Sleep (rev)

1.문제

```
제 고양이가 Flag를 먹고 잠이 들어버렸어요...
제 고양이는 한번 자면 일어나질 않는데 어쩌죠?
제발 고양이를 깨워주세요ㅠㅠ

sleep.exe는 encode.bin 파일을 해독하는 프로그램 입니다.
```

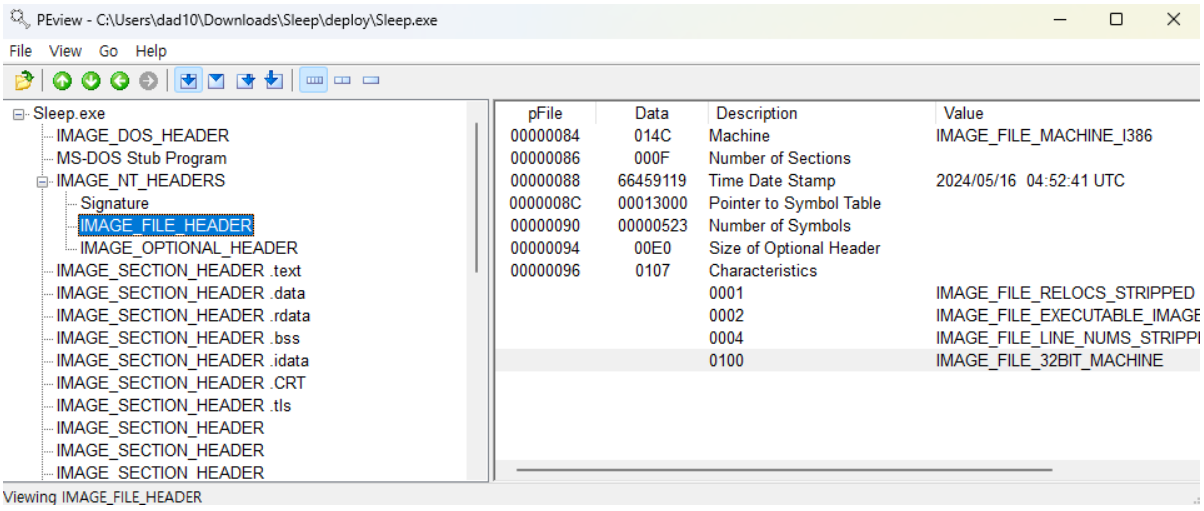
`Sleep` 문제를 실행하면 귀여운 고양이가 자고 있는 걸 볼 수 있습니다.

아무래도 Sleep() 함수가 아주 길게 동작하고 있는 것 같네요..

분석을 시작해 봅시다!

2. 분석

> PEview



- 우선 PE 파일의 구조를 쉽게 보여주는 PEview를 이용해 컴파일 환경을 알아봅시다!
- IMAGE_FILE_HEADER에 보면 32bit 환경으로 컴파일된 것을 알 수 있네요.

```
💡PE 파일이란? 윈도우 실행파일이라고 부르며 윈도우OS에서 사용되는 실행파일 형식을 의미한다.

🔗window docs: https://learn.microsoft.com/ko-kr/windows/win32/debug/pe-format
```

> Sleep 디컴파일 화면 main()

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[30]; // [esp+16h] [ebp-32h] BYREF
  char FileName[16]; // [esp+34h] [ebp-14h] BYREF

  __main();
  strcpy(FileName, "encoded.bin");
  puts("  ^--^");
  puts(asc_40403C);
  puts("Sleep .....\n");
  Sleep(0xDEA00000);
  puts("Oh.. Wa...ke... up..?\n");
  Sleep(0x7D0u);
  puts("  ^--^");
  puts(asc_40403C);
  puts("No... Sleep .....\n");
  Sleep(0xDEA00000);
  puts("  ^--^");
  puts(asc_404081);
  puts("Wake UP :) \nWait! I will give you flag!!! ");
  Sleep(0x3E8u);
  putchar(10);
  flag(FileName, v4);
  system("pause");
  return 0;
}
```

- IDA를 이용해서 Sleep 문제의 main 함수를 디컴파일해 봤습니다.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[30]; // [esp+16h] [ebp-32h] BYREF
  char FileName[16]; // [esp+34h] [ebp-14h] BYREF

  __main();
  strcpy(FileName, "encoded.bin");                  // Filename 변수에 encoded.bin 저장
  puts("  ^--^");
  puts(asc_40403C);
  puts("Sleep .....\n");
  Sleep(0xDEA00000);                                // 0xDEA00000 만큼 Sleep
  puts("Oh.. Wa...ke... up..?\n");
  Sleep(0x7D0u);                                    // 0x7D0 만큼 Sleep
  puts("  ^--^");
  puts(asc_40403C);
  puts("No... Sleep .....\n");
  Sleep(0xDEA00000);                                // 0xDEA00000 만큼 Sleep
  puts("  ^--^");
  puts(asc_404081);
  puts("Wake UP :) \nWait! I will give you flag!!! ");
  Sleep(0x3E8u);                                    // 0x3E8 만큼 Sleep
  putchar(10);
  flag(FileName, v4);                               // flag('encoded.bin', v4) : v4는 버퍼입니다.
  system("pause");
  return 0;
}
```

