Hashing Algorithm

Auteur: bWlrYQ

1. Recon

On nous présente un algorithme de hash "maison", ça sent déjà mauvais... On a à notre disposition deux éléments, la fonction de hash utilisée ainsi que la sortie de notre script qui contient le hash ainsi qu'un salt.

L'utilisation d'un sel est une bonne idée, elle rend le hash beaucoup plus difficile à casser mais il faut éviter de le donner en temps normal... Etant donné que le code source est disponible on peut essayer de l'analyser un peu

2. Analyse du code source

```
#!/usr/bin/python
# Author: bWlrYQ

from sys import argv
from time import time
from base64 import b64encode
from string import printable

def main():
    try:
        if(argv[1] == "-c" or argv[1] == "--cleartext"):
            try:
                hash_result = []
                salt = int(time())
                cleartext = argv[2]
                for char in cleartext:
                    if(char in printable):
                        continue
                    else:
                        print["[!] Cleartext should only contain printable
characters"]
                        return
                for char in cleartext:
                    cipherChar = ord(char)
                    cipherChar = cipherChar * pow(cipherChar,2) + (salt-
cipherChar)
                    hash_result.append(cipherChar)
                hash_str=""
                for i in range(len(hash_result)):
                    if(i == len(hash_result)-1):
                        hash_str += str(hash_result[i])
                    else:
                        hash_str += str(hash_result[i])+","
                cipher = b64encode(hash_str.encode('utf-8'))
                print(f"[>] Salt: {salt}")
```

```
                print(f"[>] Hash: {cipher}")
            except:
                print("[!] Please specify a cleartext to hash")
        else:
            print("[!] Unrecognized argument")
            return
    except:
        print("[!] Please specify text to hash with argument -c or --cleartext")
        return
main()
```

Dans un premier temps on remarque qu'il y a beaucoup de choses "inutiles", on va donc essayer de nettoyer la source pour garder uniquement ce qui nous intéresse pour comprendre l'algorithme.

```
from time import time
from base64 import b64encode

hash_result = []
salt = int(time())
cleartext = "cleartext ici"
for char in cleartext:
    cipherChar = ord(char)
    cipherChar = cipherChar * pow(cipherChar,2) + (salt-cipherChar)
    hash_result.append(cipherChar)
hash_str=""
for i in range(len(hash_result)):
    if(i == len(hash_result)-1):
        hash_str += str(hash_result[i])
    else:
        hash_str += str(hash_result[i])+","
cipher = b64encode(hash_str.encode('utf-8'))
print(f"[>] Salt: {salt}")
print(f"[>] Hash: {cipher}")
```

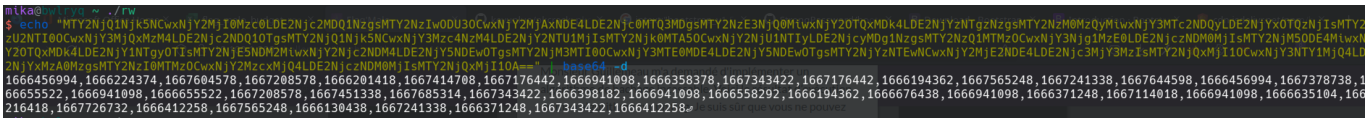
On remarque que chaque caractère de notre cleartext passe au travers de la méthode ord() qui renvoie le code ascii du caractère (donc un entier), ce résultat est ensuite utilisé pour réaliser une petite opération mathématique basée sur le salt (qui est juste un timestamp epoch).

On ajoute ensuite le résultat de cette opération à une liste, chaque caractère de notre cleartext devient donc un entier d'une taille raisonnable.

Une fois fait, on récupére chaque élement de notre liste et on les met dans un string en les séparant d'une virgule, puis on encode le tout en base64.

3. Casser l'algorithme

Le base64 est ici inutile, il sert juste à passer notre liste en une chaîne de caractères, on va donc retourner à l'état de liste.



Pour chaque élément de notre liste, on doit maintenant réaliser l'opération mathématique inverse à celle qui a été faite dans l'algorithme, pour rappel nous avons `entier_liste[i] = x * x² + (1666083818-x)` avec x le résultat de ord(char).

Pour résoudre les quelques équations, on peut utiliser la librarie sympy (parfaite pour ceux qui sont mauvais en maths comme moi et qui n'ont pas envie de réfléchir). Une implémentation mathématique à la main est aussi possible mais j'ai trop la flemme.

```
from sympy import symbols, solve
hash_list =
[1666456994,1666224374,1667604578,1667208578,1666201418,1667414708,1667176442,1666
941098,1666358378,1667343422,1667176442,1666194362,1667565248,1667241338,166764459
8,1666456994,1667378738,1666655522,1666941098,1666655522,1667208578,1667451338,166
7685314,1667343422,1666398182,1666941098,1666558292,1666194362,1666676438,16669410
98,1666371248,1667114018,1666941098,1666635104,1666216418,1667726732,1666412258,16
67565248,1666130438,1667241338,1666371248,1667343422,1666412258]
flag = "FMCTF{"
x = symbols('x')
for i in hash_list:
    expr = x * pow(x,2) + (1666083818-x) - i
    flag+=chr(solve(expr)[0])
print(flag+"}")
```

Et voilà !