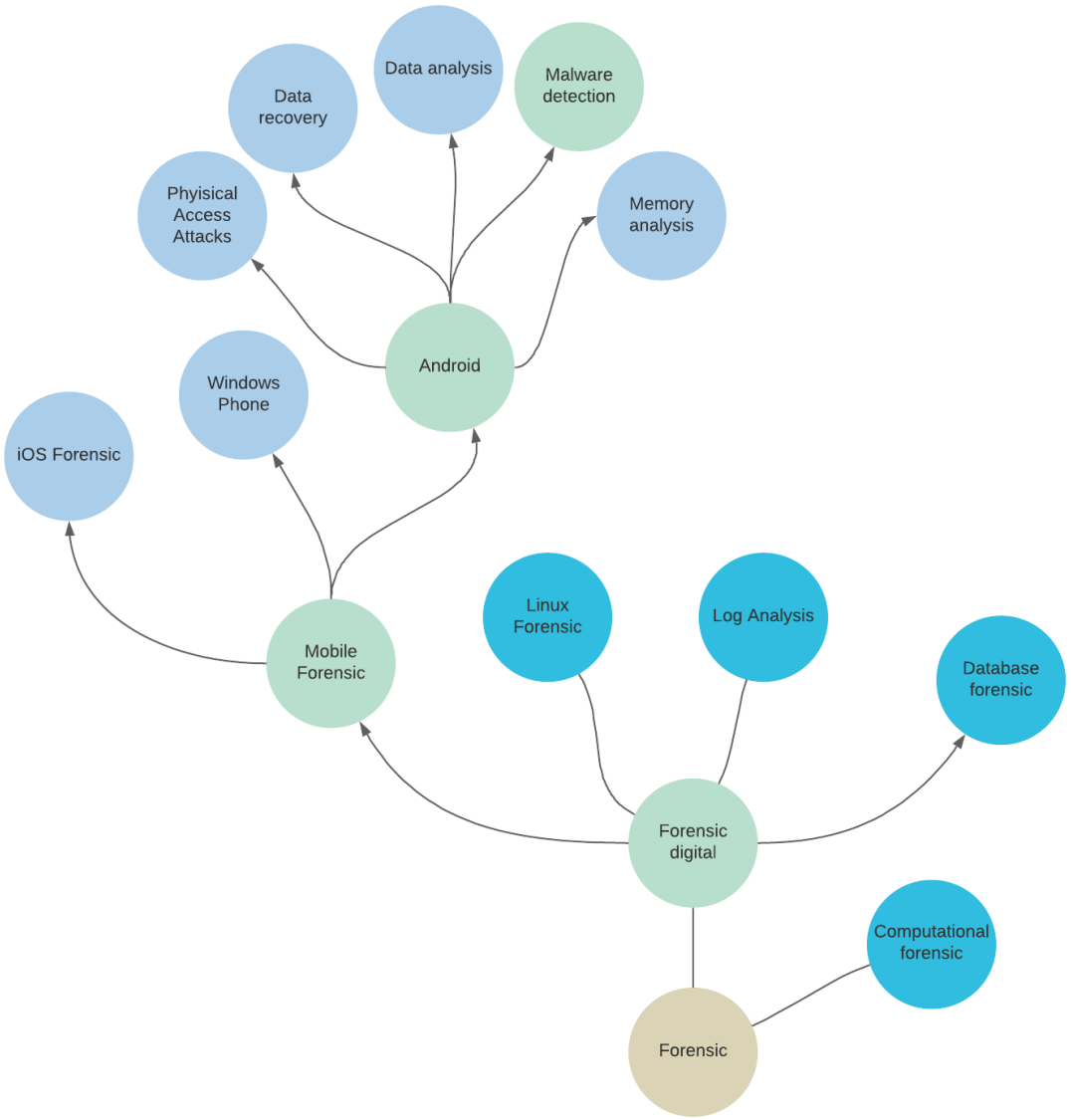


Figure 9 Vue synoptique du forensique

2.1.4. Outils de forensique digital commerciales

Vous trouverez les tableaux comparatifs dans les annexes de ce document.
Les produits ayant été retenu sont les suivants :

- Cellebrite Touch UFED[25]
- Encase Mobile Investigator[26]
- Oxygen Forensiques Detective Features[27]
- MOBILedit Forensique Express[28]
- PC-3000 Mobile[29]
- Paraben corporation E3 :DS[30]

Les applications ont été comparé en me posant les questions suivantes
« Implémente-t-il du machine learning ? »
« Combien coûte leur produit ?»
« De quoi est-ce que j'ai besoin afin d'utiliser cet outil ? »
« Quel(s) système(s) d'exploitation sont compatibles ? »
« Propose-t-il des outils spécifiques ? (Outil physique ou software »

Voici ci-dessous quelques observations sur ces outils ainsi qu'une synthèse générale.

Cellebrite Touch UFED

La détection de malware de Cellebrite est faite avec un système de signature, ce qui fait que l'entreprise doit tenir elle-même à jour ses listes de malwares.

Le matériel dédié est très intéressant pour une personne souhaitant aller en dehors de son laboratoire afin de récupérer des données et les traiter une fois qu'il retournera dans son laboratoire, ce qui est possible avec Cellebrite Touch UFED.

Figure 10 Boîtier UFED Touch2 de Cellebrite

[25] https://www.cellebrite.com/en/ufed/

[26] https://security.opentext.com/encase-mobile-investigator

[27] https://www.teeltech.com/mobile-device-forensic-tools/oxygen-forensic-detective/

[28] https://www.mobiledit.com/online-store/forensic-express

[29] https://www.acelaboratory.com/pc-3000-mobile.php

[30] https://paraben.com/paraben-for-mobile-forensics/

PC3000 Mobile

Cet outil est le seul qui s'occupe de principalement récupérer des données. Les autres outils sont des couteaux suisses, la plupart font de *l'imaging*, protègent l'intégrité des preuves, bypass de sécurité, ou permettent de récupérer des fichiers supprimés.

Constat général

En parcourant Internet, nous constatons que très peu de forum d'aide sur ces outils et aucun document ne procurant de reverse engineering du code n'ont été trouvé pour ces outils, nous dépendons donc entièrement de l'implémentation et des mises à jour des constructeurs.

Le problème principal de ces solutions réside dans le temps de support qu'ils garantissent et promettent car les mises à jour sont limitées de quelques mois pour MOBILedit Forensique Express à 2 ans pour PC-3000 Mobile. Cela veut dire qu'une fois passé ces délais, nous nous retrouvons seul face à nous-mêmes et notre solution se retrouvera, peut-être, rapidement désuète face à l'apparition de nouveaux modèles de smartphones et il sera nécessaire de repayer une licence.

Les outils proposent tous des synthèses d'analyse, ce qui facilite la compréhension des problèmes rencontrés des utilisateurs.

Certains outils sont plus compatibles que d'autres, je pense notamment à MOBILedit Forensique qui permet d'effectuer toutes ses fonctionnalités sur plusieurs types de téléphone mobile.

La plupart de ces outils fournissent également des prestations de familiarisation avec ces outils en proposant notamment des cours.

Une fonctionnalité qui revient souvent également est celle permettant de récupérer des fichiers supprimés, ce qui est très intéressant afin de pouvoir analyser des logs qu'un virus laisserait et tente de cacher en les supprimant.

Pour finir, nous observons clairement que le nom de machine learning est évoqué uniquement lorsque nous parlons d'analyse de photos. Le machine ou deep learning doit être utilisé dans quelques fonctionnalités, mais à ce jour, aucun des outils de forensique présenté utilise uniquement du machine learning, ce qui peut nous laisser croire qu'aucune application de la sorte n'existe.

Au cours de ce projet et afin d'avoir plus d'informations, des mails ont été envoyé aux différents constructeurs, seul un d'entre eux a répondu, il s'agit de Cellebrite. Mon mail spécifiait que je n'étais pas un acheteur potentiel et que j'effectuais une étude du marché actuel du forensique mobile, les questions posaient portait sur la présence de machine learning ainsi que le prix. Ce mail répond que le machine learning n'est pas implémenté dans leur produit ainsi que le prix ne peut être communiqué qu'aux potentiels acheteurs.

De part cette réponse de Cellebrite et de par l'absence de réponse des autres entreprises, la conclusion a été faites que le prix était communiqué uniquement aux acheteurs potentiels. Les prix indiqués dans le tableau sont des prix donnés dans des forums.

2.1.5. Outils de forensique digital libres

En comparaison au chapitre précédent, nous nous rendons compte que les solutions proposées gratuitement ne sont pas autant poussée que celle proposée par des spécialistes du forensique.

Nous constatons que la plupart des outils gratuits sont des petits outils spécifiques à une tâche, il n'existe pas vraiment d'outil « couteau-suisse » comme proposé par les logiciels payants.

L'avantage de ces outils est son système libre permettant de comprendre ce qui se passe derrière l'exécution. Les différents add-on permettent d'obtenir un outil adopté à notre besoin.

Il existe également de nombreux outils sur GitHub permettant de faire du forensique d'une façon ou d'une autre mais ces derniers n'ont pas été cité dans le tableau mais méritent d'être mentionné :

1. Android-forensiques[31] : permet d'extraire des données.
2. Android_Forensiques: bypassing Android Pattern Lock[32] : Permet de bypass l'écran verrouillé comme Oxygen ;
3. LiME [33]: permet de capturer toute la mémoire vive d'un système Android, au contraire de Volatility, ce dernier fonctionne en chargeant un driver kernel sur le système et de dump sa mémoire tandis que Volatility est capable d'interpréter la capture mémoire ;
4. LabCIF[34] : framework d'extraction et d'analyse avec des modules d'autopsy et permet de créer des rapports.

Conclusion

Nous avons pris conscience que notre monde est de plus en plus connecté et que les personnes le sont aussi de plus en plus avec leur smartphone et qu'il s'agit d'un point important à prendre en compte à l'avenir.

Nous savons aussi qu'il est difficile d'avoir un outil fonctionnant sur toutes les plateformes Android car celle-ci se diversifient de plus en plus.

Nous avons également pu avoir une vue sur les outils de forensique existants, notamment le manque d'information concernant l'utilisation ou non de machine learning. La rare mention qui en a été faite l'a été pour détecter des preuves incriminantes dans des photos.

Pour conclure ce chapitre, il ne semble pas exister une solution de forensique utilisant uniquement du machine learning. Afin d'approfondir ce sujet, j'ai décidé de me concentrer sur la détection de malware à l'aide de machine learning.

[31] https://GitHub.com/nowsecure/android-forensiques

[32] https://GitHub.com/c0d3sh3lf/Android_Forensiques

[33] https://GitHub.com/504ensicsLabs/LiME

[34] https://GitHub.com/labcif/FAMA

3. Détection de malware

Nous discuterons dans ce chapitre de deux études ayant été faites sur la détection de malware sur des téléphones Android. Nous parlerons ensuite des jeux de données disponible et en libre accès fourni par des chercheurs.

Nous verrons dans ce chapitre les différents critères importants pour avoir une bonne détection de modèle, les chercheurs ont cherché à avoir :

- La meilleure précision possible
- Une résistance au temps qui passe
- Une extraction de données efficaces et viables afin de produire un bon jeu de données
- L'efficacité en termes de temps d'exécution
- Détecter le type de malware

3.1. Drebin

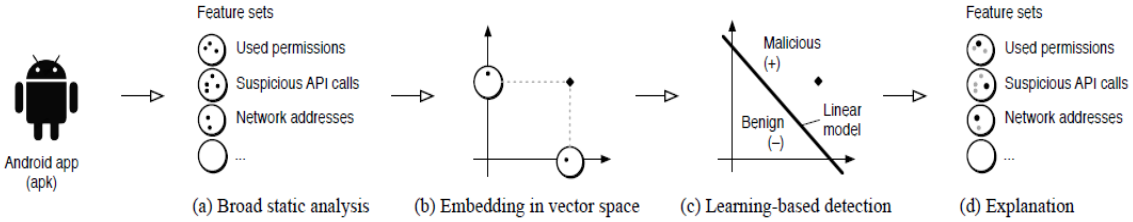


Figure 11 Fonctionnement de Drebin

Ce chapitre décrit l'étude [3] qui a analysé 123453 applications de plusieurs fournisseurs d'applications et 5560 malwares. L'objectif de cette étude était de pouvoir créer un outil permettant de détecter de nouveaux malwares en direct sur le téléphone et de pouvoir détecter de quelle famille de malware il est issu. Les analyses effectuées sont des analyses statiques car celle-ci ne sont pas spécialement exigeantes en temps de calcul.

Une analyse statique veut dire que le code n'est jamais exécuté, nous allons uniquement regarder les fichiers qui composent son APK, les librairies utilisées, les appels API. Cette analyse est divisée en 8 extractions d'informations.

Feature set	Explanation
S_1 Hardware features	App uses %s feature %s.
S_2 Requested permissions	App requests permission to access %s.
S_3 App components	App contains suspicious component %s.
S_4 Filtered intents	Action is triggered by %s.
S_5 Restricted API calls	App calls function %s to access %s.
S_6 Used permissions	App uses permissions %s to access %s.
S_7 Suspicious API calls	App uses suspicious API call %s.
S_8 Network addresses	Communication with host %s.

Figure 12 Représentation des 8 scores calculés

Les 4 premiers ensembles de fonctionnalités sont issu de l'analyse du manifest :

S1 va vérifier à quel hardware va avoir accès l'application et va surtout vérifier la combinaison des accès, il serait étrange pour un GPS d'avoir besoin d'un accès aux messages.

S2 vérifie les permissions des applications, les malwares ont tendance à demander plus de permissions que les applications bénignes.

S3 regardera les appels fais aux *components* des applications, certaine famille de malware partagent le nom de services particuliers.

S4 nous observons les *intent* (moyen de communication intra-processus et inter-processus), ces appels permettent d'envoyer des données entre processus.

Les 4 ensembles suivants analysent le code désassemblé :

S5 Nous regardons si l'application tente de faire des appels API cruciaux

S6 Nous comparons les droits que semblent avoir besoin le code et ceux effectifs

S7 Certains appels API sont suspects car fournissent des informations sensibles sur le téléphone ou les données

S8 Les connexions au réseau sont observées et une recherche d'occurrence avec des adresses IP connues est faites afin de détecter s'il s'agit d'un botnet par exemple.

Une fois ces analyses effectuées, une somme est faite pour chaque S et un vecteur est créé comme présenté ci-dessous

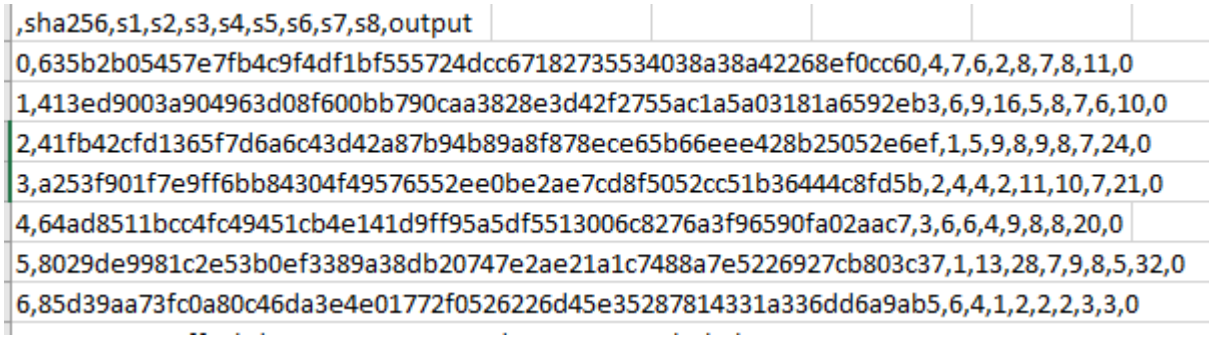


Figure 13 Exemple de vecteurs générés par Drebin

Les jeux de données de cette étude seront décrits au chapitre 3.2.

Les fonctionnalités sont extraites à l'aide de Androguard [35]et de Smali [36] qui sont des outils permettant d'extraire les fichiers d'un APK, de désassembler du code DEX/ODEX et décompiler des fichiers DEX/ODEX.

Une fois en possession de ces vecteurs, l'objectif est de placer dans un espace vectoriel les différents résultats et d'utiliser le modèle SVM (Support vector machine). Ce modèle va chercher à trouver la limite optimale entre deux classes, vous retrouverez plus d'info sur ces modèles dans le chapitre 4. Avec cette technique, ils obtiennent une détection correcte à 94% des malwares avec un taux de faux positif avoisinant les 1%.

Concernant la détection de famille de malware, il est dit qu'il est difficile pour Drebin de détecter une nouvelle famille de malware. Une expérience a été faite où l'on substituait une famille du jeu de données d'entraînement et les redonnions lors de la phase de test afin de simuler une nouvelle famille de virus. La même expérience a été faites, à la différence que, nous fournissions cette fois 10 exemples de malware d'une famille lors de la phase d'entraînement, ce qui permet de représenter l'émergence d'un nouveau malware. Les résultats montrent que Drebin est capable de détecter certaine famille de malware une fois qu'ils sont en possession d'un petit ensemble de données sur une nouvelle famille de malware.

