Mamba Lambda

Deskripsi: "Classic dis.dis() python bytecode challenge, nothing too fancy ~ "

https://docs.python.org/3/library/dis.html

Challenge:

```
  3           0 LOAD_CONST               1 (<code object <lambda> at 0x0000018839EF5190, file "chall.py", line 3>)
              2 LOAD_CONST               2 ('check.<locals>.<lambda>')
              4 MAKE_FUNCTION            0
              6 STORE_FAST               0 (___)

  4           8 LOAD_GLOBAL              0 (input)
             10 LOAD_CONST               3 ('Welcome to warmup, no XOR RE here and please give me the flegg >> ')
             12 CALL_FUNCTION            1
             14 STORE_FAST               1 (______)

  5          16 LOAD_FAST                0 (___)
             18 LOAD_FAST                1 (______)
             20 LOAD_METHOD              1 (encode)
             22 CALL_METHOD              0
             24 CALL_FUNCTION            1
             26 LOAD_CONST               4 (b'\x11\x08C-49?\x92\x9847$\x92\x1d4n-$\x92\x1d4n\x19$\x92\x1d4-9\x1d4\x089\x1dC-\xc0\x1d\\?$9\x91')
             28 COMPARE_OP               3 (!=)
             30 POP_JUMP_IF_FALSE       42

  6          32 LOAD_GLOBAL              2 (print)
             34 LOAD_CONST               5 ('Wrong!')
             36 CALL_FUNCTION            1
             38 POP_TOP
             40 JUMP_FORWARD             8 (to 50)

  8     >>   42 LOAD_GLOBAL              2 (print)
             44 LOAD_CONST               6 ('Correct! See you in the next elimination round :>')
             46 CALL_FUNCTION            1
             48 POP_TOP
        >>   50 LOAD_CONST               0 (None)
             52 RETURN_VALUE

Disassembly of <code object <lambda> at 0x0000018839EF5190, file "chall.py", line 3>:
  3           0 LOAD_GLOBAL              0 (bytes)
              2 LOAD_CONST               1 (<code object <genexpr> at 0x0000018839EF50E0, file "chall.py", line 3>)
              4 LOAD_CONST               2 ('check.<locals>.<lambda>.<locals>.<genexpr>')
              6 MAKE_FUNCTION            0
              8 LOAD_FAST                0 (___)
             10 GET_ITER
             12 CALL_FUNCTION            1
             14 CALL_FUNCTION            1
             16 RETURN_VALUE

Disassembly of <code object <genexpr> at 0x0000018839EF50E0, file "chall.py", line 3>:
  3           0 LOAD_FAST                0 (.0)
        >>    2 FOR_ITER                46 (to 50)
              4 STORE_FAST               1 (__)
              6 LOAD_FAST                1 (__)
              8 LOAD_CONST               0 (13371337)
             10 LOAD_CONST               1 (2024)
             12 LOAD_FAST                1 (__)
             14 BINARY_SUBTRACT
             16 BINARY_MULTIPLY
             18 LOAD_CONST               2 (2025)
             20 LOAD_FAST                1 (__)
             22 BINARY_SUBTRACT
             24 BINARY_MULTIPLY
             26 BINARY_ADD
             28 LOAD_CONST               3 (199)
             30 BINARY_MODULO
             32 LOAD_FAST                1 (__)
             34 LOAD_CONST               4 (1)
             36 BINARY_ADD
             38 LOAD_CONST               5 (25)
             40 BINARY_MODULO
             42 BINARY_ADD
             44 YIELD_VALUE
             46 POP_TOP
             48 JUMP_ABSOLUTE            2
        >>   50 LOAD_CONST               6 (None)
             52 RETURN_VALUE
```

REVERSE ENGINEERING
By rwx4m

Analisis Bytecode

Bagian 1

Bytecode python dapat dianalisis menggunakan modul ‘dis’ yang memungkinkan untuk melihat low level instructions. Pada file “chall.txt”, terlihat bahwa bentuk ini adalah hasil dari “disassembly” (seperti petunjuk soal yang mengarahkan pada ‘dis’ python)

```
5          16 LOAD_FAST                0 (___)
           18 LOAD_FAST                1 (______)
           20 LOAD_METHOD              1 (encode)
           22 CALL_METHOD              0
           24 CALL_FUNCTION            1
           26 LOAD_CONST               4 (b'\x11\x08C~49?\x92\x984?$\x92\x1d4n~$\x92\x1d4n\x19$\x92\x1d4~9\x1d4\x089\x1dC~\xc0\x1d\\?$9\x91')
           28 COMPARE_OP               3 (!=)
           30 POP_JUMP_IF_FALSE       42
```

