I. Modèle de base

1. Génération des Données

Le script génère deux types de données :
- **Données bénignes :** Caractérisées par une faible utilisation du CPU, des opérations disque modérées, et une activité réseau faible à modérée.
- **Données ransomware :** Caractérisées par une utilisation élevée du CPU, des opérations disque intenses, et une activité réseau importante.

a. Code correspondant :

```python
import numpy as np
import pandas as pd

# Generate benign data
num_benign = 700
benign_data = {
    "cpu_usage": np.random.uniform(5, 50, num_benign),
    "disk_io": np.random.uniform(100, 500, num_benign),
    "network_activity": np.random.uniform(100, 2000, num_benign),
    "label": 0
}

# Generate ransomware data
num_ransomware = 300
ransomware_data = {
    "cpu_usage": np.random.uniform(70, 100, num_ransomware),
    "disk_io": np.random.uniform(5000, 10000, num_ransomware),
    "network_activity": np.random.uniform(3000, 5000, num_ransomware),
    "label": 1
}
```

b. Combinaison des Données et Sauvegarde

Les deux ensembles de données sont fusionnés en un DataFrame unique, qui est ensuite sauvegardé au format CSV pour une utilisation ultérieure.

```python
# Combine datasets
benign_df = pd.DataFrame(benign_data)
ransomware_df = pd.DataFrame(ransomware_data)
df = pd.concat([benign_df, ransomware_df], ignore_index=True)

# Save dataset to CSV
df.to_csv("ransomware_dataset.csv", index=False)
```

2. Chargement des librairies

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Voici une explication des bibliothèques utilisées dans le code :

- **`pandas as pd`** :
  - `pandas` est une bibliothèque puissante pour la manipulation et l'analyse de données en Python. Elle permet de lire, écrire, transformer et analyser des données sous forme de DataFrame.
  - **Utilisation dans le code** :
    - Charger des données depuis un fichier CSV (`pd.read_csv`).
    - Créer et manipuler des DataFrames (ensembles de données tabulaires).
    - Analyser l'importance des caractéristiques en DataFrame.
- **`numpy as np`** :
  - `numpy` fournit un support pour le calcul numérique avec des tableaux multidimensionnels efficaces (ndarray) et des fonctions mathématiques performantes.
  - **Utilisation dans le code** :
    - Générer des données aléatoires pour les ensembles de données simulés (`np.random.uniform`).
- **`from sklearn.ensemble import RandomForestClassifier`** :
  - Cette classe appartient à la bibliothèque `scikit-learn`. `RandomForestClassifier` est un algorithme d'apprentissage supervisé qui utilise plusieurs arbres de décision pour améliorer la précision des prédictions.
  - **Utilisation dans le code** :
    - Créer un modèle pour différencier les comportements bénins et ceux du ransomware.
- **`from sklearn.model_selection import train_test_split`** :
  - Cette fonction divise un ensemble de données en sous-ensembles d'entraînement et de test. Cela permet de valider les performances du modèle sur des données non vues.
  - **Utilisation dans le code** :
    - Diviser les données en ensembles d'entraînement (70%) et de test (30%).
- **`from sklearn.metrics import classification_report, confusion_matrix, accuracy_score`** :
  - Ces fonctions permettent d'évaluer les performances du modèle :
    - `classification_report` : Affiche des métriques telles que la précision, le rappel et le score F1.
    - `confusion_matrix` : Donne un tableau montrant les vraies et fausses classifications.

- `accuracy_score` : Calcule le pourcentage de prédictions correctes.
- **Utilisation dans le code** :
  - Fournir des statistiques sur les performances du modèle après la prédiction.

3. Préparation des Données

a. Chargement et Séparation

Les données sont chargées depuis le fichier CSV. Les caractéristiques (CPU, disque, réseau) sont séparées des étiquettes (bénigne ou ransomware). Ensuite, elles sont divisées en ensembles de données d'entraînement (70%) et de test (30%).

```
# Load dataset
data = pd.read_csv('ransomware_dataset.csv')

# Separate features and labels
X = data.drop('label', axis=1)  # All columns except the label
y = data['label']  # The label column

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

4. Entraînement du Modèle

Un modèle Random Forest est créé avec 100 arbres de décision. Il est entraîné sur l'ensemble d'entraînement.

```
# Create the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)
```

5. Évaluation du Modèle

a. Prédiction et Métriques

Le modèle est évalué sur l'ensemble de test. Les métriques incluent la matrice de confusion, le rapport de classification et le score de précision.

