AsProtect MUP

2013-06-18

Lazly

목차

1. 목적

0x01 문서의 목적

문서의 목적은 본인이 패킹된 파일을 MUP을 하면서 가장 재미 있었던 Asprotect MUP을 문서화 시켜보려고 합니다. 많은 패킹된 파일들을 언패킹 시켜본 것은 아니지만 자동스크립트도 통하질 않았고, 문서들도 전부 외국문서였던 패커는 처음이었습니다. 다행이도 아주 설명이 잘된 문서가 있어서 몇일 간의(띄엄띄엄 한달...걸린 것 같네요 허허) 삽질 끝에 MUP를 할 수 있었습니다.

서두에 밝히지만 본 문서는 내가 보고 MUP했던 문서를 초보자들이 좀더 보기 쉽게 접하길 바라는 마음에 재해석 하는 문서입니다. 그렇기에 본 문서 내용의 저작권은 원 문서를 작성하신 그분께 있습니다. (그분이라 칭하는 이유는 coolsoft에서 받은 문서인데 coolsoft의 리뉴얼 때문에 누군지 정확히 모르겠습니다..)

0x02 문서에서 다루는 내용

본 문서는 언패킹에 대해 지식이 조금 있다는 가정하에 Asprotect 2.1 ska를 언패킹 하는 방법에 대해 설명하고자 합니다.

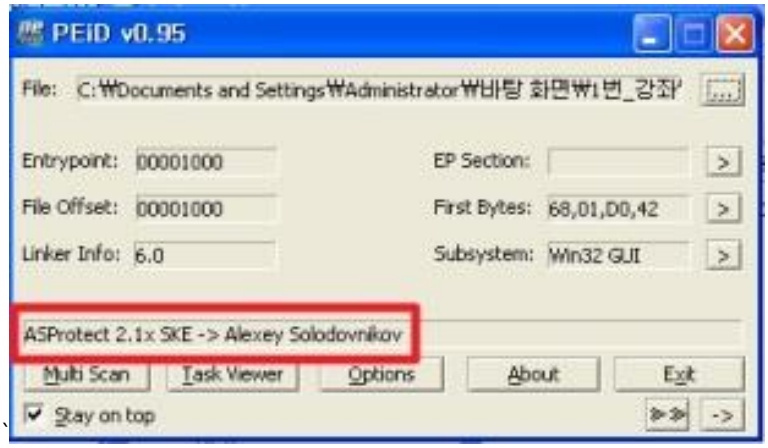
AsProtect MUP 부터 시작해서 왜 Asprotect MUP가 안되었는지, 그리고 어떻게 해 나가는지를 순차적으로 차근차근 초보자에게 맞춰서 제작 해 보려고 합니다.

2. 프로그램 설명

0x01 Asprotect

Asprotct 1.3 - > 2.x버전으로 넘어오면서 가장 크게 달라진 점은 IAT 주소공간을 제대로 연결시켜주지 않는다는 큰 특징이 있습니다.

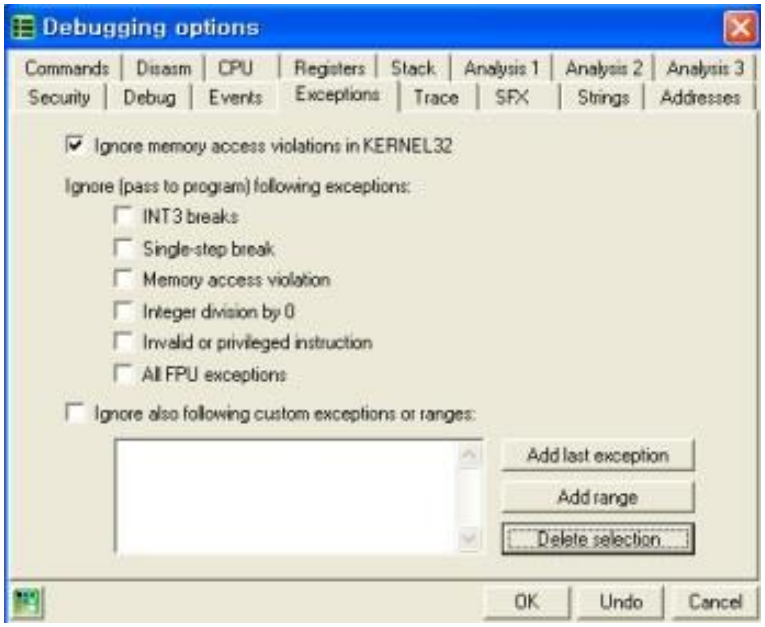
정확히는 AsProtect 내에서 연결되지만, 패킹을 풀어버리기 때문에 IAT가 제대로 연결 안 되는 것 입니다. 결국에는 IAT를 직접적으로 다 가져와서 프로그램 내에 삽입을 해주어야 한니다.



