

# Byte-Circus

Player: constantine

Kategori: Reverse Engineering



## Phase 1: Decompilation & Static Analysis

Target memberikan sebuah file *compiled Python bytecode* dengan nama:

Script.cpython-313.pyc

Karena `.pyc` adalah Python bytecode, langkah selanjutnya adalah melakukan **dekompileasi** untuk mendapatkan kembali source code-nya secara mendekati akurat.

Ekstensi `cpython-313.pyc` menunjukkan bahwa file dikompilasi menggunakan **Python 3.13**. Ini penting karena struktur bytecode berubah antar versi Python.

Untuk cara manual sebenarnya bisa digunakan:

```
pycdc script.cpython-313.pyc
# atau
python3.13 -m dis script.cpython-313.pyc
```

Tapi daripada membaca opcode Python secara manual, aku menggunakan **PyChaos/Python Decompiler online** yang secara otomatis mengembalikan struktur kode aslinya:



The screenshot shows a web-based decompiler interface. At the top, there's a navigation bar with icons for back, forward, and search, followed by the URL "pychaos.io/decompiled?uvid=a99c4a77-98e1-4836-91e4-842a5b35a6dc". Below the URL is a small link to "Home | Decompile | Demo". The main content area displays the decompiled Python code. The code defines a function 'main' that prompts for a password, encodes it, and compares it against a stored flag. If correct, it prints the flag; if wrong, it prints an error message. It also checks if the script is run from the command line and calls 'main' if true. The code uses a list of integers for the flag and performs XOR operations on user input.

```
the_flag = [1423, 1432, 1403, 1422, 1404, 1405, 1410, 1444, 1436, 1437, 1450, 1437, 1458, 1452, 1416, 1450, 1463, 1450, 1461, 1442, 1436, 1458, 1436, 1416, 1458, 1437, 1416, 1450, 1455, 1372, 1369, 1452, 1453, 1446]
def enc(string):
    return [ord(x)^16+1337 for x in string]

def main():
    inp = input('Enter password: ').strip()
    inp_enc = enc(inp)
    if inp_enc == the_flag:
        print(f'''Correct! Flag: {inp}''')
        return None
    else:
        print('Wrong password')
        return None

if __name__ == '__main__':
    main()
```

Hasil decompile menunjukkan umum:

- Program menyimpan flag dalam bentuk **integer array** (`the_flag`)
- Setiap angka diencode menggunakan 2 operasi:
  1. `+1337`
  2. `XOR ^ 16`
- Untuk mendapatkan flag asli harus dibalik:

```
original = (value - 1337) ^ 16
```

## Phase 3: Execution

Setelah memahami logika encoding, aku menuliskan ulang solver dibantu Gemini dengan versi yang lebih clean untuk memastikan hasil 100% akurat.

**Solver Script (poc.py):**

```
def solve():
    the_flag = [
        1423, 1432, 1403, 1422, 1404, 1405, 1410, 1444, 1436, 1437,
        1450, 1437, 1458, 1452, 1416, 1450, 1463, 1450, 1461, 1442,
        1436, 1458, 1436, 1416, 1458, 1437, 1416, 1458, 1436, 1416,
        1450, 1455, 1372, 1369, 1452, 1453, 1446
    ]
```

```
decoded_flag = ""

for value in the_flag:
    temp = value - 1337      # inverse dari +1337
    original_ord = temp ^ 16  # inverse dari XOR 16
    decoded_flag += chr(original_ord)

print(f"Flag Found: {decoded_flag}")

if __name__ == "__main__":
    solve()
```

```
constantine ~/0precForesty/byte-circus (solved) ➜ v3.13.7 ➜ ① 14:45 ➜ python poc.py
Flag Found: FORESTY{static_analysis_it_is_af30cd}
```

---

## Final Flag

**FORESTY{static\_analysis\_it\_is\_af30cd}**