UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

DATA CHALLENGE
PROJET
CYBER AEGIS • 4 Avril 2025

1. MÉTHODOLOGIE GÉNÉRALE

Problématique

L'objectif de ce projet est de prédire des comportements malveillants associés à des exécutables Windows, en se basant sur les instructions assembleur extraites de leurs fichiers binaires. Concrètement, il s'agit d'un problème de **classification multi-label**, où l'on cherche à prédire plusieurs comportements binaires (présence ou absence) pour chaque programme.

Extraction de caractéristiques à partir des fichiers JSON

Les données fournies consistent en fichiers structurés de type JSON, contenant les instructions assembleur issues des graphes de contrôle (CFG). Chaque ligne suit une syntaxe de type `LABEL:INSTRUCTION[:SOUS-INSTRUCTION]`.

Étape 1 – Instructions principales [1]

Dans un premier temps, nous avons utilisé le logiciel graphviz pour interpréter les codes assembleur en dot language afin de mieux comprendre les fichiers train. Nous avons ensuite procédé sous python avec l'extrait des instructions principales apparaissant immédiatement après le premier séparateur ':'. Après analyse de fréquence, les 10 instructions les plus courantes (`CALL`, `JCC`, `RET`, etc.) ont été retenues. Nous avons ainsi ajouté 10 variables binaires indiquant leur présence dans chaque fichier.

- Résultat avec LightGBM : **18 %** de Macro-F1

Étape 2 – Sous-instructions [2]

Nous avons ensuite ciblé les sous-instructions figurant après un second ':' (par exemple : `jmp`, `je`, etc.). Cette étape a permis la génération de plus de **1250 nouvelles variables binaires**. En combinant avec les instructions principales, notre vecteur de caractéristiques final comptait environ **1260 dimensions**.

- Résultat avec LightGBM : **33 %** de Macro-F1

2. MODÉLISATION ET PERFORMANCES

Modèles testés

Plusieurs algorithmes ont été évalués :

- **LightGBM** : rapide à entraîner, mais performance modérée.
- **Random Forest** : résultats proches de LightGBM.
- **XGBoost** : meilleures performances, avec un Macro-F1 atteignant **51 %** après tuning des hyperparamètres.

[1] Voir Annexe 3
[2] Voir Annexe 3

Justification du choix du modèle : pourquoi XGBoost ?

Nous avons donc comparer ces trois algorithmes selon plusieurs critères : **la capacité à gérer un problème multi-label, des données structurées et déséquilibrées,** ainsi que le **besoin d'interprétabilité** dans un contexte de cybersécurité. Nos données combinent des instructions binaires (présence/absence d'opérations comme mov, jmp, call, etc.) et des caractéristiques de graphes d'exécution (nombre de nœuds, cycles, profondeur, etc.). Ce mélange hétérogène nécessite un modèle robuste, capable de gérer à la fois la variété des features et la rareté de certains comportements.
XGBoost, basé sur le gradient boosting, construit une série d'arbres de décision de manière séquentielle, chaque arbre corrigeant les erreurs du précédent. Ce fonctionnement est idéal pour entraîner un modèle par comportement (approche multi-label), tout en conservant précision et flexibilité. Contrairement à Random Forest, XGBoost intègre des mécanismes de régularisation (L1/L2, early stopping) et gère mieux les déséquilibres de classes. Enfin, XGBoost est interprétable : il permet d'analyser l'importance des variables et de visualiser ses décisions via des outils comme SHAP, ce qui est essentiel pour justifier les prédictions en cybersécurité.
En résumé, **XGBoost** s'est imposé comme le choix le plus adapté à notre problématique, en combinant performance, capacité d'explication et adaptabilité aux contraintes du projet.

Explorations complémentaires

- **Deep Learning** : l'approche a été explorée, mais abandonnée en raison de conflits de dépendances avec TensorFlow sous python.
- **Adresses mémoire** : plus de 5 millions d'adresses uniques ont été relevées, rendant cette piste peu exploitable sans réduction de dimension ou ressources de calcul importantes ($p > n$).

3. Interprétabilité du Modèle

Les modèles basés sur les arbres de décision (LightGBM, XGBoost) offrent une bonne interprétabilité grâce à l'analyse des **importances de variables**. Cela permet d'identifier les instructions les plus déterminantes dans la prédiction de certains comportements.

