Modèle de detection de malware basé sur Multi Layer Perceptron

Amine MIRHOM

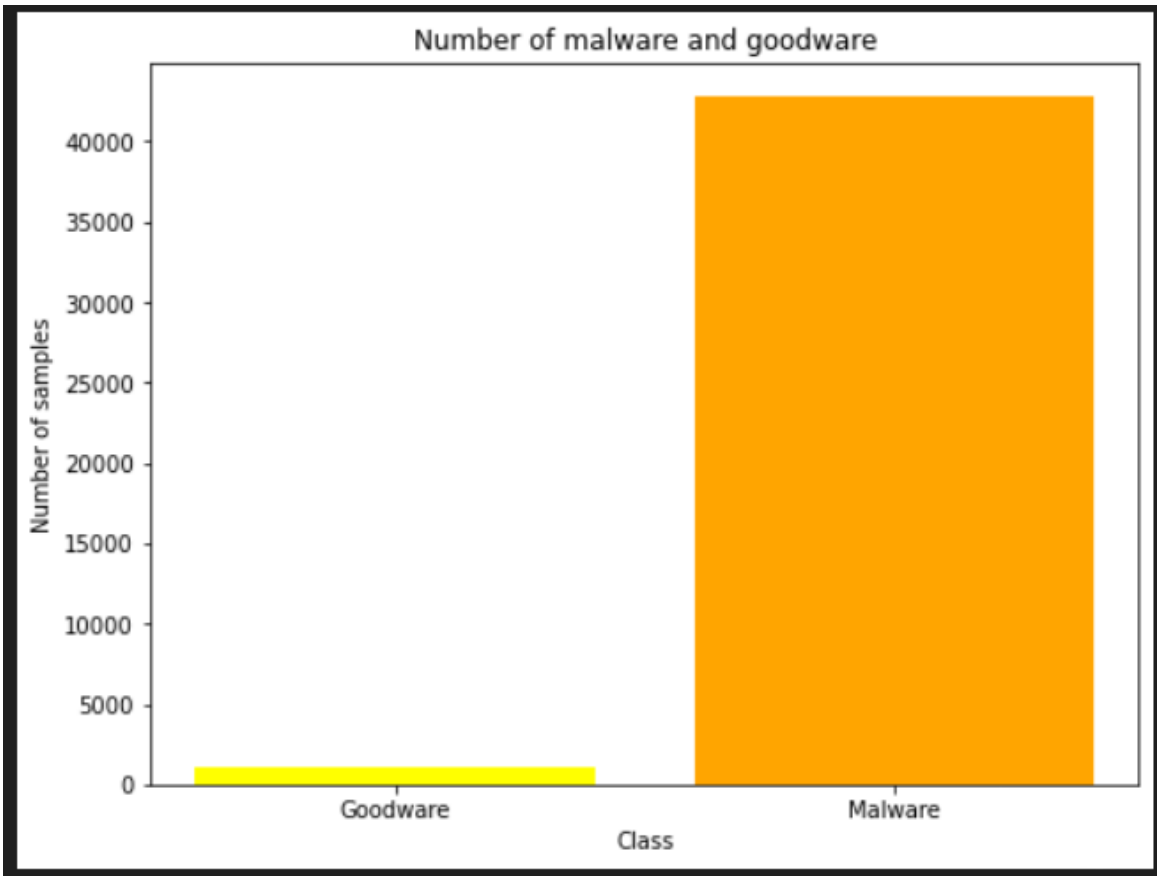
28 Juin 2023

1 Introduction

Dans le monde numérique d'aujourd'hui, la sécurité des systèmes informatiques est devenue une préoccupation majeure. Les logiciels malveillants, qui sont des logiciels conçus pour causer des dommages à un système informatique, sont une menace constante. Pour lutter contre cette menace, nous avons développé un modèle d'apprentissage profond capable de détecter les logiciels malveillants en analysant les séquences d'appels d'API Windows dans les fichiers exécutables. L'objectif est de soumettre un fichier .exe à notre modèle et de recevoir en retour une prédiction indiquant si le fichier est un logiciel malveillant ou non. Ce rapport détaille la méthodologie utilisée, les résultats obtenus et les perspectives futures de ce projet.

