**Mamba Lambda**

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

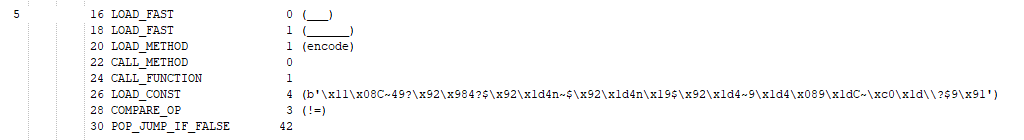
<https://docs.python.org/3/library/dis.html>

**Challenge: **

**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)



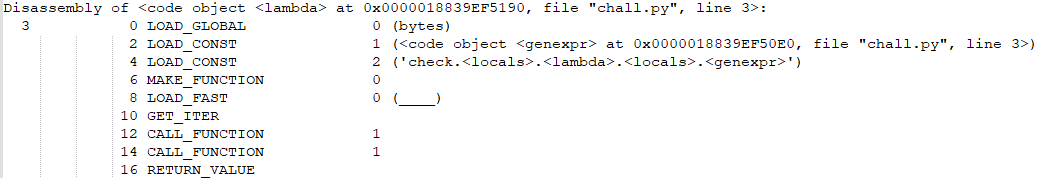
Bagian ini menjelaskan bahwa inputan user diproses sebuah fungsi *lambda* yang di load sebagai “\_\_\_” (LOAD\_FAST). kemudian inputan tersebut di *encode* dan hasilnya dibandingkan dengan konstanta biner. Jadi LOAD\_METHOD 1 (encode) dan CALL\_METHOD 0 akan mengubah inputan user menjadi byte array menggunakan 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.

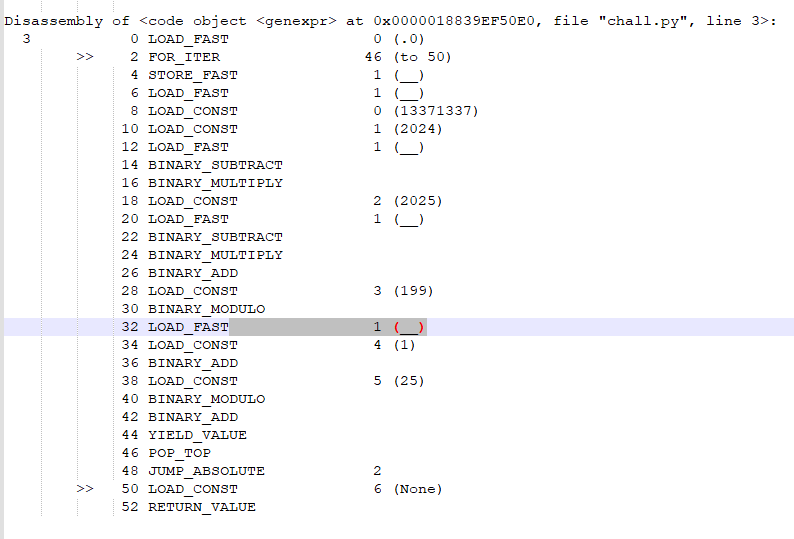
**Bagian 2**



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**



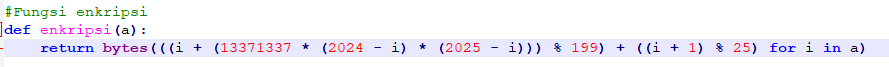
Ini adalah disassembly lanjutan dari generator. Didapat bahwa operasi aritmatika pada inputan karakter diubah menjadi nilai integer berdasarkan ASCII dan hasil akhirnya adalah sebuah byte array yang dibandingkan dengan konstanta dalam bytecode.

**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:



**Penjelasan:**

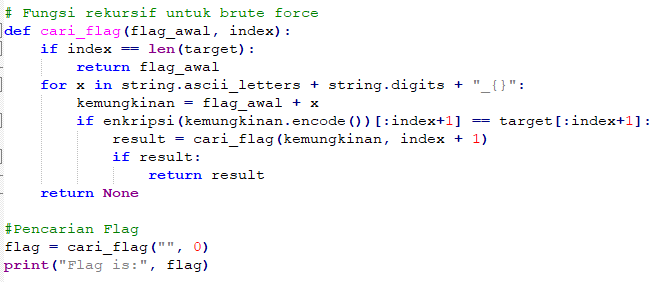
|  |  |
| --- | --- |
| **Langkah** | **Penjelasan** |
| **i** | Nilai ASCII dari setiap karakter dalam input |
| **(2024 - i)** | Mengurangi nilai ASCII karakter dari 2024 |
| **(2025 - i)** | Mengurangi nilai ASCII karakter dari 2025 |
| **13371337 \* (2024 - i) \* (2025 - i)** | Hasil pengurangan 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

**

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.

  
**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.