**PROJET DE SESSION AUTOMNE 2024**

**CYB6043**



Analyse de la mémoire des logiciels malveillants CIC-MalMem-2022

**Présenté par: « The Wolves »**

**Kamguia Kouam Eric Stephane**

**Pignandi Pappus Tchinga**

**Fadima Dioubate**

**Zoungrana Abdoul Fayçal**

**Professeurs :**

**Allili Mohand Said**

**Khoury Raphël**

# Table des matières

[Table des matières 1](#_Toc1094244993)

[Liste des Captures et Tableaux 2](#_Toc797567001)

[Introduction 2](#_Toc856293010)

[I. Présentation du jeu de données 3](#_Toc1025503659)

[II. Entraînement de Notre Système ML 5](#_Toc798028264)

[1. Prétraitement 6](#_Toc530142780)

[2. Éléments caractéristiques du choix de l’Algorithme 7](#_Toc1373195250)

[III. Evaluation des models 10](#_Toc1346513536)

[1. Classification Binaire 11](#_Toc1037943398)

[2. Classification Multi-classe 12](#_Toc1543082367)

[3. Analyse des résultats : Courbe ROC-AUC (Classification multi-classe) 13](#_Toc1082369358)

[IV. Test du model 14](#_Toc1507476924)

[1. Présentation des outils utilisés 14](#_Toc700833625)

[2. Extraction des caractéristiques des enregistrements 18](#_Toc1457021980)

[3. Prédiction 20](#_Toc180583164)

[Conclusions 21](#_Toc249404426)

[Références 22](#_Toc1992092481)

# Liste des Captures et Tableaux

[Capture1 : Présentation du jeu données 6](#_Toc185261139)

[Tableau 1 : groupe de caractéristique du jeu de données 6](#_Toc185261140)

[Capture2 : Organigramme du schéma de travail 8](#_Toc185261141)

[Capture3 : illustration de l’encodage de la classe cible 9](#_Toc185261142)

[Capture4 : matrice de choix des des algorithmes de scikit-learn 10](#_Toc185261143)

[Capture 5: Matrice de corrélation 12](#_Toc185261144)

[Tableau 2 : Comparaison des différents modèles (Classification binaire) 13](#_Toc185261145)

[Capture 6 : Histogramme de Comparaison des différents modèles (Classification binaire) 14](#_Toc185261146)

[Tableau 3 : Comparaison des différents modèles (Classification multi-classe) 14](#_Toc185261147)

[Capture 7 : Histogramme de Comparaison des différents modèles (Classification multi-classe) 15](#_Toc185261148)

[Capture 7 : Courbe ROC des différents modèle (focus mis sur la classification multi-classe) 16](#_Toc185261149)

[Capture 7 : VMware Fusion 13.6.1 17](#_Toc185261150)

[Capture 8: Windows 10 64 bits 18](#_Toc185261151)

[Capture 9: FTK Imager, exemple de capture mémoire 18](#_Toc185261152)

[Capture 12: Information de base de la capture mémoire 21](#_Toc185261153)

[Tableau 04: illustration des caractéristiques extraite sur les 02 logiciels bienveillants et le logiciel malveillant. 22](#_Toc185261154)

[Capture 14: prédiction avec l’algorithme forêt aléatoire 23](#_Toc185261155)

[Capture 15: prédiction avec l’algorithme le Plus Proche Voisin 23](#_Toc185261156)