2 Dataset

La base de données que j'ai utilisée pour mon projet s'intitule "Malware Analysis Datasets: API Call Sequences", disponible sur Kaggle. Cette base de données contient des séquences d'appels API associées à différents échantillons de logiciels. Chaque échantillon est identifié par un hash unique, et les séquences d'appels API sont représentées par une série d'identifiants numériques (t 0 à t 99), chaque identifiant correspondant à un appel API spécifique.



3 Etat de l'art de l'utilisation du Deep Learning pour la detection de malware

La détection de malwares est un domaine de recherche en constante évolution, et l'utilisation du Deep Learning a montré des résultats prometteurs dans cette tâche. Les systèmes de détection de malwares par Deep Learning exploitent la capacité des réseaux de neurones à apprendre des modèles complexes à partir de données brutes, ce qui leur permet de détecter efficacement les malwares en se basant sur des caractéristiques spécifiques.

Plusieurs approches de Deep Learning ont été explorées pour la détection de malwares, notamment les réseaux de neurones convolutifs (CNN), les réseaux de neurones récurrents (RNN), et les réseaux de neurones à propagation avant, tels que le Multi-Layer Perceptron (MLP). Chaque approche présente ses propres avantages et inconvénients en termes de représentation des données, de capacité de modélisation et de temps d'entraînement.

4 Méthodologie

La première étape de notre projet a été de collecter et de préparer les données pour l'entraînement de notre modèle. Nous avons utilisé un ensemble de données disponible sur Kaggle, qui contient des séquences d'appels d'API pour différents logiciels malveillants et non malveillants. Les appels d'API sont des points d'entrée pour le système d'exploitation pour interagir avec le logiciel, et les séquences d'appels d'API peuvent donc fournir des informations précieuses sur le comportement du logiciel.

Après avoir nettoyé et prétraité les données, nous avons divisé l'ensemble de données en un ensemble d'entraînement et un ensemble de test. Cette division est essentielle pour évaluer la performance de notre modèle sur des données qu'il n'a jamais vues auparavant, ce qui nous donne une idée de sa capacité à généraliser à de nouvelles données.

Pour le modèle lui-même, nous avons choisi d'utiliser un perceptron multicouche (MLP), qui est un type de réseau de neurones artificiels. Le MLP est un modèle d'apprentissage profond, ce qui signifie qu'il est capable de modéliser des relations complexes et non linéaires entre les entrées et les sorties. Nous avons entraîné le MLP sur l'ensemble d'entraînement et l'avons testé sur l'ensemble de test.

5 Modèle

L'ensemble des caractéristiques du modèle sont présentées dans le fichier malware-detection.ipnyb.

6 Résultats

Les résultats obtenus à partir de notre modèle MLP sont très prometteurs. Le modèle a obtenu un score d'entraînement de 99,67

La matrice de confusion, qui est un tableau utilisé pour décrire les performances d'un modèle de classification, montre également que notre modèle est capable de distinguer avec précision les logiciels malveillants des non-malveillants.

7 Interprétation des résultats

Bien que le modèle ait obtenu de très bons résultats sur l'ensemble de données de test, il est important de noter que ces résultats peuvent ne pas se généraliser à tous les fichiers .exe dans le monde réel. En effet, l'ensemble de données de test est une sous-partie de l'ensemble de données de formation, et il est donc possible que le modèle soit surajusté à cet ensemble de données spécifique.

Cependant, ces résultats sont encourageants et indiquent que notre approche a le potentiel de détecter efficacement les logiciels malveillants. De plus, ils soulignent l'efficacité de l'apprentissage profond pour ce type de tâche de classification.

