

babycmp_jp

```
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
```

問題がbabyファイルだ。該当ファイルに関する情報はないため、'file'コマンドを通じて該当ファイルの情報を調べた。

ELF64-bit file、Linux実行ファイルである。該当プログラムを実行した。

```
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...
```

この情報で分かるのは、該当プログラムに因子としてFLAG値を入力すれば、該当FLAGが正しいか検証してくれるということだ。

プログラム分析のためにIDAを使ってみよう。

```
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) & 0xFFFFFFFFFFFFFFFFCULL));
            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;
}
}

```

main関数のdecompileの様子である。

入力した値であるargv[1]がv5に保存され、該当文字列の長さがv10、v11に保存されて入力値の長さチェックに使われることを確認することができる。

```

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) & 0xFFFFFFFFFFFFFFFFCULL));
            v15 = v12++;
            *v13 ^= v18.m128i_u8[v15 - 2 * v14];
        }
        while ( v11 != v12 );
    }
    v5 = argv[1];
}
}

```

入力した値argv[1]を指すポインタv5を*(v5)^=0x57する。(InputVar[0]^=0x57)

反復文の中でargv[1][v12]をv13で受け、v13^=v18.m128i_u8[v15-2v14]する。

```

//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) & 0xFFFFFFFFFFFFFFFFCULL));

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

```

複雑なv14値計算のために上のコードを作成して計算をした。

```

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

```

上記の結果を得た。

つまり、 $v15 - 2 * v14 = \text{SomeVar} \% 22$

```

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

```

v18は上記のように初期化される

```

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

```

該if分岐を見て上記のコードがFLAG検証に使用されるコードであり、確認すべき変数はv18、v20、v21、v22である。

v18、v20、v21、v22の値を得るためにIDAで調べた。

```

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

```

上記の情報では初期化状態が分かりにくくデバッグを進めた。IDAでデバッグ進行のためにgdbserverを稼動させてみよう。

```
neko_hat@neko_hat: /mnt/c/U: x Windows PowerShell x + v
neko_hat@neko_hat:/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
```

```
strcpy(v19, "2022");
v20 = _mm_load_si128((const __m128i *)&__xmmword_55555557140);
v21 = _mm_load_si128((const __m128i *)&__xmmword_55555557150);
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}};
v11 = 0x18;

v20 = _mm_load_si128((const __m128i *)&__xmmword_55555557140);
v21 = _mm_load_si128((const __m128i *)&__xmmword_55555557150);
v18 = _mm_load_si128((const __m128i *)&__xmmword_55555557160);
v10 = __m128i v21; // [rsp+30h] [rbp-38h]
v11 = {__m128i_i8={0x11,0x17,0x5A,3,0x6D,0x50,0x36,7,0x15,0x3C,9,1,4,0x47,0x2B,0x36}};

v21 = _mm_load_si128((const __m128i *)&__xmmword_55555557150);
v18 = _mm_load_si128((const __m128i *)&__xmmword_55555557160);
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}} )
{
```

次の値が表示されます。その後、v18はv19と組み合わせて初期化されます。(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
```

入力値暗号化ロジックで復号化コードを作成してみよう。

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

#include <stdio.h>
```

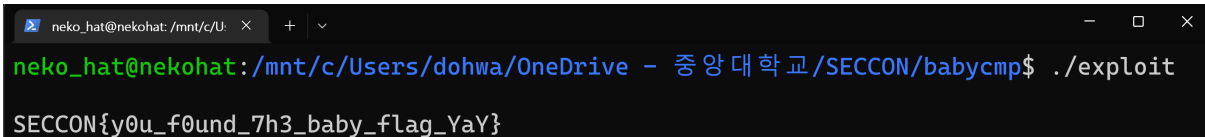
```
#include <string.h>

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;
}
```



A terminal window with a dark background. The title bar shows 'neko_hat@nekohat: /mnt/c/U:'. The prompt is 'neko_hat@nekohat: /mnt/c/Users/dohwa/OneDrive - 중앙대학교/SECCON/babycmp\$'. The command './exploit' has been entered, and the output is 'SECCON{y0u_f0und_7h3_baby_flag_YaY}'.

```
neko_hat@nekohat: /mnt/c/U:
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}