

Documentation sur l'avancement du TP : Sécurisation des objets

I – Scan des périphérique bluetooth :

On a utilisé la bibliothèque **bleak**.

Bleak est une bibliothèque Python pour communiquer avec des périphériques Bluetooth Low Energy (BLE) sur différentes plateformes. Elle offre une interface simple pour la recherche, la connexion et l'interaction avec ces périphériques, avec une documentation détaillée et une compatibilité multiplateforme.

```
async def device(mac):
    # scan for devices
    devices = await BleakScanner.discover()
    for d in devices:
        if d.address == mac:
            return d
    return None
```

Cette fonction, nous permet de faire la recherche des périphérique (discover()) et filtre la recherche en se basant sur l'adresse MAC passé en paramètre.

NB. Il faut avant tout importer les bibliothèques nécessaires :

```
import asyncio
from bleak import BleakClient, BleakScanner
```

asyncio : afin des faires des appels asynchrones.

II – Connection avec le périphérique :

```
async def main():
    d = await device('D1:6A:16:01:A2:EC')
    async with BleakClient(d.address) as client:
        if client.is_connected:
            ref = {}
            print('connected')
```

BleakClient, nous permet de se connecter à un périphérique bluetooth en se basant sur l'adresse MAC.

III –Affichage des services (API) sous forme d'un dictionnaire :

La forme souhaitée : UUID-nom du service: handle

```
if client.is_connected:
    ref = {}
    print('connected')
    services = await client.get_services()
    for service in services:
        for charac in service.characteristics:
            temp = str(str(charac) + " UNKNOWN").split()
            ref[temp[0] + '-' + temp[3]] = temp[1:][1][:2]
```

J'ajoute « UNKNOWN » pour les services sans nom. Exemple : les services 2,4,8...

(cheap solution but it works 😊).

Résultat:

```
{
  '00002a00-0000-1000-8000-00805f9b34fb-UNKNOWN': '2', '00002a01-0000-1000-8000-00805f9b34fb-UNKNOWN': '4', '00002a04-0000-1000-8000-00805f9b34fb-UNKNOWN': '6', '00002aa6-0000-1000-8000-00805f9b34fb-UNKNOWN': '8', '00002a05-0000-1000-8000-00805f9b34fb-UNKNOWN': '11', '8ec90003-f315-4f60-9fb8-838830daea50-UNKNOWN': '15', '00002a29-0000-1000-8000-00805f9b34fb-UNKNOWN': '19', '1b0d1201-a720-f7e9-46b6-31b601c4fca1-TRIP': '22', '1b0d1202-a720-f7e9-46b6-31b601c4fca1-TEMP': '26', '1b0d1203-a720-f7e9-46b6-31b601c4fca1-BATT': '30', '1b0d1204-a720-f7e9-46b6-31b601c4fca1-SOLAR': '34', '1b0d1205-a720-f7e9-46b6-31b601c4fca1-NRJ': '38', '1b0d1206-a720-f7e9-46b6-31b601c4fca1-STATUS': '42', '1b0d1207-a720-f7e9-46b6-31b601c4fca1-GPS': '46', '1b0d1208-a720-f7e9-46b6-31b601c4fca1-RESET': '50', '1b0d1301-a720-f7e9-46b6-31b601c4fca1-MEM': '54', '1b0d1302-a720-f7e9-46b6-31b601c4fca1-LIST': '58', '1b0d1303-a720-f7e9-46b6-31b601c4fca1-NUM': '62', '1b0d1304-a720-f7e9-46b6-31b601c4fca1-READ': '66', '1b0d1305-a720-f7e9-46b6-31b601c4fca1-SEND': '70', '1b0d1306-a720-f7e9-46b6-31b601c4fca1-DEL': '74', '1b0d1401-a720-f7e9-46b6-31b601c4fca1-MODE': '79', '1b0d1402-a720-f7e9-46b6-31b601c4fca1-FW': '83', '1b0d1403-a720-f7e9-46b6-31b601c4fca1-BL': '86', '1b0d1404-a720-f7e9-46b6-31b601c4fca1-RESET': '89', '1b0d1405-a720-f7e9-46b6-31b601c4fca1-STATE': '93', '1b0d1406-a720-f7e9-46b6-31b601c4fca1-TIME': '97', '1b0d1407-a720-f7e9-46b6-31b601c4fca1-ACK': '100', '1b0d1408-a720-f7e9-46b6-31b601c4fca1-PAUSE': '104', '1b0d1409-a720-f7e9-46b6-31b601c4fca1-USR': '108', '1b0d140a-a720-f7e9-46b6-31b601c4fca1-LOG': '111', '1b0d140b-a720-f7e9-46b6-31b601c4fca1-EVENT': '115', '00001503-0000-1000-8000-00805f9b34fb-NAME': '120', '00001504-0000-1000-8000-00805f9b34fb-COML': '123',
```

```
'00001505-0000-1000-8000-00805f9b34fb-COMH': '126', '00001506-0000-1000-8000-00805f9b34fb-HWVER': '129'
}
```

Maintenant ca sera plus facile d'appeler les api en se basant soit sur leur nom, UUID ou handle.

