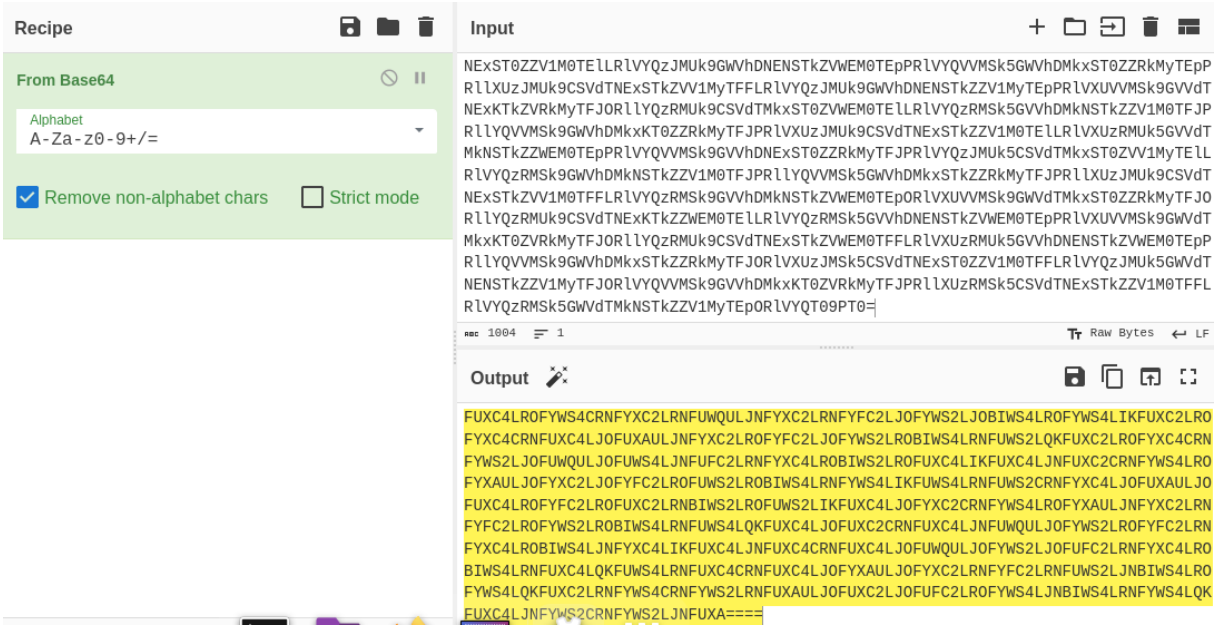
Writeups de la compétition LoveCTF, 1ère édition
Team : W3hunters
Catégorie du challenge : Crypto
Titre du challenge : Mise à l'épreuve

Une vraie mise à l'épreuve ce challenge. Un fichier txt a été soumis à notre réflexion.

Il s'agit de toute évidence d'une séquence encodée en base 64. Nous nous orientons dons vers Cyberchef pour le décoder.



Après décodage de base64, on obtient du base32 qu'on décode également.