[35] https://GitHub.com/androguard/androguard

[36] https://GitHub.com/JesusFreke/smali

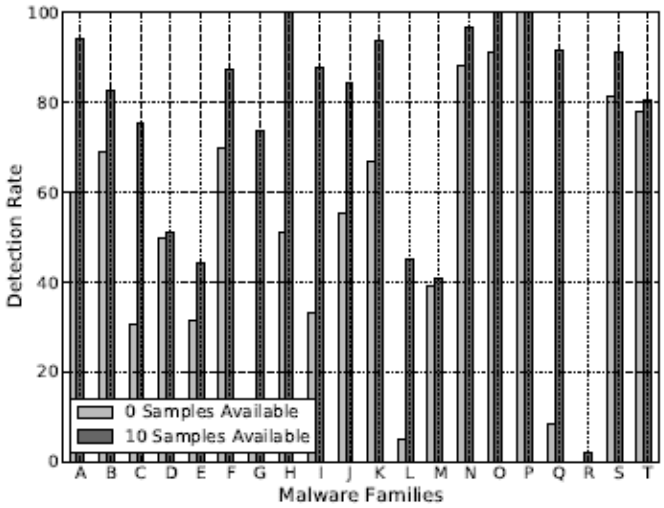


Figure 14 Expérience de détection de nouveaux malwares

Dans son implémentation initiale, les chercheurs ont bien compris qu'il y avait une distribution inégale entre le nombre de famille et ont proposé de réduire le nombre d'extrait afin que ces derniers soient égaux mais ont observé qu'effectuer ce *downsampling* n'améliorait pas de façon significative les résultats, pire, cela rendait les données moins proches de la réalité. Les résultats obtenus pour détecter une famille de malware connues est de 93% avec un taux de faux positif à 1%.

Id	Family	#	*Id*	Family	#
A	FakeInstaller	925	*K*	Adrd	91
B	DroidKungFu	667	*L*	DroidDream	81
C	Plankton	625	*M*	LinuxLotoor	70
D	Opfake	613	*N*	GoldDream	69
E	GingerMaster	339	*O*	MobileTx	69
F	BaseBridge	330	*P*	FakeRun	61
G	Iconosys	152	*Q*	SendPay	59
H	Kmin	147	*R*	Gappusin	58
I	FakeDoc	132	*S*	Imlog	43
J	Geinimi	92	*T*	SMSreg	41

Figure 15 Liste des familles de malware

Plusieurs implémentations[37] non-officielles existent sur GitHub et permette d'obtenir des résultats similaires à ceux obtenus par l'équipe de recherche. Cette implémentation a testé d'augmenter le nombre d'extrait de malware au nombre d'extrait d'application total et a obtenu un résultat de 99.37% d'*accuracy* sur l'ensemble de données en utilisant le classifieur *Random Forest*.

[37] https://GitHub.com/mokeam/malware-analysis/blob/master/scripts/Malware_Classification_Model.ipynb

3.2. MaMaDroid

L'outil MaMaDroid [9] est une méthode de détection de malware basée sur l'extraction de *Call Graph* et crée une séquence basée sur ce graphique.

Le but de cette méthode est de pouvoir garder un modèle fiable pendant une longue période. Leur résultat démontre qu'en utilisant un jeu de données de 8500 applications bénignes et 35500 applications malicieuses récupérée sur 6 années, leur f1-score est de 98%, ce qui est un très bon résultat, encore plus intéressant, la résistance au changement des malwares est bonne car une ou deux années après l'entraînement du modèle, *l'average* approxime les 86% en moyenne.

Le résultat est certes moins bon mais reste acceptable et permet de prendre le temps de déployer un nouveau modèle pour les malwares actuels.

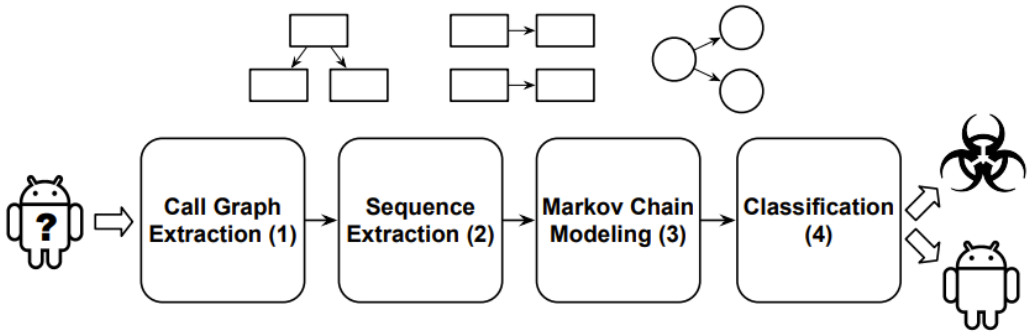


Figure 16 Schéma de representation de MaMaDroid

Un code retravaillé sur la base du code original est disponible sur GitHub [38]. Un exemple d'appel API serait :

java.lang.Throwable : String getMessage()

Lors de l'analyse de l'application, les différents points d'entrée de l'application seront énumérés et chaque nœud représentera différents chemins que peut prendre le nœud au niveau du fil de l'exécution. Une fois ce graphique obtenu, une extraction en est faîtes et les appels sont triés en fonction des packages existants, les packages inconnus, comme ceux implémentés par l'utilisateur ne sont pas pris en compte dans cette extraction.

Une chaîne de Markov [39]sera ensuite modélisée, ces chaînes basent leur prédiction future en se basant uniquement sur l'état présent du processus et ne dépend pas de ses états antérieurs. Chaque *package* est un état et une transition est représentée par la probabilité de passer d'un état à un autre. Procéder de cette façon permet de faire abstraction d'obfuscation par ajout d'appel API inutile et l'ajout de cette obfuscation ne change pas les probabilités de transition de façon significative.

Finalement, le vecteur caractéristique pour une application donnée est représenté par les probabilités de transition d'un état à un autre, les états qui ne sont pas présent dans une application sont représentées par un 0.

Finalement différents modèles ont été entraîné, *Random Forests, Nearest Neighbor* et *Support Vector Machines.* Celui produisant la plus grande satisfaction en termes de temps d'exécution et de résultat sont Random Forest et Nearest Neighbor, SVM n'est plus mentionné dans le rapport au vu de ses résultats.

[38] https://GitHub.com/IanWE/mamadroid

[39] https://www.imo.universite-paris-saclay.fr/~meliot/agreg/markov.pdf

3.3. MalDozer

Le *deep learning* ne fait pas partie du scope de ce projet, mais ce projet reste notable et est brièvement mentionné afin d'offrir une piste sur de futur travaux en lien avec le *deep learning*,

ElMouatez Billah Karbab a présenté son travail MalDozer [8] au DFRWS EU 2018. Lors de cette présentation [40]il va comparer son travail à celui de Drebin et de MaMaDroid .et présenter les avantages et inconvénients de chacun et valoriser sa solution.

Il présentera brièvement le fonctionnement des deux méthodes citées précédemment et conclura que :

- Drebin est capable de classifier un malware selon une famille, l'analyse est rapide (environs 10 secondes en moyenne) et l'application peut être déployé sur plusieurs système Android. Cependant, ses résultats de détection ne sont pas assez bons en comparaison à d'autres méthodes, l'extraction de vecteur caractéristique sont décrits à la main ce qui ne rend pas l'extraction très évolutif et adaptative.

- MaMaDroid produit lui un très bon résultat dans la classification de malware et d'application bénigne, faisant abstraction de plusieurs variables et se basant sur des appels API qui ne changent pas énormément au fil du temps, le modèle crée résiste au temps. Cet outil a également ses défauts, il ne permet pas de classifier un malware selon une famille, l'analyse d'une application prend du temps, environs 20 minutes au maximum, ce qui n'est pas adapté à de l'implémentation dans de l'IOT.

Le but de cet outil est de combiner les avantages des deux méthodes citées précédemment et d'obtenir un produit permettant de répondre aux besoins de détection de malware.

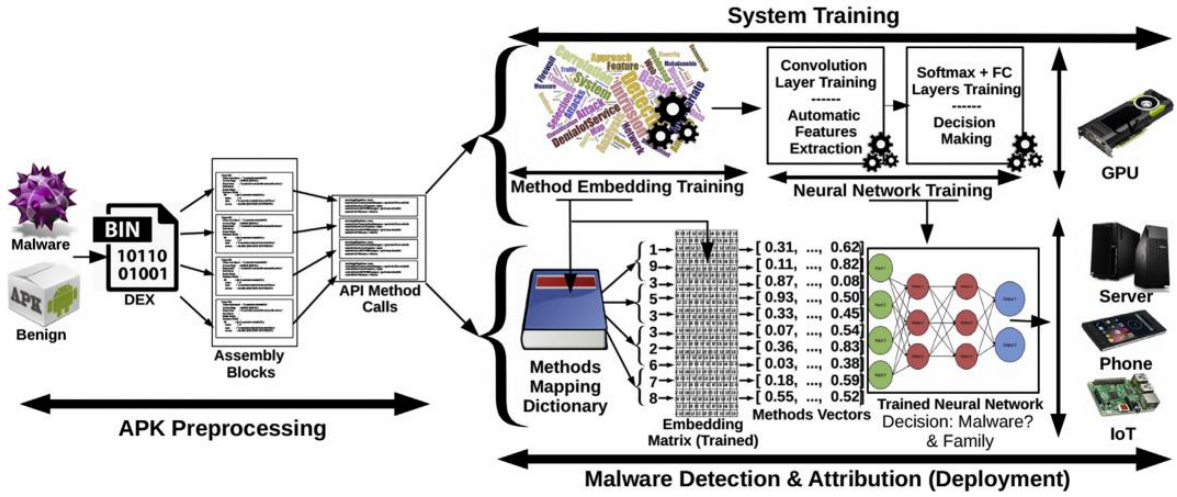


Figure 17 Approche de MalDozer

Le principe est de décompiler une APK et effectuer une recherche des tous les *API Method Callls* présent dans les fichiers .dex.

Ces dernières sont utilisées comme vecteur caractéristique d'une application, plusieurs hypothèses ont été levées quant à la façon de créer le vecteur, mais celle ayant été retenue est de traiter une application Android comme une séquence d'appel API sans filtrage et dont l'ordre d'apparition est important afin de caractériser des malwares. La partie d'entraînement se fait sur des GPU afin d'avoir une plus grande puissance de calcul et de générer plus rapidement un modèle.

[40]https://www.forensicfocus.com/webinars/maldozer-automatic-framework-for-android-malware-chasing-using-deep-learning/

Ce vecteur est passé à un modèle de réseaux de neurone relativement simple qui permet de produire des résultats qui surpassent la plupart des méthodes présentées jusqu'à présent. Ils ont utilisé un réseau de neurone similaire à Kim qui est un outil de deep learning permettant de traiter du texte.

Les résultats montrent une précision allant en moyenne jusqu'à 98% de classification correct. Pour beaucoup de famille de malware, la précision monte au-delà de 99%, avoisinant parfois 99.99%.

Que ça soit en utilisant un petit ensemble de jeux de données ou un grand, les résultats produits par MalDozer sont satisfaisants. De plus, la détection de famille de malware est très bonne. Le modèle peut également être déployé sur plusieurs architectures allant de serveur à une *raspberry p*i.

3.4. Détection basée sur les permissions

Pour donner suite à la lecture de l'étude de Drebin je me suis posé la question : « Est-ce que les permissions permettent de détecter de façon efficace un malware et une application bénigne ? ». Je me suis donc intéressé aux études de [4],[5],[6] et [10].

Dans toutes ces études le procédé a été le suivant

- Décompiler l'APK ;
- Analyser le AndroidManifest.xml ;
- Extraire les permissions selon une liste prédéfinies ;
- Créer un vecteur caractéristique en se basant sur les permissions présentes sur l'application.

Dans [10] le jeu de données est composé uniquement de 200 applications, ce qui est peu. Pour [4], [5] et [6] une implémentation sur kaggle [41] est présentée, les données utilisées est lui aussi petit, environs 200 applications aussi. Les modèle k-neighbors, naives bayes produisent une précision proche de 89-90% tandis que decision tree produit un résultat de 94%.

[41] https://www.kaggle.com/xwolf12/android-malware-analysis

3.5. Jeu de données

Ce chapitre va décrire les jeux de données trouvables sur internet. Plusieurs jeux de données utilisés lors des papiers cités précédemment ne sont plus disponible. Vous retrouverez plus d'informations dans les chapitres ci-dessous.

3.5.1. Genome[42]

[11] est un jeu de données développé par l'université de Caroline du Nord. Cette base de données a été citée dans énormément d'étude et a été utilisée à plusieurs reprises, par exemple pour Drebin. Malheureusement, depuis le 21 décembre 2015, dû à des ressources limitées, il a été décidé de ne plus partager ce projet.

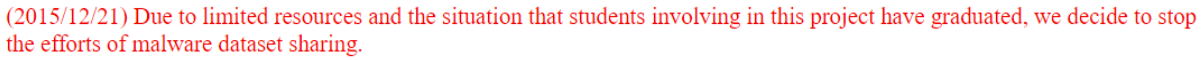


Figure 18 Capture d'écran de la page internet de Genome indiquant la fin du partage de ressource

Il existe toutefois des archives qui se retrouvent dans Androzoo présenté plus loin.

3.5.2. Drebin

L'étude Drebin a mis à disposition son jeu de données est composé de 5560 malwares issus de 179 familles de différents malwares et d'extrait de 123453 applications bénignes.

Ce jeu de données est accessible pour les universités ou pour des groupes industriels. Il est nécessaire de se présenter auprès d'eux par mail et de décliner son identité et d'amener quelques preuves de sa bonne foi, dans mon cas, je me suis présenté en temps qu'étudiant effectuant un projet au nom de l'HES-SO et mon mail a été envoyé depuis mon adresse HES.

En cas de réponse positive, nous recevons par mail un identifiant permettant de télécharger les jeux de données.

Drebin a décomposé son jeu de données ainsi :

- 6 zip contenant chacun 1000 malwares, les malwares sont identifiés à l'aide de leur hash (SHA256).
- Un fichier csv permettant de relier une famille de malware à un SHA256 est disponible.
- Un csv contenant l'analyse faites des 123453 applications bénignes.

En parcourant ses fichiers, il est possible de créer un vecteur caractéristique permettant d'identifier un malware ou une famille de malware.

3.5.3. M0Droid Dataset

[12] a mis à disposition son jeu de données mais le lien n'est plus accessible.

[42] http://www.malgenomeproject.org/

3.5.4. Kharon Malware Dataset

Le travail de [13] est un peu différent des autres jeux de données présentés, ce dernier est composé de 7 malwares. Le but est ici d'effectuer une analyse approfondie et poussée permettant de comprendre le fonctionnement de certains malwares. Nous retrouvons sur leur site [43]internet la liste de ces applications ainsi que les résultats du *reverse* effectué sur ces dernières.

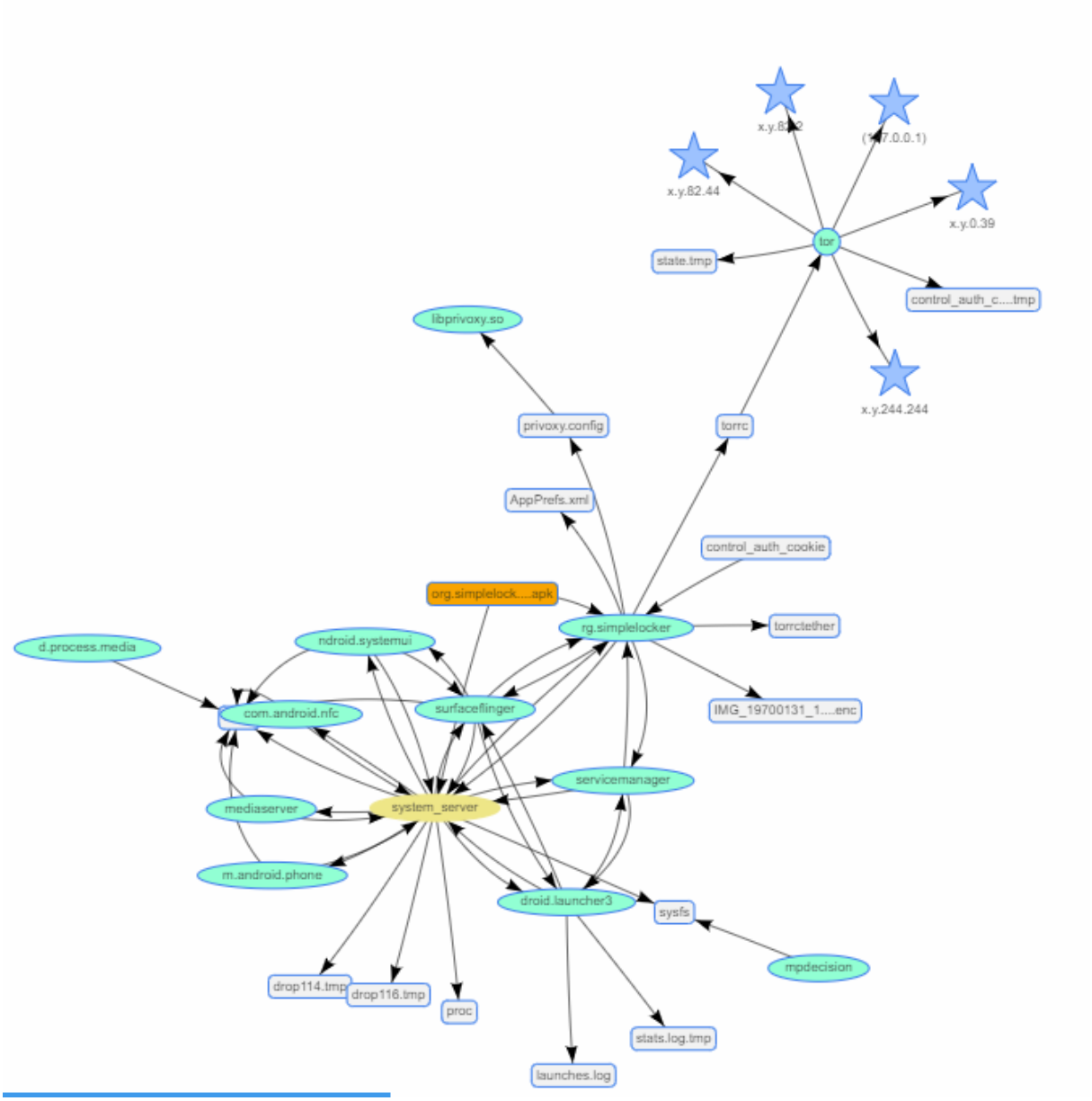


Figure 19 Graphique de SimpLocker analysé par Kharon Project

[43] http://kharon.gforge.inria.fr/dataset/

3.5.5. Androzoo

L'outil [14] est une API [44]permettant de télécharger près de 15'463'679 APK. Cette ressource a été mise à disposition par l'Université du Luxembourg. De la même façon que pour Drebin, il est nécessaire de s'identifier auprès des chercheurs pour qu'ils puissent nous fournir une clé API. Cette dernière est privée et ne devrait en aucun être distribuée sur internet.

