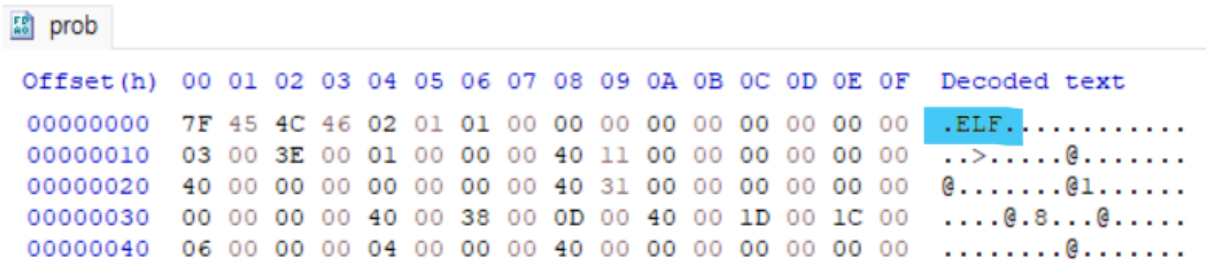
Dreamhack Long Sleep Writeup

이름
prob

들어가면 prob라는 파일이 있다.



헥스 에디터로 확인한 후 Elf 파일로 확장자를 변경한다. 이후 ida에서 디컴파일한다.

문제 제목인 long sleep과 프로그램이 실행되다가 멈춘다는 문제 설명의 뉘앙스를 볼 때 프로그램 실행을 일정 시간 동안 멈추는 sleep함수와 관련된 문제인 듯 하다.

일단 main 함수로 가 보았다.

```
1  __int64 __fastcall main(int a1, char **a2, char **a3
2  {
3    int i; // [rsp+Ch] [rbp-34h]
4    _BYTE v5[40]; // [rsp+10h] [rbp-30h] BYREF
5    unsigned __int64 v6; // [rsp+38h] [rbp-8h]
6
7    v6 = __readfsqword(0x28u);
8    puts("Wait! I'm generating flag!!");
9    sub_1411(v5, a2);
10   printf("Here's your flag: DH{");
11   for ( i = 0; i <= 31; ++i )
12     printf("%02x", (unsigned __int8)v5[i]);
13   puts("}");
14   return 0LL;
15 }
```

sub_1411(v5, a2)를 호출하여 v5에 플래그를 생성하는 부분에 주목했다. 일단 sub_1411 함수로 가 보았다.

```
__int64 __fastcall sub_1411(__int64 a1)
{
  size_t v1; // rax
  _BYTE v3[120]; // [rsp+20h] [rbp-80h] BYREF
  unsigned __int64 v4; // [rsp+98h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  if ( qword_4030 != 1 )
    exit(0);
  sub_19F5(v3);
  v1 = strlen("I will evolve into SUPER FLAG!!!!");
  sub_1A73(v3, "I will evolve into SUPER FLAG!!!!", v1);
  sub_1B14(v3, a1);
  return 1LL;
}
```

strlen과 sub_1A73에서 "I will evolve into SUPER FLAG!!!!" 문자열을 이용해 v3 버퍼를 조작하는 부분이 눈에 보인다. 함수를 계속 따라가 보았다.

```
__int64 __fastcall sub_1A73(__int64 a1, __int64 a2, unsigned __int64 a3)
{
  __int64 result; // rax
  unsigned int i; // [rsp+2Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = i;
    if ( a3 <= i )
      break;
    *(_BYTE *)(a1 + (unsigned int)(*(_DWORD *)(a1 + 64))++) = *(_BYTE *)(a2 + i);
    if ( *(_DWORD *)(a1 + 64) == 64 )
    {
      sub_1552(a1, a1);
      *(_QWORD *)(a1 + 72) += 512LL;
      *(_DWORD *)(a1 + 64) = 0;
    }
  }
  return result;
}
```

sub_1A73 함수이다. 계속해서 함수를 따라갔다.

```
v17 = __readfsqword(0x28u);
v11 = 0;
v13 = 0;
while ( v11 <= 0xF )
{
  v16[v11++] = (*(unsigned __int8 *)(v13 + 2 + a2) << 8) | (*(unsigned __int8 *)(
  v13 += 4;
}
while ( v11 <= 0x3F )
{
  v16[v11] = v16[v11 - 16]
           + (__ROL4__(v16[v11 - 15], 14) ^ __ROR4__(v16[v11 - 15], 7) ^ (v16[v11
           + v16[v11 - 7]
           + ((v16[v11 - 2] >> 10) ^ __ROL4__(v16[v11 - 2], 13) ^ __ROL4__(v16[v1
  if ( qword_4030 )
    sub_14D2();
  ++v11;
}
v3 = a1[20];
```

Sub_1552 함수이다. If문 안에 들어간 sub_14D2 함수를 따라갔다.

```
unsigned __int64 sub_14D2()
{
  signed __int64 v0; // rax
  struct timespec rqtp; // [rsp+10h] [rbp-30h] BYREF
  timespec rmtp; // [rsp+20h] [rbp-20h] BYREF
  unsigned __int64 v4; // [rsp+38h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  ++qword_4038;
  qword_4038 *= 2LL;
  rqtp.tv_sec = qword_4038;
  rqtp.tv_nsec = 1LL;
  v0 = sys_nanosleep(&rqtp, &rmtp);
  return v4 - __readfsqword(0x28u);
}
```

sub_14D2 함수에서 문제와 관련이 있어 보이는 nanosleep 함수를 찾을 수 있었다.

이 nanosleep이 프로그램이 실행이 제대로 안되는 원인이므로 nanosleep을 실행시키는 sub_14D2 함수를 실행시키기 않게 해야 한다.

즉, 아까 위에서 확인한 sub_1552 함수의 sub_14D2가 들어간 if문이 조건에 걸리지 않게 해야 한다는 의미이다.

그러면 pwndbg에서 조건을 무효화하기 위해 qword_4030 값을 0으로 설정할 수 있을 것이다. 이때는 set *(unsigned __int64 *)0x404010 = 0와 같은 명령어를 사용할 수 있다.

이 경우 if (qword_4030)는 항상 False가 되어 sub_14D2() 함수는 호출되지 않고, 프로그램이 정상적으로 실행 완료되어 플래그를 얻어낼 수 있게 된다.

플래그를 얻기 위한 계획은 모두 세워두었으나 prob.elf 파일을 리눅스로 옮겨 실행할 시 파일에 디버깅 심볼이 없어 pwndbg에서의 정상적인 실행이 불가능했다.

그래서 대신 풀이 시나리오를 작성했다.

1. pwndbg에서 prob 파일을 실행한다.

2. sub_1552 함수의 진입점에 브레이크포인트를 설정한다.

3. 프로그램을 실행하여 브레이크포인트에 도달한다.

4. sub_1552 함수 내부를 확인하여 if(qword_4030) 조건문이 있는 부분을 찾아야 한다. disassemble sub_1552이나 context disasm과 같은 명령어를 통해 어셈블리를 확인하고, qword_4030을 참조하는 부분을 찾아 해당 명령어 주소를 확인한다.

5. qword_4030 관련 명령어가 있는 주소에 도달하면 qword_4030의 값을 0으로 변경한다. 예를 들어, 주소가 0x4030인 경우 set *(long long*)0x4030 = 0과 같은 명령어를 사용할 수 있다.

6. x/gx (메모리 주소)과 같은 메모리 출력 명령을 활용하여 변경 사항을 확인한다.

7. continue 명령어로 프로그램을 계속 실행해 플래그를 얻어낸다.