Solution pour la sécurité des maisons intelligentes

Réalise par :

ECHCHRIKI IMAD

Bakkali Tahiri Mohamed

AARAB Ayoub

ABDERRAHMAN ZAOUIDI

Année Universitaire **2023-2024**

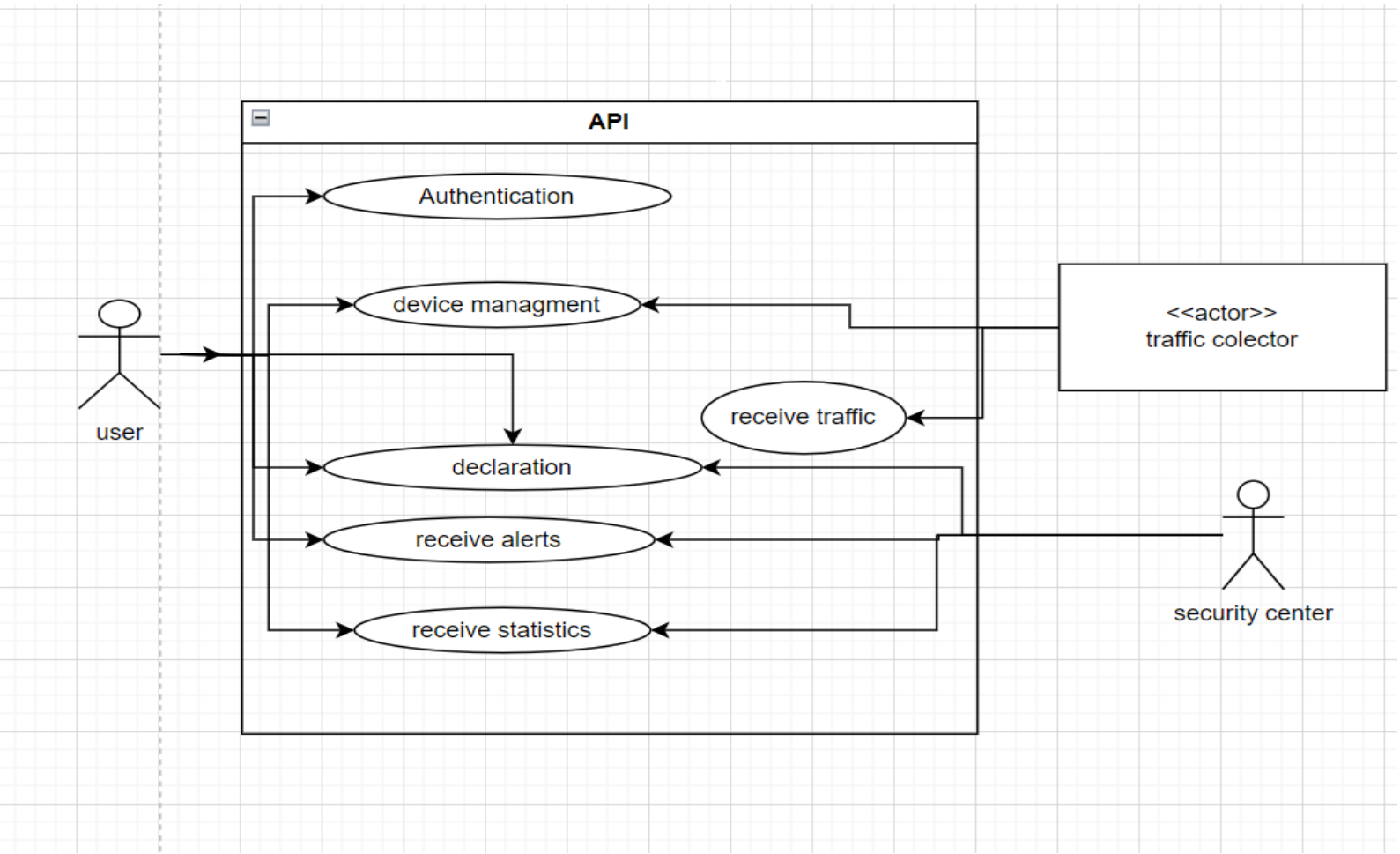
Table des matières

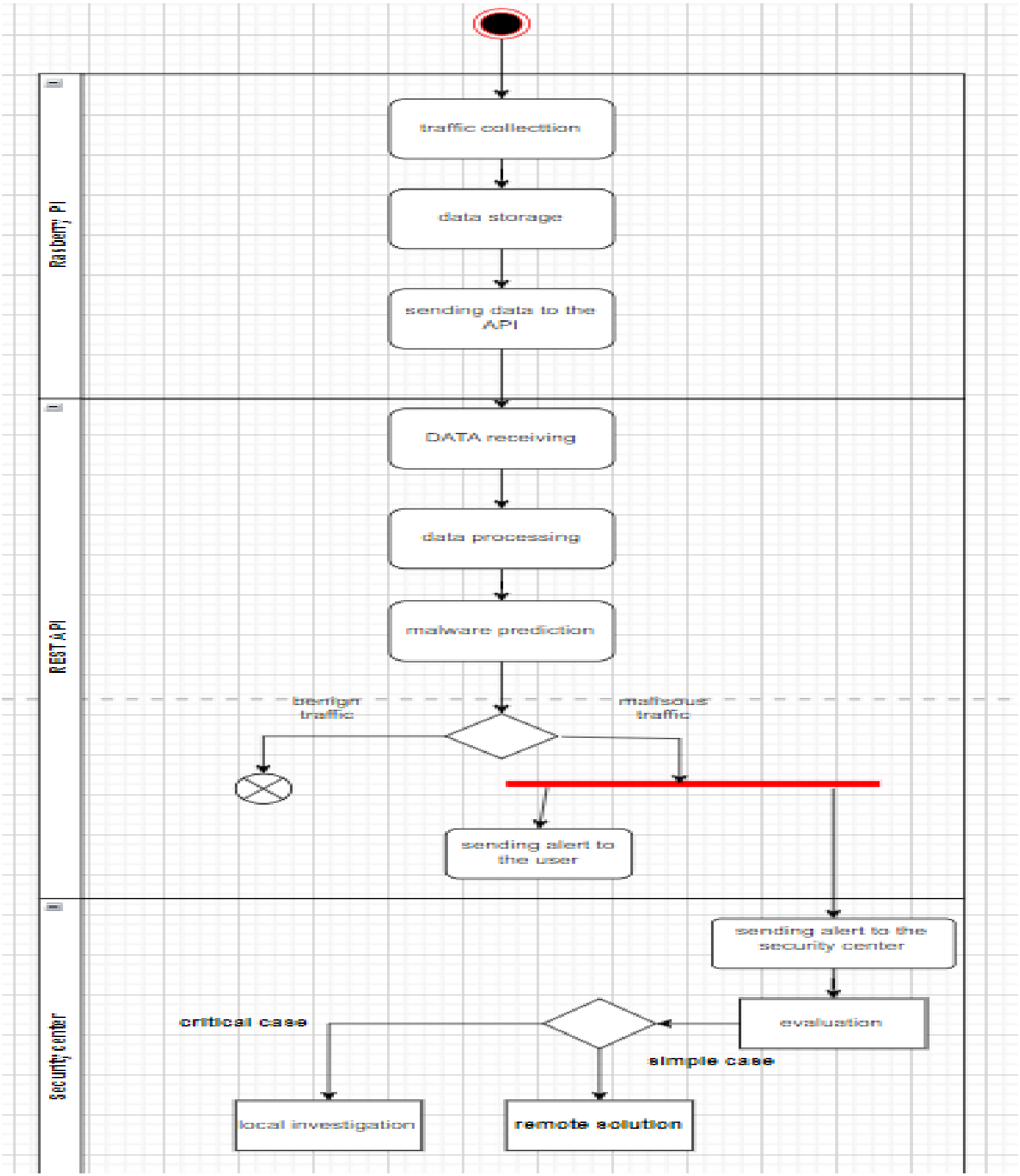
1. présentation de problematique:

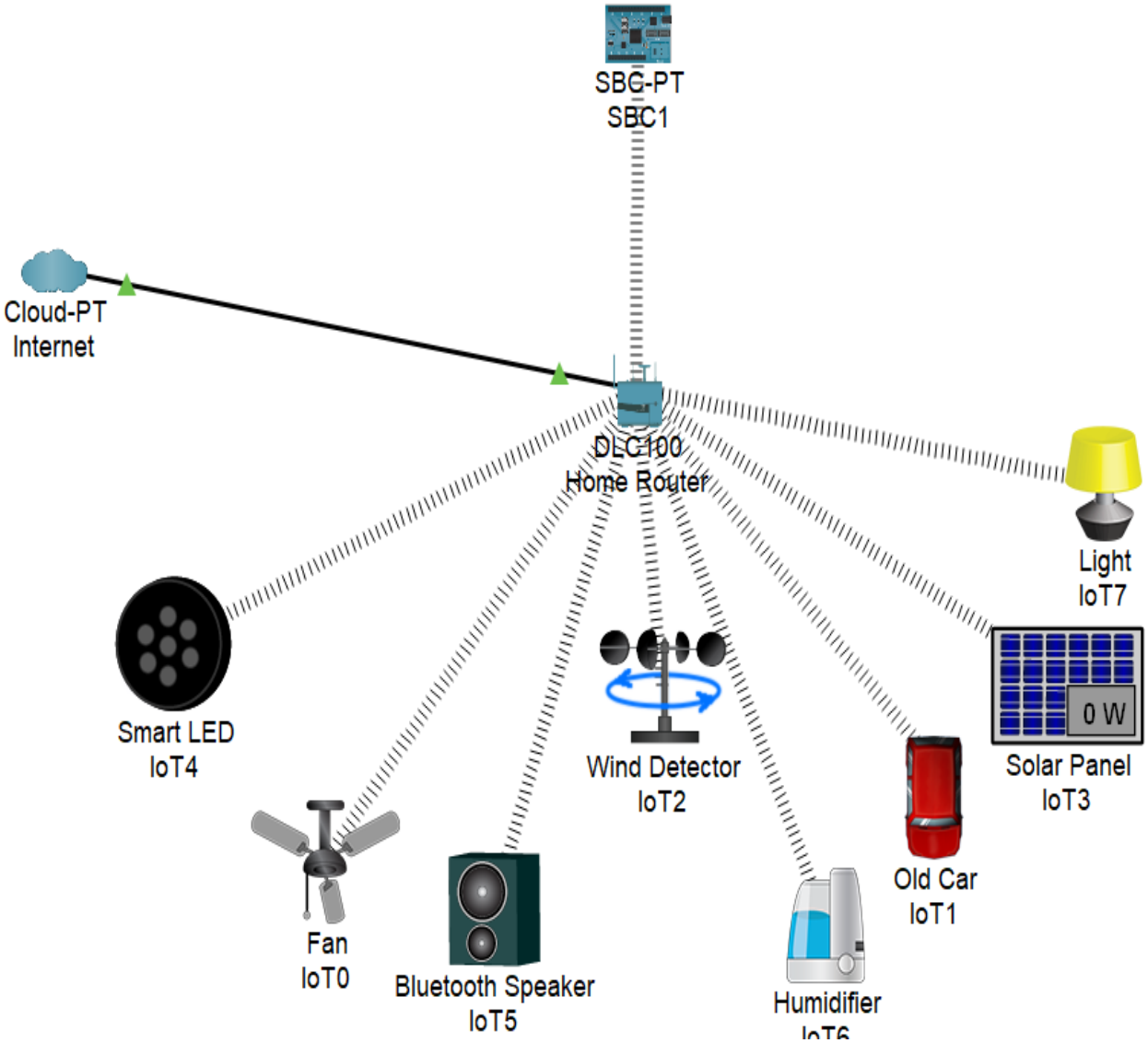
1.1. Introduction:

Avec l'avènement de la technologie connectée, les maisons intelligentes, ou "smart homes", sont devenues une réalité tangible pour de nombreux foyers. Ces habitations intègrent une variété d'appareils et de systèmes interconnectés qui offrent commodité, efficacité, et sécurité aux occupants. Des thermostats intelligents aux serrures connectées, en passant par les caméras de surveillance, ces dispositifs IoT (Internet des objets) promettent de transformer notre manière de vivre. Cependant, cette révolution technologique n'est pas sans défis. Les dispositifs IoT dans les maisons intelligentes présentent des vulnérabilités en matière de sécurité, que des cybercriminels peuvent exploiter. Les risques vont de l'accès non autorisé aux données personnelles à la prise de contrôle des appareils à distance, ce qui met en péril la vie privée et la sécurité des résidents. Dans cette section, nous explorerons les défis associés à la sécurité des dispositifs IoT dans les maisons intelligentes, ainsi que les contraintes et les limitations qui pourraient entraver leur adoption généralisée.

1.2. La conception :




Raspberry PI
traffic collection
data storage
sending data to the API
REST API
DATA receiving
data processing
malware prediction
benign traffic
malicious traffic
sending alert to the user
Security center
sending alert to the security center
evaluation
critical case
simple case
local investigation
remote solution


SBC-PT
SBC1
Cloud-PT
Internet
DLC100
Home Router
Light
IoT7
Smart LED
IoT4
Wind Detector
IoT2
0 W
Solar Panel
IoT3
Old Car
IoT1
Fan
IoT0
Bluetooth Speaker
IoT5
Humidifier

1.3. Aperçu Général:

Notre projet vise à concevoir et à développer une plateforme innovante d'analyse du trafic généré par les dispositifs IoT au sein des maisons intelligentes. Cette plateforme a pour objectif principal de surveiller et d'analyser le flux de données circulant dans le réseau domestique, en identifiant et en capturant le trafic provenant des différents appareils connectés.
La plateforme fonctionne en collectant activement le trafic réseau à domicile, en utilisant des capteurs ou des sniffers positionnés stratégiquement pour intercepter les paquets de données échangés entre les dispositifs IoT et le réseau local. Une fois collectées, ces données sont ensuite envoyées à un serveur central via une API REST pour un traitement ultérieur.