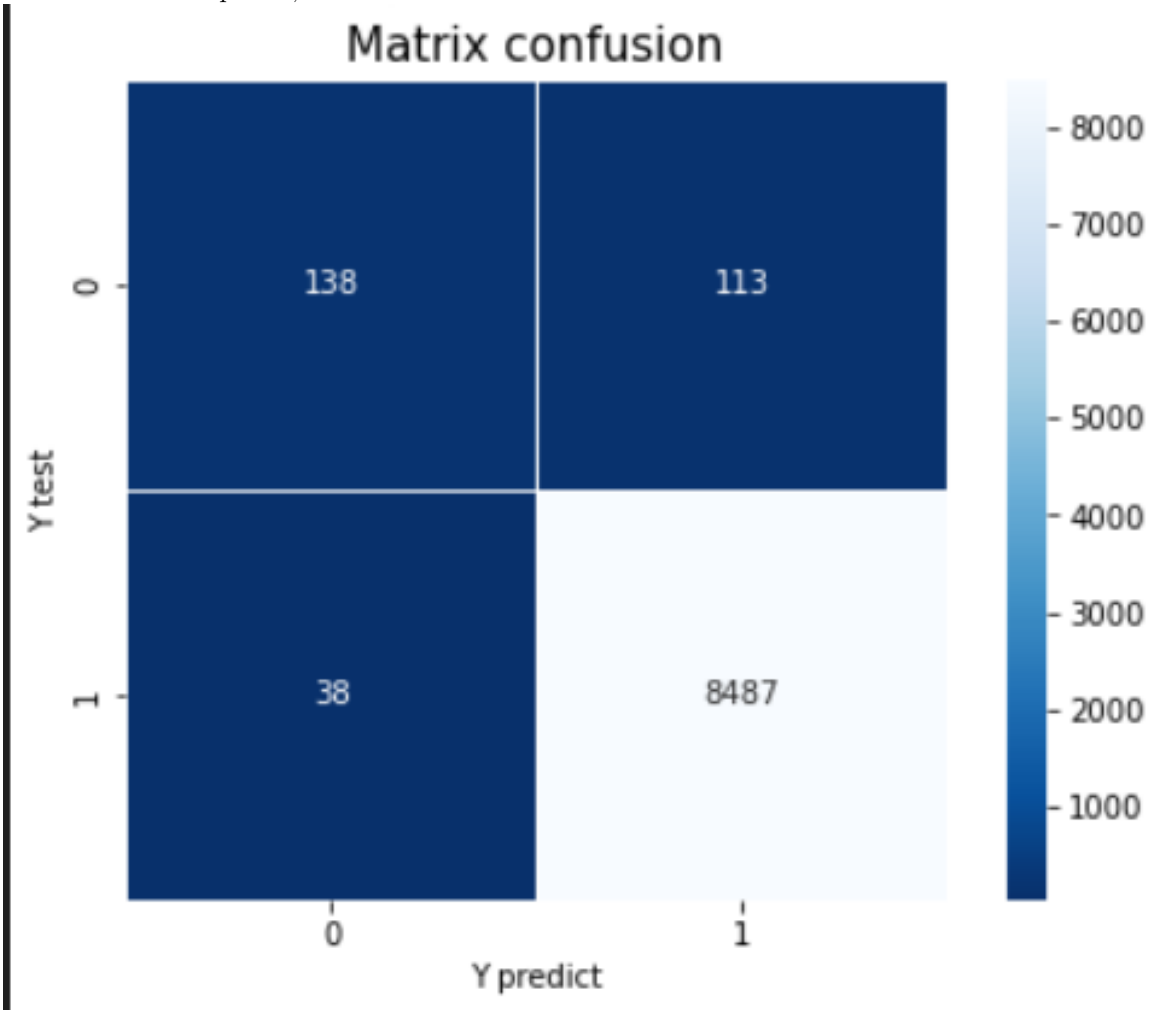
Matrice de confusion. Dans mon modèle de détection de malwares, j'ai utilisé une matrice de confusion pour évaluer les performances. Voici ce que j'ai pu déduire de cette matrice :

Vrais positifs (TP) : Mon modèle a correctement identifié 138 malwares. C'est encourageant, car cela signifie que le modèle est capable de détecter efficacement les malwares dans certains cas.

Faux positifs (FP) : Cependant, mon modèle a également prédit à tort que 113 fichiers sûrs étaient des malwares. C'est un domaine qui nécessite une amélioration, car ces erreurs pourraient entraîner des alertes inutiles pour l'utilisateur.

Faux négatifs (FN) : De plus, mon modèle a manqué 38 malwares, les classant à tort comme sûrs. C'est un autre domaine d'amélioration important, car ces erreurs pourraient permettre à des malwares de passer inaperçus.

