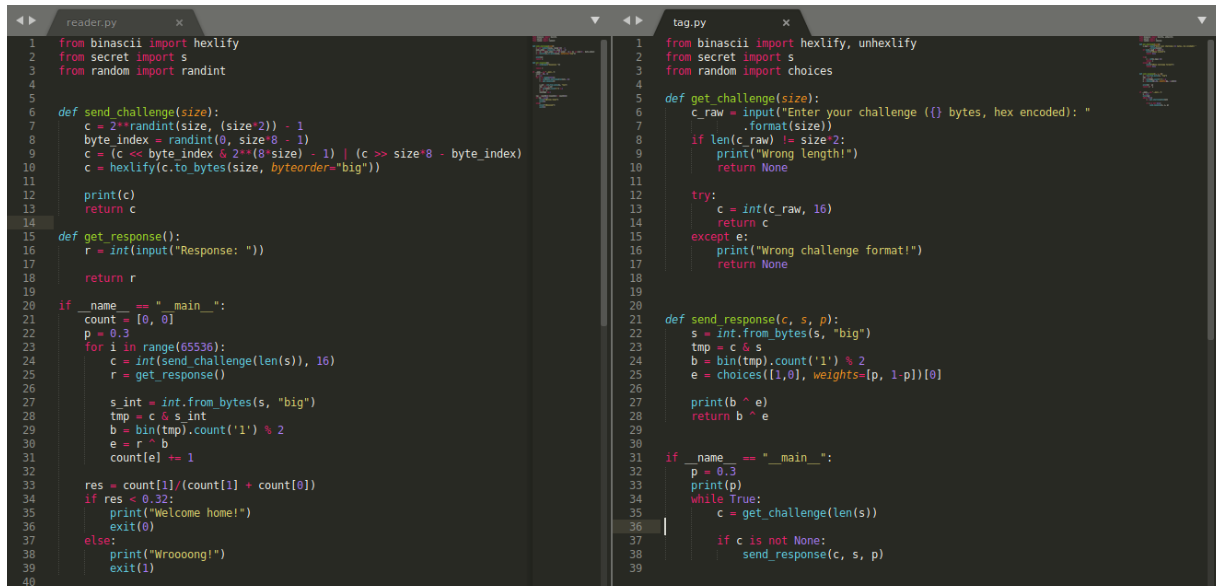
CTF Writeup : LPN - Active (Crypto)

Le challenge commence en nous fournissant une adresse IP et une archive contenant deux programmes python nommés reader.py et tag.py :



Pour commencer, nous nous connectons à l'adresse IP pour voir ce qui s'y trouve. Un prompt nous demande alors un challenge. D'après les fichiers python, nous comprenons que reader.py est le code du serveur et tag.py celui du client. De plus, il semble contenir une information intéressante : "'from secret import s"' Le but du challenge est donc probablement de récupérer cette valeur s.

Intéressons-nous maintenant au code du client qui semble modifier l'entrée utilisateur et effectuer des opérations avec la valeur secrète :

```
tmp = c & s
b = bin(tmp).count('1') \% 2
e = choices([1,0], weights=[p, 1-p])[0]
print(b ^ e)
return b ^ e
```

Notre entrée est d'abord utilisée dans un XOR avec le secret. Ensuite le programme compte le nombre de bits à 1 dans la valeur binaire du résultat du XOR. Si ce nombre est pair, b vaudra 0, sinon b vaudra 1. Une variable e est alors initialisée et sa valeur est probabiliste. La valeur de p dans la fonction main nous indique que e a 67% de chance de valoir 0 et 33% de valoir 1. Enfin la fonction retourne le XOR de b et de e.

Le secret s est donc uniquement caché par une randomisation de bits suivant une certaine probabilité ! Rien d'insurmontable !

Pour la résolution du challenge, nous suivons le raisonnement suivant : si par exemple nous passons en entrée du programme une valeur contenant uniquement des bits à 0, alors pour chaque bit de s, si ce bit vaut 0, la fonction retournera 0 (0 XOR 0) avec une probabilité de 67%, et 1 (0 XOR 1) avec une probabilité de 33%. S'il vaut 1, les probabilités sont inversées.

Par conséquent, en passant un grand nombre de fois la même entrée, et en observant quel bit est retourné avec une probabilité proche de 67%, il est possible de déduire la valeur du bit du secret utilisé. En effectuant cette opération pour chaque indice de bit, on récupère alors tout le secret :

```
out : 01000111010010000011000100111001011110110101010001110010011010010111011001
10100101100001011011000101111101100001011000110111010001101001011101100110010101
01111101100001011101000111010001100001011000110110101101110011010111110011101000
1001110010100001111101
```

Ce qui nous donne en ASCII notre fameux secret :
GH19{Trivial_active_attacks_:'(}