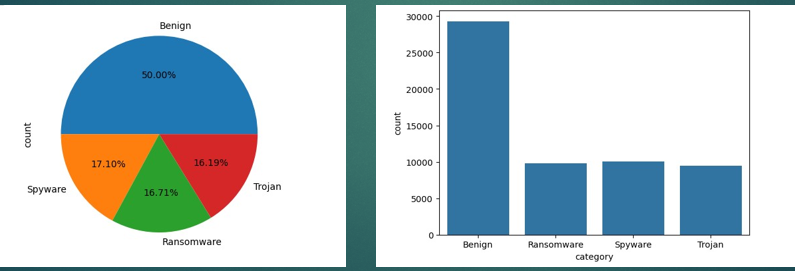
# Introduction

Dans le domaine de la cybersécurité, la détection et la classification des logiciels malveillants sont des enjeux majeurs pour la protection des systèmes informatiques. Le jeu de données CIC-MalMem-2022, spécifiquement conçu pour l'analyse des logiciels malveillants opérant en mémoire, offre une base solide pour développer des solutions de détection automatisée. Ce projet vise à exploiter ce jeu de donnée afin de créer un modèle de d’apprentissage machine capable de différencier les logiciels malveillants des logiciels bienveillants avec une grande précision. En appliquant plusieurs techniques d'apprentissage automatique et en analysant leurs performances, les objectifs sont d’identifier l'algorithme qui maximise les métriques clés, telles que la précision, le rappel, le score F1 et de le déployer dans un environnement local pour en déduire la matérialité du modèle développé. Cette approche permettra de renforcer la sécurité des systèmes en automatisant la détection des menaces potentielles, tout en optimisant la performance du modèle pour réduire les erreurs de classification.

# Présentation du jeu de données

Le jeu de donnée CIC-MalMem-2022 contient une répartition égale de fichiers bénins et malveillants, soit 50% de chaque type.

* **Objectif principal** : Analyser et détecter les malwares en mémoire.
* **Composition** :
  + **50% de fichiers bénins**
  + **50% de fichiers malveillants (Ramsomware, Spyware, Trojan)**



#### Capture1 : Présentation du jeu données

Le jeu de donnée a 58,596 enregistrements et 57 caractéristiques, chacune représentant une variable différente pour chaque instance.

Une image contenant texte, Police, capture d’écran, blanc

Description générée automatiquement

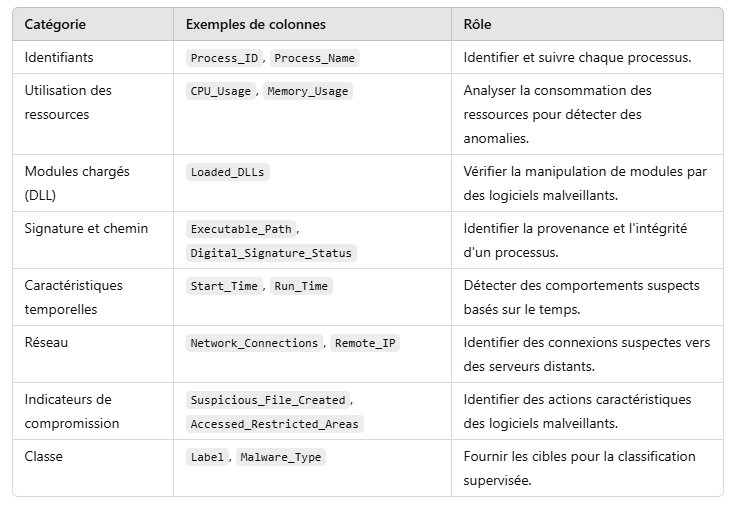
**Caractéristiques essentielles**

**Process\_ID :** Permet d’identifier chaque processus.

**CPU\_Usage :** Indique la consommation de ressources par un processus, souvent plus élevée pour les logiciels malveillants.

**Loaded\_DLLs :** Liste les modules (DLL) chargés par le processus, utile pour repérer les injections malveillantes.

**Label :** Spécifie si le processus est bienveillant (0) ou malveillant (1)."



#### Tableau 1 : groupe de caractéristique du jeu de données

# Entraînement de Notre Système ML

Le graphe si dessous illustre le flux de travail global pour le développement de notre système d’apprentissage machine (ML), comprenant les étapes suivantes :

1. **Prétraitement** : Le dataset est d'abord standardisé. Ensuite, il est divisé en ensembles d'entraînement (*TrainSet*) et de test (*TestSet*).
2. **Rééchantillonnage avec SMOTEENN** : Pour équilibrer les classes dans *TrainSet*, la méthode SMOTEENN est utilisée. Cela combine l’augmentation des classes minoritaires (SMOTE) et la suppression des échantillons bruyants (ENN).
3. **Entraînement des modèles** : Plusieurs algorithmes sont testés, incluant :
   1. RNN (Réseaux de Neurones Séquentiels),
   2. KNN,
   3. Random Forest,
   4. Naive Bayes (Complément).
4. **Optimisation avec AdaBoost** : Le modèle avec les meilleures performances est sélectionné, et son efficacité est améliorée en utilisant AdaBoost pour ajuster les prédictions.
5. **Validation des modèles** : Les modèles entraînés sont testés sur l’ensemble de test pour évaluer leurs performances.
6. **Déploiement** : Le modèle final, après validation, est déployé. Il est ensuite utilisé pour faire des prédictions sur de nouveaux échantillons (*Dataset Sample*).