본 문서에서 다룰 ASProtect 2.1 SKE 정보 입니다.

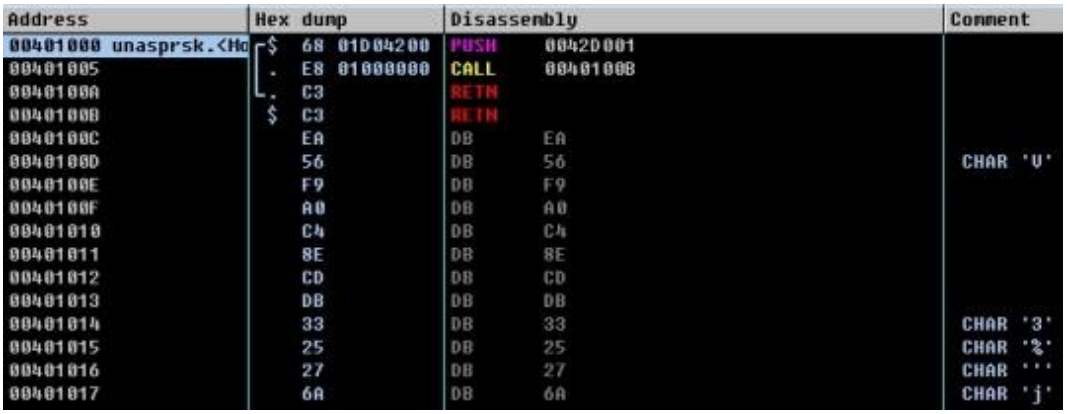
3. AsProtect MUP

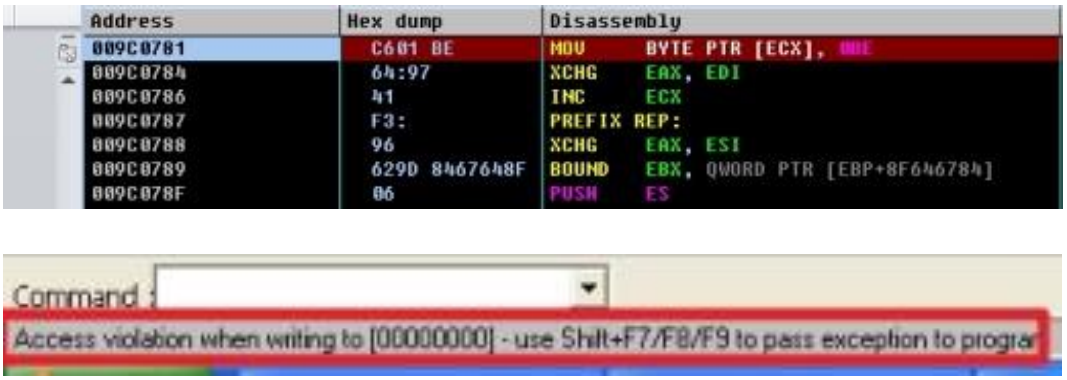
0x01. Find OEP



Ollydbg로 unaspack.exe를 실행 시킨뒤
Options -> Debugging options -> Exceptions 탭에서 설정 합니다.