- 중간중간 길게 Sleep 함수가 들어있어서 프로그램이 진행되지 못하는군요..
- 마지막에 flag 함수도 한번 들어가 봐야 할 것 같습니다.

> flag()

```
int __cdecl flag(char *FileName, char *a2)
{
  char v3; // dl
  int v4; // eax
  unsigned __int8 Buffer; // [esp+17h] [ebp-11h] BYREF
  FILE *Stream; // [esp+18h] [ebp-10h]
  int v7; // [esp+1Ch] [ebp-Ch]

  Stream = fopen(FileName, "rb");
  if ( !Stream )
    return puts("Failed to open file for reading.");
  func1();
  v7 = 0;
  while ( fread(&Buffer, 1u, 1u, Stream) == 1 )
  {
    func2(v7);
    Buffer ^= 3 * (_BYTE)v7;
    func3();
    Buffer = (Buffer >> 4) | (16 * Buffer);
    if ( (v7 & 1) != 0 )
      v3 = 95;
    else
      v3 = -57;
    Buffer ^= v3;
    Buffer ^= 0xA3u;
    Buffer = (32 * Buffer) | ((int)Buffer >> 3);
    v4 = v7++;
    a2[v4] = Buffer;
    if ( (v7 & 1) == 0 )
      func4();
  }
  a2[v7] = 0;
  func5();
  fclose(Stream);
  return printf("flag: SOTI{%s}\n", a2);
}
```

- `flag` 함수의 맨 아래 줄을 보면 `printf("flag: SOTI{%s}\n", a2);` 로 `flag`를 출력하고 있습니다.
- 우리는 Sleep 함수만 조작해서 `flag()` 함수가 실행되게만 하면 될 것 같네요!
- 풀이는 다음 장에 있습니다.