Bagian ini menjelaskan bahwa inputan user diproses sebuah fungsi *lambda* yang di load sebagai “___” (LOAD_FAST). kemudian inputan tersebut di *encode* dan hasilnya dibandingkan dengan konstanta biner. Jadi LOAD_METHOD 1 (encode) dan CALL_METHOD 0 akan mengubah inputan user menjadi byte array menggunakan metode ‘encode()’.

Penjelasan:

1. encode() adalah metode string yang mengubah sebuah string menjadi representasi byte. Biasanya diperlukan sebelum adanya operasi bitwise/compare byte-level.

2. LOAD_CONST: ini disebut byte array yang biasa direpresentasikan sebagai konstanta dalam bentuk diatas (b’\x11\x08c…). jadi didalam bagian LOAD_CONST berisi nilai biner yang berarti nilai byte ini perlu dibandingkan dengan operasi logika.

Bagian 2

```
Disassembly of <code object <lambda> at 0x0000018839EF5190, file "chall.py", line 3>:
  3           0 LOAD_GLOBAL              0 (bytes)
              2 LOAD_CONST               1 (<code object <genexpr> at 0x0000018839EF50E0, file "chall.py", line 3>)
              4 LOAD_CONST               2 ('check.<locals>.<lambda>.<locals>.<genexpr>')
              6 MAKE_FUNCTION            0
              8 LOAD_FAST                0 (____)
             10 GET_ITER
             12 CALL_FUNCTION            1
             14 CALL_FUNCTION            1
             16 RETURN_VALUE
```

Dari bagian ini didapatkan bahwa Disassembly dari fungsi lambda membentuk byte array dari sebuah ekspresi generator (genexpr) dan melakukan operasi matematika pada tiap karakter input.

Penjelasan: ‘bytes’ adalah fungsi untuk mengubah iterasi dari integer menjadi sebuah objek byte. Di dalam bytecode terlihat instruksi LOAD_GLOBAL 0 (bytes) diikuti oleh pembuatan fungsi dengan MAKE_FUNCTION 0.

Bagian 3

```
Disassembly of <code object <genexpr> at 0x0000018839EF50E0, file "chall.py", line 3>:
  3           0 LOAD_FAST                0 (.0)
        >>    2 FOR_ITER                46 (to 50)
              4 STORE_FAST               1 (__)
              6 LOAD_FAST                1 (__)
              8 LOAD_CONST               0 (13371337)
             10 LOAD_CONST               1 (2024)
             12 LOAD_FAST                1 (__)
             14 BINARY_SUBTRACT
             16 BINARY_MULTIPLY
             18 LOAD_CONST               2 (2025)
             20 LOAD_FAST                1 (__)
             22 BINARY_SUBTRACT
             24 BINARY_MULTIPLY
             26 BINARY_ADD
             28 LOAD_CONST               3 (199)
             30 BINARY_MODULO
             32 LOAD_FAST                1 (__)
             34 LOAD_CONST               4 (1)
             36 BINARY_ADD
             38 LOAD_CONST               5 (25)
             40 BINARY_MODULO
             42 BINARY_ADD
             44 YIELD_VALUE
             46 POP_TOP
             48 JUMP_ABSOLUTE            2
        >>   50 LOAD_CONST               6 (None)
             52 RETURN_VALUE
```

Ini adalah disassembly lanjutan dari generator. Didapat bahwa operasi aritmatika pada inputan karakter diubah menjadi nilai integer berdasarkan ASCII dan hasil akhirnya adalah sebuah byte array yang dibandingkan dengan konstanta dalam bytecode.