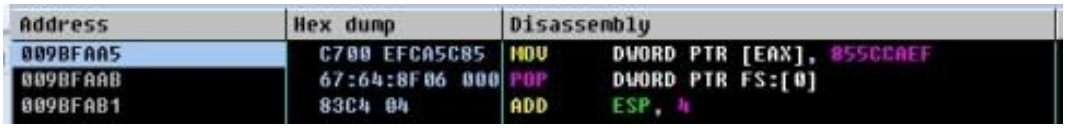
Ollydbg로 실행시키면 다음과 같이 익숙한 화면으로 넘어가게 됩니다.
AsPack과 마찬가지로 AsProtect는 실행 되기 바로 이전까지 실행 시킨 후 메모리뷰에서 Code영역에 Break를 걸어 주는 것이 정석 입니다.. F9로 진행 시켜 보겠습니다.



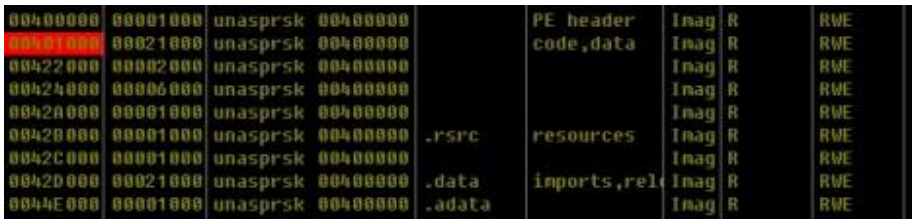


다음과 같은 화면이 걸리고 Access vilation When writing to [00000000] – use shift + f7/f8/f9 to pass exception to program이 걸림을 확인 할 수 있습니다..

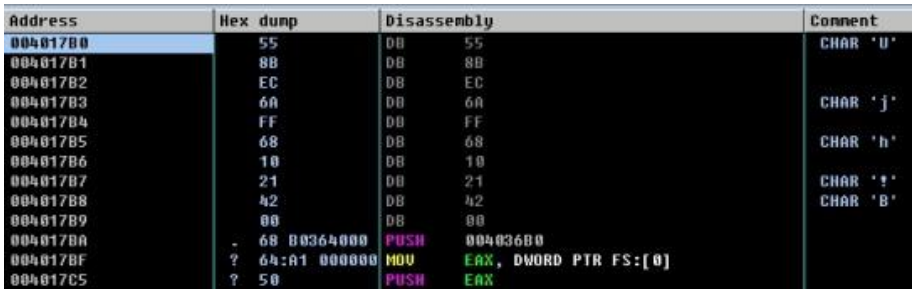
Shift + F9로 실행 되기 바로 직전까지의 실행 횟수를 세보겠습니다.
즉 26번째에 실행이 된다면 바로 25번째까지 shift + F9를 해주면 된다는 이야기 입니다.



바로 위에 주소까지 멈춤을 확인 할 수 있다. 필자 컴퓨터 기준으로 25번의 실행입니다.
이렇게 까지 실행을 해준 후 Ctlr + M을 눌러줘서 Code영역에 Break Point를 걸어 줍니다.

