HACKIT CTF 2018 WRITE-UP

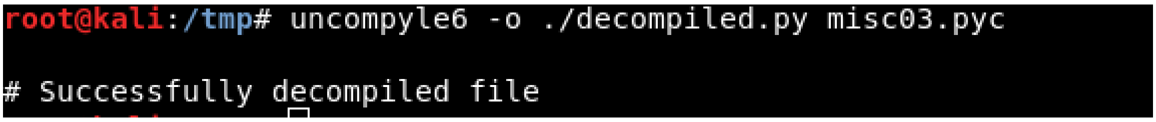
Misc Challenges – Trap-O-Saur

Challenge Description

> You have two choices: Either use your hard earned leet skill indefinitely or be calm and think deep. One of them will yield you the path to flag.
>
> misc03.pyc - attached

Solution

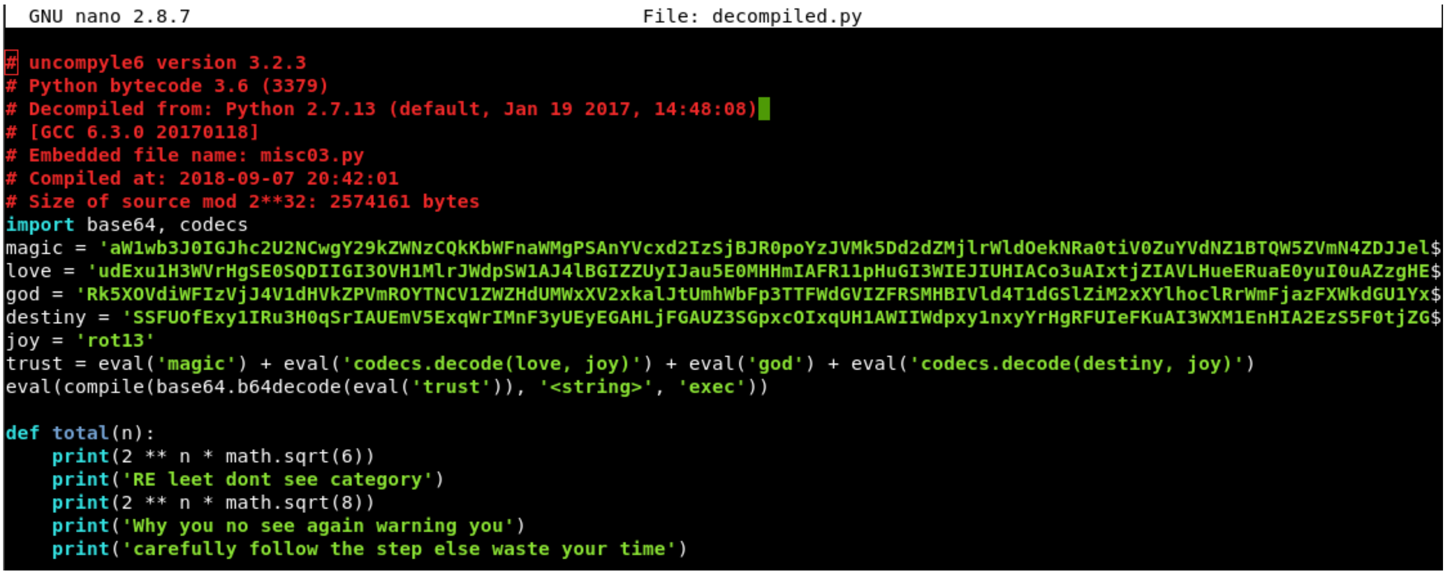
We have got a compiled python program. The first step was to decompile it using ***Uncompyle6*** tool:



```
root@kali:/tmp# uncompyle6 -o ./decompiled.py misc03.pyc

# Successfully decompiled file
```

