

Assignment2

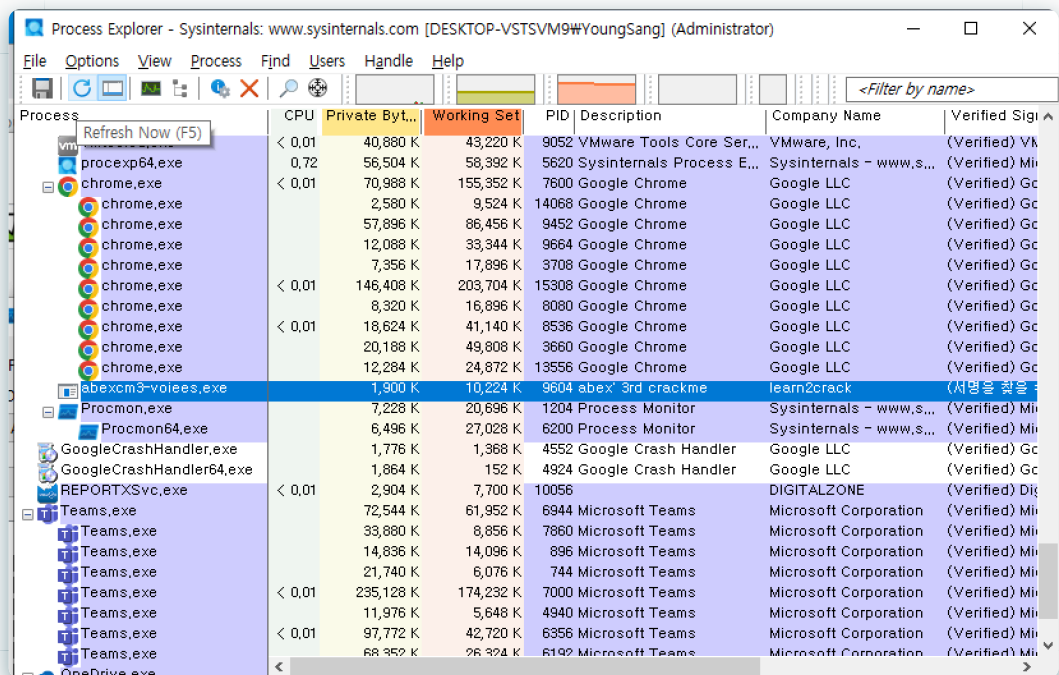
강민준

✓ Description.

```
1 abexcm3-voieee.exe를 분석 및 코드 패치 없이 keyfile을 인증
2 1. 실행흐름 파악
3 2. 분석 지점을 파악 (=> 분석 이후, 어떤 지점을 주안점으로 삼아야하는가?)
4 3. 코드 동적 분석 및 내부 알고리즘 파악 (=> 분석 지점에 대한 동적 분석 수행)
5 /**
6     keyfile을 만들어내는 것, binary 수정X, 분석한 알고리즘을 파악한대로     필요한 parameter를 채워넣는 방식
7 **/
```

✓ 분석 과정

1. 프로세스가 실행될 때, 단일 모듈인지 확인하기 위하여 **Process Explorer** 를 이용한다.



abexcm3-voieee.exe 는 단일 모듈의 실행 파일인 것을 확인해볼 수 있다.

2. xdbg를 이용하여 실행흐름을 분석한다.