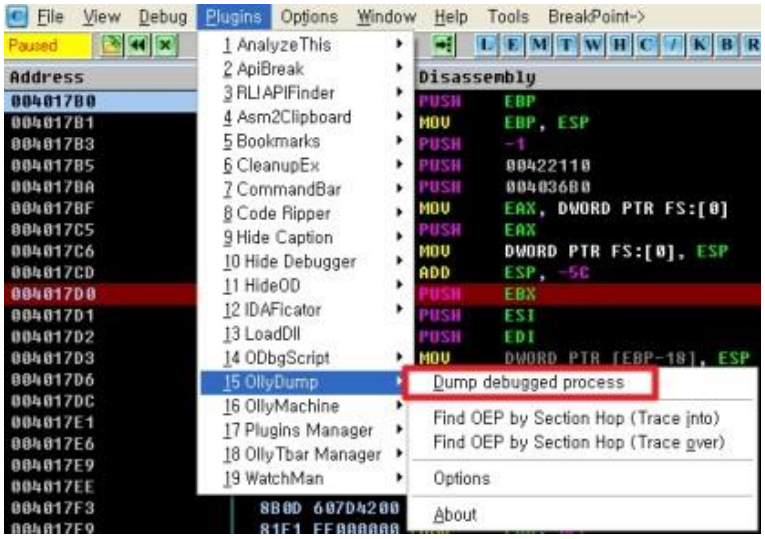


Break Point까지 걸어 준 뒤에 Shift + F9를 해주면 OEP를 찾을 수 있다.

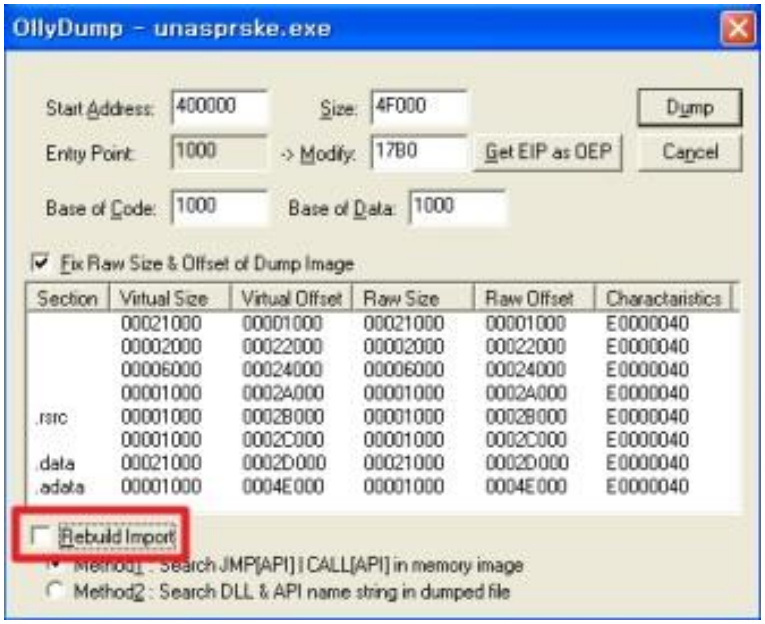


위의 화면은 패킹으로 인해 제대로 Code가 안 보이는 것 입니다.
오른쪽버튼 -> Analysis -> Remove analysis from module을 해주면 소스가 제대로 표시 됩니다.

0x02. DUMP

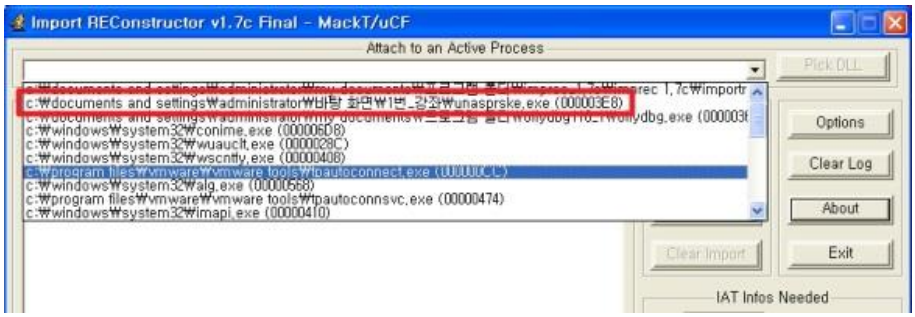


오른쪽버튼 -> Dump debugged process 혹은 Plugins -> OllyDump -> Dump debugged process의 옵션으로 파일을 Dump 할 수 있습니다.