From Base32

Alphabet
A-Z2-7=

Remove non-alphabet chars

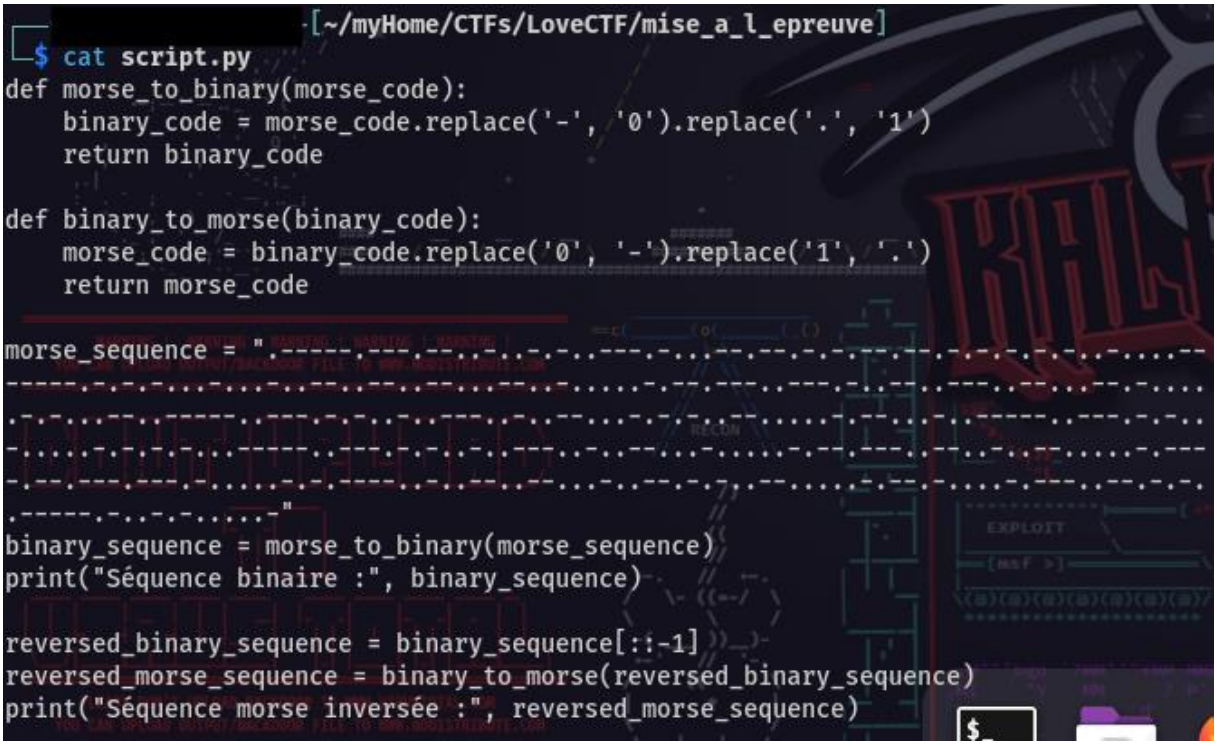
```
FUXC4LROFYWS4CRNFYXC2LRNFUWQULJNFYXC2LRNFYFC2LJOFYWS2LJOBIWS4LROFYWS4LIKFUXC2LRO
FYXC4CRNFUXC4LJOFUXAULJNFYXC2LROFYFC2LJOFYWS2LROBIWS4LRNFUWS2LQKFUXC2LROFYXC4CRN
FYWS2LJOFUWQULJOFUWS4LJNFUFC2LRNFYXC4LROBIWS2LROFUXC4LIKFUXC4LJNFUXC2CRNFYWS4LRO
FYXAULJOFYXC2LJOFYFC2LROFUWS2LROBIWS4LRNFYWS4LIKFUWS4LRNFUWS2CRNFYXC4LJOFUXAULJO
FUXC4LROFYFC2LROFUXC2LRNBIWS2LROFUWS2LIKFUXC4LJOFYXC2CRNFYWS4LROFYXAULJNFYXC2LRN
FYFC2LROFYWS2LROBIWS4LRNFUWS4LQKFUXC4LJOFUXC2CRNFUXC4LJNFUWQULJOFYWS2LROFYFC2LRN
FYXC4LROBIWS4LJNFYXC4LIKFUXC4LJNFUXC4CRNFUXC4LJOFUWQULJOFYWS2LJOFUFC2LRNFYXC4LRO
BIWS4LRNFUXC4LQKFUWS4LRNFUXC4CRNFUXC4LJOFYXAULJOFYXC2LRNFYFC2LRNFUWS2LJNBIWS4LRO
FYWS4LQKFUXC2LRNFYWS4CRNFYWS2LRNFUXAULJOFUXC2LJOFUFC2LROFYWS4LJNBIWS4LRNFYWS4LQK
FUXC4LJNFYWS2CRNFYWS2LJNFUXA====
```

752 1 Raw Bytes LF

Output

```
-.....-.
-..-.---
--..-.-.
--.---.
-....-.-
-.-.....
--..-.-.
--..-...
--..--..
-..----.
-.-.....
```