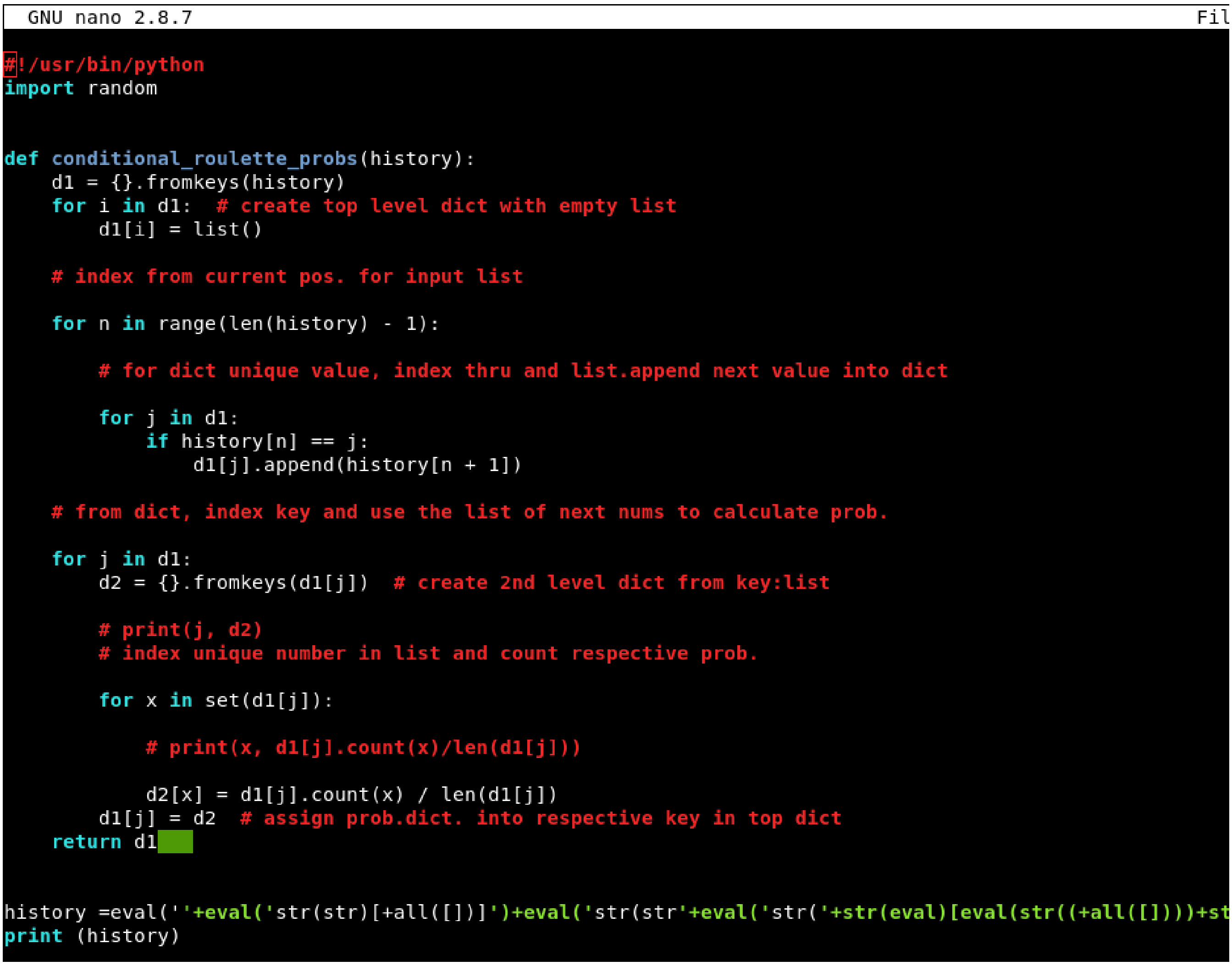
> Uncompyle6 is a Native Python cross-version decompiler and fragment decompiler. The successor to decompyle, uncompyle, and uncompyle2

The resulting code after the decompiling is the following:



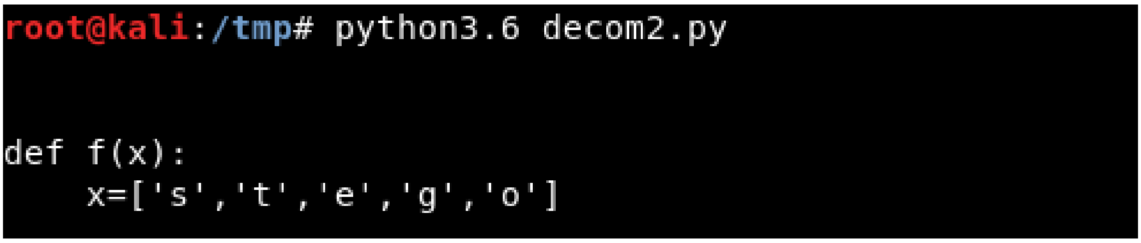
```
  GNU nano 2.8.7                                        File: decompiled.py

# uncompyle6 version 3.2.3
# Python bytecode 3.6 (3379)
# Decompiled from: Python 2.7.13 (default, Jan 19 2017, 14:48:08)
# [GCC 6.3.0 20170118]
# Embedded file name: misc03.py
# Compiled at: 2018-09-07 20:42:01
# Size of source mod 2**32: 2574161 bytes
import base64, codecs
magic = 'aW1wb3J0IGJhc2U2NCwgY29kZWNzCQkKbWFnaWMgPSAnYVcxd2IzSjBJR0poYzJVMk5Dd2dZMjlrWldOekNRa0tiV0ZuYVdNZ1BTQW5ZVmN4ZDJJel$
love = 'udExu1H3WVrHgSE0SQDIIGI3OVH1MlrJWdpSW1AJ4lBGIZZUyIJau5E0MHHmIAFR11pHuGI3WIEJIUHIACo3uAIxtjZIAVLHueERuaE0yuI0uAZzgHE$
god = 'Rk5XOVdiWFIzVjJ4V1dHVkZPVmROYTNCV1ZWZHdUMWxXV2xkalJtUmhWbFp3TTFWdGVIZFRSMHBIVld4T1dGSlZiM2xXYlhoclRrWmFjazFXWkdGU1Yx$
destiny = 'SSFUOfExy1IRu3H0qSrIAUEmV5ExqWrIMnF3yUEyEGAHLjFGAUZ3SGpxcOIxqUH1AWIIWdpxy1nxyYrHgRFUIeFKuAI3WXM1EnHIA2EzS5F0tjZG$
joy = 'rot13'
trust = eval('magic') + eval('codecs.decode(love, joy)') + eval('god') + eval('codecs.decode(destiny, joy)')
eval(compile(base64.b64decode(eval('trust')), '<string>', 'exec'))

def total(n):
    print(2 ** n * math.sqrt(6))
    print('RE leet dont see category')
    print(2 ** n * math.sqrt(8))
    print('Why you no see again warning you')
    print('carefully follow the step else waste your time')
```

After analyzing the code, we tried extract the base64 and encrypted data and by taking the resulting object code after compilation function of ***trust*** variable we successfully extracted the base64 encoded data and after decoding the result, we had the following code:



```
  GNU nano 2.8.7                                                                                        Fil

#!/usr/bin/python
import random


def conditional_roulette_probs(history):
    d1 = {}.fromkeys(history)
    for i in d1:  # create top level dict with empty list
        d1[i] = list()

    # index from current pos. for input list

    for n in range(len(history) - 1):

        # for dict unique value, index thru and list.append next value into dict

        for j in d1:
            if history[n] == j:
                d1[j].append(history[n + 1])

    # from dict, index key and use the list of next nums to calculate prob.

    for j in d1:
        d2 = {}.fromkeys(d1[j])  # create 2nd level dict from key:list

        # print(j, d2)
        # index unique number in list and count respective prob.

        for x in set(d1[j]):

            # print(x, d1[j].count(x)/len(d1[j]))

            d2[x] = d1[j].count(x) / len(d1[j])
        d1[j] = d2  # assign prob.dict. into respective key in top dict
    return d1


history =eval(''+eval('str(str)[+all([])]')+eval('str(str'+eval('str('+str(eval)[eval(str((+all([]))+st
print (history)
```

As we notice the ***history*** variable has an obfuscated value. And the result of ***history*** was the following function which leads us to the final inclusion.

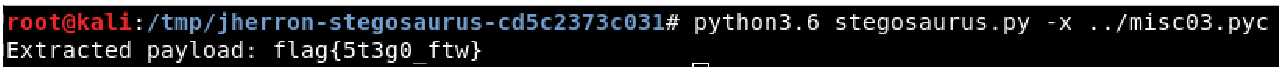


```
root@kali:/tmp# python3.6 decom2.py


def f(x):
    x=['s','t','e','g','o']
```

From the content of the array "***stego***", and after thinking deep enough. We managed to connect the strings of the word stego to the challenge's title, which is ***Stegosaur***.

> Stegosaur is a steganography tool that allows embedding arbitrary payloads in Python bytecode (pyc or pyo) files.
> It also can extract embedded bytecodes.

Looking to ***Stegosaur*** help menu, we used "***-x***" option to extract the embedded bytecodes which result to our flag:



```
root@kali:/tmp/jherron-stegosaurus-cd5c2373c031# python3.6 stegosaurus.py -x ../misc03.pyc
Extracted payload: flag{5t3g0_ftw}
```