Au niveau du serveur, les données recueillies sont soumises à une analyse approfondie à l'aide de modèles de machine learning avancés. Ces modèles sont spécialement conçus pour détecter des schémas et des comportements anormaux qui pourraient indiquer une activité suspecte ou une intrusion potentielle dans le réseau domestique. L'objectif est de détecter rapidement les activités malveillantes telles que les tentatives d'accès non autorisé, les attaques par déni de service, ou toute autre forme d'intrusion visant à compromettre la sécurité du système.
En intégrant l'analyse du trafic réseau et l'apprentissage automatique, notre plateforme offre une solution robuste et proactive pour renforcer la sécurité des maisons intelligentes face aux menaces croissantes liées aux dispositifs IoT. Elle permet aux utilisateurs de surveiller en temps réel l'intégrité de leur réseau domestique et de prendre des mesures préventives pour protéger leurs données et leur vie privée contre les cyberattaques.

2. la réalisation de raspberry Pi

2.1. c’est quoi Le Raspberry Pi :

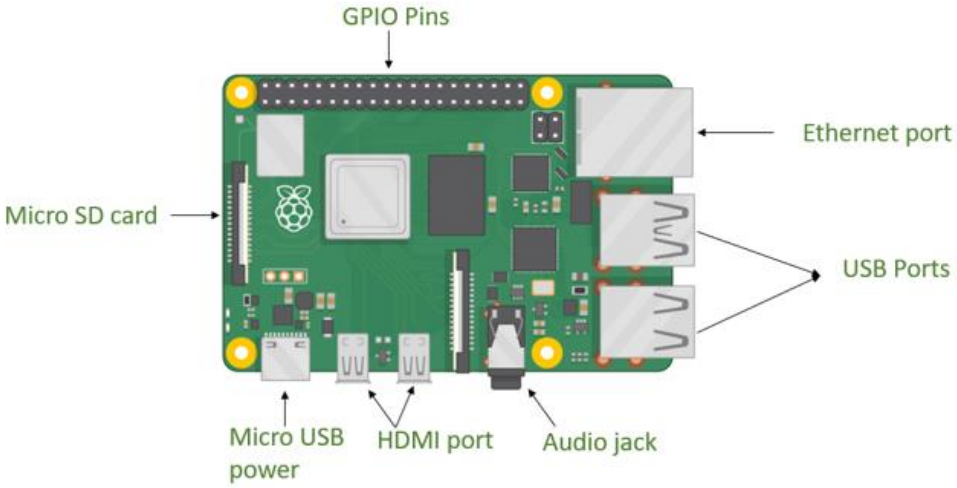
Le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM de la taille d'une carte de crédit conçu par des professeurs du département informatique de l'université de Cambridge dans le cadre de la fondation Raspberry Pi.

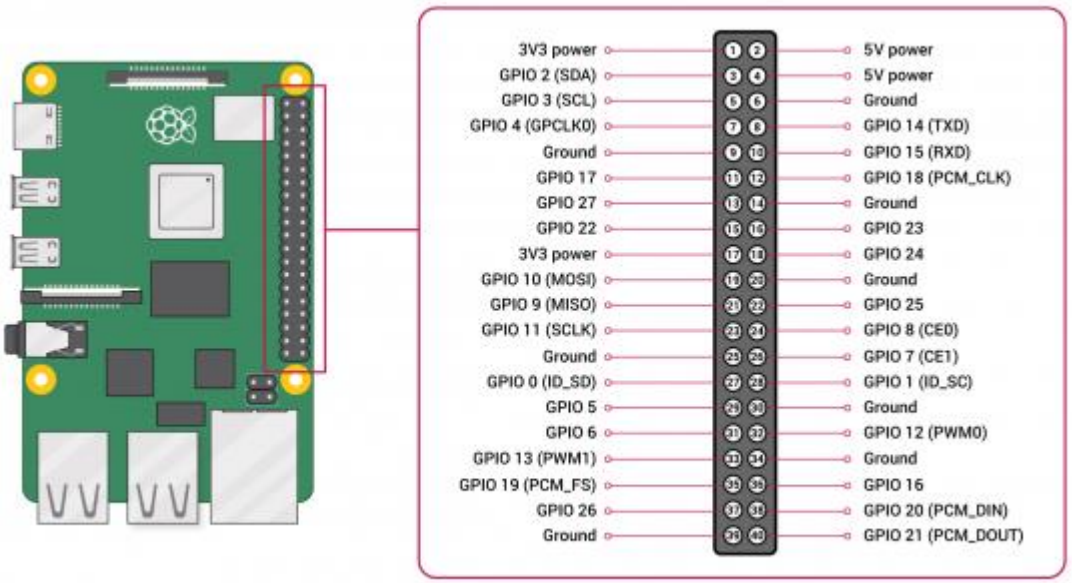
2.2. le choix de Raspberry Pi :

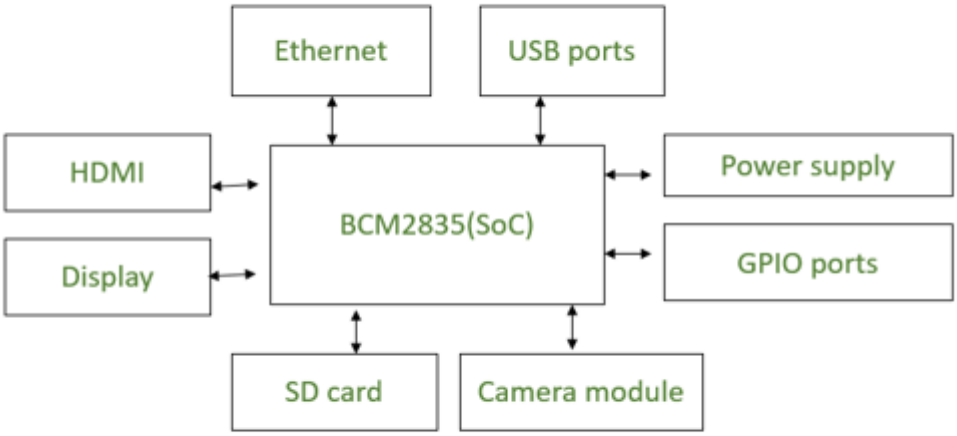
Puissance de calcul : Le Raspberry Pi a une puissance de calcul significative par rapport à l'Arduino à cause de processeur ARM. Il peut exécuter un système d'exploitation complet comme Linux, ce qui permet de faire fonctionner des applications plus complexes.
Polyvalence : Le Raspberry Pi peut être utilisé traitement de données, le streaming vidéo,

l'hébergement de serveur, etc. Il est idéal pour les projets qui nécessitent des fonctionnalités avancées ou une connectivité réseau.
Interfaces : Le Raspberry Pi dispose de diverses interfaces intégrées, telles que le Wi-Fi, le Bluetooth, les ports USB, HDMI, etc., ce qui le rend adapté à une large gamme de projets nécessitant une connectivité externe.
Prix : le Raspberry Pi offre une meilleure puissance de calcul pour son prix et peut être une meilleure option pour ce projets.
On va choisir pour ce projet le **raspberry pi model B**, car il contient une carte Bluetooth et Wi-Fi à l'intérieur, et donc on n'a pas besoin d'importer une carte wifi extérieur par USB.

