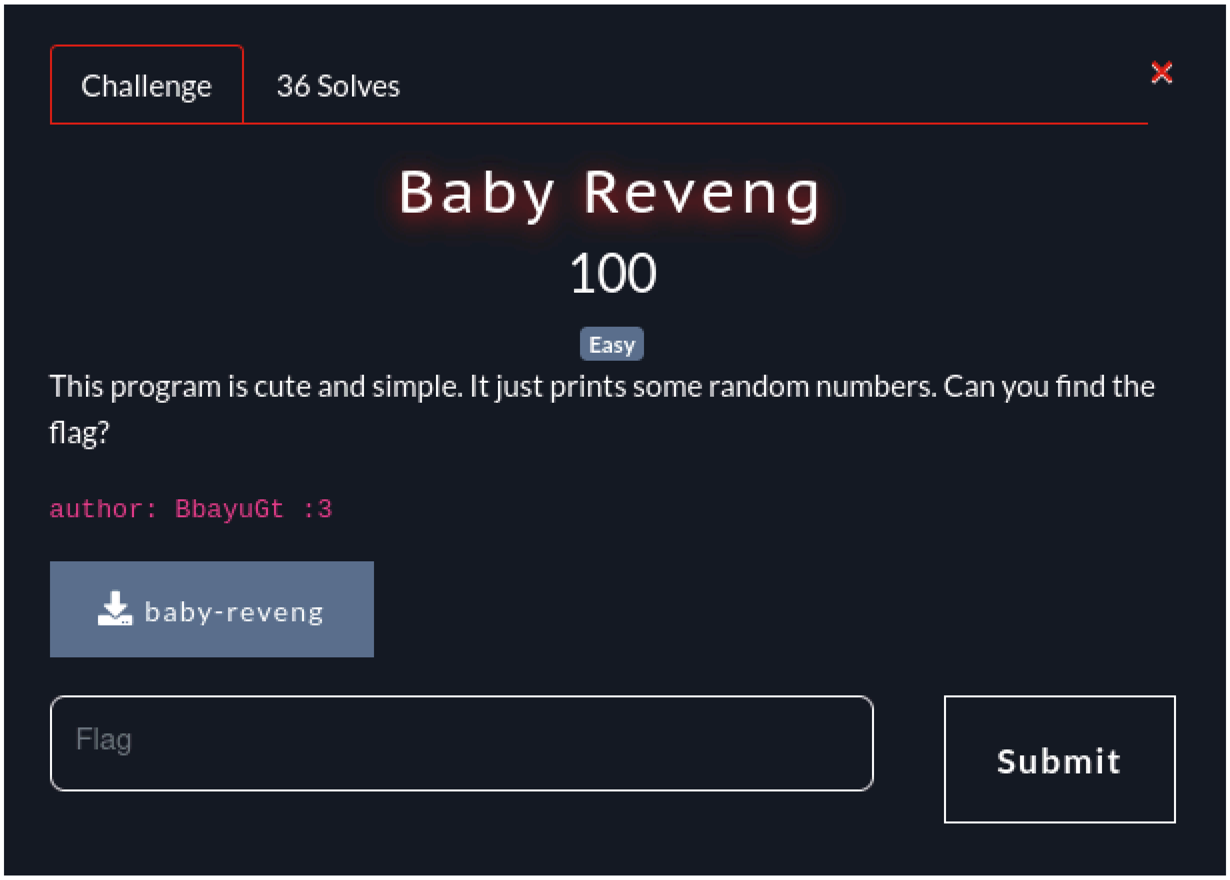
# baby-reveng

**Player:** Constantine
**Kategori:** Reverse Engineer



Challenge
36 Solves

Baby Reveng

100

Easy

This program is cute and simple. It just prints some random numbers. Can you find the flag?

`author: BbayuGt :3`

---

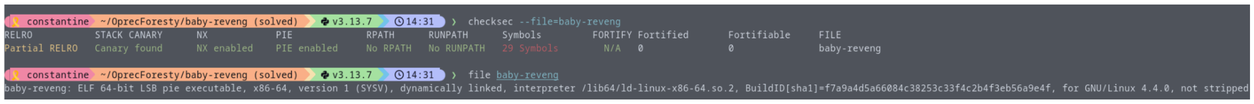
## 1. Initial Reconnaissance

Langkah pertama cek identitas file untuk memahami arsitektur binary.

Bash

file baby-reveng

checksec --file=baby-reveng



```
constantine ~/OprecForesty/baby-reveng (solved) v3.13.7 14:31 › checksec --file=baby-reveng
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable     FILE
Partial RELRO   Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   29 Symbols        N/A   0               0               baby-reveng

constantine ~/OprecForesty/baby-reveng (solved) v3.13.7 14:31 › file baby-reveng
baby-reveng: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=f7a9a4d5a66084c38253c33f4c2b4f3eb56a9e4f, for GNU/Linux 4.4.0, not stripped
```

File terkonfirmasi sebagai **ELF 64-bit LSB executable, x86-64**. Karena simbolnya *not stripped*, kita bisa langsung mencari fungsi main tanpa kesulitan.