Quand nous décodons le base 32, on obtient un output qui ressemble à du pur morse, on se lance à chercher comment décoder le morse mais tous les outils en ligne n'ont donné aucune piste sérieuse. Et bam bim bam, l'idée était venue. Et si c'était que chaque '-' est un '0' et chaque '.' un '1' et vice versa peut-être. Ce serait donc du binaire ? Essayons voir.



```
[~/myHome/CTFs/LoveCTF/mise_a_l_epreuve]
$ cat 4_script.py
def ajouter_espaces(texte, taille_bloc=8):
    blocs = [texte[i:i+taille_bloc] for i in range(0, len(texte), taille_bloc)]
    return ' '.join(blocs)

# Exemple d'utilisation
sequence_binaire = "01111101011010000011010100110001011110100101111100110101001101
10011001101100001010111110100010001001000010111110011011001100010010111110111001101
10001101101010001100000111010101011111011010100011000001101110010111110011010101110
01101100011011010100011000001100111010111110100111001100011001101000110001001011111
01100111001100110011011101110101010000000111101101010101001001010100100111010001 1
01011011001000100001"
sequence_avec_espaces = ajouter_espaces(sequence_binaire)
print("Séquence avec un espace entre chaque bloc de 8 caractères :\n", sequence_ave
c_espaces)

[~/myHome/CTFs/LoveCTF/mise_a_l_epreuve]
$ python3 4_script.py
Séquence avec un espace entre chaque bloc de 8 caractères :
 01111101 01101000 00110101 00110001 01111010 01011111 00110101 00110111 00110011 0
1100001 01011111 01000100 01001000 01011111 00110110 01100010 01011111 01110011 011
00011 01101010 00110000 01110101 01011111 01101010 00110000 01101110 01011111 00110
101 01110011 01100011 01101010 00110000 01100111 01011111 01001110 01100011 0011010
0 01100010 01011111 01100111 00110011 00110111 01110101 01000000 01111011 01010101
01001001 01010010 01110100 01101011 01100100 01000001
```

Donc nous avons deux possibilités de séquences binaires. Nous écrivons donc un script pour traduire le binaire en texte.

```
[~/myHome/CTFs/LoveCTF/mise_a_l_epreuve]
$ cat 2_script.py
def binary_to_text(binary_string):
    # Divise la chaîne binaire en octets de 8 bits
    octets = [binary_string[i:i+8] for i in range(0, len(binary_string), 8)]

    # Convertit chaque octet en un caractère ASCII et les concatène
    text = ''.join(chr(int(octet, 2)) for octet in octets)

    return text

# Exemple d'utilisation
binary_string = "01111101 01101000 00110101 00110001 01111010 01011111 00110101 001
10111 00110011 01100001 01011111 01000100 01001000 01011111 00110110 01100010 01011
111 01110011 01100011 01101010 00110000 01110101 01011111 01101010 00110000 0110111
0 01011111 00110101 01110011 01100011 01101010 00110000 01100111 01011111 01001110
01100011 00110100 01100010 01011111 01100111 00110011 00110111 01110101 01000000 01
111011 01010101 01001001 01010010 01110100 01101011 01100100 01000001"
text = binary_to_text(binary_string.replace(" ", ""))
print("Texte converti :", text)

[~/myHome/CTFs/LoveCTF/mise_a_l_epreuve]
$ python3 2_script.py
Texte converti : }h51z_573a_DH_6b_scj0u_j0n_5scj0g_Nc4b_g37u@{UIRtkdA
```

AAAAAAAAH ! L'une des deux séquences binaires traduites en caractères ASCII semble être une piste pour obtenir le flag.

Ce résultat a d'abord l'air inversé, nous l'inversons à nouveau.

```
└─$ python3
Python 3.11.8 (main, Feb  7 2024, 21:52:08) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> sequence = "}h51z_573a_DH_6b_scj0u_j0n_5scj0g_Nc4b_g37u@{UIRtkdA"
>>> print (sequence[::-1])
AdktRIU{@u73g_b4cN_g0jcs5_n0j_u0jcs_b6_HD_a375_z15h}
>>>
```

A cette étape, nous sommes à 200% sûrs d'être sur la bonne voie, il reste comment déchiffrer cette séquence. Pourquoi ne pas essayer dcode.fr ? Dcode nous parle donc de chiffre ROT, on essaie de déchiffrer avec et dans la liste de propositions voilà le coquin FLAG !!!!