2.3. Architecture de Raspberry Pi:


GPIO Pins
Ethernet port
Micro SD card
USB Ports
Micro USB power
HDMI port
Audio jack

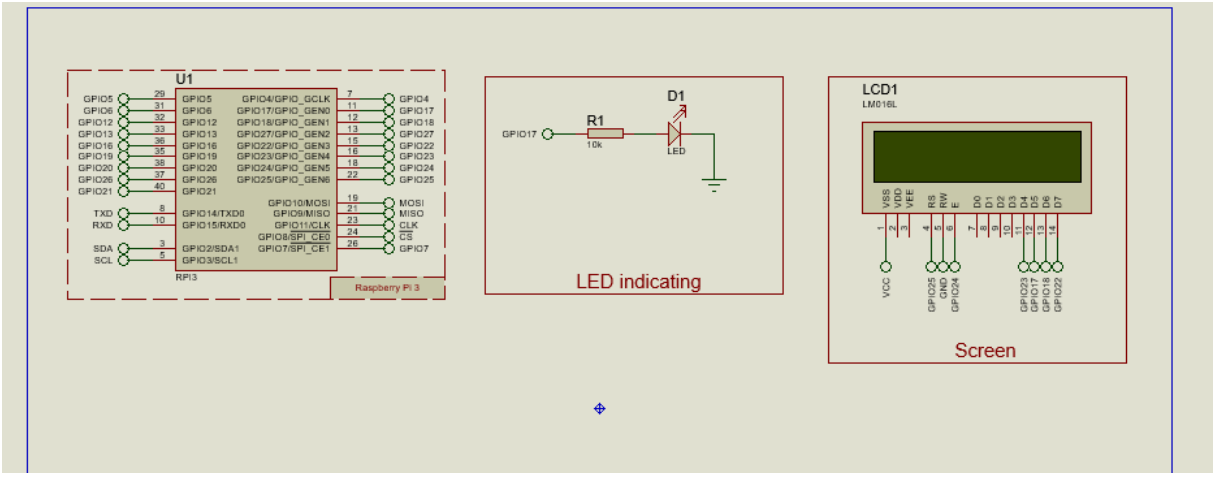

3V3 power
GPIO 2 (SDA)
GPIO 3 (SCL)
GPIO 4 (GPCLK0)
Ground
GPIO 17
GPIO 27
GPIO 22
3V3 power
GPIO 10 (MOSI)
GPIO 9 (MISO)
GPIO 11 (SCLK)
Ground
GPIO 0 (ID_SD)
GPIO 5
GPIO 6
GPIO 13 (PWM1)
GPIO 19 (PCM_FS)
GPIO 26
Ground
5V power
5V power
Ground
GPIO 14 (TXD)
GPIO 15 (RXD)
GPIO 18 (PCM_CLK)
Ground
GPIO 23
GPIO 24
Ground
GPIO 25
GPIO 8 (CE0)
GPIO 7 (CE1)
GPIO 1 (ID_SC)
Ground
GPIO 12 (PWM0)
Ground
GPIO 16
GPIO 20 (PCM_DIN)
GPIO 21 (PCM_DOUT)



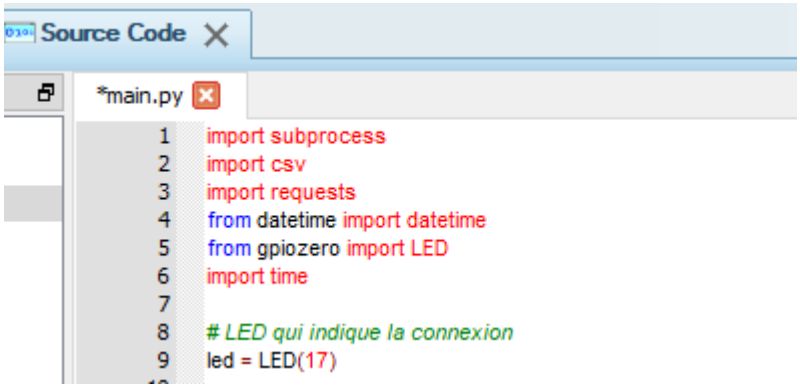
2.4. La simulation sur proteus 8 (ISIS) :

Proteus est un logiciel de simulation électronique qui permet de concevoir, simuler et tester des circuits électroniques avant leur réalisation physique.

a-partie hardware :



U1 : le raspberry PI
LED indicating : si le raspberry est connectés la LED va allumer
Screen : c'est pour voir l'état du raspberry
b- Partie programme :



En important ces modules, le code peut utiliser les fonctionnalités qu'ils offrent pour accomplir différentes tâches, telles que l'exécution de commandes système, la manipulation de fichiers CSV, l'envoi de requêtes HTTP, la gestion des dates et heures, le contrôle des broches GPIO, et la gestion du temps.
Subprocess : Ce module permet d'exécuter des commandes système et de gérer leurs entrées et sorties.
CSV : Ce module fournit des fonctionnalités pour lire et écrire des fichiers CSV (Comma Separated Values), il est utilisé pour stocker des données tabulaires.
requests : Ce module est utilisé pour envoyer des requêtes HTTP dans le but d'interagir avec des serveurs web. Il est souvent utilisé pour récupérer des données à partir d'API web ou pour effectuer des requêtes vers des sites web.
datetime : Ce module fournit des classes pour manipuler des dates et des heures en Python.
gpiozero : Ce module offre une interface simple pour contrôler les broches GPIO (General Purpose Input/Output) sur un Raspberry Pi.
time : Ce module fournit des fonctions pour travailler avec le temps en Python.

```
11
12  def connect_to_wifi(ssid, password):
13      print("Connecting to Wi-Fi...")
14      led.off()
15
16      time.sleep(5)
17
18      if "connected":
19          print("Wi-Fi connected!")
20          led.on()
21      else:
22          print("Failed to connect to Wi-Fi.")
23
```

Ce code définit une fonction nommée connect_to_wifi qui prend deux arguments : ssid (identifiant du réseau Wi-Fi) et password (mot de passe du réseau Wi-Fi). La fonction est conçue pour se connecter au réseau Wi-Fi .
time.sleep(5): Cette ligne met le programme en pause pendant 5 secondes. Cela peut être utilisé pour attendre un court laps de temps avant de poursuivre l'exécution du programme.

```
23
24  # Fonction qui capture le traffic reseau par tcpdump
25  def capture_traffic(interface, pcap_file):
26      command = ["sudo", "tcpdump", "-i", interface, "-w", pcap_file]
27      process = subprocess.Popen(command)
28      return process
```

