UNIVERSITE ABDELMALEK ESSAADI
FACULTE DES SCIENCES ET TECHNIQUES DE TANGER
DEPARTEMENT GENIE INFORMATIQUE

Rapport de projet

Intégration des MLOps avec les systèmes de détection des malwares basés sur les signatures et sur les images

Cycle Master en Sciences et Techniques

Filière : Sécurité IT et Big Data

Réalisé par :

- **BOULBEN Firdaous**
- **EL BOURKADI Abderrazzak**
- **EL HAYANI Adnan**
- **ET-TOUBI Ayoub**

Encadré par :

- **Pr. EL AACHAK Lotfi**

Année Universitaire : 2023/2024

1. Introduction

La détection de malware est un élément essentiel de la cybersécurité. Les approches traditionnelles basées sur les signatures et les images jouent un rôle important dans l'identification des logiciels malveillants. Cependant, l'évolution rapide des techniques employées par les cybercriminels nécessite des solutions plus adaptables et performantes. L'utilisation de l'apprentissage automatique (machine learning) permet d'automatiser la détection et la classification des logiciels malveillants.

Ce projet s'intéresse à l'intégration des principes MLOps (Machine Learning Operations) dans les systèmes de détection de malware basés sur les signatures et les images. Les MLOps visent à gérer efficacement le cycle de vie des modèles de machine learning, de leur développement à leur déploiement en production. En appliquant des pratiques MLOps, nous cherchons à améliorer la fiabilité, la reproductibilité et la scalabilité de la détection de malware.

Ce rapport couvrira en détail la méthodologie utilisée, incluant l'acquisition et le prétraitement des données, ainsi que la sélection et l'entraînement des modèles. En outre, il présentera les outils utilisés et l'application utilisateur de l'application réalisée.

2. Objectifs

L'objectif principal de ce projet est d'évaluer l'efficacité de l'apprentissage automatique (machine learning) et des principes MLOps (Machine Learning Operations) pour la détection de malware en intégrant des approches basées sur les signatures et les images. Il vise à :

- **Évaluer l'efficacité des modèles de machine learning :** Comparer différents modèles de machine learning pour la détection de malware, notamment les techniques basées sur les signatures et les images.
- **Intégrer les principes de MLOps :** Appliquer les meilleures pratiques de MLOps pour améliorer le cycle de vie des modèles de machine learning, y compris la versionnage, le suivi des expériences, et le déploiement continu.
- **Automatiser le pipeline de détection de malware :** Mettre en place un pipeline automatisé pour l'acquisition, le prétraitement des données, l'entraînement des modèles, et l'évaluation des performances.
- **Déployer les modèles en production :** Assurer la scalabilité et la robustesse des modèles déployés en utilisant des outils de conteneurisation et d'orchestration.

3. Méthodologie

Pour atteindre les objectifs fixés, la méthodologie suivante a été adoptée :

3.1. Acquisition de données

Pour mener à bien notre projet, nous avons acquis deux ensembles de données distincts pour la détection de logiciels malveillants : un ensemble de données d'images et un autre de signatures.

3.1.1. Ensemble de données de signatures :

L'ensemble de données de signatures utilisé pour ce projet contient des informations détaillées sur 138 047 fichiers PE (Portable Executable). Chaque entrée de cet ensemble de données comprend diverses caractéristiques techniques des fichiers, permettant une analyse approfondie et la classification des logiciels malveillants. Les colonnes principales de cet ensemble de données incluent :

- **Name**: Nom du fichier.
- **md5**: Hachage MD5 du fichier.
- **Machine**: Type de machine pour lequel le fichier est destiné.
- **SizeOfOptionalHeader**: Taille de l'en-tête optionnel.
- **Characteristics**: Caractéristiques spécifiques du fichier.
- **MajorLinkerVersion**, **MinorLinkerVersion**: Versions du linker.
- **SizeOfCode**, **SizeOfInitializedData**, **SizeOfUninitializedData**: Tailles des segments de code et de données.
- **AddressOfEntryPoint**: Adresse du point d'entrée du fichier.
- **BaseOfCode**, **BaseOfData**: Adresses de base du code et des données.
- **ImageBase**: Adresse de base de l'image.
- **SectionAlignment**, **FileAlignment**: Alignement des sections et des fichiers.
- **MajorOperatingSystemVersion**, **MinorOperatingSystemVersion**: Versions du système d'exploitation.
- **MajorImageVersion**, **MinorImageVersion**: Versions de l'image.
- **MajorSubsystemVersion**, **MinorSubsystemVersion**: Versions du sous-système.
- **SizeOfImage**, **SizeOfHeaders**: Tailles de l'image et des en-têtes.
- **CheckSum**: Somme de contrôle du fichier.
- **Subsystem**: Sous-système pour lequel le fichier est destiné.