Le but d'Androzoo est de fournir des ressources aux chercheurs afin que ces derniers puissent faire des expériences reproductibles pour tout le monde et ainsi faire avancer nos connaissances.

Une fois la clé API obtenue, il nous faudra télécharger la liste des applications utilisant le hash des applications pour les reconnaître. Ce fichier fait 1.4GB.

Le fichier contient les champs suivants :

- *SHA256* : permettant d'identifier l'APK.
- *Dex_size* : Indique la taille des classes.dex.
- *Dex_date* : N'est presque plus utilisable car la majorité des applications venant de Google Play ont 1980.
- *Pkg_name* : Permet de spécifier un package Android spécifique.
- *Vt_detection* : Indique le nombre d'antivirus ayant détecté l'application comme étant un virus (si disponible).
- *Markets* : Permet de spécifier la plateforme sur laquelle l'application a été téléchargée.

Lors de la sortie de l'étude, des informations ont été fournies par rapport aux applications déjà présente. En voici quelques-unes tirées de leur papier cité précédemment.

Marketplace	# of Android apps	Percentage
Google Play	1 899 883	59.70%
Anzhi	605 646	19.03%
AppChina	577 662	18.15%
1mobile	57 525	1.81%
AnGeeks	55 804	1.75%
Slideme	52 145	1.64%
torrents	5 294	0.17%
freewarelovers	4 145	0.13%
proandroid	3 683	0.12%
HiApk	2 491	0.08%
fdroid	2 023	0.06%
genome	1 247	0.04%
apk_bang	363	0.01%
Total	**3 182 590 Unique apps**	

Figure 20 Distribution des applications selon les plateformes de téléchargement

Nous pouvons constater que les 3 marchés les plus utilisés pour télécharger des applications sont Google Play, Anzhi et AppChina.

[44] https://androzoo.uni.lu/

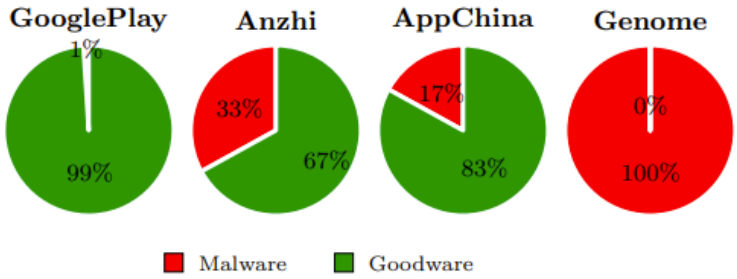


Figure 21 Pourcentage d'application désignée comme virus par 10 antivirus selon le marché utilisé

Nous pouvons constater que 1% des applications sur GooglePlay sont détectées comme virus par les 10 antivirus, cela démontre un grand soin prix par cette plateforme dans la distribution de ses APK.

Quant à Anzhi et AppChina sont des plateformes de partages d'APK orientées sur le marché chinois et la plupart de ses applications ne sont pas disponible sur GooglePlay pour différentes raisons, ici nous pourrons déduire que la plupart de ces applications ne sont pas présente sur GooglePlay car ces dernières sont des malwares.

Finalement, présenté précédemment dans ce document, Genome est bien composé à 100% de malwares.

Toutes les APK sont vouées à être analysées par les antivirus. Il se peut que des données manquent parfois dans le csv.

3.6. Conclusion

Nous pouvons conclure à la suite de la lecture de ces études que MalDozer est un très bon compromis entre tous ces critères et est une solution viable de détection de malware. Drebin et MaMaDroid possèdent certaines de ces caractéristiques mais MaMaDroid ne permet pas de détecter une famille de malware tandis que Drebin y arrive.

L'analyse statique permet d'obtenir des informations sur les malwares avec un haut pourcentage de précision et permet, dans le cas de Drebin d'effectuer des analyses et extraction de nouvelles applications installées en 10 secondes tandis que la solution de MaMaDroid prend 20 minutes d'analyse par application.

Nous avons également parlé de détection de malware Android basé sur l'analyse de permission, Les résultats observés lors des recherches m'impressionnent et me donnent envie d'implémenter une solution similaire en me basant sur le code existant de son créateur.

La lecture des études sur la détection basée sur les permissions me laisse porter à croire qu'il serait possible d'avoir de meilleurs résultats en utilisant un plus gros jeu de données. Une autre raison me donnant envie d'explorer cette solution provient du fait que les études citées datent de 2013 et de 2016. Ces données pourront être téléchargées à l'aide d'AndroZoo et différents jeux de données pourraient être crées.

4. Partie expérimentale

4.1. Solution proposée

Nous trouverons dans ce chapitre la description de l'implémentation des différents codes produits lors de ce projet d'approfondissement. Il y sera expliqué :

1. L'extraction des APK d'un téléphone Android ;
2. Le téléchargement et le traitement des APK afin d'entraîner un modèle ;
3. L'extraction des APK du téléphone ;
4. Les résultats obtenus sur les différents modèles.
5. Les problèmes rencontrés
6. Les améliorations à faire

La motivation de cette implémentation est décrite dans la conclusion du chapitre 3.6, je rajouterais que créer un code facilement *scalable* était un objectif à atteindre pour moi et plusieurs modèles sont implémentés afin que pour une autre entrée que celle que j'ai définie, nous puissions avoir un comparatif entre ces différents modèles.

Mon but est dans un premier temps de tester une analyse de permission d'APK Android et de voir si les résultats sont meilleurs que ceux obtenus dans les études mentionnées au chapitre 3.4. L'objectif secondaire est de rendre accessible à un novice ce genre d'analyse. Un soin sera apporté aux commentaires dans le code de l'application que vous pouvez retrouver en annexe et sûr GitHub : https://GitHub.com/panticne/Android_Forensic_ML/tree/main/drebin-easier

4.1.1. Environnement de développement

- Le projet a été réalisé sur une VM Ubuntu (https://www.osboxes.org/ubuntu/)
- Le code a été développé sur l'IDE PyCharm Community (https://www.jetbrains.com/fr-fr/pycharm/download/#section=windows)
- Un fichier requirements.txt a été généré afin de faciliter l'installation des librairies
- Version de python : 2.7.17
- Version de pip : 9.0.1

4.1.2. Fonctionnement général

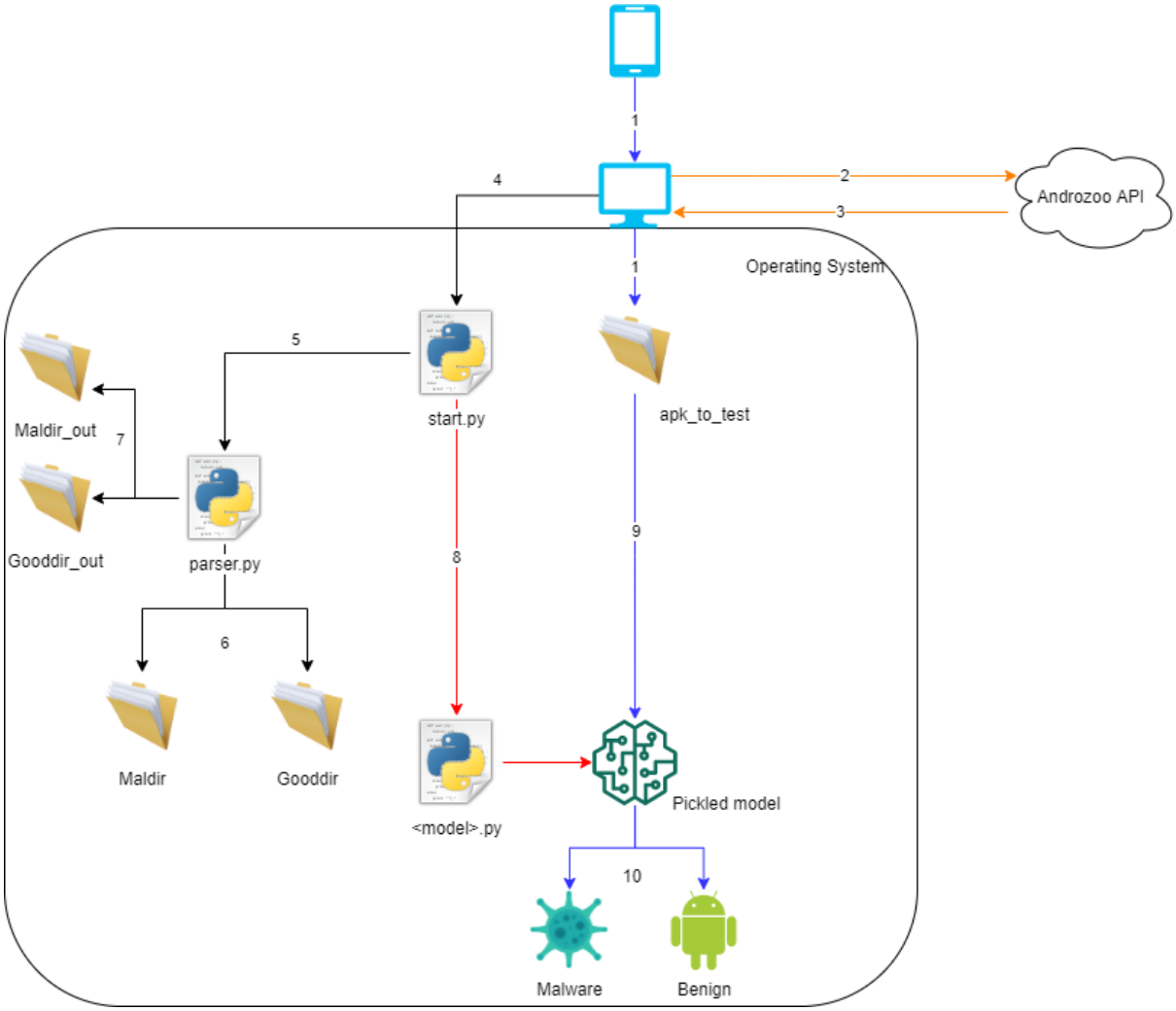


Figure 22 Schéma représentant le fonctionnement de mon projet

La solution imaginée est la suivante et peut se diviser en cinq axes

1) L'extraction des APK du téléphone
2) Le téléchargement des APK
3) Le *parsing* et annotation des APK
4) L'entraînement d'un ou de plusieurs modèles
5) *Parser* les applications extraites du téléphone

Lors de ce projet, seul les parties 1,2,3 et 4 ont été implémentée. Des propositions seront faites pour la 5ème partie afin de pouvoir facilement reprendre ce projet. Des explications seront données sur les choix pris lors de l'implémentation.

Les différents projets dont je me suis inspiré afin de créer ce code sont directement commenté dans les fichiers python crée lors du projet.

4.1.3. Extraction des APK

Ce chapitre décrit le point 1 de la figure 22.

L'extraction peut se faire à l'aide de plusieurs outils, celui étant le plus intéressant est adb

ADB

Aussi appelé *Android Debug Bridge* est un outil permettant de lancer un serveur permettant de communiquer avec une cible Android. Ce lien [45]vous permettra d'installer ADB peu importe sur quel OS vous comptez l'utiliser

Plusieurs commandes existent et permettent d'installer ou de télécharger des dossiers.

La connexion peut s'établir par connexion USB ou par connexion au même réseau Wi-Fi

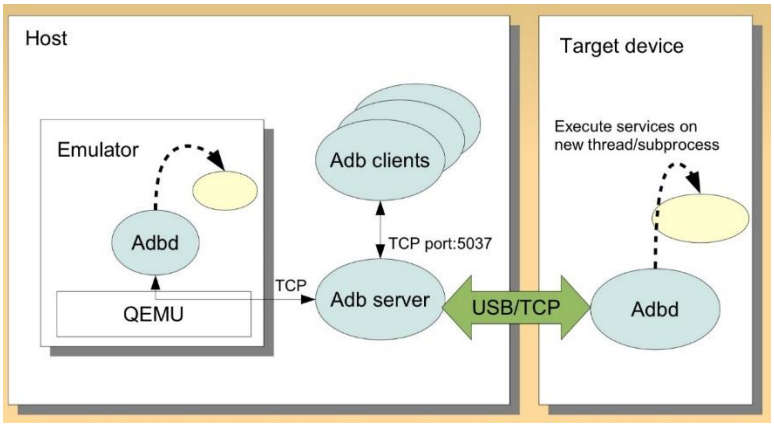


Figure 23 Fonctionnement d'ADB, adb-broker[46]

Notre cible étant les APK, ces dernières sont stockées dans le dossier /data/data/ mais ce dossier n'est pas lisible par l'agent ADB installé sur le téléphone. Il y a donc plusieurs solutions pour récupérer ces données, la solution retenue est la suivante :

Rooter le téléphone afin de pouvoir y installer un kernel personnalisé et d'installer un nouvel agent ADB sur le téléphone permettant de se connecter à l'aide de *adb root*. J'ai utilisé cette méthode lors de mon implémentation afin de récupérer ces APK. Le téléphone utilisé est un Nokia 8 et utilisant Android 11.

Afin de pouvoir modifier le bootloader de Nokia et de le rooter à l'aide de Magisk [47] il est nécessaire d'obtenir une clé pour déverrouiller ce *bootloader*. Afin de l'obtenir il suffit de se rendre sur cette page [48]web et de saisir une adresse électronique ainsi que le numéro IMEI qui signifie *International Mobile Equipment Identity* et sert à identifier de manière unique chaque terminal de téléphonie. Une fois ces données saisies et validées, vous recevrez par mail « unlock.key » vous permettant ainsi de rooter votre téléphone. J'ai utilisé ce tutoriel [49]afin de m'aider.

Toutefois j'ai été confronté à un problème que je décris dans la rubrique problèmes rencontrés.

Si vous ne souhaitez pas rooter votre téléphone, des applications permettant de récupérer les APK des applications installées sur votre téléphone sont disponibles sur le store et permettent de rapidement récupérer ces APK sur notre ordinateur. Il est également possible d'extraire les APK sans rooter le téléphone mais cette méthode [50]n'a pas été utilisée dans le projet mais mérite d'être mentionnée

[45] https://www.xda-developers.com/install-adb-windows-macos-linux/

[46] https://GitHub.com/lxs137/adb-broker

[47] https://GitHub.com/topjohnwu/Magisk/releases/tag/v23.0

[48] https://www.nokia.com/phones/en_int/bootloader

[49] https://www.androidauthority.com/unlock-nokia-8-bootloader-908160/

[50] https://gist.GitHub.com/ctrl-freak/24ac0e61b7cf550a6945

AirDroid

Cet outil [51]permet de gérer notre téléphone à travers une interface web. Pour que cette solution fonctionne, il faudra que votre téléphone et votre ordinateur soient sur le même réseau. Un serveur web est lancé sur l'adresse : 192.168.1.40 :8888 et vous permettra d'accéder aux donnes de votre téléphone sûr votre PC.