Ce graphe met en évidence un pipeline structuré et complet pour gérer un problème de classification en ML, incluant des techniques avancées pour améliorer l’équilibre des données et optimiser les performances des modèles.

Une image contenant texte, diagramme, Plan, Dessin technique

Description générée automatiquement

#### Capture2 : Organigramme du schéma de travail

## Prétraitement

Le prétraitement est une étape essentielle dans la construction de tout système d’apprentissage machine. Voici les différentes étapes suivies pour préparer les données :

- Chargement des données : Les données ont été importées à partir d'un fichier CSV contenant des informations sur des échantillons de malwares.

- Transformation des colonnes : La colonne `Category`, contenant des valeurs textuelles, a été transformée. Une nouvelle colonne `Category\_v0` a été créée pour extraire une valeur plus générale de la catégorie en utilisant la fonction `find\_category`.

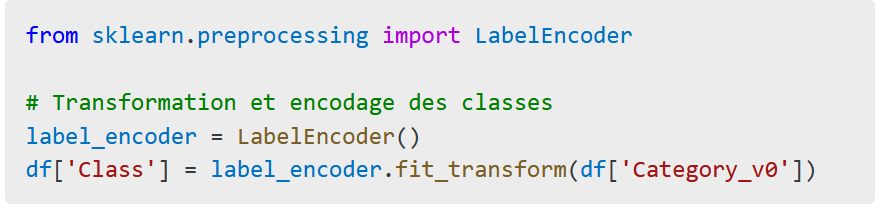
- Encodage des classes : La colonne `Category\_v0` a été encodée en valeurs numériques à l'aide de `LabelEncoder` pour faciliter la classification. La correspondance entre les valeurs originales et les classes encodées a été enregistrée pour interprétation ultérieure.

- Suppression des colonnes inutiles : Les colonnes `Category` et `Category\_v0`, ainsi que les colonnes ayant des valeurs constantes (ne fournissant aucune information utile pour l'entraînement), ont été supprimées.

- Traitement des valeurs manquantes : Une analyse des valeurs manquantes a été effectuée via une carte thermique pour identifier les colonnes contenant des lacunes.

- Visualisation des distributions : La distribution des colonnes numériques a été explorée à l’aide de graphiques de densité et d’histogrammes, permettant d’évaluer la répartition des valeurs et de repérer éventuellement des anomalies.

Exemple de code : Encodage des classes



#### Capture3 : illustration de l’encodage de la classe cible

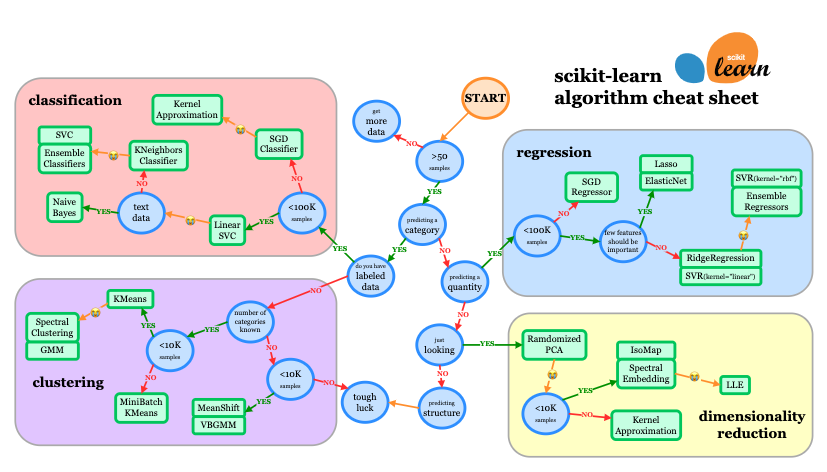
Ces étapes ont permis d’obtenir un jeu de données propre et structuré, prêt pour l'entraînement.

## Éléments caractéristiques du choix de l’Algorithme

Dans cette partie nous présenterons les caractéristiques des algorithmes qui nous ont pousser à les choisir pour cela nous commencerons par utiliser scikit-learn algorithm cheat sheet dans notre processus de travail.