Cette fonction (capture_traffic) est conçue pour capturer le trafic réseau à partir d'une interface réseau spécifiée et l'enregistrer dans un fichier au format pcap.
command = ["sudo", "tcpdump", "-i", interface, "-w", pcap_file]: Cette liste contient les éléments nécessaires pour exécuter la commande tcpdump via subprocess. Cette commande est construite de la manière suivante :
"**sudo**": Permet d'exécuter la commande en tant qu'administrateur.
"**tcpdump**": Le nom de la commande utilisée pour capturer le trafic réseau.
"-i", **interface**: Spécifie l'interface réseau à partir de laquelle capturer le trafic.
"-w", **pcap_file**: Spécifie le chemin du fichier pcap dans lequel enregistrer le trafic capturé.
process : subprocess.Popen(command): C'est exécution de la commande construite dans la variable command à l'aide de subprocess.Popen().

```
29
30  # PCAP to CSV file
31  def pcap_to_csv(pcap_file, csv_file):
32      with open(csv_file, 'w', newline='') as csvfile:
33          writer = csv.writer(csvfile)
34          writer.writerow(['Timestamp', 'Source IP', 'Destination IP', 'Protocol', 'Length'])
35          command = ["tcpdump", "-r", pcap_file, "-nn", "-tttt"]
36          process = subprocess.Popen(command, stdout=subprocess.PIPE)
37          for line in process.stdout:
38              line = line.decode("utf-8").strip()
39              parts = line.split(' ')
40              if len(parts) >= 7 and parts[1] != 'IP':
41                  timestamp = parts[0]
42                  source_ip = parts[2]
43                  destination_ip = parts[4]
44                  protocol = parts[5]
45                  length = parts[-1]
46                  writer.writerow([timestamp, source_ip, destination_ip, protocol, length])
47
```

En ajout une fonction vise à convertir le fichier pcap (Packet Capture) en un fichier CSV (Comma-Separated Values) contenant des informations sur les paquets capturés.
writer.writerow(['Timestamp', 'Source IP', 'Destination IP', 'Protocol', 'Length']): c'est pour écrire une ligne d'en-tête dans le fichier CSV.
command = ["tcpdump", "-r", pcap_file, "-nn", "-tttt"]: construction d'une commande pour lire le fichier pcap à l'aide de tcpdump, -nn sont utilisées pour afficher les adresses IP numériques, et -tttt est utilisé pour afficher le timestamp.

```
47
48  # envoyer le CSV to web API
49  def send_to_api(csv_file, api_url):
50      with open(csv_file, 'rb') as file:
51          files = {'file': file}
52          response = requests.post(api_url, files=files)
53          return response
```

Cette fonction prend un fichier CSV, l'envoie à une API web via une requête POST multipart et retourne la réponse de l'API. Cela permet d'automatiser l'envoi de données vers une API à partir de fichiers CSV
with open(csv_file, 'rb') as file: : c'est pour ouvrir le fichier CSV en mode lecture binaire ('rb') pour permettre la lecture de son contenu.

```
55
56  def main():
57      wifi_ssid = "your_wifi_ssid"
58      wifi_password = "your_wifi_password"
59      interface = "eth0"
60      pcap_file = "network_traffic.pcap"
61      csv_file = "network_traffic.csv"
62      api_url = "http://localhost/create/{id}"
63
64      try:
65
66          connect_to_wifi(wifi_ssid, wifi_password)
67
68          # Capture le traffic du reseau
69          print("Capturing network traffic. Press Ctrl+C to stop.")
70          capture_process = capture_traffic(interface, pcap_file)
71          capture_process.wait()
72
73          print("Converting PCAP file to CSV...")
74          pcap_to_csv(pcap_file, csv_file)
75
76          print("Conversion complete. Sending CSV file to API...")
77
78          response = send_to_api(csv_file, api_url)
79
80          print("Response from API:", response.status_code)
81          print("CSV file sent successfully.")
82
83      except KeyboardInterrupt:
84
85          print("\nKeyboardInterrupt detected. Stopping capture.")
86          capture_process.terminate()
87
88  if __name__ == "__main__":
89      main()
```

La fonction principale main() compris la connexion au Wi-Fi, la capture du trafic réseau, la conversion du fichier PCAP en fichier CSV, et l'envoi du fichier CSV à une API web.
Un bloc try est utilisé pour gérer les exceptions potentielles pendant l'exécution du programme.
Un bloc except KeyboardInterrupt est utilisé pour intercepter l'interruption du programme via Ctrl+C, auquel cas la capture du trafic est arrêtée.

3. modèle de machine learning

3.1. description général :

Le choix de l'algorithme Random Forest pour l'analyse des données de trafic réseau générées par différents types de malwares repose sur plusieurs raisons solides. Random Forest est un algorithme d'apprentissage automatique basé sur une approche d'ensemble, dans laquelle de nombreux arbres de décision sont construits et combinés pour produire un modèle plus robuste et plus précis.

La robustesse du Random Forest provient de sa capacité à réduire le surapprentissage, car chaque arbre de décision est construit sur un échantillon aléatoire de données, ce qui accroît la diversité du modèle global. Cette caractéristique est particulièrement utile dans le contexte de la détection de trafic malveillant, où les types de malwares et de comportements anormaux peuvent varier considérablement. En outre, Random Forest est capable de gérer des ensembles de données complexes avec des caractéristiques hétérogènes et peut également fournir des indications sur l'importance des différentes caractéristiques, ce qui peut être utile pour identifier les indicateurs les plus pertinents de comportement malveillant.

Dans le cadre de l'analyse de trafic réseau, ce module Random Forest peut être utilisé pour classifier le trafic comme bénin ou malveillant, ainsi que pour identifier des types spécifiques de malwares, tels que les attaques DDoS, les tentatives de scan de ports, ou les communications

3.2.développement du modèle :

Third Party Library Imports

```
import pandas as pd
import numpy as np
import sklearn
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import matplotlib.pyplot as plt
import seaborn as sns
import io
import glob
```

[1] ✓ 6.4s

Loading datasets into respective dataframes

```
gagfyt_df = pd.read_csv(r'C:\Users\imade\OneDrive\Bureau\IOT defender\Gagfyt.csv')
Hide_and_seek_df = pd.read_csv(r'C:\Users\imade\OneDrive\Bureau\IOT defender\Hide_and_seek.csv')
Kenjiro_df = pd.read_csv(r'C:\Users\imade\OneDrive\Bureau\IOT defender\Kenjiro.csv')
Linux_Hajime_df = pd.read_csv(r'C:\Users\imade\OneDrive\Bureau\IOT defender\Linux_Hajime_df.csv')
Mirai_df = pd.read_csv(r'C:\Users\imade\OneDrive\Bureau\IOT defender\Mirai.csv')
Muhstik_df = pd.read_csv(r'C:\Users\imade\OneDrive\Bureau\IOT defender\Muhstik.csv')
Okiru_df = pd.read_csv(r'C:\Users\imade\OneDrive\Bureau\IOT defender\Okiru.csv')
```

[2] ✓ 7.3s

```
C:\Users\imade\AppData\Local\Temp\ipykernel_15160\3073905347.py:3: DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or set low_memory=False.
  Kenjiro_df = pd.read_csv(r'C:\Users\imade\OneDrive\Bureau\IOT defender\Kenjiro.csv')
```

