

# 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

 baby-reveng

Submit

---

## Phase 1: Initial Reconnaissance

Langkah pertama adalah melakukan identifikasi awal terhadap binary untuk mengetahui format file, arsitektur, proteksi, serta potensi indikator lainnya.

## 1. checksec

```
CTF & Cybersecurity/OprecForestry/baby-reveng (solved) via v3.13.7
> 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
```

Output menunjukkan:

- **ELF 64-bit LSB executable**
- Arsitektur: **x86-64**
- **No PIE, NX enabled, RELRO Partial**
- **Symbols NOT STRIPPED** → sangat membantu untuk reversing.

## 2. File Signature

```
CTF & Cybersecurity/OprecForestry/baby-reveng (solved) via v3.13.7
> 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]=f7a9a4d5a65084c38253c33f4c2b4f3eb56a9e4f, for GNU/Linux 4.4.0, not stripped
```

Hasil menegaskan bahwa ini adalah ELF x86-64 dengan simbol lengkap.

Artinya kita bisa langsung membuka fungsi main tanpa harus melakukan simbol recovery.

---

## Phase 2: Static Analysis & Execution

Binary kemudian dianalisis menggunakan **Ghidra** untuk mendapatkan struktur high-level code:

```
45 for (local_9c = 0; local_9c < 0x21; local_9c = local_9c + 1) {
46     this = (ostream *)
47         std::ostream::operator<<((ostream *)std::cout,local_98[(int)local_9c] + local_9c);
48     std::ostream::operator<<(this,std::endl<>);
49 }
50 if (local_10 != *(long *) (in_FS_OFFSET + 0x28)) {
51     /* WARNING: Subroutine does not return */
52     __stack_chk_fail();
53 }
54 return 0;
55 }
```

Dalam fungsi **main()** ditemukan:

- Sebuah array lokal bernama **local\_98**
- Berisi puluhan nilai **hex literal** yang membentuk string flag, tetapi **dalam bentuk integer array**
- Ini dikenal sebagai **stack strings obfuscation**, di mana string tidak disimpan sebagai literal, tetapi diisi byte-per-byte ke array stack.

Contoh polanya:

```
local_10 = *(long *) (in_FS_OFFSET + 0x28);  
local_98[0] = 0x46;  
local_98[1] = 0x4f;  
local_98[2] = 0x52;  
local_98[3] = 0x45;  
local_98[4] = 0x53;  
local_98[5] = 0x54;  
local_98[6] = 0x59;  
local_98[7] = 0x7b;
```

Terdapat trap logic atau anti-dump trick seperti loop menggunakan logika:

```
local_98[(int)local_9c] + local_9c
```

Artinya:

- Setiap karakter flag ditambahkan dengan dynamic offset.
- Akibatnya output runtime adalah **angka acak**, bukan flag asli (bisa dilihat dibawah, hasil dari ./baby-reveng).
- Flag asli sebenarnya **tersimpan murni dalam array local\_98 sebelum dimodifikasi**.

CTF & Cybersecurity/0precForesty/baby-reveng (solved) via 🐍 v3.13.7

> ./baby-reveng

70  
80  
84  
72  
87  
89  
95  
130  
106  
61  
108  
132  
107  
127  
65  
133  
67  
127  
121  
82  
53  
116  
138  
124  
128  
126  
130  
128  
132  
130  
88  
82  
157

CTF & Cybersecurity/0precForesty/baby-reveng (solved) via 🐍 v3.13.7

> \_

Output dari `./baby-reveng` . Kesimpulannya:

➡ Tidak perlu reversing algoritma distorsi runtime-nya.

➡ Cukup ambil raw hex values dari array `local_98` dan decode langsung ke ASCII.

---

## Phase 3: Extraction & Decoding

Untuk mengekstraksi flag, kita cukup mengcopy nilai hex dari Ghidra dan membuat script Python untuk mengubahnya ke karakter ASCII.

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

```
def solve_flag():
    hex_values = [
        0x46, 0x4f, 0x52, 0x45, 0x53, 0x54, 0x59, 0x7b,
        0x62, 0x34, 0x62, 0x79, 0x5f, 0x72, 0x33, 0x76,
        0x33, 0x6e, 0x67, 0x3f, 0x21, 0x5f, 0x74, 0x65,
        0x68, 0x65, 0x68, 0x65, 0x68, 0x65, 0x3a, 0x33,
        0x7d
    ]

    flag = "".join([chr(val) for val in hex_values])

    return flag

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

Tinggal kita execute saja scriptnya:

```
constantine ~/0precForesty/baby-reveng (solved) v3.13.7 14:35 > python3 translate.py
[+] Decoding flag...
[+] Flag ditemukan: FORESTY{b4by_r3v3ng?!_tehehehe:3}
```

---

## Final Flag

FORESTY{b4by\_r3v3ng?!\_tehehehe:3}