* **Choix éclairé** : Ce graphe offre un guide structuré pour sélectionner des algorithmes en fonction de la taille des données, du type de problème et des besoins spécifiques (précision, vitesse, ou interprétabilité).
* **Exploration rapide** : Il permet de tester rapidement plusieurs algorithmes pour comparer leurs performances, en validant les meilleurs choix par des métriques comme le F1-score ou la courbe ROC.
* **Adaptabilité** : Il montre que les algorithmes simples (comme Naive Bayes) peuvent être un bon point de départ avant d’explorer des modèles plus complexes comme Random Forest ou RNN.

En résumé, le graphe peut servir de référence pour structurer l’expérimentation et rationaliser le choix des algorithmes en fonction des contraintes et objectifs de votre projet. Si besoin, je peux inclure cette analyse dans votre rapport.



#### Capture4 : matrice de choix des des algorithmes de scikit-learn

Le choix de l'algorithme repose sur plusieurs autres critères :

- **Nature du problème** : Le problème abordé ici est une classification multi-classes, où chaque échantillon doit être associé à une seule catégorie parmi plusieurs.

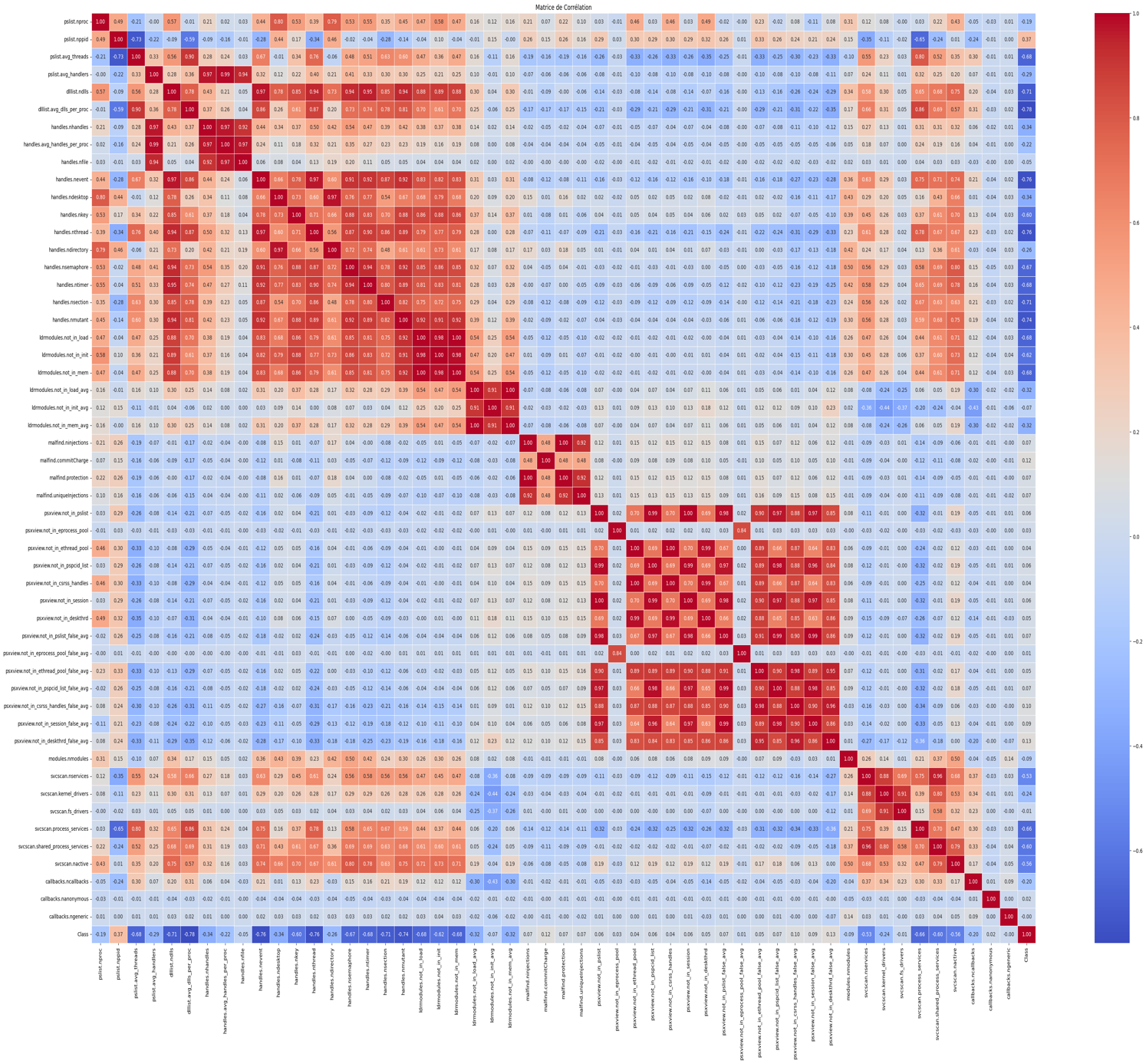
- **Distribution des données** : Après analyse, les données présentent un certain déséquilibre entre les classes. Cela a orienté le choix vers des algorithmes robustes au déséquilibre ou compatibles avec des techniques de rééchantillonnage, comme SMOTE.