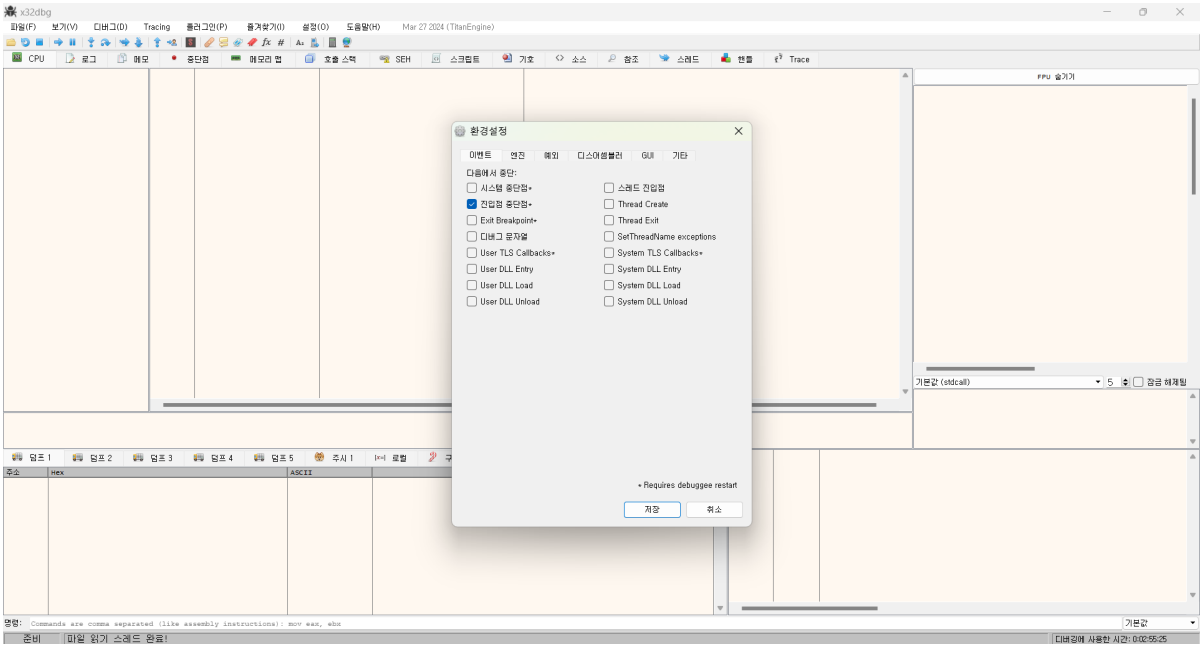
3. 풀이

`flag()` 함수까지 가게하는 다양한 방법이 있지만, 우리는 `sleep()` 함수의 값을 직접 조작해볼 겁니다. `x64dbg`를 사용해 봅시다 ✨

> 💡 x64dbg 란? 윈도우의 디버거 중 하나로, 실행파일을 실행하면서 동적으로 분석할 수 있게 도와주는 도구이다. x64dbg는 64bit형식을 지원하고, x32dbg는 32bit 형식을 지원한다.
>
> 🔗 x64dbg 홈페이지
> https://x64dbg.com/

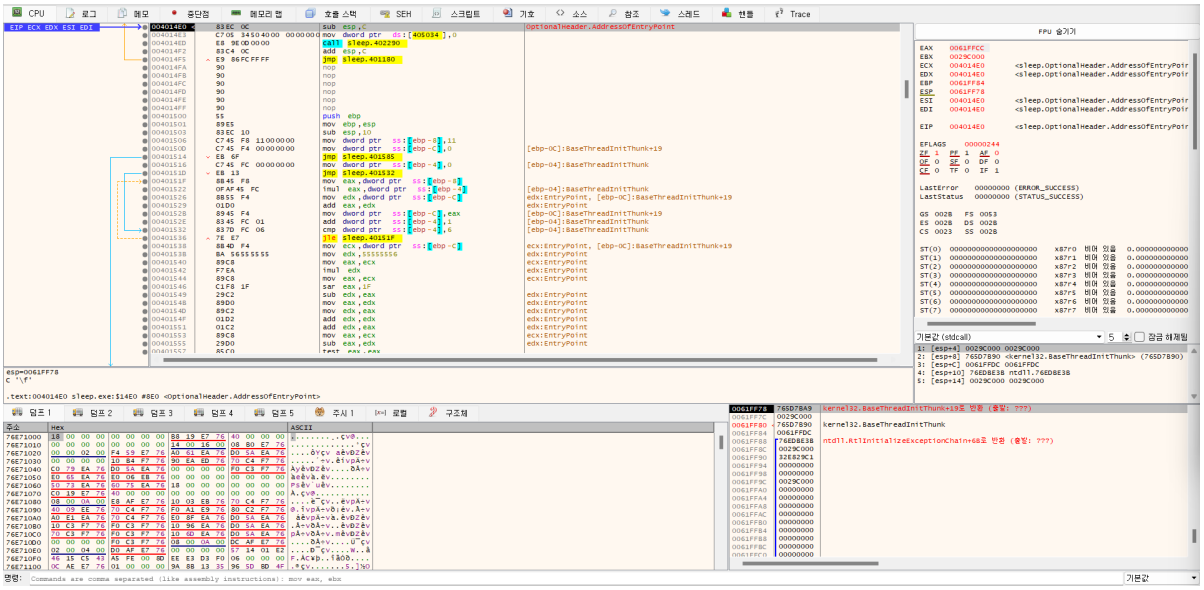
> x32dbg 설정

우리는 위에서 `sleep.exe` 파일이 32bit로 컴파일 되었다는 것을 알고 있었습니다. `x32dbg`를 일단 열어봅시다!