- **DllCharacteristics**: Caractéristiques des DLL.
- **SizeOfStackReserve**, **SizeOfStackCommit**, **SizeOfHeapReserve**, **SizeOfHeapCommit**: Tailles réservées et engagées pour la pile et le tas.
- **LoaderFlags**: Indicateurs de chargement.
- **NumberOfRvaAndSizes**: Nombre de RVA (Relative Virtual Addresses) et de tailles.
- **SectionsNb**: Nombre de sections.
- **SectionsMeanEntropy**, **SectionsMinEntropy**, **SectionsMaxEntropy**: Entropie moyenne, minimale et maximale des sections.
- **SectionsMeanRawsize**, **SectionsMinRawsize**, **SectionMaxRawsize**: Taille brute moyenne, minimale et maximale des sections.
- **SectionsMeanVirtualsize**, **SectionsMinVirtualsize**, **SectionMaxVirtualsize**: Taille virtuelle moyenne, minimale et maximale des sections.
- **ImportsNbDLL**, **ImportsNb**, **ImportsNbOrdinal**: Nombre de DLL importées, nombre total d'importations, et nombre d'importations ordinales.
- **ExportNb**: Nombre d'exportations.
- **ResourcesNb**, **ResourcesMeanEntropy**, **ResourcesMinEntropy**, **ResourcesMaxEntropy**: Nombre de ressources, entropie moyenne, minimale et maximale des ressources.
- **ResourcesMeanSize**, **ResourcesMinSize**, **ResourcesMaxSize**: Taille moyenne, minimale et maximale des ressources.
- **LoadConfigurationSize**: Taille de la configuration de chargement.
- **VersionInformationSize**: Taille des informations de version.
- **legitimate**: Indicateur de légitimité du fichier (0 pour malveillant et 1 pour légitime), avec 96 724 fichiers malveillants et 41 323 fichiers légitimes.

Ces caractéristiques détaillées permettent de capturer les nuances et les comportements des logiciels malveillants, facilitant ainsi l'entraînement de modèles de machine learning pour leur détection précise.

Source : https://github.com/chihebchebbi/Mastering-Machine-Learning-for-Penetration-Testing/blob/master/Chapter03/MalwareData.csv.gz

3.1.2. Ensemble de données d'images :

L'ensemble de données que nous avons utilisé pour la classification des logiciels malveillants fusionne les deux jeux de données Malimg et Malevis. Ce jeu de données combiné contient toutes les classes Malevis, qui sont équilibrées en termes de distribution des classes, ainsi que cinq classes sélectionnées du jeu de données Malimg. L'objectif principal de cet ensemble de données est de permettre la classification multiclasse des byteplots représentant des fichiers PE malveillants, en prenant en compte à la fois les images en couleur RGB et en niveaux de gris dans un même ensemble de données.

Cet ensemble de données est organisé en deux répertoires distincts pour les ensembles d'entraînement et de validation, chacun comprenant un total de 31 classes (**Adposhel**, **Agent**, **Allaple**, **Alueron.gen!J**, **Amonetize**, **Androm**, **Autorun**, **BrowseFox**, **C2LOP.gen!g**, **Dialplatform.B**, **Dinwod**, **Elex**, **Expiro**, **Fakerean**, **Fasong**, **HackKMS**, **Hlux**, **Injector**, **InstallCore**, **Lolyda.AA1**, **Lolyda.AA2**, **MultiPlug**, **Neoreklami**, **Neshta**, **Regrun**, **Sality**, **Snarasite**, **Stantinko**, **VBA**, **VBKrypt**, **Vilsel**). Cette structure permet de séparer les données pour l'entraînement initial des modèles et pour l'évaluation de leur performance sur des données non vues.

Source : https://www.kaggle.com/datasets/gauravpendharkar/blended-malware-image-dataset

3.2. Prétraitement des données

Le prétraitement des données est une étape essentielle pour garantir que les ensembles de données sont prêts à être utilisés pour l'entraînement des modèles de détection de logiciels malveillants. Voici les étapes de prétraitement effectuées pour les ensembles de données d'images et de signatures.

3.2.1. Prétraitement des données de signatures

Pour les données de signatures, les étapes de prétraitement suivantes ont été effectuées :

- **Traitement des données dupliquées, manquantes et outliers** : L'ensemble de données n'avait pas des entrées dupliquées ou des valeurs manquantes. Les valeurs aberrantes (outliers) ont été identifiées et traitées pour garantir la robustesse du modèle.
- **Encodage des variables catégorielles :** Les colonnes catégorielles, telles que Name et md5, ont été encodées en utilisant un LabelEncoder.