Cleannig malware sub-set dataframes

```
pdList = [gagfyt_df, Hide_and_seek_df, Kenjiro_df, Mirai_df, Muhstik_df,Okiru_df, Linux_Hajime_df]
df = pd.concat(pdList,ignore_index=True)
```

[3] ✓ 0.2s

oncatinating the column Traffic labeled

oncatinating the column Traffic labeled

```
df['Traffic_Labeled'] = np.where(df['Traffic_Labeled']!="Benign", df['Traffic_Labeled'] + df['Malware'], df['Traffic_Labeled'])


df.drop(["tunnel_parents   label   detailed-label", "Tunnel", "Malware","index"], axis =1, inplace = True)
```

[4] ✓ 0.6s Python

```
df.head()
```

[5] ✓ 0.0s Python

	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	...	conn_state	local_orig	local_resp	missed_bytes	history	orig_pkts
0	2019-09-20 22:27:19.984081920	CjnBrQ3lWRLcSTnJV6	192.168.1.195	4512	162.248.88.215	62336	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	C	0
1	2019-09-20 22:25:46.017224960	CHL0ex2ROJt8hD3up6	192.168.1.195	49327	162.248.88.215	62336	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	C	0
2	2019-09-20 22:25:34.246748928	C7bfRx4kiGSypeozCd	162.248.88.215	62336	192.168.1.195	614	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	^c	0
3	2019-09-20 22:26:06.340357120	CE7K0r1ua7NTzBHGDf	192.168.1.195	29776	162.248.88.215	62336	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	C	0
4	2019-09-20 22:23:52.564219904	CyIM264HX2dYzLJAI6	162.248.88.215	62336	192.168.1.195	3389	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	^c	0

5 rows × 21 columns

Data Pre Processing

```
df['Traffic_Labeled'] = df['Traffic_Labeled'].str.strip()
```

[6] ✓ 0.3s Python

```
df['Traffic_Labeled'].value_counts()
```

[7] ✓ 0.1s Python

```
Traffic_Labeled
Benign                                1885954
Malicious PartOfAHorizontalPortScan   1526588
Malicious Okiru                        356530
Malicious DDoS                           2476
Malicious Attack                          199
Malicious C&C-HeartBeat                   154
Malicious C&C                               7
Name: count, dtype: int64
```

+ Code + Markdown

Converting Originating endpoint's TCP/UDP port (or ICMP code) to Categorical

```
df["id.orig_p"]= df["id.orig_p"].values.astype(str)
df["id.resp_p"]= df["id.resp_p"].values.astype(str)
```

[8] ✓ 1.4s Python

```
df
```

[9] ✓ 3.4s Python

	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes	...	conn_state	local_orig	local_resp	missed_bytes	his
0	2019-09-20 22:27:19.984081920	CjnBrQ3IWRLcSTnJV6	192.168.1.195	4512	162.248.88.215	62336	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	
1	2019-09-20 22:25:46.017224960	CHL0ex2ROJt8hD3up6	192.168.1.195	49327	162.248.88.215	62336	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	
2	2019-09-20 22:25:34.246748928	C7bfRx4kiGSypeozCd	162.248.88.215	62336	192.168.1.195	614	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	
3	2019-09-20 22:26:06.340357120	CE7K0r1ua7NTzBHGDf	192.168.1.195	29776	162.248.88.215	62336	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	
4	2019-09-20 22:23:52.564219904	CyIM264HX2dYzLJAI6	162.248.88.215	62336	192.168.1.195	3389	tcp	NaN	NaN	NaN	...	OTH	NaN	NaN	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3771903	2018-07-25 18:25:12.036609792	Cf2JQI4Fk3in6TJJsg	192.168.100.111	29930	197.255.125.11	18246	udp	NaN	0 days 00:00:00.247373	106.0	...	SF	NaN	NaN	0	
3771904	2018-07-25 22:21:24.001981952	CuUxQ5I0YFXm7aurf	192.168.100.111	864	222.176.49.129	23	tcp	NaN	0 days	0.0	...	S0	NaN	NaN	0	
	2018-07-26															

Subsetting the raw data set due of Hardware Memory issue

```
df_Benign = df.loc[df['Traffic_Labeled'] == 'Benign']
df_Benign=df_Benign.loc[np.random.permutation(df_Benign.index)[:30000]]
df_Benign.reset_index(inplace=True)
df_PortScan = df.loc[df['Traffic_Labeled'] == 'Malicious PartOfAHorizontalPortScan']
df_PortScan=df_PortScan.loc[np.random.permutation(df_PortScan.index)[:20000]]
df_PortScan.reset_index(inplace=True)
df_Okiru = df.loc[df['Traffic_Labeled'] == 'Malicious Okiru']
df_Okiru = df_Okiru.loc[np.random.permutation(df_Okiru.index)[:8000]]
df_Okiru.reset_index(inplace=True)
df_Others = df[df['Traffic_Labeled'].isin(['Malicious DDoS', 'Malicious Attack',
                                           'Malicious C&C-HeartBeat', 'Malicious C&C'])]
pdList = [df_Benign, df_Okiru, df_PortScan, df_Others]
df = pd.concat(pdList,ignore_index=True)
df = df.drop('index',axis=1)
```

[10] ✓ 1.4s

```
df.shape
```

[11] ✓ 0.0s

```
(60836, 21)
```

New Variable - Classification Target

+ Code + Markdown

```
df.loc[df['Traffic_Labeled'] != 'Benign', 'Malicious'] = 1
df.Malicious=df.Malicious.fillna(0)
df.Malicious.value_counts()* 100 / len(df)
```

[12] ✓ 0.0s

```
Malicious
1.0    50.687093
0.0    49.312907
Name: count, dtype: float64
```

```
df.Malicious.value_counts()* 100 / len(df)
```
[13] ✓ 0.0s

```
Malicious
1.0    50.687093
0.0    49.312907
Name: count, dtype: float64
```

Null Values

```
df.isna().sum()* 100 / len(df)
```
[14] ✓ 0.0s

```
ts                   0.000000
uid                  0.000000
id.orig_h            0.000000
id.orig_p            0.000000
id.resp_h            0.000000
id.resp_p            0.000000
proto                0.000000
service             99.516734
duration            93.518640
orig_bytes          93.518640
resp_bytes          93.518640
conn_state           0.000000
local_orig         100.000000
local_resp         100.000000
missed_bytes         0.000000
history              0.941877
orig_pkts            0.000000
orig_ip_bytes        0.000000
resp_pkts            0.000000
resp_ip_bytes        0.000000
Traffic_Labeled      0.000000
Malicious            0.000000
dtype: float64
```

Dropping off Variable 'uid' Unique ID of Connection, present 100% unique values

```
df.drop(['local_resp','local_orig','uid'],axis=1,inplace=True)
```
[15] ✓ 0.0s

Transport Layer Protocol

```
df.proto.value_counts()
```
[16] ✓ 0.0s

```
proto
tcp     53511
udp      6752
icmp      573
Name: count, dtype: int64
```