Vrais négatifs (TN) : D'un autre côté, mon modèle a correctement identifié 8487 fichiers sûrs. C'est un résultat très positif, car cela s



8 Détection et détails sur detection.py

Une fois le modèle formé, nous avons développé un script Python, detection.py, pour permettre aux utilisateurs de soumettre des fichiers .exe pour analyse. Le script charge le modèle MLP formé, extrait les séquences d'appels d'API du fichier soumis, et utilise le modèle pour prédire si le fichier est un logiciel malveillant ou non. Les résultats de la prédiction sont ensuite affichés à l'utilisateur.

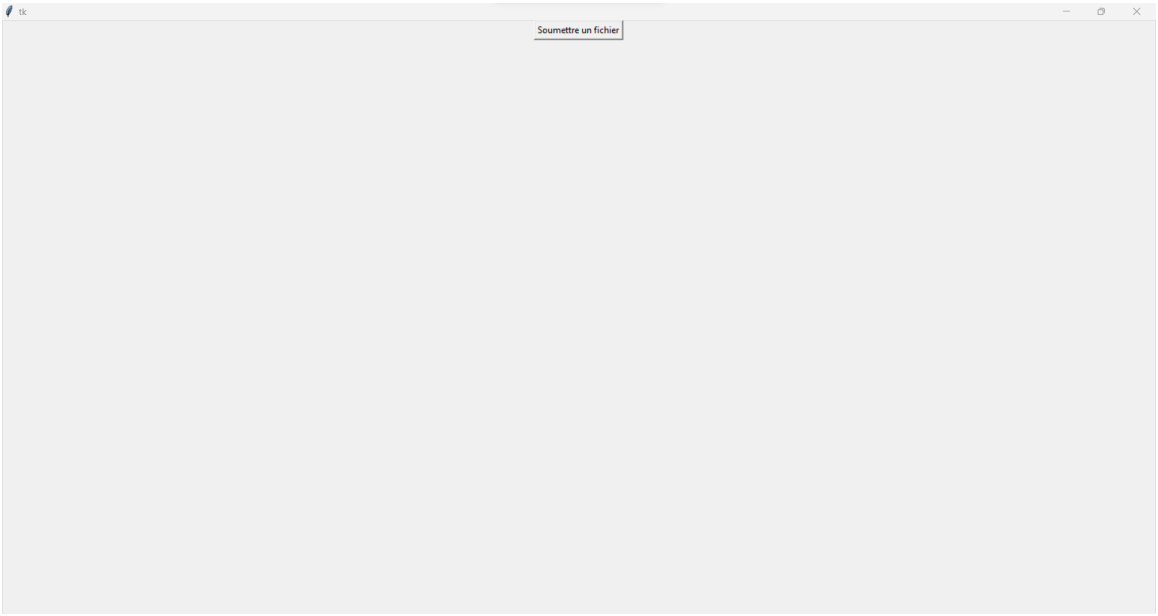
L'interface utilisateur est construite en utilisant la bibliothèque tkinter de Python, qui permet de créer des interfaces utilisateur graphiques. L'utilisateur peut soumettre un fichier en cliquant sur un bouton, et le résultat de la prédiction est affiché dans une étiquette.

Voici le code correspondant:

```
import tkinter as tk
from tkinter import filedialog
import joblib
import pefile
import os

# Charger le mod le form      partir du fichier
model = joblib.load('malware-detection.pkl')

print("le modele est", model)
print(type(model))


# Fonction pour extraire les s quences d'appels API    partir d'un fichier ex cutabl

def extraire_sequences_appels_api(filepath):
    sequences_api = []

    try:
        pe = pefile.PE(filepath)  # Ouvrir le fichier ex cutable avec pefile
        # Acc s aux sections pertinentes ou trouver les points d'entr e des s quenc
        for entry in pe.DIRECTORY_ENTRY_IMPORT:
            for API in entry.imports:
                sequences_api.append(API.name)
        # Extraire les s quences d'appels API et les stocker dans sequences_api

    except pefile.PEFormatError:
        # G rer les erreurs lors du traitement du fichier ex cutable
        print("Erreur de format de fichier ex cutable")

    return sequences_api

# Fonction pour soumettre le fichier ex cutable et effectuer la pr diction
def soumettre_fichier():
    # Ouvrir une bo te de dialogue pour s lectionner le fichier ex cutable
    filepath = filedialog.askopenfilename()

    # Pr traitement des donn es (assurez-vous de suivre le m me pr traitement que
```

```
    sequences_api = extraire_sequences_appels_api(filepath)  # Extraction des s quenc
    donnees_d_entree = [sequences_api]  # Cr er une liste de s quences d'appels API

    # Pr parez les donn es d'entr e que vous souhaitez soumettre au mod le pour la

    # Utiliser le mod le pour la pr diction
    prediction = model.predict(donnees_d_entree)

    # Afficher les r sultats de la pr diction sur l'interface utilisateur
    if prediction == 'malware':
        result_label.config(text="Le fichier est d tect  comme un malware.")
    else:
        result_label.config(text="Le fichier est d tect  comme bon.")

# Cr er l'interface utilisateur
root = tk.Tk()

# Bouton pour soumettre le fichier ex cutable
submit_button = tk.Button(root, text="Soumettre un fichier", command=soumettre_fichier
submit_button.pack()



#  tiquette  pour afficher les r sultats
result_label = tk.Label(root, text="")
result_label.pack()

# Lancer la boucle principale de l'interface utilisateur
root.mainloop()
```