- **Standardisation des caractéristiques** : Les caractéristiques numériques ont été mises à l'échelle en utilisant StandardScaler pour garantir que toutes les caractéristiques sont sur une échelle similaire, facilitant ainsi l'entraînement des modèles.
- **Feature selection** : Pour réduire la dimensionnalité et améliorer les performances du modèle, une sélection de caractéristiques a été effectuée. Les caractéristiques suivantes ont été retenues, réduisant l'ensemble de données à 20 caractéristiques pertinentes.

```
support = featureSelection.get_support()
print(support)
```

```
[ True  True  True False  True False False  True  True False False  True
 False False False  True False False False False  True False False  True
 False  True  True  True False False False False False  True  True  True
  True False False False False False False False False  True  True
 False False False False False False False  True]
```

```
X_selected = X_new[:, support]
X_selected.shape
```

```
(138047, 20)
```

```
selected_columns = X.columns[support]

# Print the names of selected columns
print(selected_columns)
```

```
Index(['Name', 'md5', 'Machine', 'Characteristics', 'SizeOfCode',
       'SizeOfInitializedData', 'BaseOfCode', 'FileAlignment',
       'MajorSubsystemVersion', 'SizeOfHeaders', 'Subsystem',
       'DllCharacteristics', 'SizeOfStackReserve', 'SectionsNb',
       'SectionsMeanEntropy', 'SectionsMinEntropy', 'SectionsMaxEntropy',
       'ExportNb', 'ResourcesNb', 'VersionInformationSize'],
      dtype='object')
```

3.2.2. Prétraitement des données d'images

Pour les données d'images, les étapes de prétraitement suivantes ont été effectuées :

- **Encodage des labels** : Les étiquettes de classe des images ont été encodées en utilisant LabelEncoder pour les convertir en valeurs numériques.

```
# Encode labels
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_valid = label_encoder.transform(y_valid)
```

- **Normalisation des pixels** : Les valeurs de pixel des images ont été mises à l'échelle dans une plage de 0 à 1 pour faciliter la convergence lors de l'entraînement des modèles.

```
# Normalize images
X_train = X_train / 255.0
X_valid = X_valid / 255.0
```

- **Redimensionnement des données pour l'entrée du modèle** : Les données d'image ont été redimensionnées à une forme uniforme (128x128 pixels) et remodelées pour être compatibles avec l'entrée du modèle.

```
# Reshape data for model input
X_train = X_train.reshape(-1, 128, 128, 1)
X_valid = X_valid.reshape(-1, 128, 128, 1)
```

3.3. Sélection et entraînement des modèles

Pour identifier les modèles les plus performants pour la détection de logiciels malveillants, plusieurs algorithmes de machine learning ont été entraînés et évalués. Pour les données de signatures, nous avons utilisé des modèles de **Logistic Regression**, des **Decision Tree** et des **Random Forest**. Ces modèles sont bien adaptés pour traiter les caractéristiques discrètes et continues présentes dans les données de signatures. Pour les données d'images, nous avons testé plusieurs modèles de classification d'images, notamment les **Decision Tree**, **Random Forest**, les **Support Vector Machine (SVM)** et le **Naive Bayes**.

3.4. Évaluation des modèles

L'évaluation des modèles a été réalisée en utilisant diverses métriques de performance pour déterminer l'efficacité des modèles de détection de logiciels malveillants. Voici les résultats obtenus pour les données de signatures et les données d'images.

3.4.1. Évaluation des modèles pour les signatures

Pour les données de signatures, nous avons utilisé l'accuracy pour évaluer la performance des modèles. Les résultats obtenus sont les suivants :

- **Logistic Regression** : `0.9776168055052518`
- **Decision Tree** : `0.981238681637088`
- **Random Forest** : `0.9991669684896777`