00401000	6A 00	push 0	EntryPoint
00401002	68 00204000	push abexcm3-voies.402000	402000:"abex' 3rd crackme"
00401007	68 12204000	push abexcm3-voies.402012	402012:"Click OK to check for the keyfile."
0040100C	6A 00	push 0	
0040100E	E8 8C000000	call <JMP.&MessageBox>	
00401013	6A 00	push 0	
00401015	68 80000000	push 80	
0040101A	6A 03	push 3	
0040101C	6A 00	push 0	
0040101E	6A 00	push 0	
00401020	68 00000080	push 80000000	
00401025	68 B9204000	push abexcm3-voies.4020B9	4020B9:"abex.l2c"
0040102A	E8 5E000000	call <JMP.&CreateFileA>	
0040102F	A3 CA204000	mov dword ptr ds:[4020CA],eax	
00401034	83F8 FF	cmp eax,FFFFFFFF	
00401037	74 3C	je abexcm3-voies.401075	
00401039	6A 00	push 0	
0040103B	FF35 CA204000	push dword ptr ds:[4020CA]	
00401041	E8 4D000000	call <JMP.&GetFileSize>	
00401046	83F8 12	cmp eax,12	
00401049	75 15	jne abexcm3-voies.401060	
0040104B	6A 00	push 0	
0040104D	68 35204000	push abexcm3-voies.402035	402035:"Well done!"
00401052	68 40204000	push abexcm3-voies.402040	402040:"Yep, keyfile found!"
00401057	6A 00	push 0	
00401059	E8 41000000	call <JMP.&MessageBox>	
0040105E	EB 28	jmp abexcm3-voies.401088	
00401060	6A 00	push 0	
00401062	68 79204000	push abexcm3-voies.402079	402079:"Error"
00401067	68 1F204000	push abexcm3-voies.40207F	40207F:"The Found file is not a valid keyfile!"
0040106C	6A 00	push 0	
0040106E	E8 2C000000	call <JMP.&MessageBox>	
00401073	EB 13	jmp abexcm3-voies.401088	
00401075	6A 00	push 0	
00401077	68 54204000	push abexcm3-voies.402054	402054:"Error"
0040107C	68 5A204000	push abexcm3-voies.40205A	40205A:"HMMMM, I can't find the file!"
00401081	6A 00	push 0	
00401083	E8 17000000	call <JMP.&MessageBox>	
00401088	E8 0C000000	call <JMP.&ExitProcess>	
0040108D	FF25 54304000	jmp dword ptr ds:[<&CreateFileA>]	JMP.&CreateFileA
00401093	FF25 58304000	jmp dword ptr ds:[<&GetFileSize>]	JMP.&GetFileSize
00401098	FF25 5C304000	jmp dword ptr ds:[<&ExitProcess>]	JMP.&ExitProcess
0040109F	FF25 64304000	jmp dword ptr ds:[<&MessageBox>]	JMP.&MessageBox
004010A5	0000	add byte ptr ds:[eax],al	
004010A7	0000	add byte ptr ds:[eax],al	

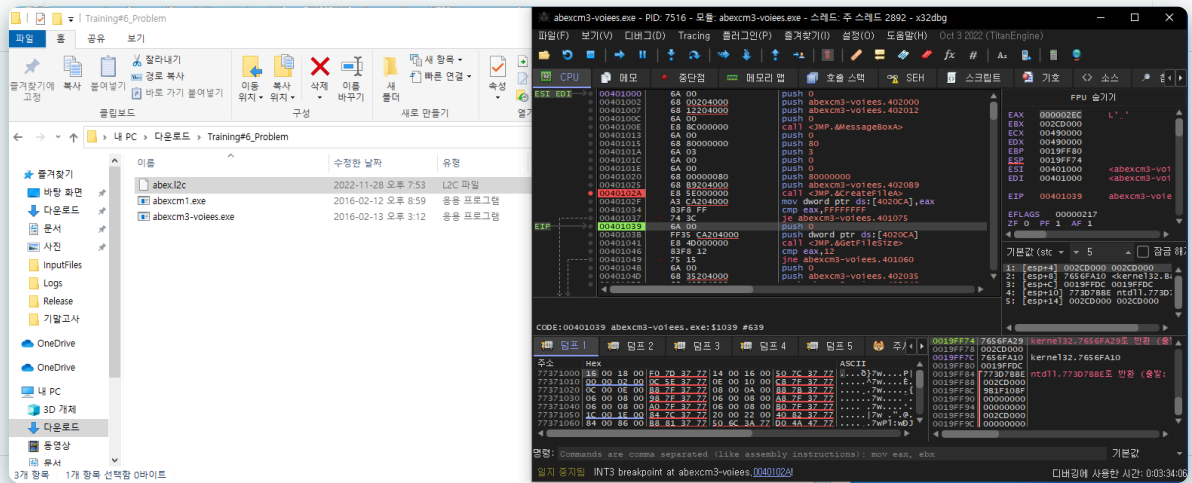
위와 같이, `CreateFileA()` 의 인자값으로 {"abex.l2c", 80000000, 0, 0, 3, 80, 0}을 넘겨 특정 operation을 수행한 다음, 리턴 값과 0xFFFFFFFF를 비교한다. 만약 같다면, Error 문으로 분기하게 된다. 코드 패치 없이 해당 분기문으로 분기하지 않기 위해서 `CreateFileA()` 의 명세를 살펴보아야한다. [CreateFileA 명세](#)