Duration: Time of last packet seen – time of first packet seen

+ Code + Markdown

```
from datetime import time, timedelta, datetime
df.duration = pd.to_timedelta(df.duration)
df.duration = pd.to_timedelta(df['duration']).dt.total_seconds()
```
[17] ✓ 0.0s

```
df[['duration','orig_bytes','resp_bytes']]=df[['duration','orig_bytes','resp_bytes']].fillna
```
[18] ✓ 0.0s

```
df = df.fillna("Unknown")
```
[19] ✓ 0.0s

```
df.isna().sum()
```
[20] ✓ 0.0s

```
ts                  0
id.orig_h           0
id.orig_p           0
id.resp_h           0
id.resp_p           0
proto               0
service             0
duration            0
orig_bytes          0
resp_bytes          0
conn_state          0
missed_bytes        0
history             0
orig_pkts           0
orig_ip_bytes       0
resp_pkts           0
resp_ip_bytes       0
Traffic_Labeled     0
Malicious           0
dtype: int64
```

Variables with more than 90% unique values

```
df=df.drop(['id.resp_h','id.orig_h'],axis=1)
```
[21] ✓ 0.0s

```
Traffic_Labeled
Benign                                30000
Malicious PartOfAHorizontalPortScan   20000
Malicious Okiru                        8000
Malicious DDoS                         2476
Malicious Attack                        199
Malicious C&C-HeartBeat                 154
Malicious C&C                             7
Name: count, dtype: int64
```

```
df.to_csv('preprocessed_Labeled_IoT.csv', encoding='utf-8')
```
[23] ✓ 2m 42.8s

Feature Engineering

```
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn import preprocessing

df_dumie = df

def encode_text_dummy(df_dumie, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df_dumie[dummy_name] = dummies[x]
    df_dumie.drop(name, axis=1, inplace=True)


encode_text_dummy(df_dumie,"proto")

encode_text_dummy(df_dumie,"service")
```
[24] ✓ 0.0s

```
import zat
from sklearn.model_selection import train_test_split
from zat.log_to_dataframe import LogToDataFrame
from zat.dataframe_to_matrix import DataFrameToMatrix
print('zat: {:s}'.format(zat.__version__))
import pandas as pd
print('Pandas: {:s}'.format(pd.__version__))
import numpy as np
print('Numpy: {:s}'.format(np.__version__))
import sklearn
from sklearn.ensemble import IsolationForest
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
print('Scikit Learn Version:', sklearn.__version__)
```

[27] ✓ 0.1s

```
zat: 0.4.7
Pandas: 2.2.2
Numpy: 1.26.2
Scikit Learn Version: 1.3.2
```

```
labels = np.array(df_dumie['Malicious'])
features = df_dumie.drop(['Traffic_Labeled','Malicious'], axis = 1)
features = features.select_dtypes(exclude=['object'])
feature_list = list(features.columns)
```

[28] ✓ 0.0s

```
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.25, rando
```

[29] ✓ 0.0s

feature scaling

```
sc_X = StandardScaler()
train_features = sc_X.fit_transform(train_features)
test_features = sc_X.transform(test_features)
```

[30] ✓ 0.0s

```
df_dumie.info()
```

[25] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 60836 entries, 12563 to 53538
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ts               60836 non-null  object
 1   id.orig_p        60836 non-null  object
 2   id.resp_p        60836 non-null  object
 3   duration         60836 non-null  object
 4   orig_bytes       60836 non-null  object
 5   resp_bytes       60836 non-null  object
 6   conn_state       60836 non-null  object
 7   missed_bytes     60836 non-null  int64
 8   history          60836 non-null  object
 9   orig_pkts        60836 non-null  int64
 10  orig_ip_bytes    60836 non-null  int64
 11  resp_pkts        60836 non-null  int64
 12  resp_ip_bytes    60836 non-null  int64
 13  Traffic_Labeled  60836 non-null  object
 14  Malicious        60836 non-null  float64
 15  proto-icmp       60836 non-null  bool
 16  proto-tcp        60836 non-null  bool
 17  proto-udp        60836 non-null  bool
 18  service-Unknown  60836 non-null  bool
 19  service-dns      60836 non-null  bool
 20  service-http     60836 non-null  bool
 21  service-ssh      60836 non-null  bool
dtypes: bool(7), float64(1), int64(5), object(9)
memory usage: 7.8+ MB
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

[26] ✓ 0.1s

```
clf = RandomForestClassifier(n_estimators = 200, max_depth=20,)
```
[31] ✓ 0.0s

```
train_features[0]
```
[32] ✓ 0.0s

```
array([ 0.        ,  1.49132127,  1.09832459, -0.06735393, -0.06479325,
       -0.09615585,  0.36849844, -0.35227064,  0.0676726 , -0.0256503 ,
       -0.02690315, -0.05646304])
```

Training the model on the training dataset

```
clf.fit(train_features, train_labels)

# performing predictions on the test dataset
y_pred = clf.predict(test_features)

# metrics are used to find accuracy or error
from sklearn import metrics
print()

# using metrics module for accuracy calculation
print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(test_labels, y_pred))
```
[33] ✓ 1.0s

```
ACCURACY OF THE MODEL:  0.6342297323952922
```

```
import pickle
pickle.dump(clf, open('model.pkl', 'wb'))
```
[43]

4. Description des Composants de l'API :

```python
import pickle
import joblib
from fastapi import FastAPI,Path
from pydantic import BaseModel
from route2 import router
from ml_model import model2
app=FastAPI()
app.include_router(router)
#model = joblib.load("model.pkl")
#model = pickle.load(open('model.pkl', 'rb'))
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
uri = "mongodb+srv://admin:test@cluster0.9lip2wo.mongodb.net/?retryWrites=true&w=majority"
client = MongoClient(uri, server_api=ServerApi('1'))
try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
except Exception as e:
    print(e)
```

IA_model.py:

Ce fichier charge un modèle ML à partir d'un fichier model.pkl en utilisant joblib.

```python
import joblib

model2 = joblib.load("model.pkl")
```

baseclass.py:

Définit une classe BaseClass héritant de BaseModel de pydantic pour représenter des données de base pour l'API.

```python
class BaseClass(BaseModel):
    missed_bytes:int
    orig_pkts: int
    orig_ip_bytes: int
    resp_pkts: int
    resp_ip_bytes: int
    proto_icmp: int
    proto_tcp: int
    proto_udp:int
    service_Unknown: int
    service_dhcp: int
    service_dns: int
    service_ssh: int
```

RAdatabase2.py:

- Configure une connexion à une base de données MongoDB à l'aide de pymongo.
- Il semble utiliser cette connexion pour interagir avec une collection MongoDB nommée iot_collection.

```python
from fastapi import APIRouter
from pymongo import MongoClient

client=MongoClient("mongodb+srv://admin:test@cluster0.9lip2wo.mongodb.net/?retryWrites=true&w=majority")
db=client.iot_db
collection=db["iot_collection"]
```

filter.py:

- Traite un fichier CSV pour le prétraitement des données avant utilisation dans le modèle ML.
- Effectue des transformations et du nettoyage de données.

```
import pandas as pd
import numpy as np
from datetime import time, timedelta, datetime
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
df = pd.read_csv(r'C:\Users\imade\OneDrive\Bureau\IOT defender\Muhstik.csv')

df['Traffic_Labeled'] = np.where(df['Traffic_Labeled']!="Benign", df['Traffic_Labeled'] + df['Malware'], df['Tr

# dropping the unecessaty columns
df.drop(["tunnel_parents   label   detailed-label", "Tunnel", "Malware"], axis =1, inplace = True)


df['Traffic_Labeled'] = df['Traffic_Labeled'].str.strip()

df["id.orig_p"]= df["id.orig_p"].values.astype(str)
df["id.resp_p"]= df["id.resp_p"].values.astype(str)

df = df.drop('Traffic_Labeled', axis=1)

df.drop(['local_resp','local_orig','uid'],axis=1,inplace=True)

df.duration = pd.to_timedelta(df.duration)                                                    26
df.duration = pd.to_timedelta(df['duration']).dt.total_seconds()

df[['duration','orig_bytes','resp_bytes']]=df[['duration','orig_bytes','resp_bytes']].fillna

df = df.fillna("Unknown")

df.isna().sum()
df=df.drop(['id.resp_h','id.orig_h'],axis=1)
df.to_csv('preprocessed_Labeled_IoT1.csv', encoding='utf-8')

#dumie dataset
df_dumie = df

def encode_text_dummy(df_dumie, name):

    dummies = pd.get_dummies(df[name])

    for x in dummies.columns:

        dummy_name = f"{name}_{x}"
```

```
        df_dumie[dummy_name] = dummies[x]

    df_dumie.drop(name, axis=1, inplace=True)




encode_text_dummy(df_dumie, name: "proto")

encode_text_dummy(df_dumie, name: "service")

features = df_dumie
features = features.select_dtypes(exclude=['object'])
feature_list = list(features.columns)

print(features.columns)

sc_X = StandardScaler()
features = sc_X.fit_transform(features)

print(features)
print(features[0])
```

schema2.py:

- Fournit des fonctions pour sérialiser et désérialiser des objets IoT en dict et vice-versa.

```python
def seria(iot)-> dict:
    return {
        "id":str(iot["_id"]),
        #"ip": str(iot["ip"]),
        #"status": str(iot["status"]),
        "missed_bytes": int(iot["missed_bytes"]),
        "orig_pkts": int(iot["orig_pkts"]),
        "orig_ip_bytes": int(iot["orig_ip_bytes"]),
        "resp_pkts": int(iot["resp_pkts"]),
        "resp_ip_bytes": int(iot["resp_ip_bytes"]),
        "proto_icmp": int(iot["proto_icmp"]),
        "proto_tcp": int(iot["proto_tcp"]),
        "proto_udp": int(iot["proto_udp"]),
        "service_Unknown": int(iot["service_Unknown"]),
        "service_dhcp": int(iot["service_dhcp"]),
        "service_dns": int(iot["service_dns"]),
        "service_ssh": int(iot["service_ssh"])
    }
2 usages
def list_deseri(iots)-> list:
    return [seria(iot) for iot in iots]
```

route2.py:

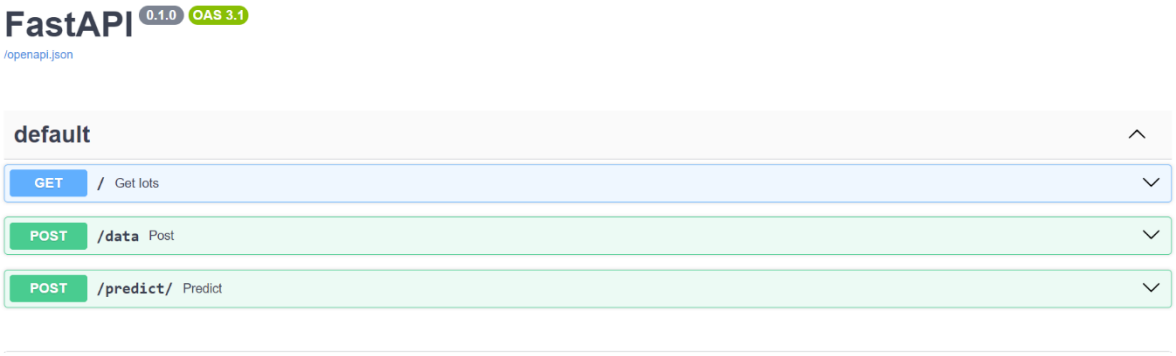
- Définit des routes spécifiques à l'API à l'aide de APIRouter de FastAPI.

- Inclut des endpoints pour récupérer des données IoT, poster de nouvelles données IoT, et effectuer des prédictions à l'aide du modèle ML chargé

```
from fastapi import FastAPI
C:\Users\User\pythonProject\RestApi.py
import numpy as np
from schema2 import list_deseri
from RAdatabase2 import collection
from fastapi import APIRouter,HTTPException
from baseclass import BaseClass
import RestApi
router =APIRouter()

@router.get("/")
async def get_iots():
    iots=list_deseri(collection.find())
    return iots

@router.post("/data")
def post(iot:BaseClass):
    collection.insert_one(dict(iot))

@router.post("/predict/")
def predict(item: BaseClass):
    try:
        features = np.array(item.features).reshape(1, -1)
        prediction = RestApi.model2.predict(features)
        return {"prediction": prediction.tolist()}
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))
```

Fonctionnement Global

- L'API permet d'interagir avec des données IoT stockées dans MongoDB.
- Les données sont prétraitées à l'aide du fichier filter.py avant d'être utilisées dans le modèle ML.
- Le modèle ML chargé (model2) est utilisé pour effectuer des prédictions sur de nouvelles données IoT.
- Les résultats sont renvoyés aux utilisateurs via les endpoints définis dans route2.py.
- La structure utilise pydantic pour la validation et la gestion des modèles de données



4. conclusion :

Les solutions proposées offrent un point de départ prometteur pour intégrer une logique de gestion de sécurité en temps réel (SEM) et des mécanismes d'ajustement destinés aux smart cities. L'utilisation de modèles d'apprentissage automatique pour analyser le trafic réseau peut contribuer à détecter rapidement les activités suspectes, renforçant ainsi la sécurité globale des systèmes interconnectés.

Ces solutions pourraient s'adapter aux environnements urbains en ajoutant des fonctionnalités supplémentaires comme l'injection de règles spécifiques au réseau, des contrôles de pare-feu, et des outils de surveillance de sécurité sophistiqués. En intégrant ces éléments, les smart cities pourraient bénéficier d'un niveau de sécurité accru, tout en offrant des services fiables et efficaces à leurs citoyens.

Ainsi, l'adoption de ces solutions pourrait représenter une avancée significative vers la sécurisation des infrastructures des villes intelligentes, tout en permettant d'anticiper et de contrer les menaces émergentes.