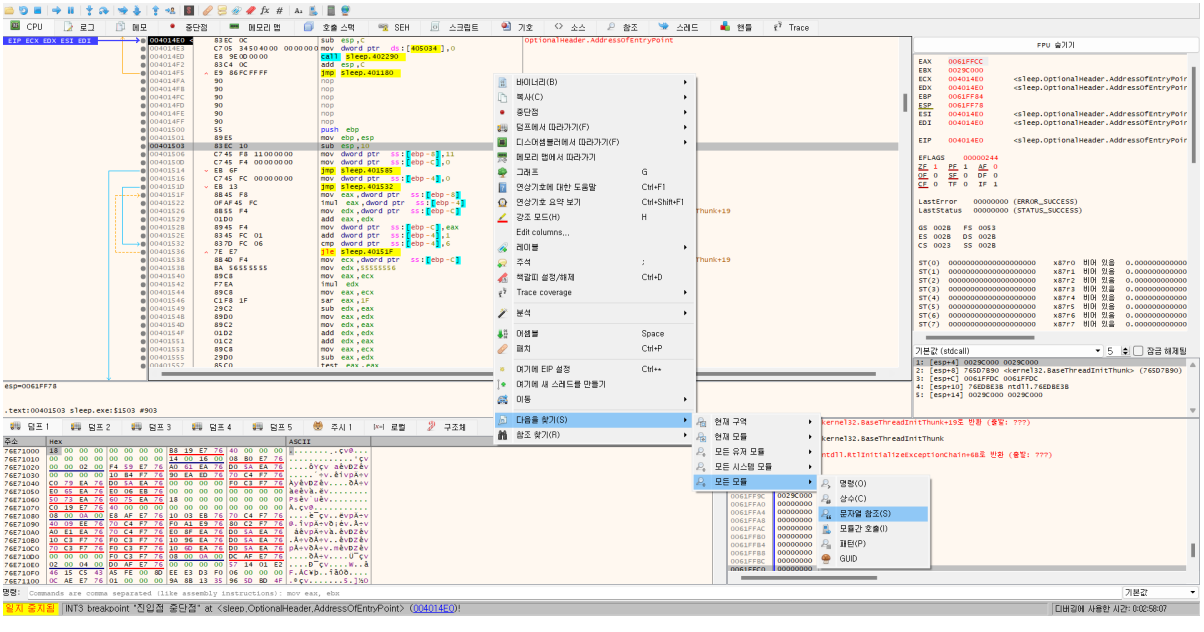


- 설정 -> 환경설정으로 들어가 `진입점 중단점`만 체크 해줄 겁니다!
- 체크가 완료되었다면 저장 버튼을 누르고 `sleep.exe`를 `x32dbg`에 드래그앤드랍 해봅시다!

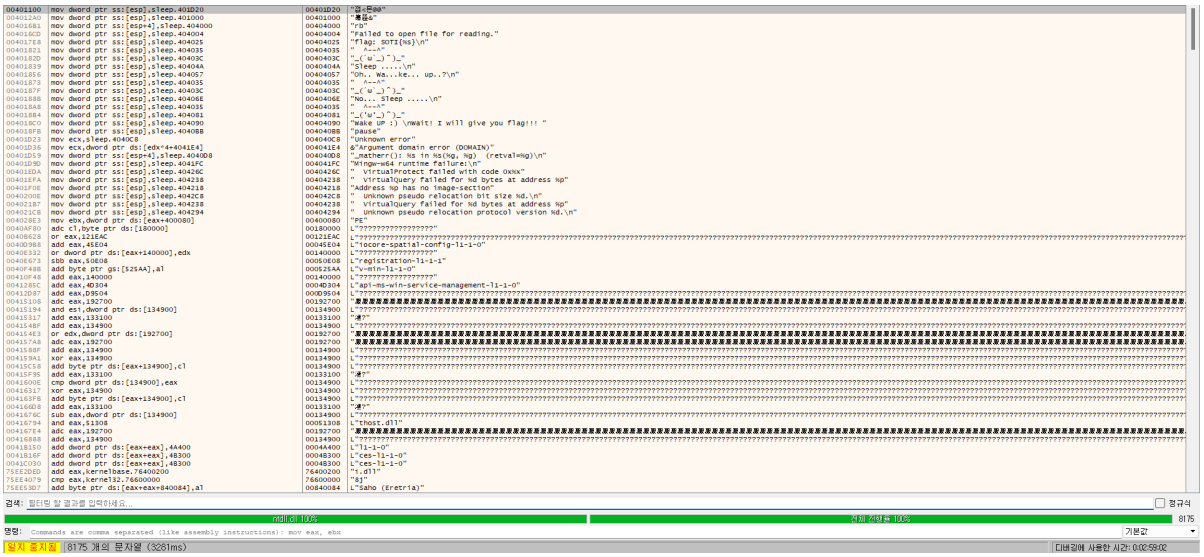
문자열 찾기



- 아래와 같이 `EntryPoint(진입지점)`에 `breakpoint(중단점)`가 걸렸습니다.
- 우리는 여기서 아까 실행했을 때 보았던 `Sleep .....`이라는 텍스트를 찾아볼 겁니다.