C++		Copy
HANDLE <code>CreateFileA</code> (
[in]	LPCSTR	lpFileName,
[in]	DWORD	dwDesiredAccess,
[in]	DWORD	dwShareMode,
[in, optional]	LPSECURITY_ATTRIBUTES	lpSecurityAttributes,
[in]	DWORD	dwCreationDisposition,
[in]	DWORD	dwFlagsAndAttributes,
[in, optional]	HANDLE	hTemplateFile
);		
OPEN_EXISTING		
3	Opens a file or device, only if it exists.	
	If the specified file or device does not exist, the function fails and the last-error code is set to ERROR_FILE_NOT_FOUND (2).	
	For more information about devices, see the Remarks section.	

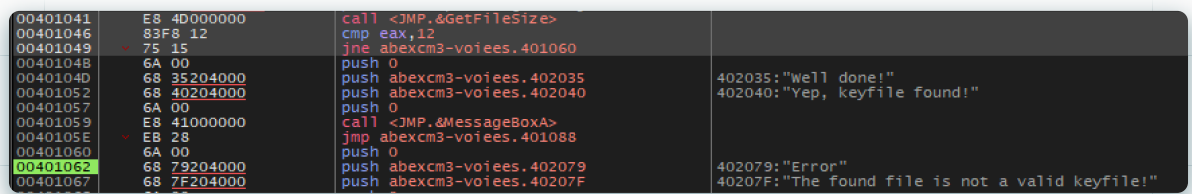
명세를 읽어보면, `CreateFileA()` 의 첫번째 인자인 "abex.l2c" 인 파일에 대해서 OPEN_EXISTING 모드로 open한다는 것을 확인할 수 있다. 또한, 해당 함수가 실패할 경우, `INVALID_HANDLE_VALUE` 값을 반환한다는 것을 알 수 있다. "abex.l2c"파일이 존재하지 않을 경우, 해당 함수가 실패한다는 것을 확인할 수 있었는데 이 때의 반환 값은 0xFFFFFFFF 이다.

3. `CreateFileA()` 함수가 실패하지 않도록 적절하게 인자를 채워준다.

CreateFileA() 은 WIN32 API로서 시스템콜을 내부적으로 호출한다. "abex.l2c" 파일을 적절하게 생성해준다면 아래와 같이 원하는 분기 내부로 들어올 수 있게된다.



그럼에도 불구하고, 이번에는 **The found file is not a valid keyfile!** 라는 결과를 확인할 수 있었다. 해당 실패 분기로 분기하게 된 원인을 살펴보자.



GetFileSize() 함수의 반환 값이 0x12이 아니기 때문에, Error로 분기한 것을 확인할 수 있다.