3.4.2. Évaluation des modèles pour les images

Pour les données d'images, nous avons évalué les modèles en utilisant plusieurs métriques, y compris l'accuracy, la précision, le rappel et le F1-score. Les résultats obtenus sont les suivants:

```python
# Train and evaluate models
results = {}
for model_name, model in models.items():
    model.fit(X_train_flat, y_train)
    y_pred = model.predict(X_valid_flat)
    results[model_name] = {
        "Accuracy": accuracy_score(y_valid, y_pred),
        "Precision": precision_score(y_valid, y_pred, average='weighted'),
        "Recall": recall_score(y_valid, y_pred, average='weighted'),
        "F1-Score": f1_score(y_valid, y_pred, average='weighted')
    }
```

```python
# Print results
for model_name, metrics in results.items():
    print(f"Model: {model_name}")
    for metric_name, metric_value in metrics.items():
        print(f"{metric_name}: {metric_value}")
    print("\n")
```

```
Model: Decision Tree
Accuracy: 0.8780613560195927
Precision: 0.8753943549979862
Recall: 0.8780613560195927
F1-Score: 0.8758712986219503


Model: Random Forest
Accuracy: 0.9409641660221707
Precision: 0.9446544595766387
Recall: 0.9409641660221707
F1-Score: 0.9417037574897283


Model: SVM
Accuracy: 0.9218870843000774
Precision: 0.9326725191366753
Recall: 0.9218870843000774
F1-Score: 0.9241379524341644


Model: Naive Bayes
Accuracy: 0.7525135344160866
Precision: 0.7940389843747464
Recall: 0.7525135344160866
F1-Score: 0.7564289237319274
```

3.5. Intégration des pratiques de MLOps

Nous avons intégré les pratiques MLOps dans notre pipeline de machine learning pour assurer la fiabilité et l'évolutivité de notre système de détection de logiciels malveillants. Cela inclut le

contrôle de version pour suivre les modifications, le déploiement continu pour des mises à jour rapides, le suivi des expériences pour évaluer les performances, le versionnement des modèles pour une gestion efficace et l'orchestration de l'infrastructure pour une exécution fluide des tâches.

3.6. Déploiement des modèles

Nous avons déployé le modèle Random Forest, qui s'est révélé être le meilleur performer avec une précision de 0,99 pour l'ensemble de données des signatures et de 0,94 pour les données des images. Pour garantir la scalabilité et la fiabilité du déploiement, nous avons utilisé des techniques de conteneurisation et d'orchestration. Le modèle a été encapsulé dans un conteneur Docker pour assurer sa portabilité et son isolation. Ensuite, nous avons utilisé des outils d'orchestration tels que Kubernetes pour gérer et coordonner le déploiement des conteneurs dans un environnement de production, garantissant ainsi une disponibilité élevée et une gestion efficace des ressources.

4. Outils Utilisés

Pour le développement et le déploiement de notre système de détection de logiciels malveillants, nous avons utilisé une combinaison d'outils et de technologies bien établis. Voici une liste des principaux outils utilisés :

4.1. Bibliothèques Python:

- **Scikit-learn:** Bibliothèque d'apprentissage automatique populaire pour l'entraînement et l'évaluation de modèles de détection de malwares.
- **OpenCV:** Bibliothèque de traitement d'images puissante pour la manipulation et l'analyse d'images de malwares.

4.2. Outils de conteneurisation et d'orchestration:

- **Docker:** Plateforme de conteneurisation pour l'empaquetage, la distribution et l'exécution d'applications dans des conteneurs isolés.
- **Kubernetes:** Système open-source pour l'automatisation du déploiement, de la mise à l'échelle et de la gestion des applications conteneurisées.

4.3. Développement web:

- **Flask:** Un framework web Python léger et polyvalent pour la création d'applications web et d'API. Il est utilisé pour construire l'API RESTful qui alimente l'interface utilisateur et gère les interactions avec le modèle de détection de malwares.
- **Angular:** Cadre de développement JavaScript populaire pour la création d'applications web dynamiques et riches en fonctionnalités. Il est utilisé pour développer l'interface utilisateur du système de détection de malwares.
- **Tailwind:** Framework CSS performant et personnalisable pour la conception et le stylisme d'interfaces utilisateur responsives. Il est utilisé pour créer une interface utilisateur élégante et intuitive pour l'application.