- **Complexité des relations entre variables** : Les corrélations entre les variables ont été étudiées via une matrice de corrélation. Les relations complexes entre les caractéristiques suggèrent l'utilisation de modèles non linéaires tels que Random Forest ou les réseaux neuronaux.

- **Exigences en temps d'exécution** : Une analyse préalable a permis de prendre en compte le temps d'entraînement et d'inférence des modèles, favorisant un compromis entre précision et efficacité.

- **Complexité des relations entre variables :** Les corrélations entre les variables ont été étudiées via une matrice de corrélation. Les relations complexes entre les caractéristiques suggèrent l'utilisation de modèles non linéaires tels que Random Forest ou les réseaux neuronaux.

- Exigences en temps d'exécution : Une analyse préalable a permis de prendre en compte le temps d'entraînement et d'inférence des modèles, favorisant un compromis entre précision et efficacité.



#### Capture 5: Matrice de corrélation

Une matrice de corrélation a été générée pour explorer les relations entre les différentes caractéristiques, aidant à sélectionner les variables les plus influentes.

Ces analyses ont permis de choisir un modèle adapté, garantissant une performance optimale sur ce jeu de données.

# Evaluation des models

L’évaluation des performances des modèles a été réalisée pour deux types de tâches de classification à savoir :

* **Classification binaire** où l’objectif est de différencier deux classes, représentées par 0 et 1.
* **Classification multiclass** où l’objectif est de distinguer plusieurs classes simultanément (quatre classes dans ce cas).

Ces deux types de classification nous ont permis de mesurer la capacité des modèles à généraliser sur des tâches simples (binaire) et complexes (multiclass). L’analyse a porté sur plusieurs modèles que sont : Réseau **de Neurones, Régression Logistique, KPPV, Forêt Aléatoire** et **Classificateur de Bayes**.

Nous avons mis un accent particulier sur la **classification multiclass**, car elle représente un défi plus complexe et révèle les performances réelles des modèles sur un problème de classification varié. Voici les résultats comparatifs pour chacune de ces tâches de classification :

## Classification Binaire

Les résultats obtenus pour les cinq modèles sont présentés ci-dessous :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

#### Tableau 2 : Comparaison des différents modèles (Classification binaire)

Une image contenant texte, capture d’écran, ligne, Caractère coloré

Description générée automatiquement

#### Capture 6 : Histogramme de Comparaison des différents modèles (Classification binaire)

**Forêt Aléatoire, Régression Logistique** et **KPPV** ont atteint une performance parfaite avec **100 %** de précision, de rappel et de F1-Score. Le **Réseau de Neurones** a également obtenu un excellent résultat avec une **accuracy** de **99.79 %**. En revanche, le **Classificateur de Bayes** a montré des limites significatives avec une précision de seulement **54 %**, ce qui indique une difficulté à classer correctement les données.

## Classification Multi-classe

Pour la classification multi-classe, nous avons évalués les modèles sur quatre classes. Voici un résumé des résultats :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

#### Tableau 3 : Comparaison des différents modèles (Classification multi-classe)

Une image contenant texte, capture d’écran, ligne, Caractère coloré

Description générée automatiquement

#### Capture 7 : Histogramme de Comparaison des différents modèles (Classification multi-classe)

Le **modèle Forêt Aléatoire** se distingue comme le plus performant, avec une **accuracy** de **96 %** et des métriques cohérentes (précision, rappel, F1-Score). Le **KPPV** suit avec une performance solide à **89 %**. Les **Réseaux de Neurones** et **Régression Logistique** montrent des résultats moyens, tandis que le **Classificateur de Bayes** présente des performances nettement inférieures (47 %).