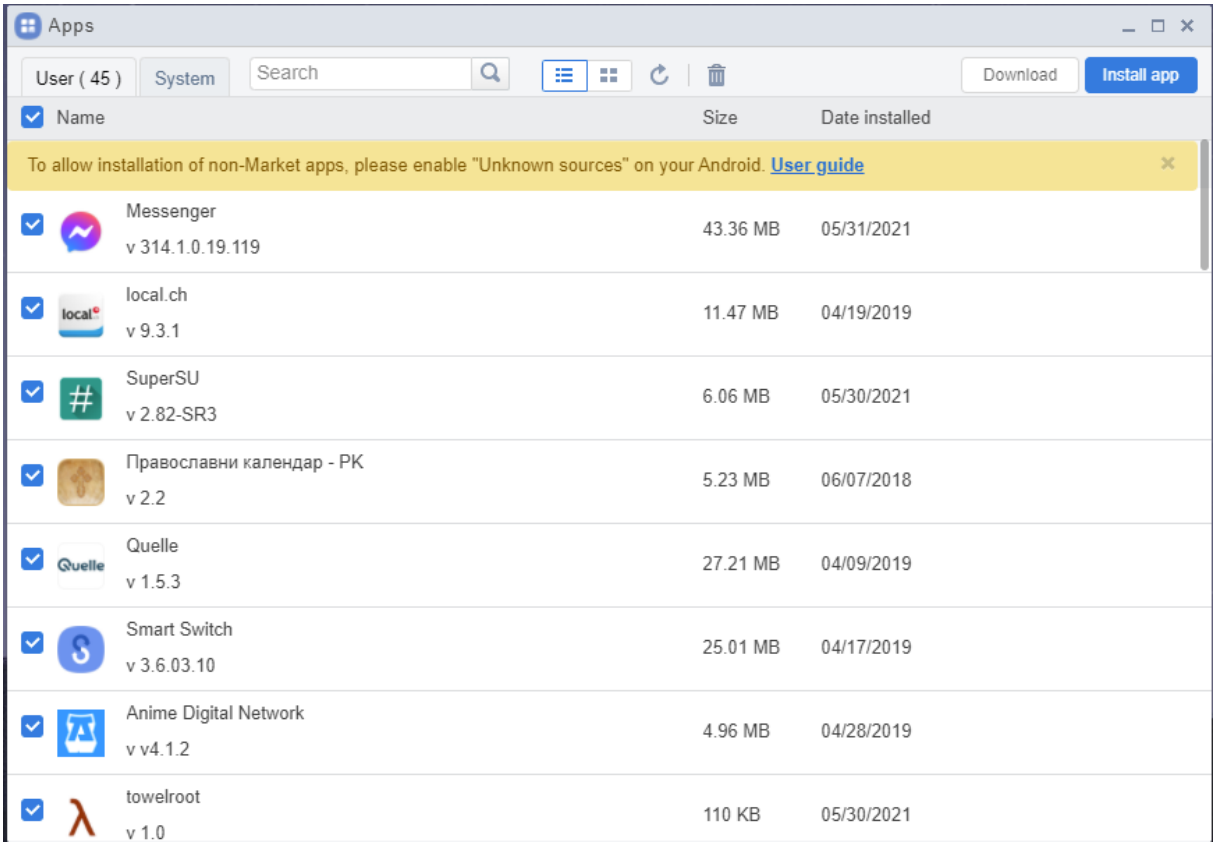
- 

Figure 24 Interface graphique de AirDroid affichant les applications présente

Il est possible de toutes les sélectionner et d'appuyer sur *Download*. Le téléchargement se lancera sur votre navigateur.

Apk Extractor

Plusieurs applications ayant la même fonctionnalité existent, ici je vais citer [52]celle que j'ai utilisé afin d'extraire mes APK.

Une fois installée, une liste des applications sur votre téléphone va s'ouvrir et vous permettre de choisir quelle APK extraire. Une fois que vous avez sélectionné l'application que vous souhaitez exporter, cette dernière se retrouvera sur votre carte SD et vous sera accessible. Vous pourrez finalement les récupérer en branchant votre téléphone en USB à votre ordinateur et en récupérant les APK présents.

[51] https://play.google.com/store/apps/details?id=com.sand.airdroid&hl=fr_CH&gl=US

[52] https://play.google.com/store/apps/details?id=com.iraavanan.apkextractor&hl=en

4.1.4. Téléchargement des APK

Ce chapitre reprend les points 1 et 2 de la figure 22

Comme décrit dans le chapitre 3, la bibliothèque d'APK utilisé afin d'entraîner notre modèle sera Androzoo.

En parcourant leur site, nous pouvons y trouver un lien vers GitHub [53]d'un outil permettant de simplifier le téléchargement des APK. Détail important, lorsque vous serez amené à devoir créer le fichier «~/.az », il vous faudra fournir votre clé API ainsi que le chemin vers le fichier contenant tous les SHA256 des applications.

Nous pouvons y spécifier les mêmes arguments que ceux décrit par Androzoo, il est cependant possible de multithreader ce processus de téléchargement et d'y injecter une seed afin de télécharger de façon aléatoire une APK répondant à nos critères.

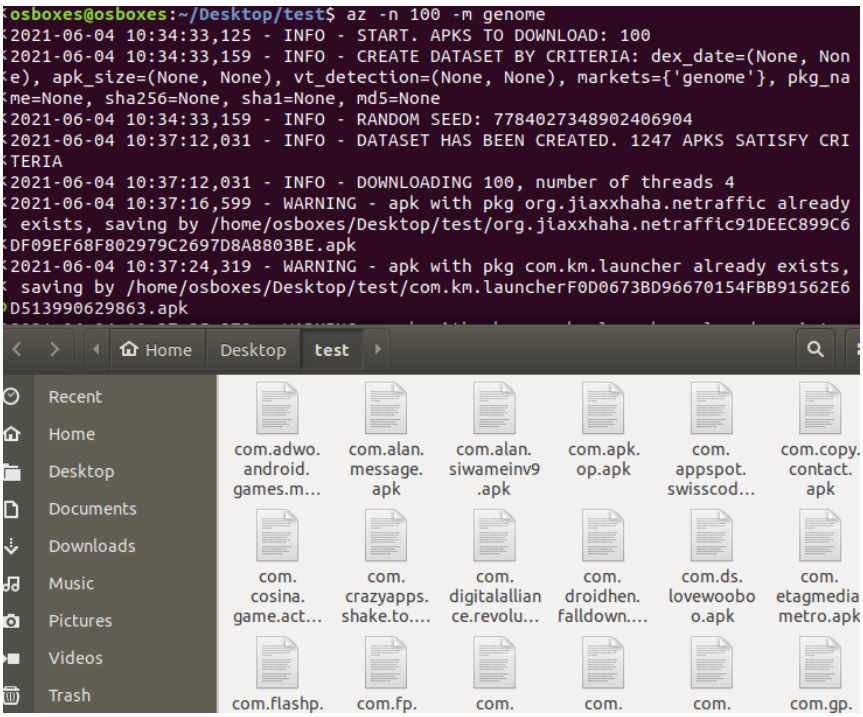


Figure 25 Exemple de commande permettant de récupérer 100 applications présentes sur le marché Genome

Notons que par défaut, l'application va utiliser le nombre de cœurs disponibles afin de télécharger au plus vite ces applications.

Afin de récupérer des APK malicieuses, j'ai utilisé la commande

az -n 1000 -vt 10:

Elle me permet de télécharger 1000 applications dont le score -vt est 10 ou plus (10 étant le maximum)

Afin de récupérer les APK bénignes, j'ai utilisé la commande

az -n 1000 -vt :0

Elle me permet de télécharger 1000 applications dont le score -vt est 0 ou 0

L'intérêt d'utiliser cet outil est de pouvoir diversifier nos jeux de données, nous pourrions par exemple spécifier une base de données de virus sorti depuis 2019 jusqu'à aujourd'hui et voir si un modèle entraîné avec des malwares des années 2013 peut détecter ces nouvelles menaces.

[53] https://github.com/ArtemKushnerov/az

4.1.5. Traitement des APK

Ce chapitre représente les points 5,6 et 7 de la figure 22.

Ce chapitre va décrire le fonctionnement de parser.py qui se trouve en annexe de ce document.

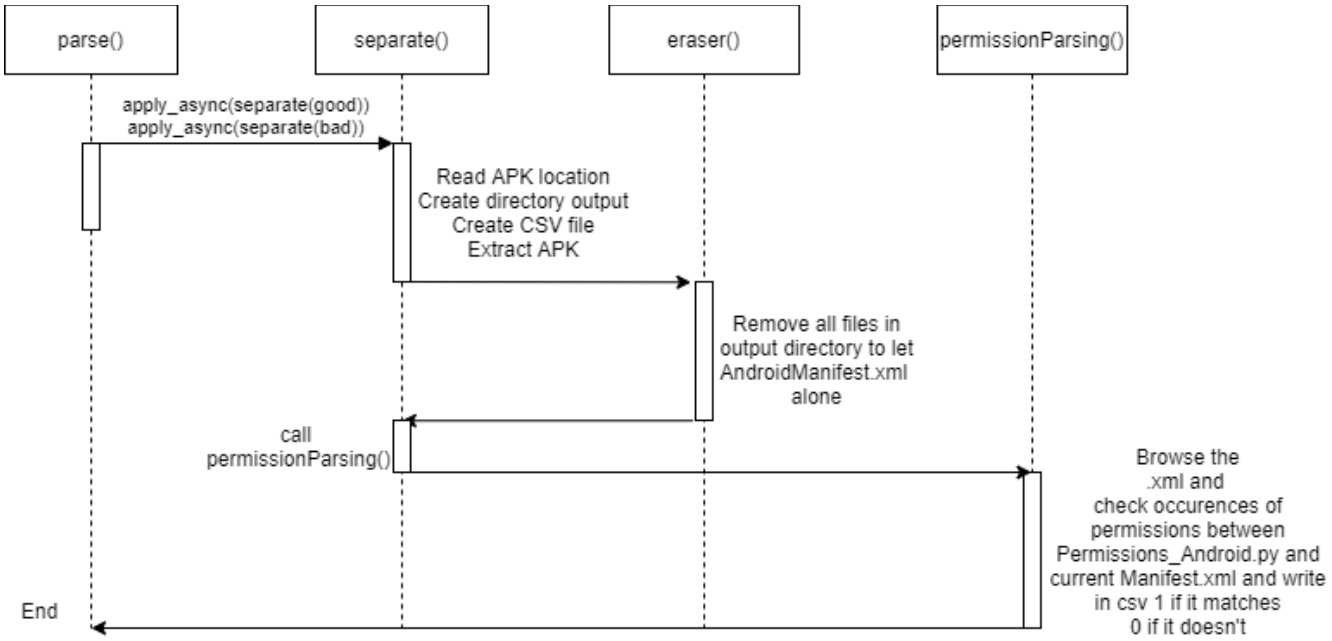


Figure 26 Schéma séquentiel de l'exécution de la méthode de *parsing*

Par souci de clarté, les paramètres des méthodes ont été omis dans ce schéma.

Le fonctionnement du parser est le suivant. Il reçoit en paramètre la localisation des APK téléchargé avec Androzoo et reçoit également le dossier où l'utilisateur souhaite extraire ses APK.

Le programme va lancer un processus pour parcourir les APK bénignes et un autre processus va être crée afin de parcourir les APK malicieuses. En parallèles les deux processus vont parcourir tous les fichiers présents et lancer une commande apktool[54] et stocker le résultat de cette commande dans le dossier de sortie spécifié par l'utilisateur.

Il sera ensuite appelé la méthode ereaser(filename) qui est chargée de vider tous les fichiers présentes dans le dossier de sortie crée par apktool et d'y laisser uniquement AndroixManifest.xml afin d'économiser de l'espace disque.

Ensuite nous allons *parser* le fichier XML et sortir les données présentent dans *name* de l'attribut *https://schemas.android.com/apk/res/android*. Si vous souhaitez reprendre mon projet et entraîner des modèles avec de nouvelles données, il serait facile de remplacer cette ligne par un autre élément que vous souhaiteriez récupérer, il suffit pour cela de changer l'attribut et vous pourrez récupérer une autre information qu'une permission.

Une fois tous les éléments récupérés, ces derniers seront comparés aux permissions présentes dans le fichier Permissions_Android.py. Si l'application actuelle existe, nous allons saisir 1, autrement nous saisirons -1 dans le csv.

Une fois tous les fichiers parcourus, l'application se termine et aura crée vos deux fichiers « binaryApps_BEN.csv » et « binaryApps_MAL.csv ».

[54] https://doc.ubuntu-fr.org/apktool

« 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1 »

Voici donc une représentation d'application bénigne faites par notre extraction de données.

« 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 »

Voici la représentation d'une application malicieuse faites par notre extraction de données.

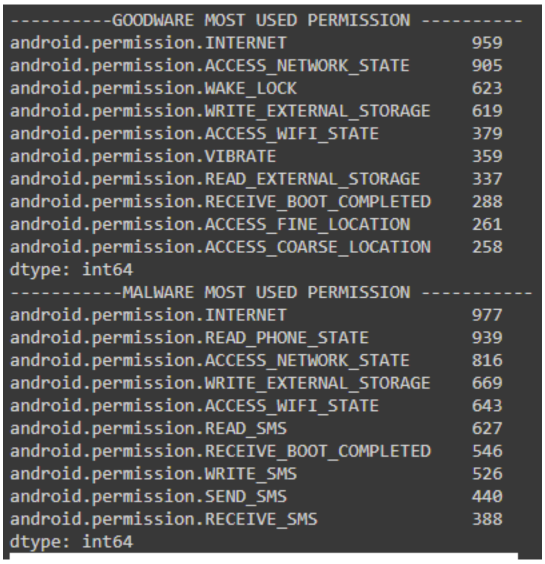

```
----------GOODWARE MOST USED PERMISSION ----------
android.permission.INTERNET                   959
android.permission.ACCESS_NETWORK_STATE       905
android.permission.WAKE_LOCK                  623
android.permission.WRITE_EXTERNAL_STORAGE     619
android.permission.ACCESS_WIFI_STATE          379
android.permission.VIBRATE                    359
android.permission.READ_EXTERNAL_STORAGE      337
android.permission.RECEIVE_BOOT_COMPLETED     288
android.permission.ACCESS_FINE_LOCATION       261
android.permission.ACCESS_COARSE_LOCATION     258
dtype: int64
-----------MALWARE MOST USED PERMISSION -----------
android.permission.INTERNET                   977
android.permission.READ_PHONE_STATE           939
android.permission.ACCESS_NETWORK_STATE       816
android.permission.WRITE_EXTERNAL_STORAGE     669
android.permission.ACCESS_WIFI_STATE          643
android.permission.READ_SMS                   627
android.permission.RECEIVE_BOOT_COMPLETED     546
android.permission.WRITE_SMS                  526
android.permission.SEND_SMS                   440
android.permission.RECEIVE_SMS                388
dtype: int64
```

Figure 27 Top 10 des permissions utilisées pour les applications malicieuses et bénigne

Nous pouvons observer dans ce top 10 que les malwares de notre jeu de données ont souvent demandé des accès à nos informations personnels

READ_PHONE_STATE	Permet à l'application de connaître votre numéro de téléphone, des informations sur votre réseau, le statut de vos appels.
READ_SMS, WRITE_SMS et RECEIVE_SMS	Permet d'accéder en lecture, écriture et de recevoir des SMS sur l'application installée.

Ces permissions sont utilisées afin de récupérer des infos personnelles et sont souvent rapatriée vers un serveur distant afin de dresser un profil sur vous ou faire du chantage selon la sensibilité de vos données.

En annexe vous retrouverez une mise en comparaison des droits les plus utilisés par les malwares sur l'axe des x et le nombre d'implémentation de ces dernières chez les bonnes et mauvaises applications.

4.1.6. Fonctionnement général

Ce chapitre va décrire de façon général les différents points de la Figure 22. L'implémentation de tous les modèles ainsi que le code de start.py se trouvent en annexe de ce document.

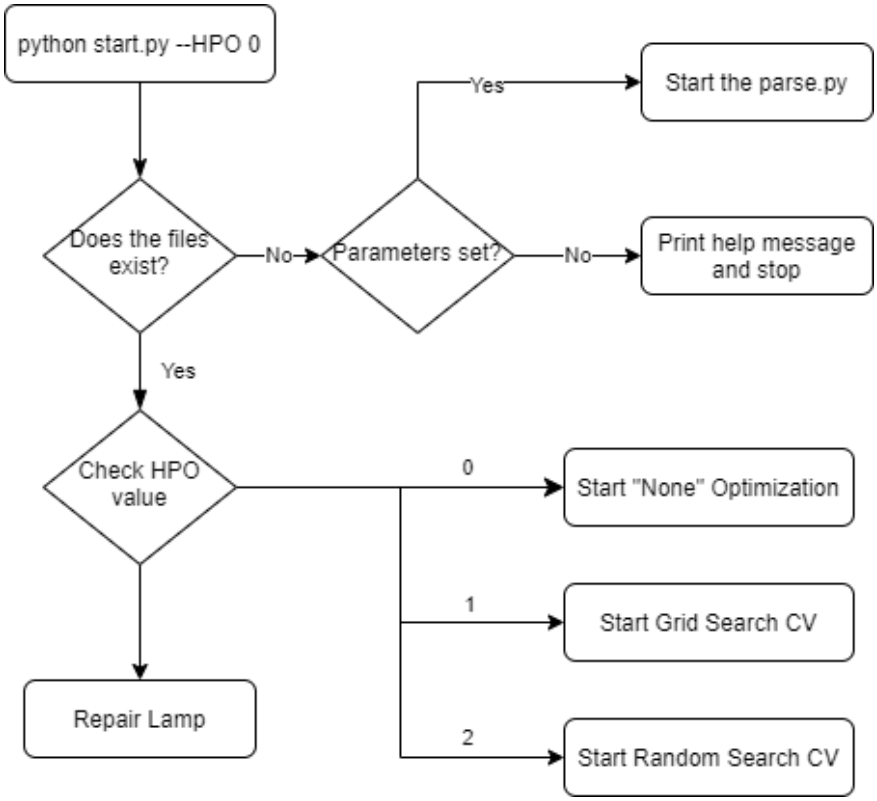


Figure 28 Fonctionnement logique de start.py

La librairie « parse », permet de récupérer les données saisies par l'utilisateur en paramètres de la commande.

```
"--maldir", help="Absolute path to directory containing malware apks.")
"--maldir_dest", help="Absolute path to directory where to store the malware apktool output.")
"--goodddir", help="Absolute path to directory containing benign apks.")
"--goodddir_dest", help="Absolute path to directory where to store the benign apktool output.")
"--testsize", help="Size of the test set when split (value between ]0;1[).")
"--HPO", help="Apply a Hyper-parameters optimization. 2 = Random Search, 1 = GridSearch and 0 = None")
```

Figure 29 Paramètres à saisir pour lancer le programme

Les noms de fichiers nécessaires sont codés en dur dans l'application, les CSV doivent porter le nom qui est indiqué dans ce document (par défaut le parser.py crée ces fichiers au nom adéquat).