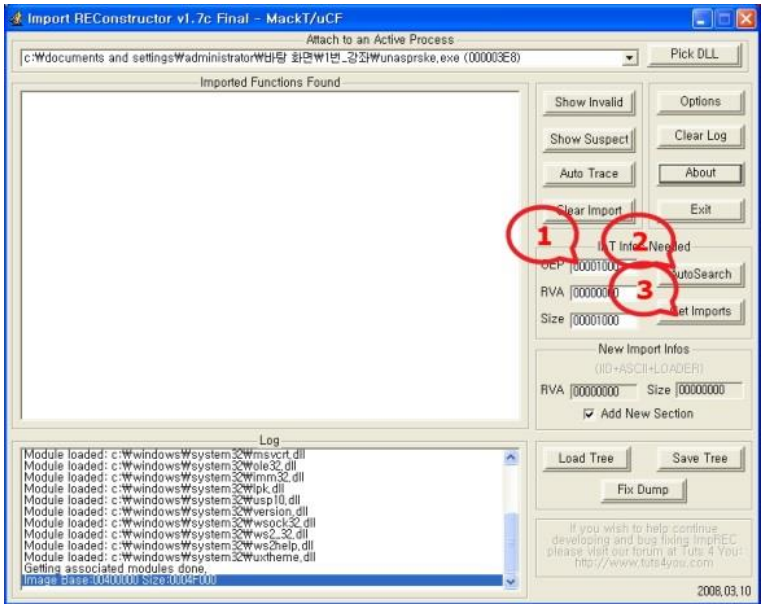


Rebulid Import의 옵션을 제거 해 준뒤에 Dump버튼을 눌러 저장 해줍니다.. Rebulid Improt의 경우 Ollydbg 자체 에서 지원하는 IAT 복구 옵션입니다. 많은 패치가 이뤄져서 성공률이 꽤 높다고 하지만 본인의 경우는 ImportREC를 쓰고 있기에 옵션을 제거 했습니다.

이렇게 Dump파일 까지 만들어 준뒤에 Ollydbg를 종료 시키지않고 바로 ImportREC 프로그램을 실행 시킵니다.



보시는 바와 같이 Ollydbg로 실행되고 있는 unasprske.exe에 붙을 수 있습니다.



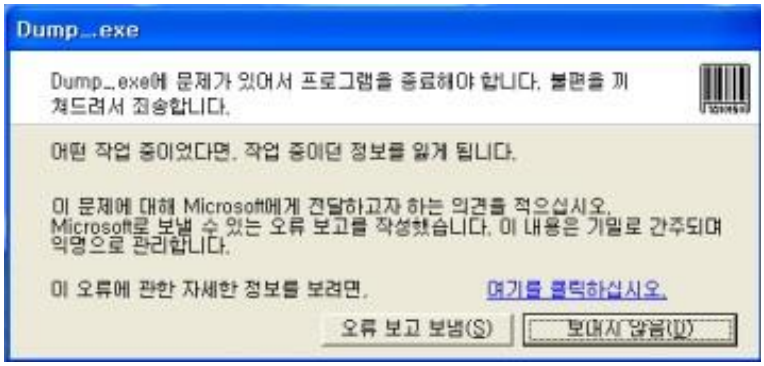
DUMP 했을 때에 OEP값을 적어주고 AutoSerch로 RVA와 Size를 알아내고 Get Imports를 해줍니다.
하지만 이 프로그램 역시 확실하게 값을 구해주는 것은 아닙니다.
IAT값들을 확실하게 알 수 있는 방법을 알려드리도록 하겠습니다.

OEP 살짝 아래 부분을 보면은 Kernel32를 Call하고 있는 부분이 보인다. 바로 IAT 부분들이 모여있는 부분임을 추측 할 수 있습니다.. DWORD PTR[42A250] 으로 Ctlr + G로 가보면 좀더 정확 하게 알 수 있다.