## Analyse des résultats : Courbe ROC-AUC (Classification multi-classe)

L’accent étant mis sur la **classification multiclass**, et la **courbe ROC-AUC** du modèle **Forêt Aléatoire** a été tracée pour illustrer ses performances.

* **Interprétation de la courbe ROC** :

**Classe 0** : AUC = 1.0000 : Classification parfaite.

**Classe 1** : AUC = 0.9694 : Excellente performance avec un léger risque de confusion.

**Classe 2** : AUC = 0.9746 : Très bonne capacité de discrimination.

**Classe 3** : AUC = 0.9660 : Performances solides, bien que légèrement inférieures.

**Micro-average AUC = 0.9775 :** Cela confirme la **robustesse globale** du modèle Forêt Aléatoire sur la classification multiclass.

Une image contenant texte, capture d’écran, ligne, diagramme

Description générée automatiquement

#### Capture 7 : Courbe ROC des différents modèle (focus mis sur la classification multi-classe)

Toutes les courbes sont proches du coin supérieur gauche, ce qui indique une **capacité élevée de discrimination** entre les classes. La moyenne micro-AUC de 0.9775 démontre que le modèle est **bien adapté** pour la tâche multi-classe."

Les résultats montrent que le **modèle Forêt Aléatoire** surpasse largement les autres modèles pour les deux types de classification.

# Test du model

Une fois que nous avons terminé l’entrainement de notre modèle et que nous avons identifié le meilleur algorithme qui est adapté pour la classification de notre jeu de données, nous avons décidé de mettre en place un environnement de test. Cet environnement de test a été réalisé tout en tenant compte des conditions dans lesquelles notre jeu de donnée a été constitué.

## Présentation des outils utilisés

Pour effectuer le test, nous avons procédé par l’extraction des mêmes caractéristiques que nous avions dans le jeu de données. Nous avons effectué également un enregistrement de notre meilleur modèle en « joblib » que nous appellerons lors de l’exécution du test.

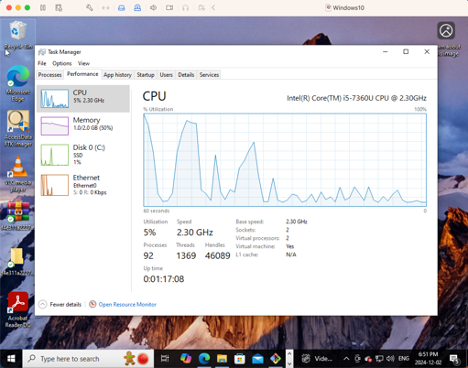
Nous avons utilisé essentiellement deux outils pour obtenir exactement les mêmes caractéristiques que nous avons dans le jeu de données d’entrainement à Savoir FTK Imager et Volatility. Étant donné que nous effectuons une détection des logiciels malveillants en mémoire, nous devions exécuter les logiciels malveillants afin d’observer leur comportement en mémoire et récupérer leurs caractéristiques par la même occasion.



#### Capture 7 : VMware Fusion 13.6.1

* Installation de Windows 10 64 bits

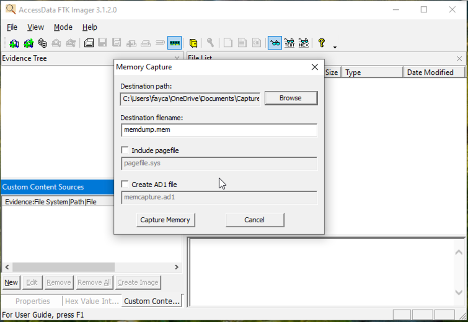
Ainsi, dans un premier temps, nous avons mis en place l’environnement qui nous permettra d’exécuter les malwares en Installant sous VMWare Fusion 13.6.1 une machine virtuelle sous Windows 10 64bits. Nous avons installé Windows car l’extraction des caractéristiques notre jeu de données a été effectué sous le système d’exploitation Windows 10 64 bits. Nous avons utilisé 2Giga de mémoire de RAM et un disque dur de 60 Giga.