4.4. Contrôle de version et CI/CD:

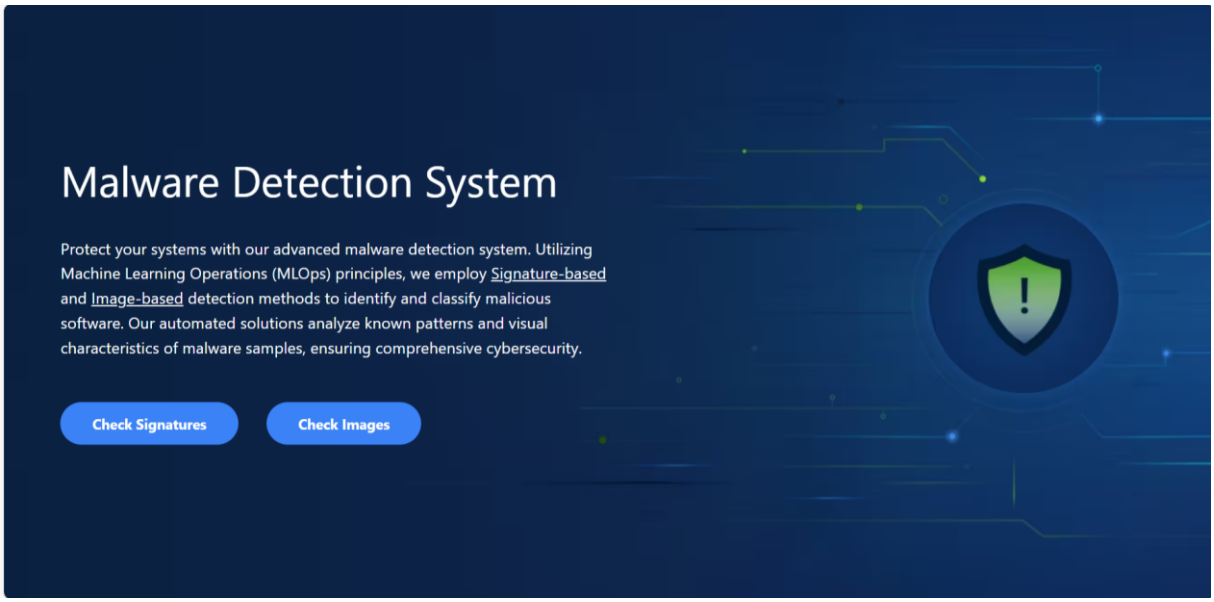
- **GitHub:** Plateforme de développement logiciel basée sur Git, utilisée pour l'hébergement de code source, le contrôle de version et l'intégration continue/déploiement continu (CI/CD). Il facilite la collaboration et l'automatisation du processus de développement.

5. Présentation de l'interface web

L'interface Web développée dans le cadre de ce projet offre une interface conviviale et intuitive pour la détection de malware basée sur les signatures et les images. Elle permet aux utilisateurs de soumettre des fichiers ou des images suspects et d'obtenir des résultats précis et rapides sur leur nature malveillante.