IAT가 있는 것을 확인 할 수 있다. 그리고 그 주위를 보면은 00 00 00 등이 중간에 껴있고 여러값들여 있는 것을 확인 할 수 있다. 필자의 경우 검색을 2A000부터 2000을 해 주었습니다.
어차피 범위검색이기 때문에 정확 하지 않아도 됨으로, Import REC가 제대로 RVA와 Size를 잡아 주지 못하는 것 같다면은 이런 식 으로 검색 해줍니다.

모두 검색 해 준뒤에 Fix Dump로 Dump 한 파일에 IAT를 수정 해 줍니다.

0x03. 에러 발생

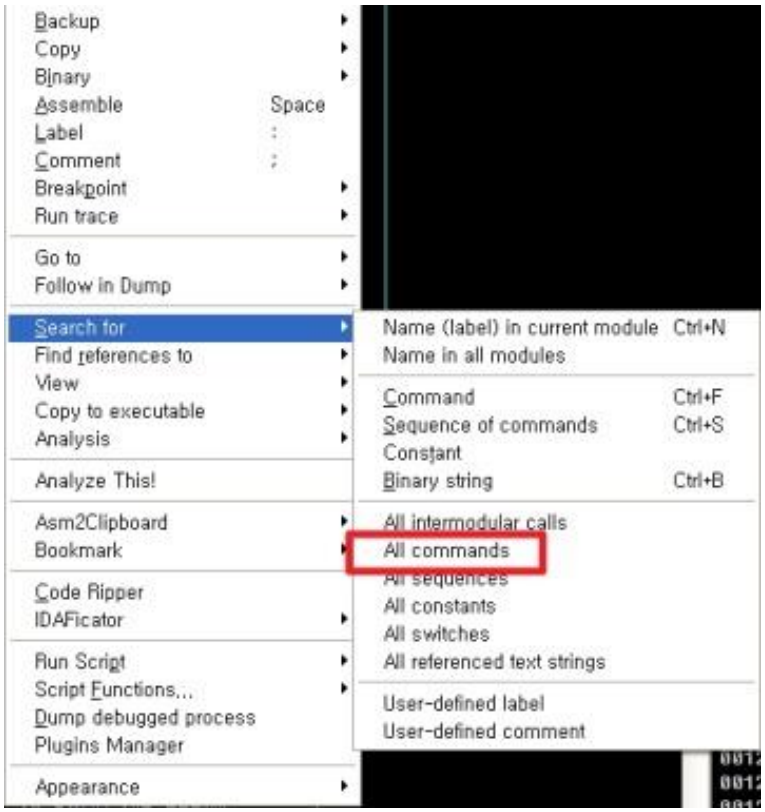


정상적인 MUP를 수행을 다 한 것 이라면 실행이 되는 것이 정상 이지만 AsProtect의 경우 실행이 되지않고 오류가 뜹니다. 왜 오류가 나는것인지 확인 해 보겠습니다.

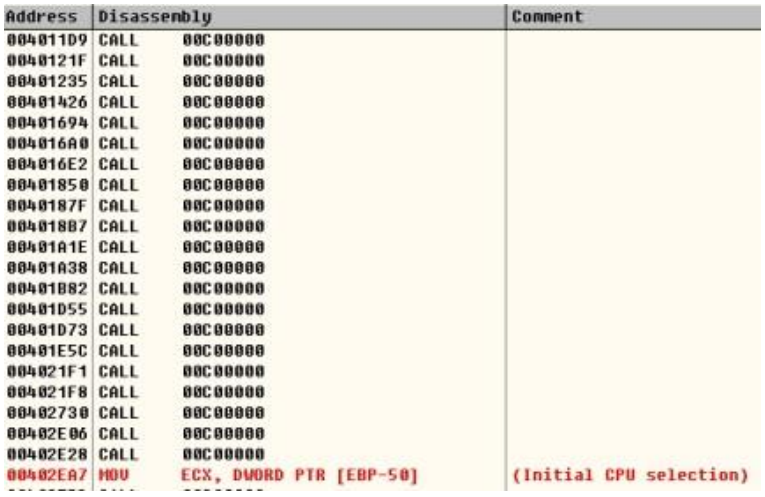
IAT까지 삽입한 DUMP 파일을 실행시켰을 때의 에러 창이다.