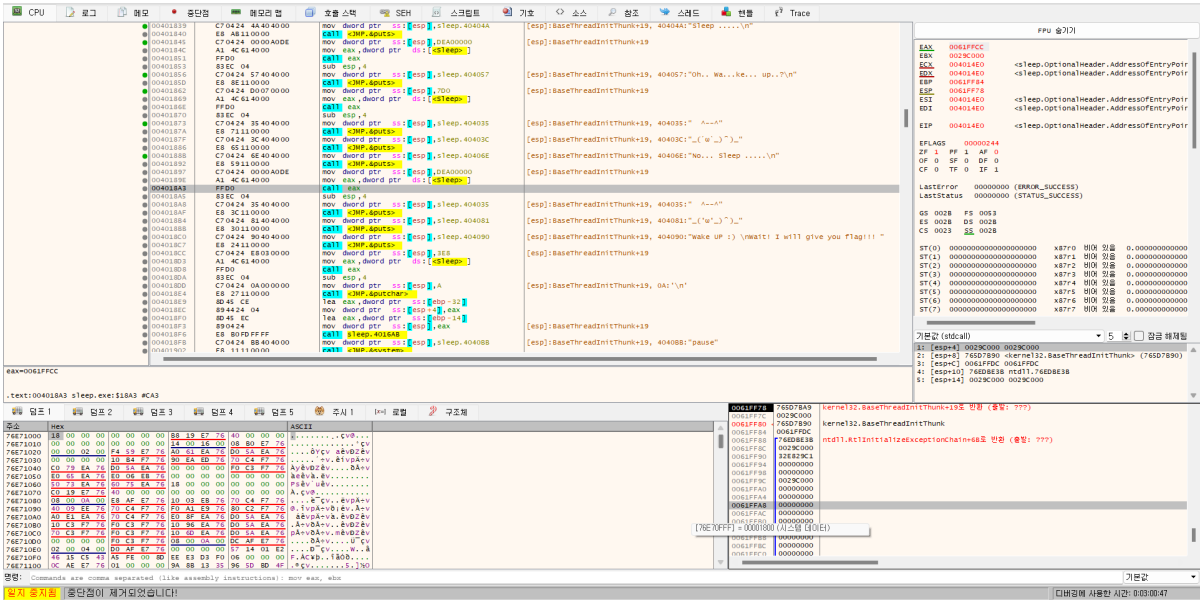


- 마우스를 우클릭 하면 위와 같은 탭이 나옵니다.
- 다음을 찾기 -> 모든 모듈 -> 문자열 참조를 클릭해주세요!



- 다음과 같이 현재 실행파일에 들어가 있는 문자열들이 검색이 됐습니다!
- 우리는 이 중에서 `Sleep .....`이라고 써있는 걸 더블클릭 해서 따라가 볼 겁니다.
- 왜냐하면 `Sleep .....`가 출력되고 나서 `Sleep()` 함수가 동작하고 있으니까요 !