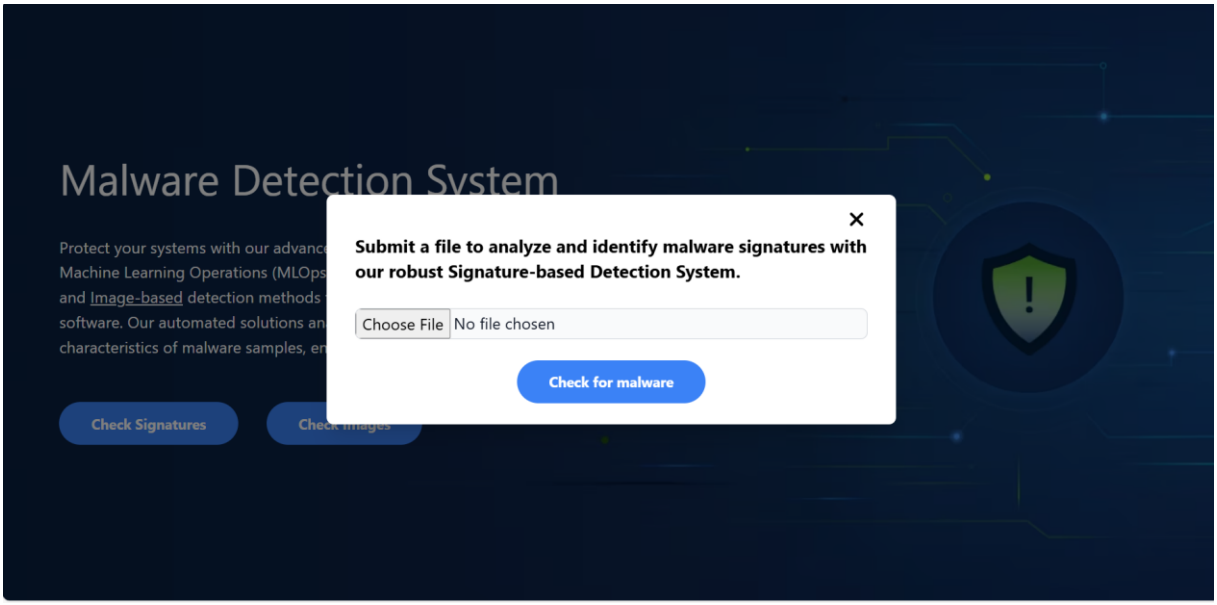
5.1. Page d'accueil

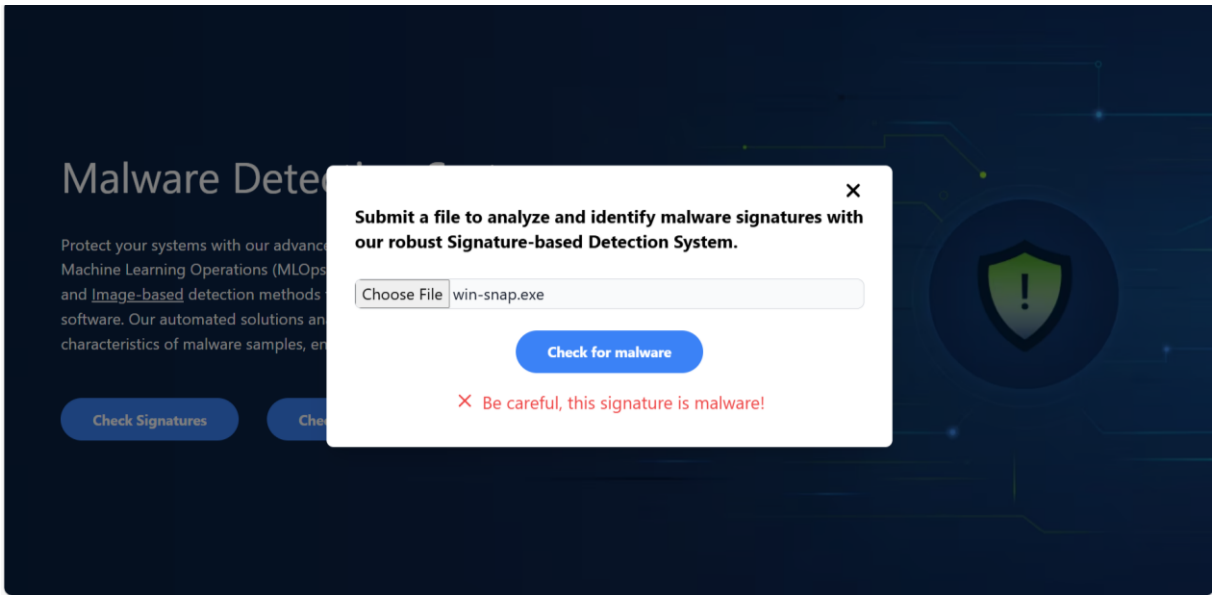
La page d'accueil présente une interface claire et simple, avec une brève description du système et de ses fonctionnalités, ainsi que deux boutons principaux pour la vérification des signatures et la vérification des images.



5.2.Détection par signature

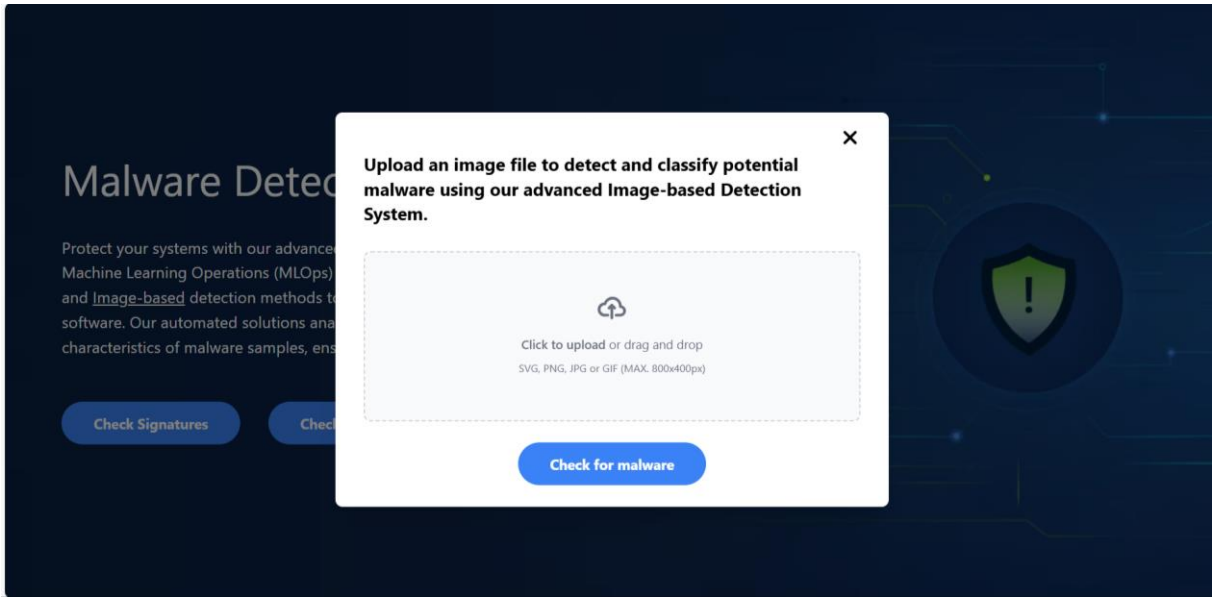
Cette section permet aux utilisateurs de soumettre des fichiers suspects pour les analyser. Une fois le fichier soumis, l'interface affiche le résultat de l'analyse, indiquant si le fichier est un malware ou non.