Si les fichiers n'existent pas, start.py va exécuter la méthode d'entrée du fichier parse.py. Une vérification est faite au préalable afin de d'assurer que tous les paramètres nécessaires ont été renseigné, si ce n'est pas le cas, nous affichons un message d'aide et indiquons qu'il manque des paramètres à l'utilisateur.

Si tous les paramètres sont renseignés, nous créons les fichiers comme indiqué au point 4.1.5.

Si vous n'aviez pas de csv, il vous faudrait relancer une fois la commande python start.py --HPO <0,1,2>

Cette fois-ci, selon si vous avez saisis la valeur 0,1 ou 2 pour HPO, le fil d'exécution sera différent.

HPO 0 va entraîner les modèles suivants : *SVM*, *kNeighbors*, *Random Forest*, *Linear Regression*, *Naive Bayes* et *Decision Tree.* 0 indique qu'aucune optimisation n'est faites sur les paramètres de ces méthodes et que les valeurs par défaut implémentée par sklearn sont choisies.

Pour HPO 1 et HPO 2, les méthodes implémentées sont : *SVM, kNeighbors* et *Random Forest*

J'ai décidé de ne pas implémenter les autres méthodes car, celles sélectionnées sont celles ayant obtenus les meilleurs résultats lors de la majorité des études.

Lorsque vous appelez **HPO 1** vous allez effectuer un Grid Search CV, cette fois-ci nous allons spécifier une liste de paramètres que nous souhaiterions optimiser afin d'obtenir le meilleur score possible et nous allons spécifier les différents paramètres à tester. La méthode de GridSearchCV de sklearn va alors pré-entraîner un modèle afin d'obtenir les meilleurs paramètres et va ensuite utiliser les données restantes afin d'entraîner et tester le modèle.

Pour **HPO 2** nous allons utiliser cette fois la méthode de Random Search CV, cette fois-ci, selon une portée donnée, RandomSearchCV va sélectionner des valeurs aléatoires dans la range du paramètre et va nous retourner les meilleurs modèles possibles afin de l'entraîner.

Pour chacune des implémentations, il est mis en commentaire dans le code un lien vers les paramètres que nous pouvons modifier, changer ainsi que les score que nous souhaiterions optimiser.

Les méthodes ont été le plus commenté possible afin de permettre à une personne souhaitant reprendre et améliorer le projet puisse le faire. Des pistes de développement pour de futurs travaux sont disponibles dans le chapitre 4.1.9

Il existe d'autres méthodes *d'hyperparameter optimization* mais ces dernières n'ont pas été implémentée car les résultats obtenus avec les deux méthodes implémentées étaient satisfaisants.

4.1.7. Résultat des modèles

Les résultats détaillés obtenus par l'application sont disponibles en annexe de ce document.

Tout d'abord, les jeux de données utilisés dans ces résultats sont ceux présents dans le dossier « 1000B1000M »

Il y a exactement 997 applications bénignes et 999 malwares. Ces chiffres ne reflètent pas vraiment la réalité, sur les stores officiels, nous n'aurons jamais une telle proportion d'applications malveillantes, mais avoir un équilibre comme celui-ci permet aux modèles de bien pouvoir apprendre ce qu'est une bonne application et ce qu'est une mauvaise application. Un modèle déséquilibré peut parfois mener à de mauvaise prédiction car le modèle n'est pas assez entraîné à différencier les deux classes.

La répartition entre les données de tests et les données d'entraînements est de 20% pour les tests et 80% pour l'entraînement. Des tests ont été effectués en utilisant 30/70 et 40/60 mais aucun résultat notable n'a été observé.

Les différents plots et score sont calculés sur la base de la matrice de confusion

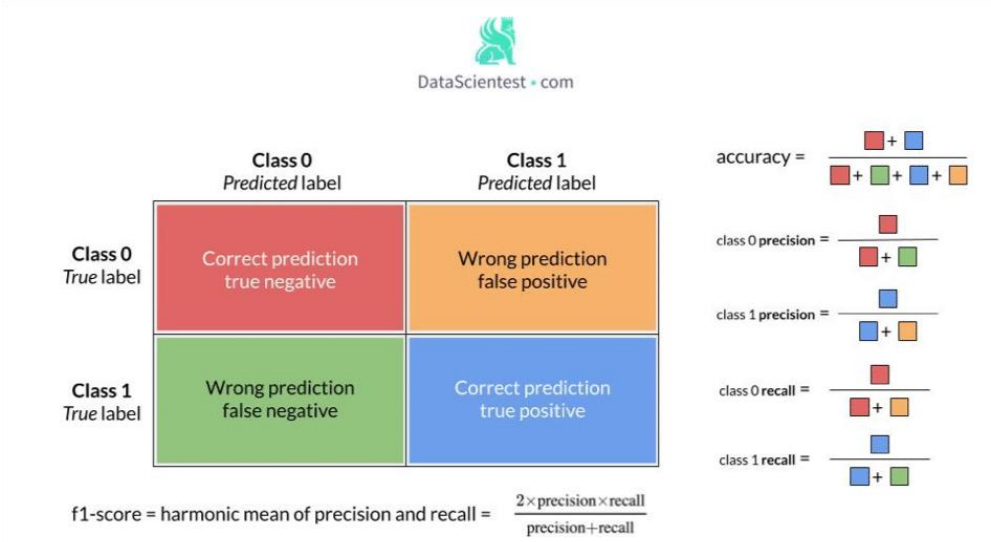


Figure 30 Représentation d'une matrice de confusion et des scores calculables, datascientest.com[55]

Vrai positive : Nous avons présenté au modèle une application malicieuse et l'a détecté.

Faux positif : Nous avons présenté au modèle une application bénigne et a détecté que c'était un maliciel.

Faux négatif : Nous avons présenté au modèle une application malicieuse et a détecté qu'elle était bénigne.

Vrai négatif : Nous avons présenté au modèle une application bénigne et a détecté qu'elle était bénigne.

En nous basant sur ces résultats nous pouvons calculer différents scores.

Les tests ont été lancés une dizaine de fois et je vous présente ici le résultat médian sur ces 10 lancers

HPO 0

Modèle	Precision	Recall	F1-score	Support BEN/MEN
SVM	67 %	68%	67%	320 / 80
(3)k-Neighbor	97 %	96%	97%	198/202
Random Forest	97%	97%	97%	197/203
Linear Regression	96 %	96 %	96 %	192/208
Naive Bayes	62%	62%	62%	44/356
Decision Tree	94%	94%	94%	192/208

[55] https://datascientest.com/comment-gerer-les-problemes-de-classification-desequilibree-partie-i

Quelques constatations peuvent être faites, tout d'abord, les résultats pour les méthodes k-neighbor, random forest, linear regression et décision tree sont très bons. Afin de confirmer l'hypothèse que les permissions suffisent à différencier un malware d'une application bénigne, il nous faudrait avoir ici aussi un ensemble de données encore plus grand, mais au vu des résultats, cela semble possible avec les modèles cités précédemment. Une raison pouvant expliquer les mauvais résultats de SVM et Naive Bayes pourrait venir de la mauvaise répartition des valeurs tests qui est trop importantes, 320/80 revient constamment et 44/356 n'est pas adéquat non plus. Choisir de meilleurs paramètres devrait régler ces problèmes. Nous pouvons déduire que les modèles cités précédemment n'ont rien appris en regardant leur matrice de confusion. Les modèles ne savent pas différencier un malware d'une application bénigne.

Le choix d'implémentation de ces modèles vient du besoin d'obtenir le meilleur résultat possible pour un ensemble de données quelconques.

HPO 1

Modèle	Precision	Recall	F1-score	Support BEN/MEN
SVM	96 %	96%	96%	193 / 207
(5)k-Neighbor	97 %	97%	97%	193/207
Random Forest	97%	97%	97%	197/203

Dans ce cas-ci nous pouvons voir l'importance de choisir de bons paramètres pour un modèle. Le score de SVM s'est clairement amélioré et est maintenant plus ou moins égal aux autres modèles même s'il produit des résultats moins bons. Nous pouvons voir ici que le choix du kernel a été important, l'utilisation du kernel rbf a permis d'obtenir de meilleurs résultats.

k-Neighbor a obtenu de meilleurs résultats en utilisant 5 voisins et Random Forest est resté proche de son résultat avec HPO 0 car les paramètres de base lui convienne mieux.

HPO 2

Modèle	Precision	Recall	F1-score	Support BEN/MEN
SVM	96 %	96%	96%	193 / 207
(4)k-Neighbor	97 %	96%	97%	193/207
Random Forest	97%	97%	97%	200/200

Les résultats entre HPO 1 et 2 sont relativement similaires car ils font varier les mêmes paramètres, mais en théorie HPO 2 sera meilleur que HPO 1 car il pourra être plus précis sur certains paramètres qu'il choisit aléatoirement que HPO 1 qui va lui dépendre des listes de paramètres fournis.

Pour conclure sur tous ces résultats, nous avons pu voir que la plupart des modèles de classification permettent de bien différencier les applications bénignes des applications malicieuses, le grand nombre d'entrées possibles ainsi que certaines combinaisons de permissions doit permettre une distinction facile de ces malwares. Il serait maintenant intéressant de reproduire une expérience similaire mais cette fois-ci en utilisant des malwares récents.

Il pourrait également être intéressant de limiter en données de test le nombre de malware afin de tester un cas réel où un téléphone aurait été infecté par 1-2 applications sur une quarantaine. Cela permettrait également de s'assurer que le modèle est fiable.

Afin d'en tirer plus de conclusion il aurait été intéressant d'implémenter les différents éléments présents dans le chapitre 4.1.9. Je pense notamment à représenter le poids des entrées ayant le plus d'influence sur la détection de maliciels et ainsi optimiser les paramètres d'entrée afin d'éviter de surcharger le modèle et de voir quelle permission influence le choix de la prédiction. Il serait également intéressant de pouvoir

4.1.8. Problèmes rencontrés

Erreur lors de la création de jeux de données

Je n'avais au départ pas compris qu'il fallait fournir une *range* pour spécifier le nombre de -vt, j'ai donc lancé un téléchargement de 8000 *malwares* et de 8000 bonnes applications et en comparant le nombre d'occurrence des droits, j'ai réalisé que les deux ensembles avaient presque les mêmes droits.

L'erreur venait du fait que j'avais saisi -vt **:10** au lieu de **10 :** ce qui change la range [0 à 10] contre [10 :max].

IMEI

L'IMEI de mon téléphone Nokia n'est pas valide. Ce qui veut dire que mon téléphone n'utilise pas une version d'usine du système installé de base lors de l'achat du téléphone ou que mon téléphone est une contrefaçon. Je n'ai donc pas pu recevoir ma clé pour modifier le boot et ainsi craquer mon téléphone.

Kali Linux

J'utilisais par défaut Kali Linux afin de tester, télécharger des applications et les temps de calcul étaient très longs, les téléchargements très lents. En passant tout mon code sûr Ubuntu, j'ai obtenu de bien meilleurs résultats.

4.1.9. Futurs travaux

Voici une liste de différentes améliorations que nous pourrions faire afin d'améliorer ce produit

- Implémentations de nouveaux HPO [56]
  - Hyperband
  - BO-GP
  - BO-TPE
  - PSO
- Proposer à l'utilisateur d'afficher ou non des plots.
- Ajouter de nouveaux graphiques [57] [58]afin de pouvoir effectuer des analyses plus détaillées .
- Exporter[59], importer et gérer nos modèles .
- Implémenter une interface graphique à l'application afin de pouvoir gérer nos fichiers et exécuter un modèle de façon isolée.
- Créer des jeux de données plus grand et plus spécifiques, en utilisant que des applications récentes par exemple.
- Implémenter des méthodes de deep learning et des algorithmes génétiques. J'ai été amené à essayer d'en implémenter au début du projet, mais les temps de calcul très lent et ma capacité de calcul étant limitée, l'optimisation de paramètres étaient trop lentes à faire et cette idée est devenue une idée de futurs travaux.
- Utiliser un fichier de configuration .json afin de stocker les différents paramètres que nous souhaitons implémenter à nos modèles
- Utiliser d'autres éléments observables dans du code statique, par exemple implémenter une méthode similaire à celle de MaMaDroid.
- Utiliser des threads avec le max de cœurs disponible sur la machine exécutant le code afin de pouvoir accélérer le processus de *parsing* des XML
- Automatiser les tests, en lançant par exemple automatiquement n fois les tests, puis en calculant les scores moyens afin d'obtenir une estimation du fonctionnement moyen des modèles.

---

[56]https://github.com/LiYangHart/Hyperparameter-Optimization-of-Machine-Learning-Algorithms/blob/master/HPO_Classification.ipynb

[57] https://www.scikit-yb.org/en/latest/api/classifier/classification_report.html

[58] https://thedatafrog.com/en/articles/matplotlib-machine-learning/

[59] https://cloud.google.com/ai-platform/prediction/docs/exporting-for-prediction

- Imprimer le poids des entrées une fois l'entraînement terminé afin de savoir quel élément pèse le plus dans la classification.

5. Conclusion

Nous avons vu à travers ce document un état des lieux de la technique et de l'art nous faisant prendre conscience que l'implémentation de machine learning dans des modules de forensique peut être efficace et produire des résultats satisfaisants. De par le manque d'information partagée par les entreprises, il est difficile de savoir ce qui se cache vraiment derrière les outils qu'ils nous proposent et par conséquent, implémenter et améliorer leur outil est impossible.

L'outil développé à terme de ce projet a permis de tester la validité de deux études faites il y a plusieurs années et qui ont les deux eu un problème quant à la quantité de données utilisées, l'utilisation d'Androzoo permet de pallier ce manque et d'obtenir une banque de données colossales. Nous avons obtenu de meilleurs résultats que ceux présentés il y a plusieurs années, ce qui démontre l'importance de deux choses. La première, avoir un bon modèle et de mauvaises données produit un mauvais résultat et la deuxième, de mauvais modèle couplé à de bonnes données ne donne également pas de bons résultats. Pour conclure, générer un bon jeu de données et tout aussi important que bien choisir et paramétrer son modèle d'apprentissage, la combinaison de ces deux points font que les résultats qui en sortent seront bons.

Lausanne, le 04.06.2021

Nemanja Pantic

6. Références

La liste des travaux sera mise ici manuellement, je linkerais les études par la suite

[1] Salman Iqbal and Soltan Abed Alharbi, « Advancing Automation in Digital Forensique Investigations Using Machine Learning Forensiques »

[2] K. Allix, T. F. Bissyandé, J. Klein and Y. L. Traon, "AndroZoo: Collecting Millions of Android Apps for the Research Community," 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), 2016, pp. 468-471.

[3] Arp, Daniel & Spreitzenbarth, Michael & Hübner, Malte & Gascon, Hugo & Rieck, Konrad. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. Symposium on Network and Distributed System Security (NDSS). 10.14722/ndss.2014.23247.

[4] Urcuqui López, Christian & Navarro, Andres. (2016). Machine learning classifiers for android malware analysis. 1-6. 10.1109/ColComCon.2016.7516385.

[5] Urcuqui López, Christian & Navarro, Andres. (2016). Framework for malware analysis in Android. Sistemas & Telemática; Vol 14, No 37 (2016)DO - 10.18046/syt.v14i37.2241. 14. 45-56. 10.18046/syt.v14i37.2241.

[6] Urcuqui López, Christian. (2016). Módulo de machine learning para detección de malware en Android. 10.13140/RG.2.2.35287.06564.

[7] W. Enck, M. Ongtang, and P. D. McDaniel. On lightweight mobile phone application certification. In Proc. of ACM Conference on Computer and Communications Security (CCS), pages 235–245, 2009.

[8] Karbab, Elmouatez & Debbabi, Mourad & Derhab, Abdelouahid & Mouheb, Djedjiga. (2018). MalDozer: Automatic framework for android malware detection using deep learning. Digital Investigation. 24. S48-S59. 10.1016/j.diin.2018.01.007.