#### Capture 8: Windows 10 64 bits

* FTK Imager 3.1.20

Etant donné que nous souhaitons analyser le comportement du logiciel malveillant une fois qu’il a été exécuté sur l’ordinateur, nous avons utilisé FTK Imager qui est un outil permettant de faire une capture mémoire de notre ordinateur. Ainsi, une fois la capture effectué, toutes taches que nous avions en exécution pendant la capture sont sauvegarder en « .mem »



#### Capture 9: FTK Imager, exemple de capture mémoire

* Volatility3-2.8.0

Après avoir obtenu la capture mémoire, il faudrait maintenant l’analyser afin d’extraire les caractéristiques dont nous aurons besoins pour effectuer le test avec le modèle. De ce fait, Volatility est l’outil que nous utiliserons pour extraire exactement les mêmes caractéristiques sur nos logiciels de test afin de les tester ensuite avec notre modèle que nous avons entrainé. Volatility permet d’analyser les captures mémoires effectué sous différents systèmes d’exploitation à savoir MAC OS, Linux, Windows. Ainsi, il intègre des symboles qui nous permettrons d’extraire les caractéristiques qui ont été utilisé dans notre jeu de données.

Par ailleurs, il faudrait noter qu’il y’a un certain nombre de dépendance qu’il installer au préalable sur l’ordinateur avec de pouvoir utiliser volatility. Vous trouverez ci-dessous les dépendances qu’il faudrait installer.

Télécharger et Installer Microsoft Visual Studio

<https://visualstudio.microsoft.com/downloads/?q=build+tools>

Télécharger et installer python

<https://www.python.org/downloads/>

Télécharger et installer Git ( optionnel )

Télécharger le dossier de volatility3 2.8.0

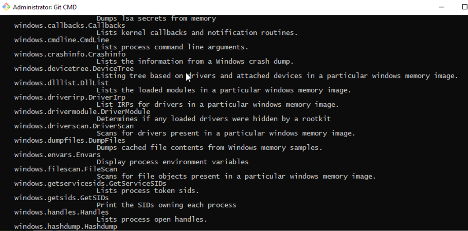
<https://pypi.org/project/volatility3/>

Télécharger les Symbole Windows et le déplacer dans le Dossier Symboles dans volatility3

<https://downloads.volatilityfoundation.org/volatility3/symbols/windows.zip>

Puis se placer dans le dossier volatility3

pip install -r requirements.txt

Capture 10: Plug-in Windows sous Volatility3

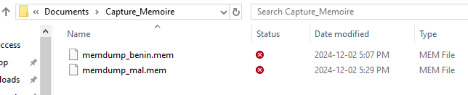
## Extraction des caractéristiques des enregistrements

Une fois que nous avons préparé l’environnement qui nous permettra d’obtenir les caractéristiques, nous procéderons à présent à l’extractions des caractéristiques des logiciels que nous testerons. Dans notre cas nous tester deux logiciel bienveillants et un logiciel malveillant.

Pour les logiciels bienveillants que nous testé, il s’agit de Wordpad et de Vlc que nous avons considéré pour le test. En ce qui concerne le logiciel malveillant qui a été testé il s’agit d’un malware que nous avons télécharger sur **MalwareBazaar.**

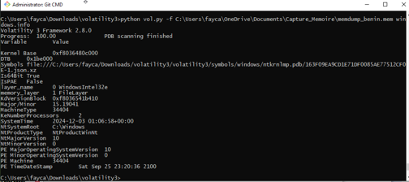
* Pour effectuer la capture, nous avons d’abord exécuter les logiciels bienveillants sans le malware puis nous avons effectué la capture avec FTK Imager.

Ensuite, nous avons exécuter le malware et avons effectuer ensuite une autre capture mémoire.

Capture 11: Capture mémoire de logiciels bienveillants et malveillant

* Après avoir effectué la capture, nous passons à présent à l’extraction des caractéristiques sous volatility.

Nous vérifions d’abord les informations de base du fichier mem. Ainsi, nous avons le type de Système d’exploitation sur lequel la capture a été effectué, le nombre de bit, l’heure de la capture, le type de processeur ainsi que le nombre.