```
# Predict on the test set
y_pred = rf_model.predict(X_test)

# Evaluation Metrics
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
 [[213   0]
 [  0  87]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       213
           1       1.00      1.00      1.00        87

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300


Accuracy Score: 1.0
```

b. Importance des Caractéristiques

L'importance des caractéristiques est analysée pour identifier les plus influentes dans la prédiction.

```python
# Feature importance analysis
feature_importances = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

print("Feature Importances:\n", feature_importances)
```

```
Feature Importances:
              Feature  Importance
0         cpu_usage        0.36
1           disk_io        0.36
2  network_activity        0.28
```

6. Sauvegarde du Modèle

Le modèle entraîné est sauvegardé au format **`.pkl`** pour une utilisation future.

```python
import joblib
joblib.dump(rf_model, "ransomware_detector.pkl")
```

II. Modèle de détection en temps réel

```python
import psutil
import joblib
import numpy as np
import time

# Load model
model = joblib.load('ransomware_detector.pkl')

while True:
    # Collect hardware metrics
    cpu_usage = psutil.cpu_percent(interval=1)
    disk_io = psutil.disk_io_counters().write_bytes
    network_out = psutil.net_io_counters().bytes_sent

    # Prepare data for the model
    features = np.array([cpu_usage, disk_io, network_out]).reshape(1, -1)

    # Predict
    prediction = model.predict(features)

    if prediction[0] == 1:
        print("ALERT: Potential ransomware detected!")
    else:
        print("System is normal.")

    # Sleep before next check
    time.sleep(5)
```

Ce code Python est conçu pour surveiller en temps réel un système et détecter des activités potentielles de ransomware à l'aide du modèle de Machine Learning préalablement entraîné.

1. **Importation des Bibliothèques**

Le script commence par importer les bibliothèques suivantes :

- **psutil :** Permet de collecter des métriques système comme l'utilisation du CPU, les opérations disque et l'activité réseau.
- **joblib :** Utilisé pour charger le modèle de Machine Learning sauvegardé au format **`.pkl`**.
- **numpy :** Fournit des outils pour manipuler les données sous forme de tableaux multidimensionnels.
- **time :** Gère les temporisations entre les cycles de surveillance.

2. **Chargement du Modèle**

Ce modèle de détection de ransomware est chargé à partir du fichier préalablement entraîné et sauvegardé :

```python
# Load model
model = joblib.load('ransomware_detector.pkl')
```

Ce modèle est un classificateur qui détermine si un comportement système est **normal (0)** ou d'un **Ransomware (1)**.

3. Boucle de Surveillance

Le script entre dans une boucle infinie pour surveiller le système en temps réel. Chaque itération effectue les étapes suivantes :

a. Collecte des Métriques Système

```
while True:
    # Collect hardware metrics
    cpu_usage = psutil.cpu_percent(interval=1)
    disk_io = psutil.disk_io_counters().write_bytes
    network_out = psutil.net_io_counters().bytes_sent
```

- **Utilisation du CPU :**
  cpu_usage = psutil.cpu_percent(interval=1)
  Cette fonction mesure l'utilisation du CPU sur un intervalle d'une seconde.

- Opérations disque :
disk_io = psutil.disk_io_counters().write_bytes
Cette fonction récupère le nombre total d'octets écrits sur le disque.

- Activité réseau :
network_out = psutil.net_io_counters().bytes_sent
Cette fonction mesure le nombre total d'octets envoyés via le réseau.

b. Préparation des Données

Les métriques collectées sont regroupées dans un tableau à une dimension et redimensionnées pour correspondre à la structure attendue par le modèle :

```
# Prepare data for the model
features = np.array([cpu_usage, disk_io, network_out]).reshape(1, -1)
```

c. Prédiction

Le modèle prédit l'état du système en utilisant les métriques préparées :

```
# Predict
prediction = model.predict(features)
```

d. Alertes

En fonction du résultat de la prédiction :
- Si le modèle prédit `**1**` (comportement suspect) : Une alerte est affichée.
- Sinon (comportement normal) :

```
if prediction[0] == 1:
    print("ALERT: Potential ransomware detected!")
else:
    print("System is normal.")
```

e. Temporisation

Une pause de 5 secondes est introduite avant le prochain cycle de surveillance :

```
# Sleep before next check
time.sleep(5)
```

AMAHROUK Asmae
Cyber Security Engineering Student