[9] Mariconti, Enrico & Onwuzurike, Lucky & Andriotis, Panagiotis & De Cristofaro, Emiliano & Ross, Gordon & Stringhini, Gianluca. (2016). MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models.

[10] Aung, Zarni & Zaw, Win. (2013). Permission-Based Android Malware Detection. International Journal of Scientific and Technology Research. 2. 228-234.

[11] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," 2012 IEEE Symposium on Security and Privacy, 2012, pp. 95-109, doi: 10.1109/SP.2012.16.

[12] Damshenas M, Dehghantanha A, Choo K K R, et al. M0droid: An android behavioral-based malware detection model[J]. Journal of Information Privacy and Security, 2015, 11(3): 141-157

[13] Kiss, N & Lalande, J.-F & Leslous, Mourad & Viet Triem Tong, Valérie. (2016). Kharon dataset: Android malware under a microscope.

[14] K. Allix, T. F. Bissyandé, J. Klein and Y. L. Traon, "AndroZoo: Collecting Millions of Android Apps for the Research Community," 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), 2016, pp. 468-471.

Document utilisé en vrac

7. Planification

7.1. Planification initiale

Vous trouverez en annexe la planification initiale. Cette dernière a été difficile à faire pour moi car j'ai eu énormément de peine à démarrer et comprendre les tenant et aboutissant de mon projet, j'ai donc effectué un planning en imaginant avoir besoin de beaucoup de temps afin de comprendre les tenant et aboutissant du machine learning et du forensique.

7.2. Planification finale

Il a été difficile de mélanger les cours et la régularité dans ce projet, en regardant bien la répartition du temps, nous voyons bien que le projet a été fait à environs 80% les 3-4 dernières semaines avant le rendu final. Dans l'ensemble cette méthode de travail me correspond bien, dédier des semaines entières à la réalisation d'un projet me permet de rester concentrer dessus et d'y réfléchir en dehors des périodes de travail. Je reste très content de ce que j'ai accomplis et je trouve que les résultats de la reprise de l'étude sont très bons. De plus l'outil permet à une personne novice de pouvoir prendre en main des outils de machine learning et de comprendre une première implémentation simple de détection de maliciel à l'aide de machine learning.

8. Tables des figures

9. Annexe

Vous trouverez dans ce chapitre toutes les annexes de ce document, notamment, les résultats des tableaux comparatifs des outils de forensique, les résultats de mon implémentation de machine learning ainsi que le code crée.

9.1. Spidermap

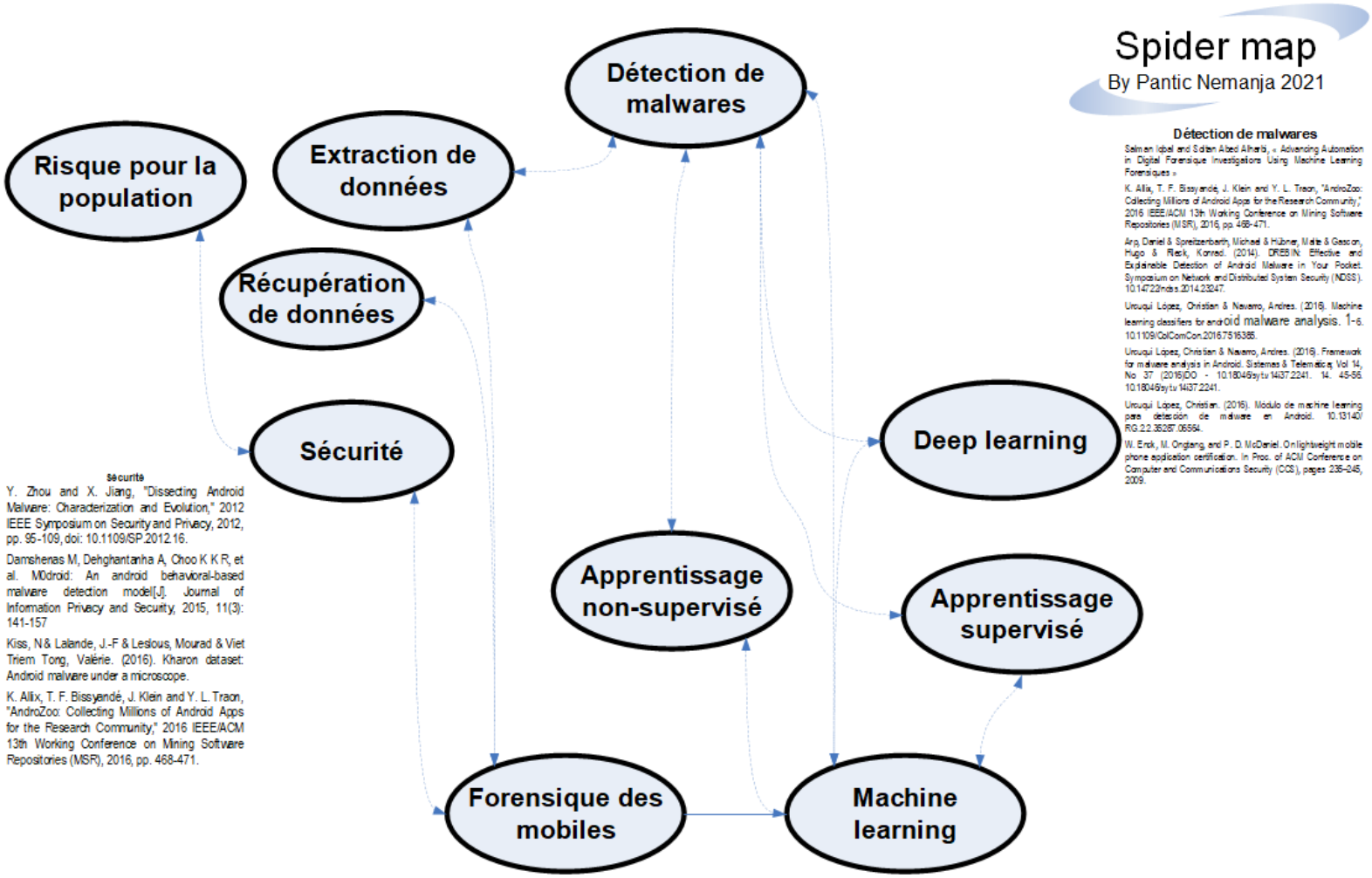


Figure 31 Spidermap du projet

9.2. Tableaux comparatifs des outils de forensique commerciaux

Nom des outils	Description	Fonctionnalités	ML ?	Prix	Spécifications	OS compatible	Outils spécifiques
Cellebrite Touch UFED	Utilisé pour récupérer et obtenir des données d'un téléphone pris sous perquisition de la Justice.	• Développer afin de pouvoir extraire le plus de données possibles • Utiliser un bootloader exclusif, une capacité EDL automatique et des commandes ADB • Extrait les données des téléphones, drones, carte SIM, GPS. • Passer à travers la sécurité de téléphone verrouillé • Retrouve les données supprimées	Pas de mention	Refus de communiquer le prix car je ne suis pas un acheter potentiel.	**UFED 4PC :** • Tourne sur Windows 8 et 10 64bit • Minimum 4Go RAM, recommandé 8Go RAM **UFED Touch2 :** • 128 Go SSD • High res. screen • 8 Go RAM DDR3L USB3.0 ou Wifi	iOS et Android	• **UFED 4PC** : est un logiciel économique, flexible et pratique sur ordinateur • **UFED Touch2** Petite tablette permettant de récupérer des données n'importe où • **UFED Ruggedized Panasonic Laptop** : chargé avec le logiciel UFED 4PC et résiste aux chocs et aux températures extrêmes afin de garantir des transferts.
Encase Mobile Investigator	Permet d'analyser notre téléphone et de report toute action suspecte sur ce dernier sans altérer à l'intégrité des preuves.	• Présentation simplifiée des preuves • OCR Puissant • Passer à travers la sécurité de téléphone verrouillé • Bonne compréhension de différents types de fichiers (PDF, HTML, SQLite, archives.) • Retrouve les données supprimées • Avec les bons droits, peut se connecter aux réseaux sociaux	Pas de mention	Refus de communiquer le prix car je ne suis pas un acheter potentiel.	• Tourne sur Windows 10 64 bits • Câble USB • Pas d'informations sans contacter	iOS, Android, Windows Mobile, Blackberry	Le logiciel est à installer et fonctionne immédiatement

Nom des outils	Description	Fonctionnalités	ML ?	Prix	Spécifications	OS compatible	Outils spécifiques
Oxygen Forensiques Detective Fonctionnalités	Il s'agit d'un outil tout en un permettant d'extraire, analyser des données de différents device (IOT, Mobile, Windows, Linux).	• Décrypte les mots de passes, les tokens d'authentification des utilisateurs afin de récupérer leurs réseaux sociaux • Peut faire abstraction de la sécurité des téléphones afin de récupérer plus d'informations • Permet de gérer des images et des backups • Permet de faire de la reconnaissance faciale • Utilise la geo data • Permet de créer des graphiques sociaux (liens entre ses contacts et lui) • Retrouve les données supprimées	Pas de mention	Refus de communiquer le prix car je ne suis pas un acheter potentiel.	• Tourne sur Windows 10 64 bits • Câble USB • Pas d'information sans contacter	Android et iOS	Le logiciel est à installer et fonctionne immédiatement
MOBILedit Forensique Express	Cet outil est utilisé afin de détecter des preuves d'intrusion. Il est destiné aux professionnels et particuliers	• Déverrouiller des téléphones • Accès aux données du téléphone • Analyse les photos (détecter armes, etc). • Reconnaissance faciale • Analyse les réseaux sociaux • Détection de malware	Utilisation de deep learning pour la reconnaissance de photos.	**Single phone :** Pai**e** par téléphone, retrouver données effacés, analyse application, 6 mois de mise à jour, **99$** **Express Standard** : Pas de limite de téléphone, frais de	USB3.0	Android, iOS, Blackberry, Windows Phone, Bada, téléphone CDMA, Bada, Symbian …	Le logiciel est à installer et fonctionne immédiatement

				licence unique, 12 mois de mise à jour, analyse physique et importe illimité **1500$** **Express Pro :** Tous les éléments dans fonctionnalités possibles avec un pricing ajustable en les contactant			
Nom des outils	**Description**	**Fonctionnalités**	**ML ?**	**Prix**	**Spécifications**	**OS compatible**	**Outils spécifiques**
PC-3000 Mobile	Ce logiciel permet de retrouver des données effacées sur son téléphone à l'aide de forensique et il permet également d'analyser les données afin d'en faire un rapport. Cet outil est principalement désigné à récupérer des données	• Récupérer des données effacées sur un téléphone endommagé • Bypass des sécurités • Faire des copies d'image • Extraire les données afin de les analyser à l'aide de nos outils de forensique	Pas de mention	4250 euros avec 2 ans de mise à jour et de garantie	Recommandé : CPU Intel Core i5 (4 coeurs) 8Gb de Ram 200Mb de disque libre Windows 7(SP1,8,8.1,10, x86 et x64) USB 3.0 Type-A	Carte SD, mSD, SATA, etc	Boîte à outils PC-3000 Mobile

Paraben corporation E3 :DS	Il s'agit d'un outil multifonctionnel permettant d'effectuer des images en préservant l'intégrité des données ainsi que d'effectuer des analyses.	• Extraire les données en utilisant les exploits connus • Mises à jour régulières • Analyse des programmes • Scan de spyware • Support l'IOT • OCR • Cloud forensique • Android Rooting Engine	Pas de mention	3495$ par licence avec une nouvelle *release* tous les 3 mois	Windows x86 et x64 bits USB3.0	iOS et Android	Le logiciel est à installer et fonctionne immédiatement

9.3. Outils de forensique digital libres

Nom des outils	Description	Fonctionnalités	ML ?	Spécifications	OS compatible
Volatility	Il s'agit d'un framework de collection d'outils implémenté en Python permettant l'extraction d'artefacts numériques à partir de la RAM.	• Extraction indépendante de si le téléphone est en cours d'utilisation ou non • Plugin de la communauté • Permet d' ?????	Pas de mention	Linux 32 et 64 bits , Windows XP SP2 et 3 à Windows 10	Android
Autopsy	Il s'agit d'un outil de forensique digital libre. Cet outil est facilement extensible à l'aide d'add-on module[60], ces derniers peuvent être écrits par nous-même, ce qui le rend très adaptatif à nos besoins. Il est également possible de partager ces modules. Son but principal est d'analyser les disques durs et smartphone et de récupérer des informations pour une analyse de preuves.	• Rapports personnalisés • Analyse et récupération des données en profondeur • Analyse des multimédia • Extraction de données (mais semble mauvaise[61]) • Analyse du volume du système • Add-on permettant des implémentation personnalisées • Documentation afin de prendre en main l'outil	Pas de mention	Disponible sur Windows 32 et 64 bits Fonctionne également sur Linux et OS X	iOS [62]et Android
Andriller CE	Il s'agit d'un assemblage de plusieurs outils de forensique pour smartphone. Ce dernier effectuer des lectures (en lecture seule) et permet de faire des acquisitions de données non-destructrice	• Débloquer un écran verrouillé • Trouver un code PIN • Permet de décoder des donnes d'applications et base de données. • Extraction des données (root ou non) • Récupérer les backup Android	Pas de mention	Environnement permettant d'exécuter du Python	Android et iOS

[60] https://GitHub.com/sleuthkit/autopsy_addon_modules

[61] https://ieeexplore.ieee.org/document/7880238

[62] https://sleuthkit.org/autopsy/docs/user-docs/4.17.0/ileapp_page.html

9.3.1. Conversation avec Cellebrite

Hi Nemanja,

As promised here's the answer:

I investigated (non-public) technical manuals, etc. but I could not find any trace of a machine-learning/AI capability of our solution.

However, to support you as much as possible I have attached the product overview for our UFED device.

I hope that you will find this useful.

Good luck with your paper!

Regards,

•••

[redacted] | Sales Development DACH
e.[redacted]@cellebrite.com | www.cellebrite.com
Cellebrite GmbH | Herzog-Heinrich-Strasse 20 80336 Munich Germany

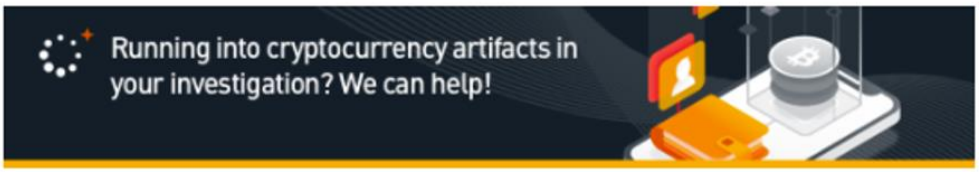


Figure 32 Conversation par mail avec un conseiller de Cellebrite

9.3.2. Start.py

```
"""
    This file is the entrypoint to 1) parse APK and 2) pass the .csv into ML m
odel
    It implements different optimisations and this program goal is to compare
different
    model.

    This file is based on : https://GitHub.com/urcuqui/WhiteHat/tree/master/Re
search/Android/scripts
                            https://GitHub.com/MLDroid/drebin/blob/master/src/
Main.py
"""
from parse import parse
from art import *

"""
If you want to implement more ML Model, you can create a file based on those e
xample and import them here and implement
method in the main with others methods.
"""
from linearRegression import linearRegression
from randomForestClassifier import randomForestClassifier
from naivesBayes import naivesBayes
from decisionTree import decisionTree
from SVC import SVC
from kNeighborsClassifier import kNeighborsClassifier
from SVCTune import SVCGS
from kNeighborsClassifierTune import kNeighborsClassifierGS
from randomForestClassifierTune import randomForestClassifierGS
from kNeighborsClassifierRandom import kNeighborsClassifierRS
from randomForestClassifierRS import randomForestClassifierGS
from SVCRS import SVCRS
import os.path
import argparse


def main(args):
    """
    Main function for
    1) If files doesn't exists : APK Parsing and generating a file
    2) If files exists : Classification of Benign and Malware application
    :param args: arguments acquired from parse_args()
    """

    # We recover here the different parameters given in command line
    malDir = args.maldir
    malDirDest = args.maldir_dest
```

```
    goodDir = args.gooddir
    goodDirDest = args.gooddir_dest
    testSize = args.testsize
    # We force the used name for CSV files, feel free to change them. By defau
lt the parsing of AndroidManifest.xml
    # will generate those filename.
    goodCsvFile = "binaryApps_BEN.csv"
    malwareCsvFile = "binaryApps_MAL.csv"
    # This arg is used to define which method of optimisation is chosen
    # 0 = None , 1 = GridSearch, 2 = RandomSearch
    HPO = args.HPO
    # We check if filename exists in current directory, if not, we create them
 with parse.py
    if os.path.isfile(goodCsvFile) and os.path.isfile(malwareCsvFile):
        # We check the value of HPO
        if HPO == 0:
            SVC(testSize)
            kNeighborsClassifier(testSize)
            randomForestClassifier(testSize)
            linearRegression(testSize)
            naivesBayes(testSize)
            decisionTree(testSize)

        if HPO == 1:
            SVCGS(testSize)
            kNeighborsClassifierGS(testSize)
            randomForestClassifierGS(testSize)

        if HPO == 2:
            SVCRS(testSize)
            kNeighborsClassifierRS(testSize)
            randomForestClassifierGS(testSize)

    else:
        # We check if parameters are specified
        if malDir == None or malDirDest == None or goodDir == None or goodDirD
est == None:
            print("You didn't specified all required parameters, check if you
have {} and {} in your current "
                  "directory or specify parameters to create them\nYou can use
 \"python start.py --h\" "
                  "to get help.".format(goodCsvFile, malwareCsvFile))
        else:
            parse(malDir, malDirDest, goodDir, goodDirDest)


def menu():
    """
```

```
    This method is used to check the value specified by the user in command li
ne
    By default values of parameters are :

    :return: ArgumentParser object which contains the parameters
    """
    args = argparse.ArgumentParser(description="Classification using Permissio
n Android.")

    args.add_argument("--
maldir", help="Absolute path to directory containing malware APK.")
    args.add_argument("--
maldir_dest", help="Absolute path to directory where to store the malware apkt
ool output.")
    args.add_argument("--
gooddir", help="Absolute path to directory containing benign APK.")
    args.add_argument("--
gooddir_dest", help="Absolute path to directory where to store the benign apkt
ool output.")
    args.add_argument("--testsize", type=float, default=0.2,
                      help="Size of the test set when split (value between ]0;
1[).")
    args.add_argument("--HPO", type=int, default=0,
                      help="Apply a Hyper-
parameters optimization. 2 = Random Search, 1 = GridSearch and 0 = None")

    return args.parse_args()


Art = text2art('DrebinEasy')
main(menu())
```

