babycmp_en

```
neko_hat@nekohat:/mnt/c/Users/dohwa/OneDrive - 중앙대학교/SECCON/babycmp$ file chall.baby
chall.baby: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-
64.so.2, BuildID[sha1]=ded5cc024f968b3087bf5d3df8649d14714e7202, for GNU/Linux 3.2.0, not stripped
```

The problem is the baby file. There is no information about the file, so we checked the information about the file through the 'file' command.

It's ELF64-bit file, Linux executable. Let's run the applicable program.

```
neko_hat@nekohat:/mnt/c/Users/dohwa/OneDrive - 중앙대학교/SECCON/babycmp$ ./chall.baby
Usage: ./chall.baby FLAG
neko_hat@nekohat:/mnt/c/Users/dohwa/OneDrive - 중앙대학교/SECCON/babycmp$ ./chall.baby neko_hat
Wrong...
neko_hat@nekohat:/mnt/c/Users/dohwa/OneDrive - 중앙대학교/SECCON/babycmp$
```

The information shows that if you enter the FLAG value as a factor in the program, it will verify that the FLAG is correct.

Let's use IDA for program analysis.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  const char *v5; // r12
  size_t v10; // rax
  size_t v11; // rdi
  unsigned __int64 v12; // rcx
  const char *v13; // rsi
  __int64 v14; // rax
  unsigned __int64 v15; // rdx
  int v16; // er12
  __m128i v18; // [rsp+0h] [rbp-68h]
  char v19[8]; // [rsp+10h] [rbp-58h] BYREF
  __m128i v20; // [rsp+20h] [rbp-48h]
  __m128i v21; // [rsp+30h] [rbp-38h]
  int v22; // [rsp+40h] [rbp-28h]
  unsigned __int64 v23; // [rsp+48h] [rbp-20h]

  v23 = __readfsqword(0x28u);
  _RAX = 0LL;
  if ( argc <= 1 )
  {
    v16 = 1;
    __printf_chk(1LL, "Usage: %s FLAG\n", *argv);
  }
  else
  {
    v5 = argv[1];
    __asm { cpuid }
    v22 = 3672641;
    strcpy(v19, "N 2022");
    v20 = _mm_load_si128((const __m128i *)&xmmword_3140);
    v21 = _mm_load_si128((const __m128i *)&xmmword_3150);
    v18 = _mm_load_si128((const __m128i *)&xmmword_3160);
    v10 = strlen(v5);
    v11 = v10;
    if ( v10 )
    {
      *v5 ^= 0x57u;
      v12 = 1LL;
      if ( v10 != 1 )
```

```
      {
        do
        {
          v13 = &argv[1][v12];
          v14 = v12 / 0x16
              + 2 * (v12 / 0x16 + (((0x2E8BA2E8BA2E8BA3LL * (unsigned __int128)v12) >> 64) & 0xFFFFFFFFFFFFFFFCLL));
          v15 = v12++;
          *v13 ^= v18.m128i_u8[v15 - 2 * v14];
        }
        while ( v11 != v12 );
      }
      v5 = argv[1];
    }
    if ( *(_OWORD *)&v20 == *(_OWORD *)v5 && *(_OWORD *)&v21 == *((_OWORD *)v5 + 1) && *((_DWORD *)v5 + 8) == v22 )
    {
      v16 = 0;
      puts("Correct!");
    }
    else
    {
      v16 = 0;
      puts("Wrong...");
    }
  }
  return v16;
}
```

This is the decompile of the main function.

An inputted value argv[1] is stored in v5, and the length of the corresponding character string is stored in v10, v11 to be used for checking the length of the input value.

```
if ( v10 )
{
      *v5 ^= 0x57u;
      v12 = 1LL;
      if ( v10 != 1 )
      {
        do
        {
          v13 = &argv[1][v12];
          v14 = v12 / 0x16
              + 2 * (v12 / 0x16 + (((0x2E8BA2E8BA2E8BA3LL * (unsigned __int128)v12) >> 64) & 0xFFFFFFFFFFFFFFFCLL));
          v15 = v12++;
          *v13 ^= v18.m128i_u8[v15 - 2 * v14];
        }
        while ( v11 != v12 );
      }
      v5 = argv[1];
}
```

A pointer v5 pointing to the input value argv[1] is *(v5)^=0x57 (InputVar[0]^=0x57)

In the repetitive sentence, argv[1][v12] is received by v13, and v13^=v18.m128i_u8[v15-2v14].