On obtient donc visuellement :



9 Utilisation d'un ransomware comme test

Dans le cadre de notre test, nous avons utilisé un ransomware que nous avions développé lors d'un projet précédent. Le code source de ce ransomware est disponible sur mon compte GitHub : lien vers le repository.

Ce ransomware est conçu pour se camoufler en tant que fichier Excel, trompant ainsi la victime qui pourrait l'exécuter sans se rendre compte de sa nature malveillante.

10 Problèmes rencontrés

L'objectif était de pouvoir soumettre un fichier .exe et d'analyser ses séquences d'appels API afin de déterminer s'il s'agit ou non d'un malware. Malheureusement, les propriétaires de la base de données n'ont pas fourni les noms des séquences d'appels API correspondant aux numéros t 1, ..., t 99 qu'ils ont inclus dans leur base de données (par exemple VirtualAllocEx : 210). Cela nous empêche de convertir les séquences d'appels API en valeurs numériques, alors que notre modèle a été entrainé sur de telles données numériques. L'absence de ces informations essentielles rend impossible l'achèvement de ce système de détection.

C'est notamment ce qui est mentionné dans le rapport des chercheurs:

1) Similar to [13], it was considered the first 100 non-consecutive repeated API calls to avoid tracking loops. 2) Since in malware detection tasks, it is prominent to recognize malicious patterns as early as possible, the sequences were extracted from the parent process only. 3) We built the list of unique API calls, considering all the samples, and then converted each API call name into a unique integer identifier equal to the index of the API call name in the list. As a result, 307 distinct API calls were identified. We produced a dataset where the first column contains the MD5 hash of the sample. The next 100 columns contain ordinal categorical values between 0 and 306, representing the API call sequence of the sample. The last column contains the label of the sample, 0 for goodware, and 1 for malware. The total running time to collect the data was about 3000 hours, resulting in approximately 50,000 Cuckoo JSON report files and 1.5 TB of raw data. Our Cuckoo sandbox environment was based on an Intel Xeon D-1540, 8 cores, 16 threads, 2.6 GHz, 64 GB RAM, and 2 TB SSD running Ubuntu Server 16.04 as the Cuckoo host and 8 32-bit Windows 7 Ultimate VirtualBox virtual machines running in parallel as Cuckoo analysis guests.

11 Amélioration futures

Pour améliorer ce projet à l'avenir, nous envisageons d'explorer d'autres types de modèles d'apprentissage profond, tels que les réseaux de neurones convolutifs (CNN) ou les réseaux de neurones récurrents (RNN). Ces types de modèles pourraient être capables de capturer des motifs plus complexes dans les séquences d'appels d'API, ce qui pourrait améliorer la précision de la détection des logiciels malveillants.

Nous aimerions également tester notre modèle sur un ensemble de données plus diversifié pour évaluer sa capacité à généraliser à différents types de logiciels malveillants.

Nous chercherons également un ensemble de données comprenant les données nécessaires à la partie detection.py.

12 Bibliographie

Dataset utilisée : "Malware Analysis Datasets - API Call Sequences" par Angelo de Oliveira. Disponible sur Kaggle.

Article de recherche : Schranko de Oliveira, Angelo; Sassi, Renato José (2019). "Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks." TechRxiv. Prépublication. DOI : 10.36227/techrxiv.10043099.v1.