9.3.3. Parse.py

```
"""
    This file is used to :
    1) Call apktool to extract apk inside the right folder;
    2) Clear the apktool output directory and only let AndroidManifest.xml;
    3) Parse it with XML DOM and compare the apk permission Manifest with the
list in Permissions_Android.py.

    This file is based on : https://GitHub.com/urcuqui/WhiteHat/tree/master/Re
search/Android/scripts
"""
import Permissions_Android
import xml.etree.ElementTree as ET
```

```
import os
from os.path import join
from multiprocessing import Pool
import csv


def eraser(filename):
    """
    Remove all data in the apktool output to save data space.
    :param filename: filename of the APK.
    :return: None.
    """
    os.system("find Output{} -type f ! -name 'AndroidManifest.xml' -
delete".format(filename))


def permissionParsing(package, writer, MORB):
    """
    This permission is used to read the XML, parse it and write in a csv file
if a specific permission is present
    in current apk Manifest.
    :param package: Path to the current APK Manifest.
    :param writer: csv.writer object which keep a file open to write a row.
    :param MORB: Indicate if it's a Malware OR Benign.
    :return: None.
    """

    array_output = []
    try:
        print(package + "/AndroidManifest.xml")
        # Create entry point in xml
        tree = ET.parse(package + "/AndroidManifest.xml")
        treeRoot = tree.getroot()
        array_permissions = []
        # Browse all the file, check all occurrences of target and append it t
o a list
        # If you want to check other characteristics in a XML File you can rep
lace the value in target by another one
        target = 'uses-permission'
        for permission in treeRoot.findall(target):
            # We get the value name of target
            per = permission.get('{http://schemas.android.com/apk/res/android}
name')
            array_permissions.append(per)
        # We browse the list of permission, check if the current Manifest perm
ission are in AOSP_PERMISSIONS
        for item in sorted(Permissions_Android.AOSP_PERMISSIONS):

            if array_permissions.__contains__(item):
```

```
                # We append to the array 1 if doesn't match
                array_output.append("1")
            else:
                # We append to the array 0 if doesn't match
                array_output.append("0")
        # We add the value of MORB to indicate if it's a malware or a benign a
pplication
        array_output.append(MORB)
        # Write a line in csvFile
        writer.writerow(array_output)
        print("wrote...")
    except:
        print("error")

def separate(filename, dataDir, dataDirDest, MORB):
    """
    This method is used to list all APK in the directory and create the header
 of the file.
    :param filename: Name of the CSV File.
    :param dataDir: Path to the directory which contains all files.
    :param dataDirDest: Path to the directory to store output.
    :param MORB: Indicates if it's a Malware or Benign directory.
    :return: None.
    """
    # We list the dataDir directory and store them
    fileList = os.listdir(dataDir)
    link = []
    i = 0
    csvFile = filename
    # We open the file and create a csv.writer
    fl = open(csvFile, 'w')
    writer = csv.writer(fl)
    array_data = []
    # We store all permissions and write them as header in csv file
    for criter in sorted(Permissions_Android.AOSP_PERMISSIONS):
        array_data.append(criter)
    # We add the malware column to indicate the type
    array_data.append("malware")
    writer.writerow(array_data)

    # For each file in the list we add the full path
    for e in fileList:
        link.append(join(dataDir, e))
    # For all files, we use apktool to extract the apk to dataDirDest
    for filename in fileList:
        os.system("apktool d -f {} -
o {}Output{}".format(link[i], dataDirDest, filename))
        i += 1
        eraser(filename)
```

```
        permissionParsing(dataDirDest + "Output{}".format(filename), writer, M
ORB)

def parse(malDir,malDirDest, goodDir, goodDirDest):
    """
    Start 1 process to extract Manifest from APK in goodDir and start another
process to extract Manifest from APK
    in malDir.
    :param malDir: Path to the malware APK directory.
    :param malDirDest: Path to the malware output directory.
    :param goodDir: Path to the benign APK directory.
    :param goodDirDest: Path to the benign output directory.
    :return: None.
    """
    # You can change filename here
    goodCsvFile = "binaryApps_BEN.csv"
    malwareCsvFile = "binaryApps_MAL.csv"

    # Create two process
    pool = Pool(processes=2)
    # Attribute data to each process and call separate()
    pool.apply_async(separate, args=(goodCsvFile, goodDir, goodDirDest, "-1"))
    pool.apply_async(separate, args=(malwareCsvFile, malDir, malDirDest, "1"))

    # Close Pool and let all the processes complete
    pool.close()
    pool.join()
```

9.3.4. SVC.py

```
"""
This file is the implementation of basic SVC model
Source : https://GitHub.com/mokeam/malware-
analysis/blob/master/scripts/Malware_Classification_Model.py
"""
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn import svm


def SVC(size_split):
    """
```

```
    Method used to call the SVC model
    :param size_split: Size of the test dataset
    :return: None
    """

    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')

    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')

    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)

    # We normalize and standardize the value with all the dataset
    # Source : https://stackoverflow.com/questions/40758562/can-anyone-
explain-me-standardscaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    # We create the SVC classifier and apply, poly will generate really bad pr
ediction, the tuning will found the
    # appropriated kernel
    classifier = svm.SVC(kernel = 'poly')
    # We train the model
    classifier.fit(X_train, y_train)

    # We now test the model
    y_pred = classifier.predict(X_test)
    print("SVC")
    print(" ")
    # We print the report of the model and calculate the confusion_matrix
    print(classification_report(y_pred, y_test, labels=None))
    SVC_cm = confusion_matrix(y_test, y_pred)
    print(SVC_cm)
    # Show confusion matrix in a separate window
    plt.matshow(SVC_cm)
    plt.title('SVC matrix')
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
    print("")
```

9.3.5. SVCRS.py

```
"""
This file is the implementation of Random Search with SVC model
Source :    https://GitHub.com/LiYangHart/Hyperparameter-Optimization-of-
Machine-Learning-Algorithms
            https://GitHub.com/sontung/drebin-malwares
"""
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV


def SVCRS(size_split):
    """
    Method used to call the SVC model with Random Search
    :param size_split: Size of the test dataset
    :return: None
    """
    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)

    # We specify the parameters that we want to test
    # Check here to see the parameters of SVC
    # https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
    rf_params = {
        'gamma': [1e-3, 1e-2, 1 / 8],
        'C': sp_randint(1, 50),
        "kernel": ['linear', 'poly', 'rbf', 'sigmoid']
    }

    # We specify the scoring that we try to optimize
    # Check here to see which score you can add in the list
    # https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-
parameter
```

```
    scores = ["precision", "accuracy", "f1"]

    # We calculate the scoring
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        print("")

        Random = RandomizedSearchCV(svm.SVC(kernel="rbf"), rf_params, cv=5, sc
oring=score, n_jobs=2)
        Random.fit(X_train, y_train)

        print("Best parameters set found on development set:")
        print("")
        # By default refit is True, then when you call one of those attribute,
 the refit is made with the best
        # parameters.
        print(Random.best_estimator_)
        print(Random.best_params_)
        print("")
        print("Grid scores on development set:")
        print("")
        print(Random.best_score_)
        print("")

        print("Detailed classification report:")
        print("")
        print("The model is selected with the best specified parameters")
        print("")

        # We test now the model and print classification report and confusion
matrix
        y_true, y_pred = y_test, Random.predict(X_test)
        print(classification_report(y_true, y_pred))
        SVC_cm = confusion_matrix(y_test, y_pred)

        print(" ")
        # Show confusion matrix in a separate window
        plt.matshow(SVC_cm)
        plt.title('SVC matrix')
        plt.colorbar()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()
        print("")
        print("")
```

9.3.6. SVCTune.py

```
"""
```

```
This file is the implementation of GridSearch with SVC model
Source :    https://GitHub.com/LiYangHart/Hyperparameter-Optimization-of-
Machine-Learning-Algorithms
            https://GitHub.com/sontung/drebin-malwares
"""
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn import svm


def SVCGS(size_split):
    """
    Method used to call the SVC model with Grid Search
    :param size_split: Size of the test dataset
    :return: None
    """

    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)

    # We specify the parameters that we want to test
    # Check here to see the parameters of SVC
    # https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
    rf_params = {
        'gamma': [1e-3, 1e-2, 1 / 8],
        'C': [1, 10, 100],
        "kernel": ['linear', 'poly', 'rbf', 'sigmoid']
    }

    # We specify the scoring that we try to optimize
    # Check here to see which score you can add in the list
    # https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-
parameter
    scores = ["precision", "accuracy", "f1"]
```

```
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        print("")

        # We create the GridSearch and train the model
        clf = GridSearchCV(svm.SVC(), rf_params, cv=5, scoring=score, n_jobs=2
)
        clf.fit(X_train, y_train)

        print("Best parameters set found on development set:")
        print("")
        # By default refit is True, then when you call one of those attribute,
 the refit is made with the best
        # parameters.
        print(clf.best_estimator_)
        print(clf.best_params_)
        print("")
        print("Grid scores on development set:")
        print("")
        print(clf.best_score_)
        print("")

        print("Detailed classification report:")
        print("")
        print("The model is selected with the best specified parameters")
        print("")

        # We test now the model and print classification report and confusion
matrix
        y_true, y_pred = y_test, clf.predict(X_test)
        print(classification_report(y_true, y_pred))
        SVC_cm = confusion_matrix(y_test, y_pred)

        print(" ")
        # Show confusion matrix in a separate window
        plt.matshow(SVC_cm)
        plt.title('SVC matrix')
        plt.colorbar()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()
        print("")
        print("")
```

9.3.7. RandomForestClassifier.py

```
"""
This file is the implementation of basic Random Forest Classifier model
Source : https://GitHub.com/mokeam/malware-
analysis/blob/master/scripts/Malware_Classification_Model.py
"""
from sklearn.metrics import confusion_matrix as cm
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import matplotlib.pyplot as plt


def randomForestClassifier(size_split):
    """
    Method used to call the Random Forest Classifier model
    :param size_split: Size of the test dataset
    :return: None
    """
    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)

    RFC_clf = RandomForestClassifier()
    # We create the model
    RFC_clf = RFC_clf.fit(X_train, y_train)

    # We now test the model
    y_pred = RFC_clf.predict(X_test)

    print("Random Forest")
    print(" ")

    # We print the report of the model and calculate the confusion_matrix
    RFC_cm = cm(y_test, y_pred)
    print(classification_report(y_pred, y_test, labels=None))
    print(RFC_cm)
    # Show confusion matrix in a separate window
    plt.matshow(RFC_cm)
```

```
    plt.title('RFC Confusion matrix')
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

9.3.8. randomForestClassifierRS.py

```
"""
This file is the implementation of Random Search with Random Forest Classifier
 model
Source :    https://GitHub.com/LiYangHart/Hyperparameter-Optimization-of-
Machine-Learning-Algorithms
            https://GitHub.com/sontung/drebin-malwares
"""
from sklearn.metrics import confusion_matrix as cm
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

def randomForestClassifierGS(size_split):
    """
    Method used to call the Random Forest Classifier model with Random Search
    :param size_split: Size of the test dataset
    :return: None
    """
    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)
    # We specify the parameters that we want to test
    # Check here to see the parameters of Random Forest
    # https://scikit-
learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.htm
l
    rf_params = {
```

```
        'n_estimators': sp_randint(10, 100),
        "max_features": sp_randint(1, 64),
        'max_depth': sp_randint(5, 50),
        "min_samples_split": sp_randint(2, 11),
        "min_samples_leaf": sp_randint(1, 11),
        "criterion": ['gini', 'entropy']
    }
    # We specify the scoring that we try to optimize
    # Check here to see which score you can add in the list
    # https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-
parameter
    scores = ["precision", "accuracy", "f1"]

    # We calculate the scoring
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        print("")

        # We create the RandomizedSearch and train the model
        Random = RandomizedSearchCV(RandomForestClassifier(), rf_params, cv=5,
 scoring=score, n_jobs=2)
        Random.fit(X_train, y_train)

        print("Best parameters set found on development set:")
        print("")
        # By default refit is True, then when you call one of those attribute,
 the refit is made with the best
        # parameters.
        print(Random.best_estimator_)
        print(Random.best_params_)
        print("")
        print("Grid scores on development set:")
        print("")
        print(Random.best_score_)
        print("")

        print("Detailed classification report:")
        print("")
        print("The model is selected with the best specified parameters")
        print("")

        # We test now the model and print classification report and confusion
matrix
        y_true, y_pred = y_test, Random.predict(X_test)
        print(classification_report(y_true, y_pred))
        print("")
        RFC_cm = cm(y_test, y_pred)

        print(" ")
```

```
        print(RFC_cm)
        print(classification_report(y_pred, y_test, labels=None))

        # Show confusion matrix in a separate window
        plt.matshow(RFC_cm)
        plt.title('RFC Confusion matrix')
        plt.colorbar()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()
```

9.3.9. randomForestClassifierTune.py

```
"""
This file is the implementation of GridSearch with Random Forest Classifier mo
del
Source :    https://GitHub.com/LiYangHart/Hyperparameter-Optimization-of-
Machine-Learning-Algorithms
            https://GitHub.com/sontung/drebin-malwares
"""
from sklearn.metrics import confusion_matrix as cm
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt


def randomForestClassifierGS(size_split):
    """
    Method used to call the Random Forest Classifier model with Grid Search
    :param size_split: Size of the test dataset
    :return: None
    """
    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)
    # We specify the parameters that we want to test
```

```
    # Check here to see the parameters of Random Forest
    # https://scikit-
learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
    rf_params = {
        'n_estimators': [10, 20, 30],
        'max_features': ["auto", "sqrt", "log2", None],
        'max_depth': [15, 20, 30, 50],
        "criterion": ['gini', 'entropy']
    }
    # We specify the scoring that we try to optimize
    # Check here to see which score you can add in the list
    # https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-
parameter
    scores = ["precision", "accuracy", "f1"]

    # We calculate the scoring
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        print("")

        # We create the GridSearch and train the model
        clf = GridSearchCV(RandomForestClassifier(), rf_params, cv=5, scoring=
score, n_jobs=2)
        clf.fit(X_train, y_train)

        print("Best parameters set found on development set:")
        print("")
        # By default refit is True, then when you call one of those attribute,
 the refit is made with the best
        # parameters.
        print(clf.best_estimator_)
        print(clf.best_params_)
        print("")
        print("Grid scores on development set:")
        print("")
        print(clf.best_score_)
        print("")

        print("Detailed classification report:")
        print("")
        print("The model is selected with the best specified parameters")
        print("")

        # We test now the model and print classification report and confusion
matrix
        y_true, y_pred = y_test, clf.predict(X_test)
        print(classification_report(y_true, y_pred))
        print("")
```

```
        RFC_cm = cm(y_test, y_pred)

        print(" ")

        print(RFC_cm)
        print(classification_report(y_pred, y_test, labels=None))

        # Show confusion matrix in a separate window
        plt.matshow(RFC_cm)
        plt.title('RFC Confusion matrix')
        plt.colorbar()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()
```

9.3.10. kNeighborsClassifier.py

```
"""
This file is the implementation of basic k-Neighbors Classifier model
Source : https://GitHub.com/mokeam/malware-
analysis/blob/master/scripts/Malware_Classification_Model.py
"""
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix as cm


def kNeighborsClassifier(size_split):
    """
    Method used to call the k-Neighbors Classifier model
    :param size_split: Size of the test dataset
    :return: None
    """
    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')

    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')

    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
```

```
                                                    test_size=size_split,
random_state=42)
    # We initiate it with 3 neighbors
    neigh = KNeighborsClassifier()

    # We train the model
    neigh.fit(X_train, y_train)
    # We test the model
    y_pred = neigh.predict(X_test)

    print("k-Neighbors 3")
    print(" ")

    # We print the report of the model and calculate the confusion_matrix
    print(classification_report(y_pred, y_test, labels=None))
    K_cm = cm(y_test, y_pred)
    print(K_cm)