Sleep 함수 우회



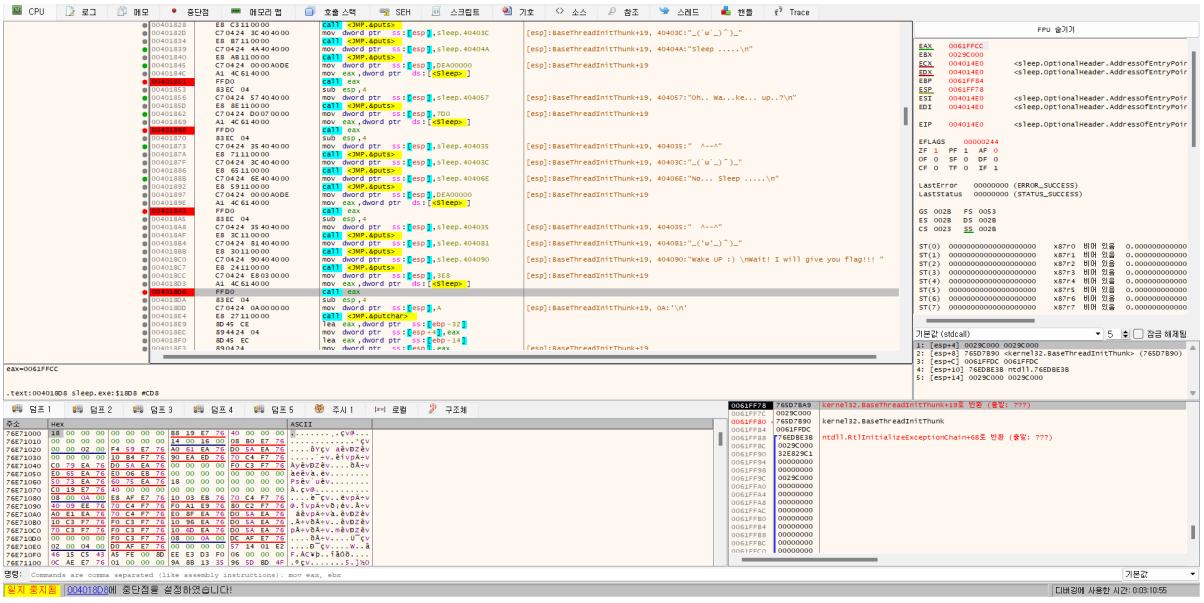
- 위와 같은 화면으로 오게 되었습니다. 익숙한 고양이가 보이는군요.
- `Sleep .....` 문자열도 여러 개가 있는 것을 볼 수 있네요.
- 어셈블리어라 읽기가 어렵네요.. 😳 하지만 우리는 중요한 부분만 보면 됩니다.
- 바로 `mov dword ptr ss:[esp],DEA00000`, `mov eax,dword ptr ds:[<Sleep>]`, `call eax` 부분입니다.

```
00401845 | C70424 0000A0DE          | mov dword ptr ss:[esp],DEA00000          |
0040184C | A1 4C614000              | mov eax,dword ptr ds:[<Sleep>]           |
00401851 | FFD0                     | call eax                                 |
```

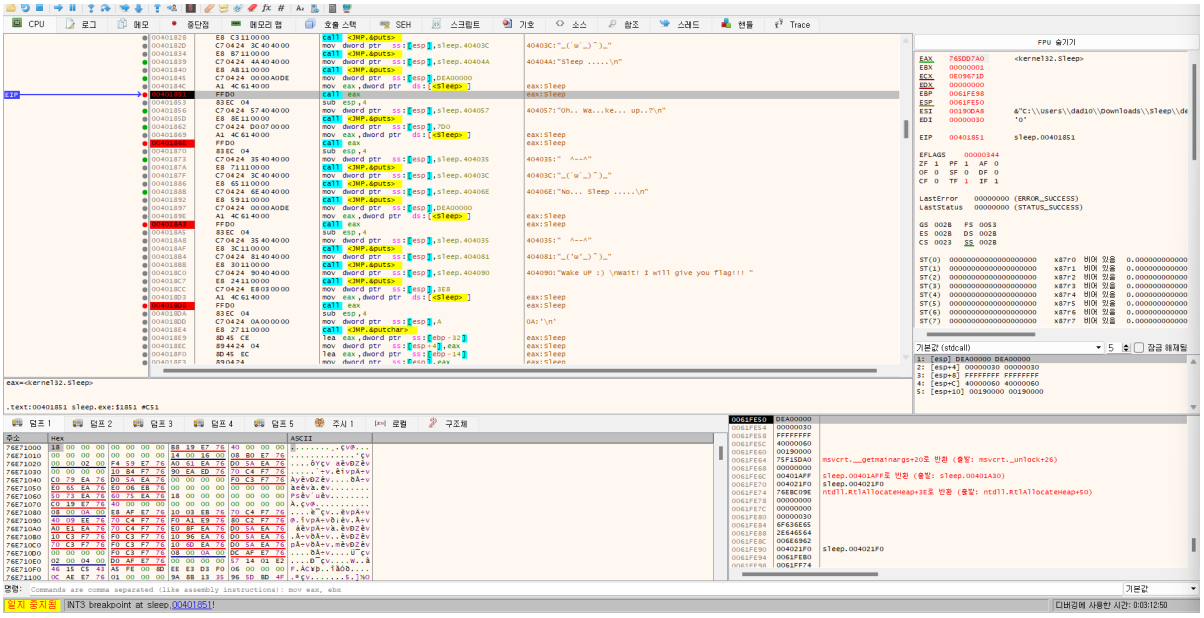
- 먼저 `mov dword ptr ss:[esp],DEA00000` 명령을 통해 스택의 최상단에 `DEA00000` 값을 집어넣었습니다. 이는 뒤에 함수를 호출하는 과정에서 사용할 값을 할당하는 과정입니다.
- 다음으로 `mov eax,dword ptr ds:[<Sleep>]` 로 `eax` 레지스터에 `Sleep()` 함수의 주솟값을 저장해줬습니다.
- 마지막으로 `call eax` 함수를 호출 해줬습니다.
- C언어 코드로 하면 아래와 같습니다.

```
Sleep(0xDEA00000);
```