```
//cal.c
//gcc -o cal cal.c

#include <stdio.h>
#include <stdint.h>
#define uint128_t __uint128_t

void main()
{
    uint64_t v15;
    int64_t v14;
    uint64_t v12 = 1LL;

    for(int i = 0; i <= 30; i++)
    {
        v14 = v12 / 0x16 + 2 * (v12 / 0x16 + (((0x2E8BA2E8BA2E8BA3LL * (unsigned __int128)v12) >> 64) & 0xFFFFFFFFFFFFFFFCLL));

        v15 = v12++;
        printf("v14 = %lld\tv15 - 2 * v14 = %lld\n", v14, v15 - 2 * v14);
    }
}
```

For complex v14 value calculations, the above code was created and calculated.

```
v14 = 0 v15 - 2 * v14 = 1
v14 = 0 v15 - 2 * v14 = 2
v14 = 0 v15 - 2 * v14 = 3
v14 = 0 v15 - 2 * v14 = 4
v14 = 0 v15 - 2 * v14 = 5
v14 = 0 v15 - 2 * v14 = 6
v14 = 0 v15 - 2 * v14 = 7
v14 = 0 v15 - 2 * v14 = 8
v14 = 0 v15 - 2 * v14 = 9
v14 = 0 v15 - 2 * v14 = 10
v14 = 0 v15 - 2 * v14 = 11
v14 = 0 v15 - 2 * v14 = 12
v14 = 0 v15 - 2 * v14 = 13
v14 = 0 v15 - 2 * v14 = 14
v14 = 0 v15 - 2 * v14 = 15
v14 = 0 v15 - 2 * v14 = 16
v14 = 0 v15 - 2 * v14 = 17
v14 = 0 v15 - 2 * v14 = 18
v14 = 0 v15 - 2 * v14 = 19
v14 = 0 v15 - 2 * v14 = 20
v14 = 0 v15 - 2 * v14 = 21
v14 = 11        v15 - 2 * v14 = 0
v14 = 11        v15 - 2 * v14 = 1
v14 = 11        v15 - 2 * v14 = 2
v14 = 11        v15 - 2 * v14 = 3
v14 = 11        v15 - 2 * v14 = 4
v14 = 11        v15 - 2 * v14 = 5
v14 = 11        v15 - 2 * v14 = 6
v14 = 11        v15 - 2 * v14 = 7
v14 = 11        v15 - 2 * v14 = 8
v14 = 11        v15 - 2 * v14 = 9
```

The above results were obtained.

That is, v15-2*v14=SomeVar%22

```
v18 = _mm_load_si128((const __m128i *)&xmmword_555555557160);
```

then, v18 is initialized as above

```
if ( *(_OWORD *)&v20 == *(_OWORD *)v5 && *(_OWORD *)&v21 == *((_OWORD *)v5 + 1) && *((_DWORD *)v5 + 8) == v22 )
    {
      v16 = 0;
      puts("Correct!"□);
    }
    else
    {
      v16 = 0;
      puts("Wrong...");
    }
```

By looking at the 'if', the code is the code used for FLAG verification, and variables to be confirmed are v18, v20, v21, v22

Checked with IDA to obtain values for v18, v20, v21, v22.

```
v20 = _mm_load_si128((const __m128i *)&xmmword_555555557140);
v21 = _mm_load_si128((const __m128i *)&xmmword_555555557150);
v22 = 3672641;
```

```
a:0000555555557140 xmmword_555555557140 xmmword 2B2D3675357F1A44591E2320202F2004h
a:0000555555557140                                          ; DATA XREF: main+31↑r
a:0000555555557150 xmmword_555555557150 xmmword 362B470401093C150736506D035A1711h
a:0000555555557150                                          ; DATA XREF: main+56↑r
```

Debugging progresses because the initialization state is difficult to understand in the above information. Try running gdbserver for debugging progress on IDA.