    # Show confusion matrix in a separate window
    plt.matshow(K_cm)
    plt.title("K-N 3 matrix")
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
    print("")
```

9.3.11. kNeighborsClassfierRandom.py

```
"""
This file is the implementation of Random Search with kNeighbors model
Source :    https://GitHub.com/LiYangHart/Hyperparameter-Optimization-of-
Machine-Learning-Algorithms
            https://GitHub.com/sontung/drebin-malwares
"""
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import confusion_matrix as cm
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt

def kNeighborsClassifierRS(size_split):
    """
    Method used to call the kNeighbors model with Random Search
    :param size_split: Size of the test dataset
    :return: None
```

```
    """

    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)
    # We specify the parameters that we want to test
    # Check here to see the parameters of kNeighbors
    # https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
    rf_params = {
        'n_neighbors': range(1, 20),
    }

    # We specify the scoring that we try to optimize
    # Check here to see which score you can add in the list
    # https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-
parameter
    scores = ["precision", "f1"]

    # We calculate the scoring
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        print("")

        # We create the RandomizedSearch and train the model
        Random = RandomizedSearchCV(KNeighborsClassifier(), rf_params, cv=5, s
coring=score, n_jobs=2)
        Random .fit(X_train, y_train)

        print("Best parameters set found on development set:")
        print("")
        # By default refit is True, then when you call one of those attribute,
 the refit is made with the best
        # parameters.
        print(Random.best_estimator_)
        print(Random.best_params_)
        print("")
        print("Grid scores on development set:")
        print("")
        print(Random.best_score_)
        print("")
```

```
        print("Detailed classification report:")
        print("")
        print("The model is selected with the best specified parameters")
        print("")

        # We test now the model and print classification report and confusion
matrix
        y_true, y_pred = y_test, Random .predict(X_test)
        print(classification_report(y_true, y_pred))

        K_cm = cm(y_test, y_pred)

        print(" ")

        print(K_cm)
        # Show confusion matrix in a separate window
        plt.matshow(K_cm)
        plt.title("K-N 3 matrix")
        plt.colorbar()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()
        print("")
        print("")
```

9.3.12. kNeighborsClassifierTune.py

```
"""
This file is the implementation of GridSearch with kNeighbors Classifier model
Source :    https://GitHub.com/LiYangHart/Hyperparameter-Optimization-of-
Machine-Learning-Algorithms
            https://GitHub.com/sontung/drebin-malwares
"""
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import confusion_matrix as cm
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

def kNeighborsClassifierGS(size_split):
    """
    Method used to call the kNeighbors model with Grid Search
    :param size_split: Size of the test dataset
    :return: None
    """
```

```
    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)
    # We specify the parameters that we want to test
    # Check here to see the parameters of kNeighbors
    # https://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
    rf_params = {
        'n_neighbors': [2, 3, 5, 10, 15, 20]
        #'kernel': ['auto', 'ball_tree', 'kd_tree', 'brute']
    }
    # We specify the scoring that we try to optimize
    # Check here to see which score you can add in the list
    # https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-
parameter
    scores = ["precision", "f1"]

    # We calculate the scoring
    for score in scores:
        print("# Tuning hyper-parameters for %s" % score)
        print("")

        # We create the GridSearch and train the model
        clf = GridSearchCV(KNeighborsClassifier(), rf_params, cv=5, scoring=sc
ore, n_jobs=2)
        clf.fit(X_train, y_train)

        print("Best parameters set found on development set:")
        print("")
        # By default refit is True, then when you call one of those attribute,
 the refit is made with the best
        # parameters.
        print(clf.best_estimator_)
        print(clf.best_params_)
        print("")
        print("Grid scores on development set:")
        print("")
        print(clf.best_score_)
        print("")

        print("Detailed classification report:")
```

```
        print("")
        print("The model is selected with the best specified parameters")
        print("")

        # We test now the model and print classification report and confusion
matrix
        y_true, y_pred = y_test, clf.predict(X_test)
        print(classification_report(y_true, y_pred))

        K_cm = cm(y_test, y_pred)

        print(" ")

        print(K_cm)
        # Show confusion matrix in a separate window
        plt.matshow(K_cm)
        plt.title("K-N matrix")
        plt.colorbar()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()
        print("")
        print("")
```

9.3.13. naivesBayes.py

```
"""
This file is the implementation of basic Naives Bayes model
Source : https://GitHub.com/mokeam/malware-
analysis/blob/master/scripts/Malware_Classification_Model.py
"""
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix as cm
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import pandas as pd


def naivesBayes(size_split):
    """
    Method used to call the Naives Bayes model
    :param size_split: Size of the test dataset
    :return: None
    """
    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
```

```
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)
    # We create the model
    gnb = GaussianNB()
    # We train the model
    gnb.fit(X_train, y_train)

    # We test the model
    y_pred = gnb.predict(X_test)

    print("Naives Bayes")
    print(" ")

    # We print the report of the model and calculate the confusion_matrix
    print(classification_report(y_pred, y_test, labels=None))
    GNB_cm = cm(y_test, y_pred)
    print(GNB_cm)

    # Show confusion matrix in a separate window
    plt.matshow(GNB_cm)
    plt.title('Naive Gauss Confusion matrix')
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

9.3.14. linearRegression.py

```
"""
This file is the implementation of basic Linear Regression model
Source : https://GitHub.com/mokeam/malware-
analysis/blob/master/scripts/Malware_Classification_Model.py
"""
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression as LG
from sklearn.metrics import confusion_matrix as cm
import pandas as pd
import matplotlib.pyplot as plt
```

```
def linearRegression(size_split):
    """
    Method used to call the Linear Regression Model
    :param size_split: Size of the test dataset
    :return: None
    """
    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)

    # We create and test the model
    clf = LG().fit(X_train, y_train)

    # We now test the model
    y_pred = clf.predict(X_test)

    # We print the report of the model and calculate the confusion_matrix
    print("Linear Regression")
    print(" ")
    LG_cm = cm(y_test, y_pred)
    print(classification_report(y_pred, y_test, labels=None))
    print(LG_cm)

    plt.matshow(LG_cm)
    plt.title('LG Confusion matrix')
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

9.3.15. decisionTree.py

```
"""
This file is the implementation of basic Decision Tree model
Source : https://GitHub.com/mokeam/malware-
analysis/blob/master/scripts/Malware_Classification_Model.py
"""
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn import tree
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix as cm

def decisionTree(size_split):
    """
    Method used to call the Decision Tree model
    :param size_split: Size of the test dataset
    :return: None
    """
    # Read both files and merge them
    df = pd.read_csv('binaryApps_BEN.csv', sep=',')
    df2 = pd.read_csv('binaryApps_MAL.csv', sep=',')
    df = pd.concat([df, df2])

    # We create the training and validation training dataset and the test and
validation test dataset
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:-
1], df['malware'],
                                                        test_size=size_split,
random_state=42)
    # We create and train the model
    clf = tree.DecisionTreeClassifier()
    clf.fit(X_train, y_train)

    # We now test the model
    y_pred = clf.predict(X_test)

    print("Decistion Tree")
    print(" ")

    # We print the report of the model and calculate the confusion_matrix
    print(classification_report(y_pred, y_test, labels=None))
    DTC_cm = cm(y_test, y_pred)
    print(DTC_cm)

    # Show confusion matrix in a separate window
    plt.matshow(DTC_cm)
    plt.title('Decistion Tree matrix')
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

9.3.16. Comparaison applications bénignes et malicieuses

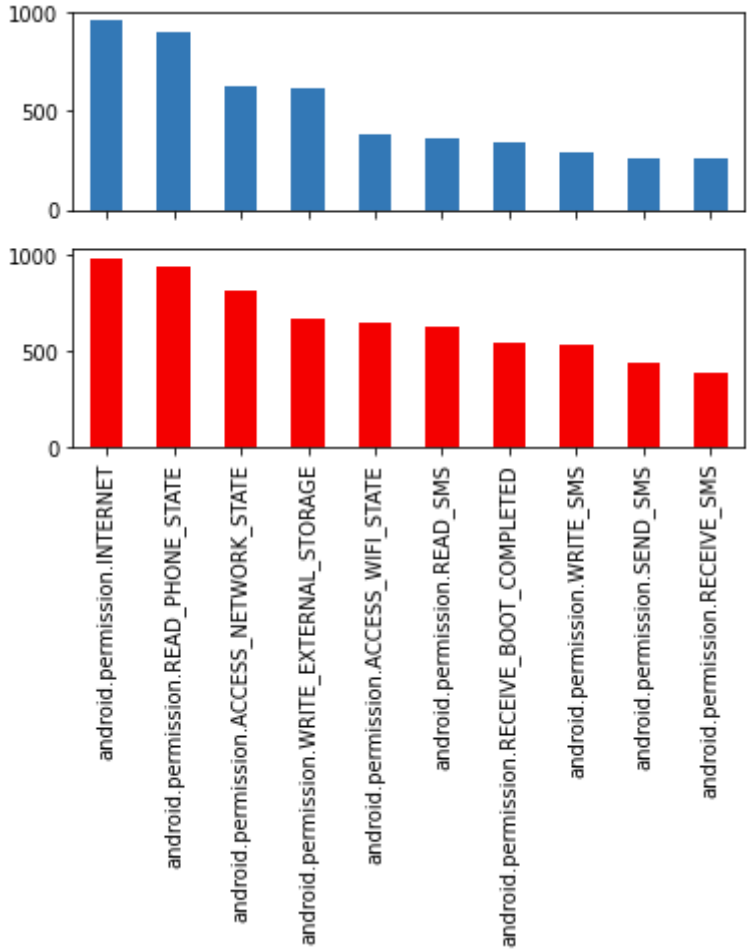


Figure 33 Tableau comparative entre les droits les plus utilisés par les applications malicieuses et la proportion de celle-ci dans les applications bénignes

9.3.17. Résultats des prédictions

HPO 0

```
SVC

              precision    recall  f1-score   support

          -1       0.99      0.60      0.74       320
           1       0.38      0.97      0.54        80

   micro avg       0.67      0.67      0.67       400
   macro avg       0.68      0.79      0.64       400
weighted avg       0.87      0.67      0.70       400

[[191   2]
 [129  78]]
```

```
k-Neighbors 3
```

```
              precision    recall  f1-score   support

          -1       0.98      0.96      0.97       197
           1       0.96      0.98      0.97       203

   micro avg       0.97      0.97      0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400

[[189   4]
 [  8 199]]
```

```
Random Forest

              precision    recall  f1-score   support

          -1       0.97      0.94      0.96       198
           1       0.95      0.97      0.96       202

   micro avg       0.96      0.96      0.96       400
   macro avg       0.96      0.96      0.96       400
weighted avg       0.96      0.96      0.96       400

[[187   6]
 [ 11 196]]
```

```
Linear Regression

              precision    recall  f1-score   support

          -1       0.96      0.96      0.96       192
           1       0.97      0.96      0.96       208

   micro avg       0.96      0.96      0.96       400
   macro avg       0.96      0.96      0.96       400
weighted avg       0.96      0.96      0.96       400

[[185   8]
 [  7 200]]
```

```
Naives Bayes

              precision    recall  f1-score   support

          -1       0.22      0.98      0.36        44
           1       1.00      0.58      0.73       356
```

```
   micro avg       0.62      0.62      0.62       400
   macro avg       0.61      0.78      0.55       400
weighted avg       0.91      0.62      0.69       400

[[ 43 150]
 [  1 206]]
```

```
Decistion Tree

              precision    recall  f1-score   support

          -1       0.95      0.94      0.95       194
           1       0.95      0.95      0.95       206

   micro avg       0.95      0.95      0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.95      0.95      0.95       400

[[183  10]
 [ 11 196]]
```

HPO 1

```
# Tuning hyper-parameters for accuracy

Best parameters set found on development set:

SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
{'kernel': 'rbf', 'C': 100, 'gamma': 0.01}

Grid scores on development set:

0.9636591478696742

Detailed classification report:

The model is selected with the best specified parameters

              precision    recall  f1-score   support

          -1       0.96      0.96      0.96       193
           1       0.97      0.96      0.96       207

   micro avg       0.96      0.96      0.96       400
   macro avg       0.96      0.96      0.96       400
weighted avg       0.96      0.96      0.96       400
```

```
# Tuning hyper-parameters for precision

Best parameters set found on development set:

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=2, p=2,
           weights='uniform')
{'n_neighbors': 2}

Grid scores on development set:

0.9878579985917318

Detailed classification report:

The model is selected with the best specified parameters

              precision    recall  f1-score   support

          -1       0.95      0.99      0.97       193
           1       0.99      0.95      0.97       207

   micro avg       0.97      0.97      0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400


[[191   2]
 [ 10 197]]
```

```
# Tuning hyper-parameters for f1

Best parameters set found on development set:

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=22, max_features=63, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=7,
            min_weight_fraction_leaf=0.0, n_estimators=74, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
{'min_samples_leaf': 1, 'n_estimators': 74, 'min_samples_split': 7, 'criterion
': 'gini', 'max_features': 63, 'max_depth': 22}

Grid scores on development set:
```

```
0.964936205036265

Detailed classification report:

The model is selected with the best specified parameters

              precision    recall  f1-score   support

          -1       0.96      0.97      0.97       193
           1       0.97      0.97      0.97       207

   micro avg       0.97      0.97      0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400



[[187   6]
 [  7 200]]
```

HPO 2

```
# Tuning hyper-parameters for accuracy

Best parameters set found on development set:

SVC(C=14, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
{'kernel': 'rbf', 'C': 14, 'gamma': 0.01}

Grid scores on development set:

0.9586466165413534

Detailed classification report:

The model is selected with the best specified parameters

              precision    recall  f1-score   support

          -1       0.96      0.97      0.96       193
           1       0.97      0.96      0.97       207

   micro avg       0.96      0.96      0.96       400
   macro avg       0.96      0.97      0.96       400
weighted avg       0.97      0.96      0.97       400
# Tuning hyper-parameters for precision
```

```
Best parameters set found on development set:

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=4, p=2,
           weights='uniform')
{'n_neighbors': 4}

Grid scores on development set:

0.9826718901186826

Detailed classification report:

The model is selected with the best specified parameters

              precision    recall  f1-score   support

          -1       0.95      0.98      0.97       193
           1       0.99      0.96      0.97       207

   micro avg       0.97      0.97      0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400


[[190   3]
 [  9 198]]
```

```
# Tuning hyper-parameters for accuracy

Best parameters set found on development set:

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=34, max_features=6, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=73, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
{'min_samples_leaf': 1, 'n_estimators': 73, 'min_samples_split': 2, 'criterion
': 'entropy', 'max_features': 6, 'max_depth': 34}

Grid scores on development set:

0.9699248120300752

Detailed classification report:
```

```
The model is selected with the best specified parameters

              precision    recall  f1-score   support

          -1       0.95      0.99      0.97       193
           1       0.99      0.96      0.97       207

   micro avg       0.97      0.97      0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400



[[191   2]
 [  9 198]]
```

9.3.18. Planification initiale

Tâche	Date (semaine)	Nombre d'heure
Lecture d'étude	26.02.2021	8
Lecture d'étude	05.03.2021	12
Discussion et spécialisation du sujet	12.03.2021	12
Analyse du marché	19.03.2021	8
Analyse du marché	26.03.2021	8
Test des outils de forensique gratuit	02.04.2021	8
Lecture d'étude	09.04.2021	8
Implémenter des premiers petits programmes de machine learning	16.04.2021	12
Implémenter des premiers petits programmes de machine learning	23.04.2021	12
Implémenter des premiers petits programmes de machine learning	30.04.2021	12
Tenter d'implémenter un outil permettant de détecter des malwares	07.05.2021	12
Tenter d'implémenter un outil permettant de détecter des malwares	14.05.2021	12
Tenter d'implémenter un outil permettant de détecter des malwares et documenter	21.05.2021	12
Tenter d'implémenter un outil permettant de détecter des malwares et documenter	28.05.2021	22
Tenter d'implémenter un outil permettant de détecter des malwares et documenter	04.06.2021	22
	Total	180

9.3.19. Planification finale

Tâche	Date (semaine)	Nombre d'heure
Lecture d'étude	26.02.2021	8
Lecture d'étude	05.03.2021	12
Discussion et spécialisation du sujet, malware	12.03.2021	14
Analyse du marché	19.03.2021	16
Analyse du marché et discussion avec collègue	26.03.2021	6
Test des outils de forensique gratuit	02.04.2021	20
Malade une semaine complète	09.04.2021	0
Implémenter des premiers petits programmes de machine learning	16.04.2021	0
Implémenter des premiers petits programmes de machine learning	23.04.2021	0
Implémenter des premiers petits programmes de deep learning, rapport	30.04.2021	14
Tenter d'implémenter un outil permettant de détecter des malwares, rapport , lecture d'étude	07.05.2021	14
Tenter d'implémenter un outil permettant de détecter des malwares et documenter	14.05.2021	12
Tenter d'implémenter un outil permettant de détecter des malwares et documenter	21.05.2021	25
Tenter d'implémenter un outil permettant de détecter des malwares et documenter	28.05.2021	35
Tenter d'implémenter un outil permettant de détecter des malwares et documenter	04.06.2021	50
	Total	218