IV- Envoyer le payload du NUM vers LIST

- Lire la valeur de NUM :

```
for key in ref:
    if key.split('-')[-1] == 'NUM':
        await client.read_gatt_char(int(ref[key]))
```

- Publier sur LIST

```
for key in ref:
    if key.split('-')[-1] == 'LIST':
        await client.write_gatt_char(int(ref[key]), bytearray(b'\x17\x00'))
```

pour afficher la résultat, il faut définir une fonction **callback**, qui va se lancer quand le LOG se met à jour :

```
def getNotified(sender,data):
    print('here is your data sir')
    print(sender, data)
```

ensuite il faut utiliser la fonction `start_notify()` en passant le handle et le nom de la fonction du callback:

```
for key in ref:
    if key.split('-')[-1] == 'ACK':
        await client.start_notify(int(ref[key]), getNotified)
```

résultat :

[illegible]

Étape suivante :

Je pense qu'il faut trouver un service avec la possibilité d'écrire le dessus, afin d'envoyer le bytearray quand on vient de récupérer.

Chapitre 2 : J'ai les fichiers

I- Lister les fichiers

```
for key in ref:
    if key.split('-')[-1] == 'LIST':
        await client.start_notify(int(ref[key]), callBackForList)
```

dans un premier temps on va se 'subscribe' à LIST et on défininie une fonction de callback pour se lancer quand il aura des nouveauté dans le services LIST.

```
def callBackForList(sender,data):
    # name, size, hash, createDate = struct.unpack('<8sI4sQ', data)
    # hash = '0x' + hash[::-1].hex()
    # fs_file_info = FsFileInfo(name=name, size=size, hash=hash,
createDate=createDate)
    # print(sender, fs_file_info)
    print(data)
```

les lignes commentés ont été utilisé pour formater l'affichages de LIST en utilisant **struct**. La LIST contient des informations sur chaque fichiers dans le périphérique (Nom,Size,Hash,DateCréation).

Output :

```
bytearray(b'T10\x00\x00\x00\x00_\x08\x00\x00n\xe1\x0b\xfefH\x08er\xdc\x00\x00\x00')
bytearray(b'T11\x00\x00\x00\x00\x00\xcc\x07\x00\x00\x0c9Y\x01\xf0\xf0ier\xdc\x00\x00\x00')
bytearray(b'T13\x00\x00\x00\x00\x00\xfd\x07\x00\x00\xf2\x0dZ9\xb0\xc7er\xdc\x00\x00\x00')
bytearray(b'T7639\x00\x00\x00\n\x03\x00\x00rYM\x1f\x88!fr\xdc\x00\x00\x00')
```

II- Récupération des fichiers

Afin de recevoir les fichiers il faut envoyer dans l'API **READ**, des messages avec la forme suivante (en bytearray) :

Nom de fichier - Début du chunk - Fin du chunk

Exemple, fichier T13 :

```
for key in ref:
    if key.split('-')[-1] == 'READ':
        await client.write_gatt_char(int(ref[key]),
bytearray(b'T13\x00\x00\x00\x00\x00\x00\x00\x00\x00\x27\x0F\x00\x00'))
```