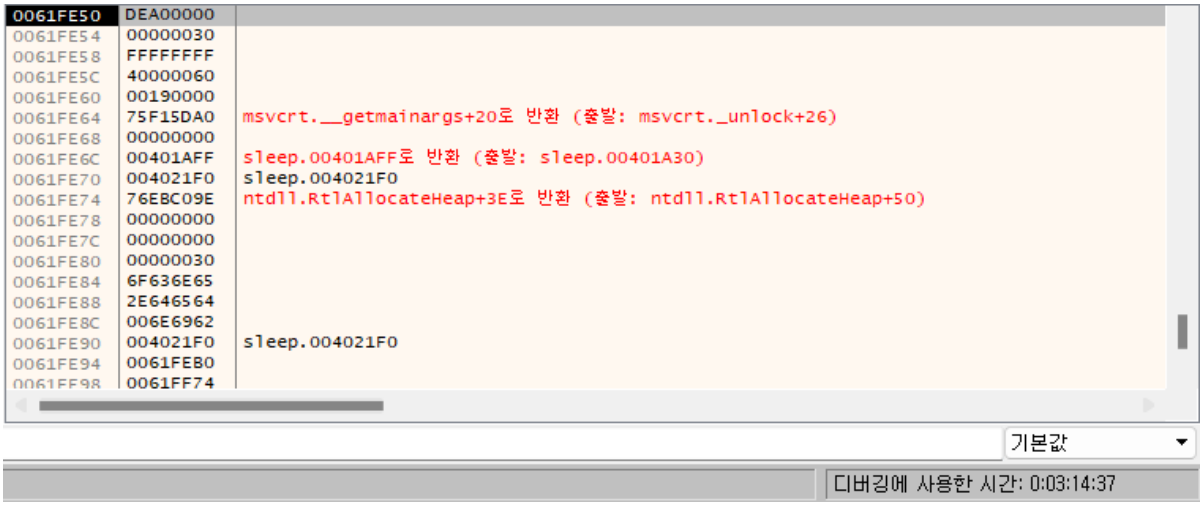
- 이제 우리는 저 값을 위조하면 된다는 것을 알게 되었습니다.
- 코드영역에서 `DEA00000` 과 같이 `Sleep()` 함수의 들어갈 인자 값을 다 0으로 고치는 것도 방법이지만, 우리는 한번 스택의 값을 조작 해봅시다.



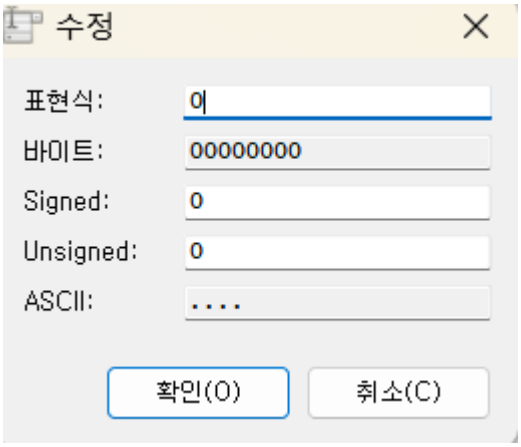
- 먼저 `call eax` 하는 부분을 클릭하고 `F2` 를 눌러 `breakpoint` 를 걸어줍니다.
- `F9` 를 눌러 걸린 `breakpoint` 까지 이동해 보겠습니다.