```
neko_hat@nekohat:/mnt/c/Users/dohwa/OneDrive - 중앙대학교/SECCON/babycmp$ gdbserver 127.0.0.1:32912
chall.baby we_ll_find_flag_soon!
Process /mnt/c/Users/dohwa/OneDrive - 중앙대학교/SECCON/babycmp/chall.baby created; pid = 494
Listening on port 32912
Remote debugging from host 127.0.0.1, port 40282
```

```
v20 = _mm_load_si128((const __m128i *)&xmmword_555555557140);
v21 = _mm_load_si128((const __m128i *)&xmmword_555555557150);
v18 = __m128i v20; // [rsp+20h] [rbp-48h]
v10 = {m128i_i8={4,0x20,0x2F,0x20,0x20,0x23,0x1E,0x59,0x44,0x1A,0x7F,0x35,0x75,0x36,0x2D,0x2B}}
```

```
v21 = _mm_load_si128((const __m128i *)&xmmword_555555557150);
v18 = __m128i v21; // [rsp+30h] [rbp-38h]
v10 = {m128i_i8={0x11,0x17,0x5A,3,0x6D,0x50,0x36,7,0x15,0x3C,9,1,4,0x47,0x2B,0x36}}
```

```
v18 = _mm_load_si128((const __m128i *)&xmmword_555555557160);
v10 = strlen(v5);
v11 = __m128i v18; // [rsp+0h] [rbp-68h]
if ( {m128i_i8={0x57,0x65,0x6C,0x63,0x6F,0x6D,0x65,0x20,0x74,0x6F,0x20,0x53,0x45,0x43,0x43,0x4F}}
{
```

The following values can be seen: Afterwards, v18 is initialized in combination with v19. (union)

```
//v18.m128i_u8
MEMORY:00007FFFFFFFDB00 db  57h ; W
MEMORY:00007FFFFFFFDB01 db  65h ; e
MEMORY:00007FFFFFFFDB02 db  6Ch ; l
MEMORY:00007FFFFFFFDB03 db  63h ; c
MEMORY:00007FFFFFFFDB04 db  6Fh ; o
MEMORY:00007FFFFFFFDB05 db  6Dh ; m
MEMORY:00007FFFFFFFDB06 db  65h ; e
MEMORY:00007FFFFFFFDB07 db  20h
MEMORY:00007FFFFFFFDB08 db  74h ; t
MEMORY:00007FFFFFFFDB09 db  6Fh ; o
MEMORY:00007FFFFFFFDB0A db  20h
MEMORY:00007FFFFFFFDB0B db  53h ; S
MEMORY:00007FFFFFFFDB0C db  45h ; E
MEMORY:00007FFFFFFFDB0D db  43h ; C
MEMORY:00007FFFFFFFDB0E db  43h ; C
MEMORY:00007FFFFFFFDB0F db  4Fh ; O
MEMORY:00007FFFFFFFDB10 db  4Eh ; N
MEMORY:00007FFFFFFFDB11 db  20h
MEMORY:00007FFFFFFFDB12 db  32h ; 2
MEMORY:00007FFFFFFFDB13 db  30h ; 0
MEMORY:00007FFFFFFFDB14 db  32h ; 2
MEMORY:00007FFFFFFFDB15 db  32h ; 2
MEMORY:00007FFFFFFFDB16 db    0
```

Let's create a decryption code with input value encryption logic.

```
//exploit.c
//gcc -o exploit exploit.c

#include <stdio.h>
```

```c
#include <string.h>

int main()
{
  char *key = "Welcome to SECCON 2022";

  char target[] = {0x4, 0x20, 0x2F, 0x20, 0x20, 0x23, 0x1E,0x59,0x44,0x1A,0x7F,0x35,0x75,0x36,0x2D,0x2B,0x11,0x17,0x5A,0x3, 0x6D,0x50,

  for(int i = 0; i<sizeof(target)/sizeof(char); i++)
    target[i] ^= key[i%strlen(key)];


  printf("%s\n", target);
  return 0;
}
```

```
neko_hat@nekohat:/mnt/c/Users/dohwa/OneDrive - 중앙대학교/SECCON/babycmp$ ./exploit

SECCON{y0u_f0und_7h3_baby_flag_YaY}
```

FLAG: SECCON{y0u_f0und_7h3_baby_flag_YaY}