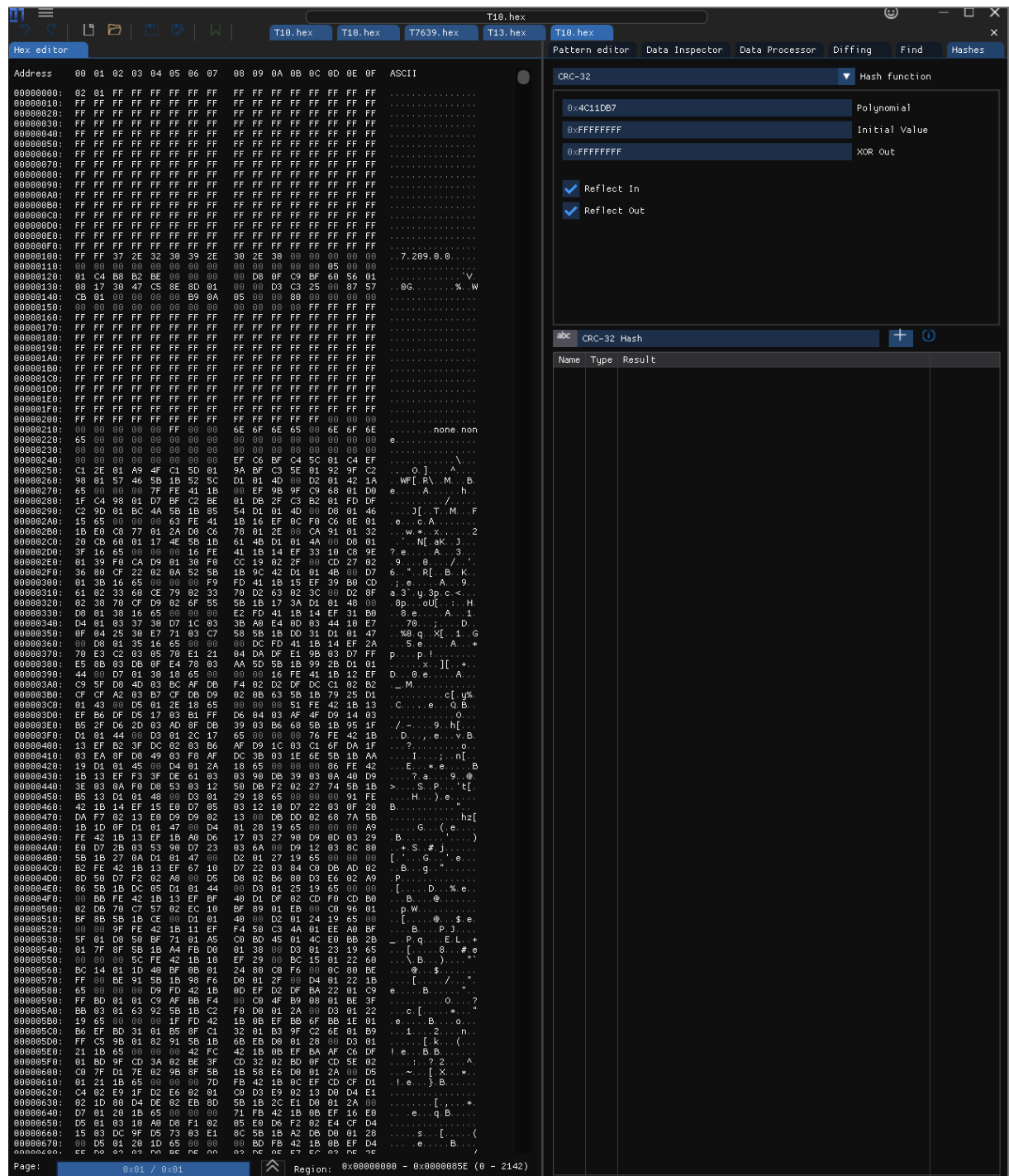
sachant que nous sommes déjà à l'écoute de READ, on écrit le résultat dans un fichier .hex

```
def getNotified(sender, data):
    file_path = "T13.hex"

    with open(file_path, "ab") as file:
        file.write(data)
```

on ouvre les fichiers dans ImHex pour vérifier les fichiers dans un premier temps.

Exemple fichier T10 :



Ca semble bien dans un premier temps, mais il faut impérativement vérifier le **HASH** pour être sûr que c'est le fichier qu'on cherche.

On écrit un simple script python pour ça :

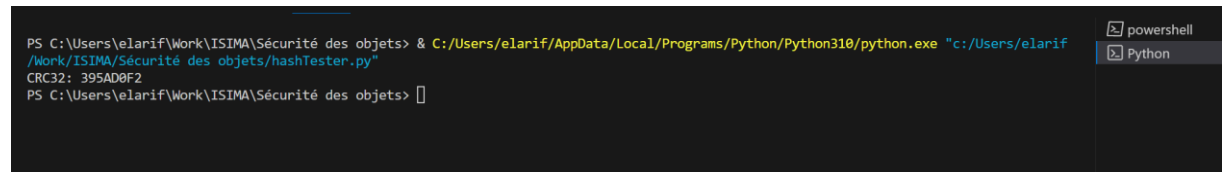
```
import binascii

def calculate_crc32(filename):
    try:
        with open(filename, 'rb') as f:
            buf = f.read()
```

```
        return binascii.crc32(buf)
    except FileNotFoundError:
        return "Le fichier n'a pas été trouvé."

filename = "T13.hex"
print(f"CRC32: {calculate_crc32(filename):08X}")
```

Résultat (fichier T13) :



The screenshot shows a PowerShell terminal window with the following text:

```
PS C:\Users\elarif\Work\ISIMA\Sécurité des objets> & C:/Users/elarif/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/elarif/Work/ISIMA/Sécurité des objets/hashTester.py"
CRC32: 395AD8F2
PS C:\Users\elarif\Work\ISIMA\Sécurité des objets> 
```

On the right side of the terminal window, there are two tabs: "powershell" and "Python", with "Python" being the active tab.

Ce qui est juste.

Conclusion :

Maintenant quand a les binaires contenant l'historique des utilisateur (GPS par exemple), Il nous reste seulement de faire un décodage des fichiers. (c'est pour la semaine prochaine :/)