

# babycmp

```
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
```

문제가 baby 파일이다. 해당 파일에 대한 정보는 없기 때문에 'file' 명령어를 통하여 해당 파일의 정보를 살펴보았다.

ELF 64-bit file, 리눅스 실행파일이다. 해당 프로그램을 실행했다.

```
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...
```

이 정보로 알 수 있는 것은 해당 프로그램에 인자로 FLAG 값을 입력하면 해당 FLAG가 맞는지 검증해준다는 것이다.

프로그램 분석을 위해 IDA를 사용해보자.

```
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 함수의 decompile 모습이다.

입력한 값인 argv[1]이 v5에 저장되고, 해당 문자열의 길이가 v10, v11에 저장되어 입력 값 길이 체크에 사용되는 것을 확인 할 수 있다.

```

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];
}

```

입력한 값 argv[1]을 가리키는 포인터 v5를 \*(v5) ^= 0x57 한다.(InputVar[0] ^= 0x57)

반복문 안에서 argv[1][v12]를 v13으로 받고, \*v13 ^= v18.m128i\_u8[v15 - 2 \* v14] 한다.

```

//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);
    }
}

```

복잡한 v14 값 계산을 위해 위의 코드를 작성하여 계산을 했다.

```

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

```

위의 결과를 얻었다.

즉,  $v15 - 2 * v14 = \text{SomeVar} \% 22$

```

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

```

v18은 위와 같이 초기화 된다.

```

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

```

해당 if문을 보아 위의 코드가 FLAG 검증에 사용되는 코드며, 확인해야할 변수는 v18, v20, v21, v22이다.

v18 v20, v21, v22의 값을 얻기 위해 IDA에서 살펴보았다.

```

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

```

위의 정보로는 초기화 상태를 알기 힘들어 디버깅을 진행하였다. IDA에서 디버깅 진행을 위해 gdbserver를 가동시켜보자.

```
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}} )
{
```

다음의 값을 확인 할 수 있다. v18은 이후 v19와 합쳐져 (union) 아래와 같이 초기화 된다.

```
//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
```

입력값 암호화 로직으로 복호화 코드를 작성해보자.

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

#include <stdio.h>
```

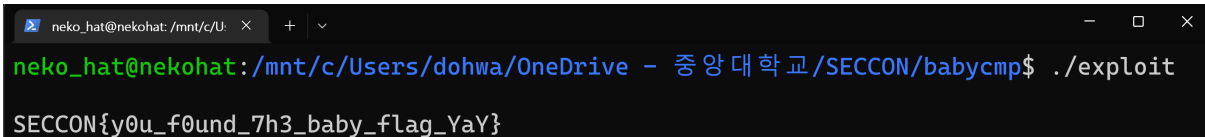
```
#include <string.h>

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 user has entered './exploit' and the output is 'SECCON{y0u\_f0und\_7h3\_baby\_flag\_YaY}'.

```
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}