GetFileSize() 의 명세를 확인하여, 반환 값이 0x12가 되도록 만들어보자. **GetFileSize 명세**

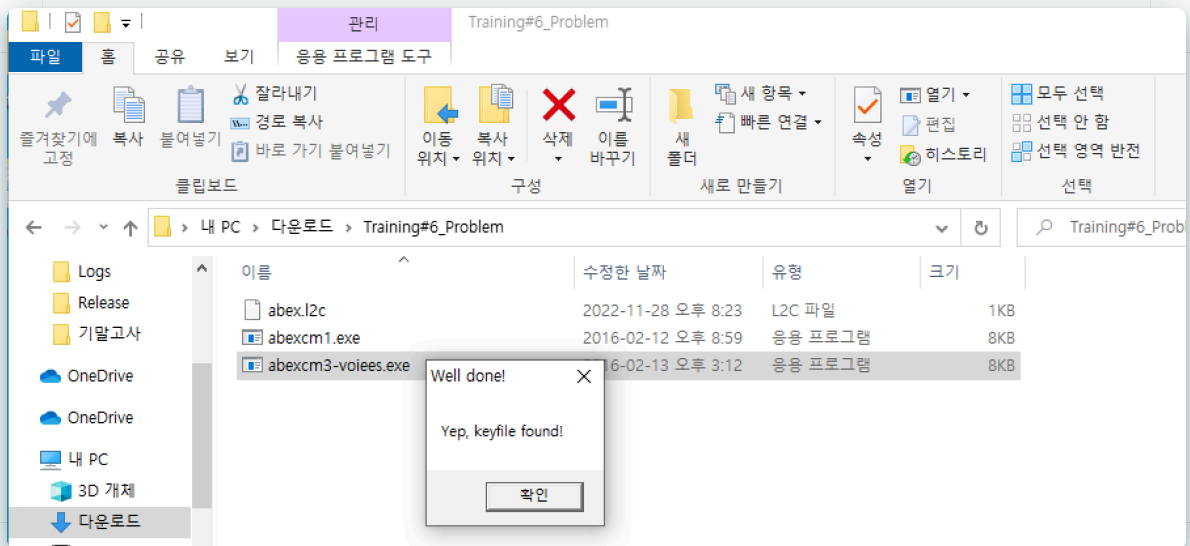
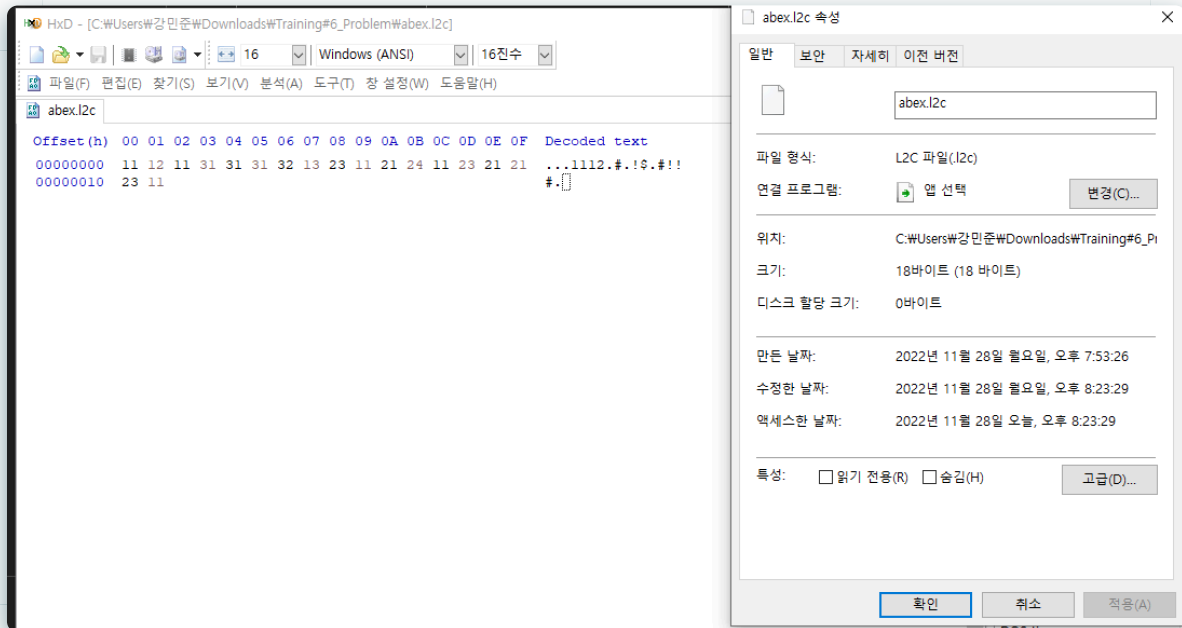
Return value

If the function succeeds, the return value is the low-order doubleword of the file size, and, if *lpFileSizeHigh* is non-NULL, the function puts the high-order doubleword of the file size into the variable pointed to by that parameter.

명세에 따라, DWORD(32bit 아키텍처 기준, 4바이트)의 형태로 실제 파일 크기를 반환하는 것을 확인할 수 있다. 이와 같은 사실을 통해서, "abex.l2c" 파일의 크기를 0x12바이트로 설정해주면 우리가 원하는 실행흐름인 **0x0040104B ~ 0x0040105E** 순으로 수행할 수 있다.

4. 마무리

HxD 를 이용하여 해당 파일의 바이너리를 12₁₆(=18₁₀)바이트 크기로 설정해준다.



성공적으로 원하는 결과를 얻을 수 있었다.