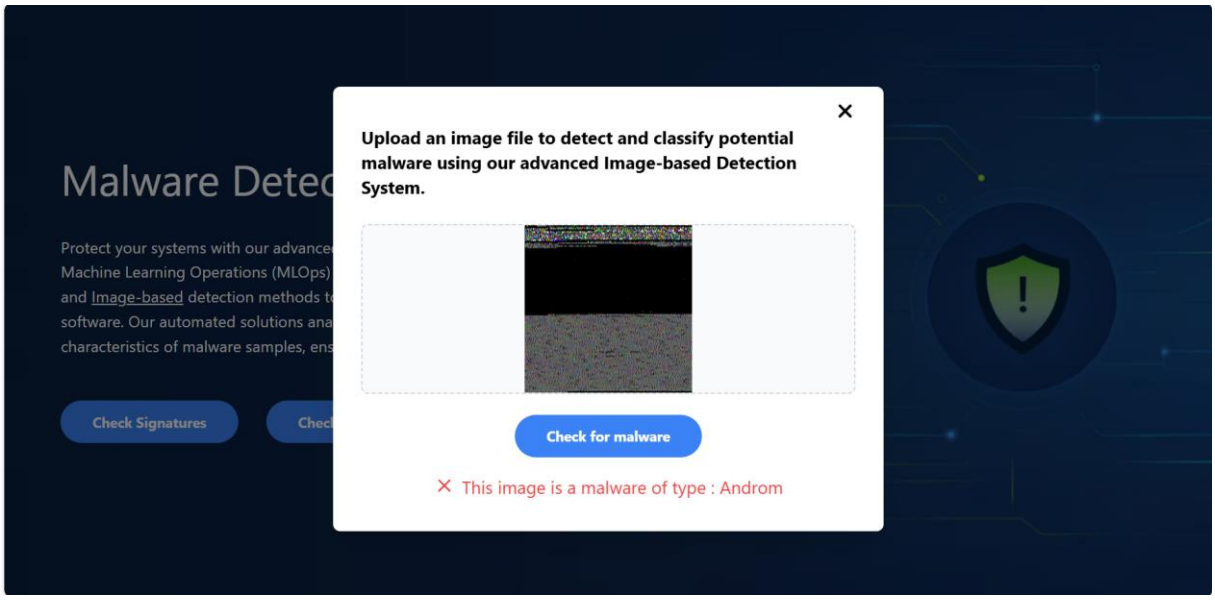


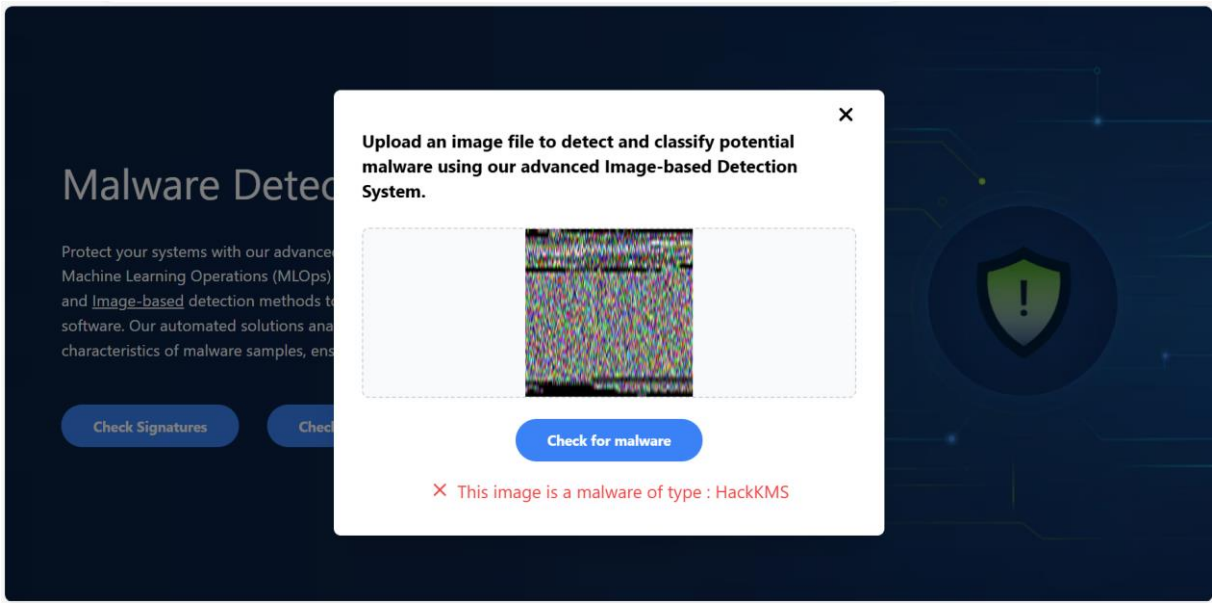


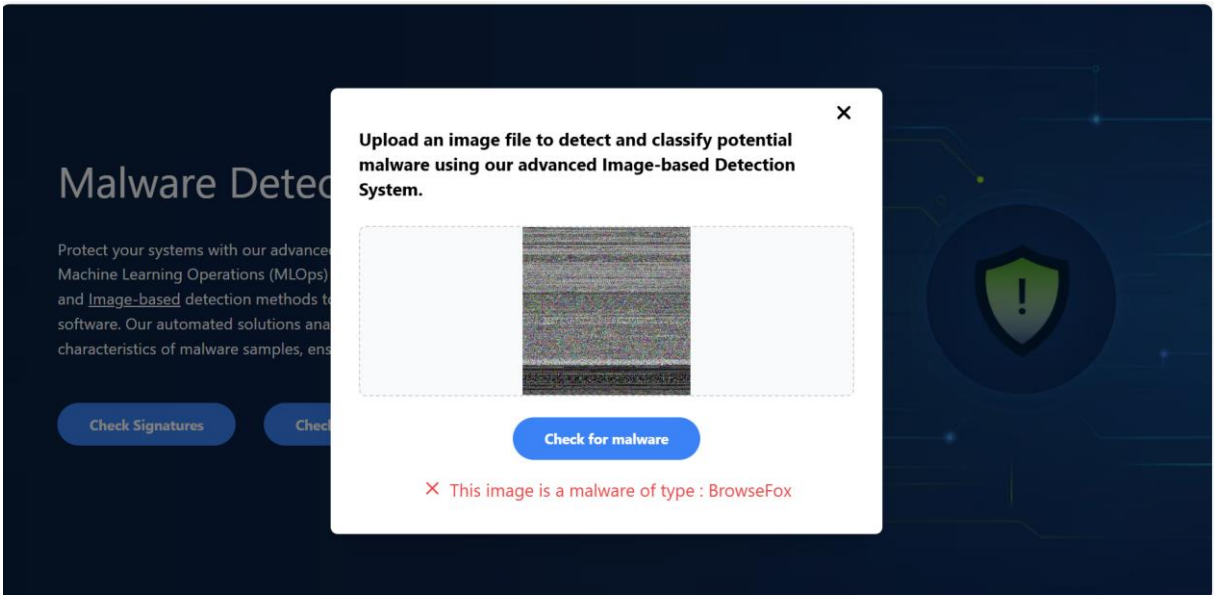
5.3.Détection par image

Cette section permet aux utilisateurs de soumettre des images suspectes pour les analyser. Une fois l'image soumise, l'interface affiche le résultat de l'analyse, indiquant le type de malware détecté (s'il y en a un).




Malware Detec
Protect your systems with our advance
Machine Learning Operations (MLOps)
and Image-based detection methods t
software. Our automated solutions ana
characteristics of malware samples, ens
Check Signatures
Upload an image file to detect and classify potential malware using our advanced Image-based Detection System.
Check for malware
This image is a malware of type : Androm


Malware Detec
Protect your systems with our advance
Machine Learning Operations (MLOps)
and Image-based detection methods t
software. Our automated solutions ana
characteristics of malware samples, ens
Check Signatures
Upload an image file to detect and classify potential malware using our advanced Image-based Detection System.
Check for malware
This image is a malware of type : HackKMS



6. Conclusion

Ce projet a démontré le potentiel de l'apprentissage automatique et des MLOps pour améliorer la détection de malware. L'utilisation de modèles de machine learning entraînés et l'intégration de pratiques MLOps peuvent conduire à des systèmes de détection de malware plus précis, robustes et évolutifs, contribuant ainsi à la cybersécurité des systèmes informatiques. Les résultats et les connaissances acquises au cours de ce projet peuvent servir de base pour des recherches futures visant à optimiser davantage les techniques de détection de malware et à développer des solutions encore plus performantes.

7. Annexe

Lien vers le dépôt GitHub: https://github.com/firdaous-boulben/malware-detection
Ce dépôt GitHub contient le code source complet du projet de détection de malware utilisant l'apprentissage automatique et les MLOps.