## 2. Static Analysis (Ghidra)

Langsung load binary ke Ghidra. Di fungsi main, terlihat struktur kode yang cukup menarik. Program memuat serangkaian nilai hex ke dalam array integer (local_98), karakter demi karakter. Ini teknik klasik *stack strings*.

Namun, ada jebakan (trap) di bagian *printing* loop:

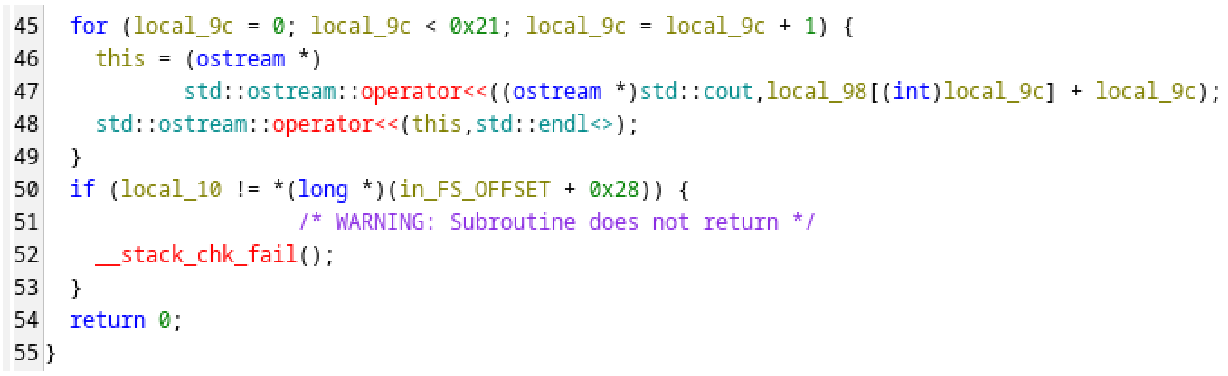
for (local_9c = 0; local_9c < 0x21; local_9c = local_9c + 1) {

  // ...

  std::ostream::operator<<((ostream *)std::cout, local_98[(int)local_9c] + local_9c);

  // ...

}



```
  for (local_9c = 0; local_9c < 0x21; local_9c = local_9c + 1) {
    this = (ostream *)
           std::ostream::operator<<((ostream *)std::cout,local_98[(int)local_9c] + local_9c);
    std::ostream::operator<<(this,std::endl<>);
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```

Logika local_98[(int)local_9c] + local_9c berarti program mengubah nilai asli flag saat runtime. Inilah kenapa kalau binary dijalankan (./baby-reveng), outputnya ngaco. Flag aslinya tersimpan murni di variabel local_98 sebelum dimodifikasi loop tersebut.

## 3. Decoding & Solving

Kita tidak perlu melakukan *reverse* pada algoritma penjumlahan tersebut. Cukup ambil *raw hex values* dari array local_98 dan konversi ke ASCII.

Untuk efisiensi dan menghindari *human error*, saya menggunakan script Python sederhana untuk parsing nilai hex tersebut.

Solver Script (solve.py):

(Credit: Script logic & generation dibantu oleh ChatGPT)

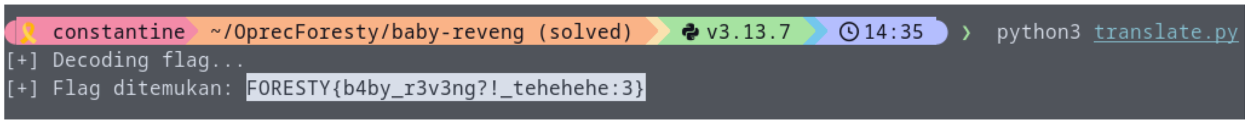
```
# Solver untuk baby-reveng
# Berdasarkan data dari array local_98 di Ghidra
```

```python
def solve_flag():
    # Array hex yang kita ambil manual dari hasil decompile
    # local_98[0] sampai local_98[0x20]
    hex_values = [
        0x46, 0x4f, 0x52, 0x45, 0x53, 0x54, 0x59, 0x7b, # FORESTY{
        0x62, 0x34, 0x62, 0x79, 0x5f, 0x72, 0x33, 0x76, # b4by_r3v
        0x33, 0x6e, 0x67, 0x3f, 0x21, 0x5f, 0x74, 0x65, # 3ng?!_te
        0x68, 0x65, 0x68, 0x65, 0x68, 0x65, 0x3a, 0x33, # hehehe:3
        0x7d                                            # }
    ]

    # Convert setiap angka hex ke karakter ASCII
    flag = "".join([chr(val) for val in hex_values])

    return flag

if __name__ == "__main__":
    print(f"[+] Decoding flag...")
    flag = solve_flag()
    print(f"[+] Flag ditemukan: {flag}")
```

**Execution:**

Bash

python3 solve.py



```
 constantine  ~/OprecForesty/baby-reveng (solved)  v3.13.7  14:35 › python3 translate.py
[+] Decoding flag...
[+] Flag ditemukan: FORESTY{b4by_r3v3ng?!_tehehehe:3}
```

Flag:

**FORESTY{b4by_r3v3ng?!_tehehehe:3}**