Capture 12: Information de base de la capture mémoire

Les plug in Windows qui ont été utilisé pour constituer le jeu de donnés sont au nombre de 09 à savoir :

windows.callbacks.Callbacks

windows.dlllist.DllList

windows.handles.Handles

windows.ldrmodules.LdrModules

windows.malfind.Malfind

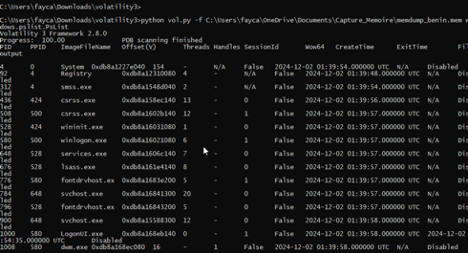
windows.modules.Modules

windows.pslist.PsList

windows.psxview.PsXView

windows.svcscan.SvcScan

Pour chacun nous avons extrait les caractéristiques pour la capture des logiciels bienveillants et pour la capture du logiciel malveillant.

Capture 13: affichage des plug ins windows disponible sous volatility

Après avoir extraite les caractéristiques, nous avons effectué un tri pour avoir les caractéristiques sous le même format que nous jeu de donnée afin de pour l’analyser avec le modèle que nous avons entrainé.

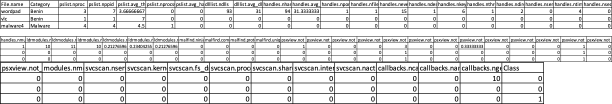


Tableau 04: illustration des caractéristiques extraite sur les 02 logiciels bienveillants et le logiciel malveillant.

## Prédiction

Pour effectuer la prédiction, nous avons utilisé notre modèle que nous avons enregistrer en joblib. Ensuite, nous avons chargé notre jeu de données de test qui a été formaté avec les mêmes caractéristiques.

Nous avons constaté qu’avec notre meilleur modèle c’est-à-dire la forêt aléatoire, le modèle prédit que toutes les entrées sont des logiciels malveillants y compris le logiciel malveillant.



Capture 14: prédiction avec l’algorithme forêt aléatoire

Puis nous avons effectué le test avec un autre modèle, le plus proche voisin qui n’était pas notre meilleur modèle et lui identifie toutes les entrées comme étant des logiciels bienveillants



Capture 15: prédiction avec l’algorithme le Plus Proche Voisin

De ces résultats, nous pensons que le modèle ne parvient pas à faire la classification avec des logiciels récent et cela peut être dû à différentes raison entre autres le fait que nous devons allouer plus de ressource à notre machine virtuelle ou entrainer nos modèles en incluant des caractéristiques de logiciels malveillant récent.

# Conclusions

En définitive, notre étude, basée sur le dataset CIC-MalMem-2022, a permis de développer des modèles de machine learning capables de détecter efficacement les logiciels malveillants, aussi bien en classification binaire qu'en multiclasses. Ces résultats constituent une base solide pour renforcer les systèmes de sécurité en automatisant la détection des menaces.

Des améliorations peuvent être envisagées, notamment l'optimisation des hyperparamètres à l'aide de techniques telles que la recherche par grille ou les méthodes bayésiennes, ainsi que l'exploration de modèles supplémentaires comme XGBoost ou LightGBM, largement reconnus pour leur efficacité dans des contextes similaires. Cependant, certaines limites subsistent à savoir le déséquilibre des classes reste un défi malgré l’utilisation de techniques telles que SMOTEEN, et le temps d’entraînement des modèles pourrait être optimisé grâce à des solutions parallélisées ou à une infrastructure matérielle plus performante. À l'avenir, l'intégration de ces modèles dans des environnements en temps réel pourrait transformer la lutte contre les cyberattaques, améliorant significativement la réactivité face aux menaces et réduisant les risques associés. Ces perspectives offrent un cadre prometteur pour renforcer la robustesse des systèmes de sécurité et leur adaptation aux défis constants du paysage cybernétique.

# Références

1. Kanaker, H., Abdel Karim, N., Awwad, S. A. B., Ismail, N. H. A., Zraqou, J., & Al Ali, A. M. F. Trojan Horse Infection Detection in Cloud-Based Environment Using Machine Learning. International Journal of Interactive Mobile Technologies, 2024
2. Kamalakanta Sethi, Shankar Kumar Chaudhary. Indian Institute of Technology Bhubaneswar. A Novel Malware Analysis for Malware Detection and Classification using Machine Learning Algorithms
3. <https://www.kaspersky.fr/resource-center/threats/trojans> Consulté le 20/11/2024 à 12 :24 pm
4. <https://scikit-learn.org/stable/> multiples consultation du 12/10/2024 au 02/12/2024