- Exemple : la sous-instruction `jmp` est fortement corrélée à la présence du comportement `SWITCH`.

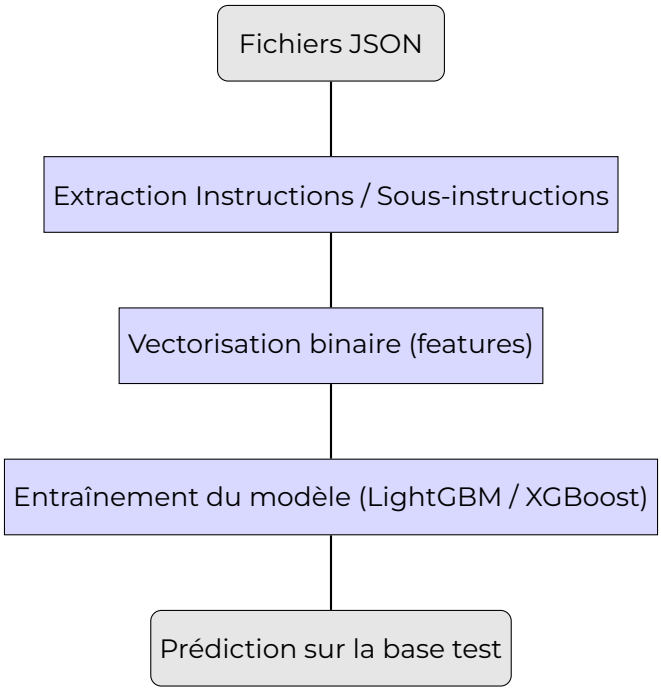
Conclusion

Nous avons mis en œuvre un pipeline efficace et interprétable de classification multi-label, basé sur l'analyse sémantique des instructions assembleur. En combinant une vectorisation binaire des instructions et l'usage de modèles de gradient boosting (XGBoost), nous avons atteint une performance satisfaisante de **51 %** en Macro-F1.
Après avoir comparer les trois méthodes (voir Annexe 3), nous avons ainsi choisi le modèle XGBoost.

Ce résultat met en évidence la pertinence d'une approche fondée sur des features explicites, tout en offrant une interprétation claire des prédictions – atout essentiel dans un contexte de cybersécurité.

ANNEXES

Annexe 1 : Pipeline de traitement des données JSON jusqu'à la prédiction



Annexe 2 : Tableau des instructions et sous-instructions

Instruction	Exemples de sous-instructions
CALL	–
CMOV	cmova, cmovb, cmove, cmovg, cmovl
HLT	–
INST	–
INVALID	–
JCC	ja, jb, je, jg, jl, jne, jnp
JMP	jmp, jrcxz
RET	ret, retf, retfq
SWITCH	–
UNDEF	–

Annexe 3 : Comparaison entre Random Forest, LightGBM et XGBoost pour le projet

Critère	**Besoin du projet**	**Random Forest**	**LightGBM**	**XGBoost (choix final)**
Nature du problème	Multi-label, données structurées et déséquilibrées	Gère le multi-label	Gère le multi-label	Gère très bien le multi-label
Données déséquilibrées	Certains comportements rares	Moins performant	Bon	Excellente gestion avec pondération
Types de features	Données binaires (instructions) + numériques (structure du graphe)	Compatible	Compatible	Optimisé pour ce type de données
Performance / Précision	Maximiser le score F1-macro	Moyenne à bonne	Très bonne	Très haute précision
Vitesse d'entraînement	Entraînement sur 20 000 fichiers	Rapide (parallélisable)	Très rapide	Moins rapide que LightGBM
Interprétabilité	Comprendre pourquoi un binaire est malveillant	Importance des variables disponible	Importance des variables disponible	Compatible avec SHAP, très explicable
Régularisation (overfitting)	Éviter l'apprentissage excessif du bruit	Pas de régularisation avancée	L1/L2 intégrés	L1/L2 + early stopping
Maturité et stabilité	Usage en contexte critique (cybersécurité)	Très stable	Moderne et rapide	Très mature et bien documenté