Penjelasan:

Instruksi Bytecode	Operasi
LOAD_FAST 0 (.0) FOR_ITER	Iterasi (for loop) dimulai untuk setiap karakter dalam input
LOAD_FAST 1 (__)	Setiap karakter disimpan dalam variabel sementara '__'.
LOAD_CONST 0 (13371337)	Nilai konstanta 13371337 di-load
LOAD_CONST 1 (2024)	Nilai konstanta 2024 di-load
BINARY_SUBTRACT	Mengurangi nilai ASCII karakter dari konstanta 2024
BINARY_MULTIPLY	Mengalikan hasil pengurangan dengan 13371337
LOAD_CONST 2 (2025)	Nilai konstanta 2025 di-load
BINARY_SUBTRACT	Mengurangi nilai ASCII karakter dari 2025
BINARY_MULTIPLY	Mengalikan hasil pengurangan dengan 2025
BINARY_ADD	Menambahkan hasil dari dua perkalian di atas
LOAD_CONST 3 (199)	Nilai konstanta 199 di-load
BINARY_MODULO	Mengambil sisa dari pembagian hasil penjumlahan dengan 199
LOAD_FAST 1 (__)	Mengambil kembali nilai ASCII karakter asli
LOAD_CONST 4 (1)	Nilai konstanta 1 di-load
BINARY_ADD	Menambahkan 1 ke nilai ASCII karakter
LOAD_CONST 5 (25)	Nilai konstanta 25 di-load
BINARY_MODULO	Mengambil sisa dari pembagian hasil penjumlahan dengan 25
BINARY_ADD	Menambahkan dua hasil operasi terakhir

CODE

Melalui analisis ini maka operasi matematika akan jadi seperti dibawah ini:

```
#Fungsi enkripsi
def enkripsi(a):
    return bytes(((i + (13371337 * (2024 - i) * (2025 - i))) % 199) + ((i + 1) % 25) for i in a)
```

Penjelasan:

Langkah	Penjelasan
i	Nilai ASCII dari setiap karakter dalam input
(2024 - i)	Mengurangi nilai ASCII karakter dari 2024
(2025 - i)	Mengurangi nilai ASCII karakter dari 2025
13371337 * (2024 - i) * (2025 - i)	Hasil pengurangan dikalikan dengan konstanta 13371337
(i + (13371337 * (2024 - i) * (2025 - i))) % 199	Hasil operasi diatas ditambahkan ke karakter asli nilai ASCII dan diambil hasil dari modulo 199
(i + 1) % 25	Menambahkan 1 ke nilai karakter asli ASCII dan diambil hasil modulo 25
Penjumlahan akhir	Dua hasil operasi di atas ditambahkan untuk mendapatkan byte akhir

Setelah itu yang akan menjadi **target** yaitu

```
#Target bytes
target = b'\x11\x08C~49?\x92\x984?$\x92\x1d4n~$\x92\x1d4n\x19$\x92\x1d4~9\x1d4\x089\x1dC~\xc0\x1d\\?$9\x91'
```

Melalui target ini, fungsi enkripsi harus menemukan hasil yang benar/cocok dengan dengan byte aray (target) agar mendapatkan hasil flag yang benar.

Maka dibuat brute force dengan menggunakan fungsi rekursif untuk menemukan string input yang benar.

```
# Fungsi rekursif untuk brute force
def cari_flag(flag_awal, index):
    if index == len(target):
        return flag_awal
    for x in string.ascii_letters + string.digits + "_{}":
        kemungkinan = flag_awal + x
        if enkripsi(kemungkinan.encode())[:index+1] == target[:index+1]:
            result = cari_flag(kemungkinan, index + 1)
            if result:
                return result
    return None

#Pencarian Flag
flag = cari_flag("", 0)
print("Flag is:", flag)
```

Penjelasan:

Jika panjang ‘flag_awal’ sama dengan panjang ‘target’, fungsi akan mengembalikan ‘flag_awal’ sebagai hasil karena sesuai dengan kriteria panjang ‘target’. Kemudian melakukan perulang yang berdasar pada format flag (gemastik{_ _ _}). Jadi pada setiap karakter dalam *string.ascii_letters + string.digits + “_{}”*, akan dicoba penambahan karakter ke ‘flag_awal’ dan memeriksa hasil enkripsi apakah cocok dengan ‘target’. Jika cocok/sesuai, fungsi akan memanggil dirinya sendiri dengan ‘flag_awal’ yang diperbarui serta ‘index’ yang bertambah. Apabila ‘flag’ ditemukan maka fungsi akan mengembalikkannya dan jika tidak maka akan dikembalikan ‘None’. Kemudian pada bagian ‘flag’

akan dimulai pencarian dengan string kosong (“”) dan index ke-0, dan akan dicetak flag jika ditemukan.