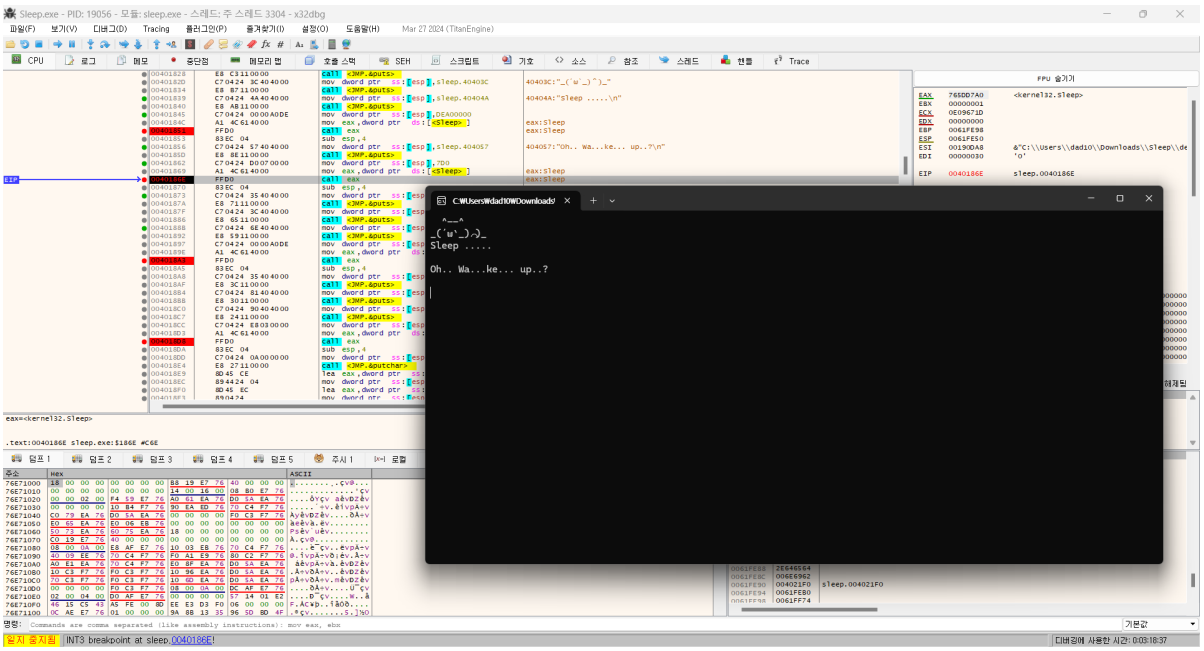
- 현재 화면과 같이 `EIP(다음에 실행될 영역을 가리키는 레지스터)`가 우리가 건 `breakpoint`까지 왔습니다.
- 우리는 전에 분석에서 `Sleep()` 함수에서 실행될 값을 스택에 최상단에 넣는다는 걸 알았습니다.
- 오른쪽 맨 아래 화면을 보겠습니다.



- 이 화면은 스택 영역을 비춰주는 화면입니다!
- `DEA00000` 값이 들어가 있는 것을 볼 수 있습니다.
- 이걸 더블클릭 후 0으로 바꿔 줄 겁니다.



- 확인을 누르고 다음 `F9`를 눌러봅시다!



- 실행화면을 보면 우리가 볼 수 없었던 `Oh.. Wa...ke... up..?`를 볼 수 있네요!
- 우리는 `sleep()` 함수를 우회했습니다 😊
- 위와 같은 방법으로 `flag()` 함수까지 가면 되겠죠? 가봅시다!!!

> Flag

```
  ^__^
_(´ω`_)ゞ_
Sleep .....

Oh.. Wa...ke... up..?

  ^__^
_(´ω`_)ゞ_
No... Sleep .....

  ^__^
_('ω'_)ゞ_
Wake UP :)
Wait! I will give you flag!!!

flag: SOTI{paTch_@Nd_WAKe_up!!!}
계속하려면 아무 키나 누르십시오 . . .
```

- Flag를 받아냈습니다 😁

```
SOTI{paTch_@Nd_WAKe_up!!!}
```