바로 이 부분에서 에러가 남을 확인 할 수 있다. 00402EBD E8 3ED17F00 CALL 00C00000 이 주소는 Asprotect 2.1x SKE가 IAT를 에뮬레이션 해주어 실행되던 HIMEM Call 입니다.

그럼 이 프로그램에서 몇 개의 HIMEM Call이 존재하는지 알아 봅시다.



오른쪽마우스 -> Search for -> All Commands 후 CALL 00C00000을 검색 해 봅니다.



Address	Disassembly		Comment
004011D9	CALL	00C00000	
0040121F	CALL	00C00000	
00401235	CALL	00C00000	
00401426	CALL	00C00000	
00401694	CALL	00C00000	
004016A0	CALL	00C00000	
004016E2	CALL	00C00000	
00401850	CALL	00C00000	
0040187F	CALL	00C00000	
00401887	CALL	00C00000	
00401A1E	CALL	00C00000	
00401A38	CALL	00C00000	
00401B82	CALL	00C00000	
00401D55	CALL	00C00000	
00401D73	CALL	00C00000	
00401E5C	CALL	00C00000	
004021F1	CALL	00C00000	
004021F8	CALL	00C00000	
00402730	CALL	00C00000	
00402E06	CALL	00C00000	
00402E28	CALL	00C00000	
00402EA7	MOV	ECX, DWORD PTR [EBP-50]	(Initial CPU selection)

꽤 많은 양의 HIMEM CALL들이 존재 함을 알 수 있습니다. 이렇게 수많은 HIMEM CALL들이 프로그램 내 AsProtect에서 IAT를 에뮬레이팅 해주는 것이다. 근데 AsProtect 패킹을 언패킹 했기 때문에

IAT를 에뮬레이팅 해줄 수 없어서 에러가 났었던 것 입니다.

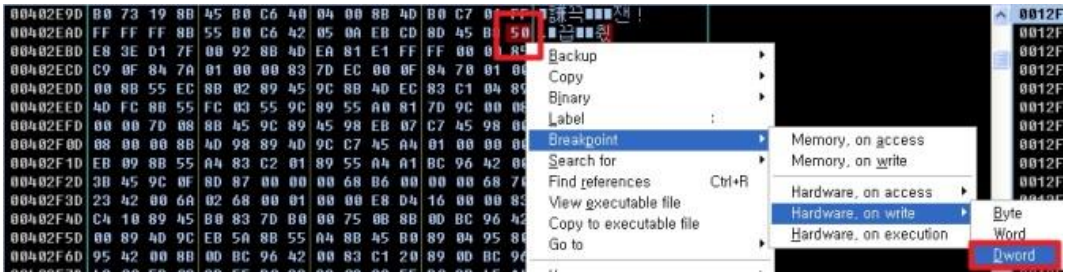
0x04. 에러 확인

우선적으로 확인 해야 할 사항은 바로, Dump로 뜬 파일의 00402EBD E8 3ED17F00 CALL 00C00000 이 부분에서 에러가 남을 확인 할 수 있습니다. 그렇다면 언패킹 되기 전에 저 주소에는 무엇이 있을까와 언패킹 되기 이전에 저곳에서 어떤 일이 벌어질 것 인가가 확인 해봐야 할 사항 인 것입니다. 무엇이 Wirte 되던지 Acess가 되던지 저곳을 중심으로 이뤄 질 것이기 때문입니다.

초보자의 입장으로 쓴 문서이기에 한번 생각 할 수 있도록 이야기를 길게 써두었습니다.
약 5분정도 어떻게 접근해야 할 것 인가 생각하신다면은 실력향상에 적지 않은 도움이 될 것 이라 예상 됩니다.

패킹된 파일을 열고 Shift + F9의 작업을 25번 해줘서 OEP를 찾아주고, 에러가 나는 지점인 00402EBD로 가보겠습니다.

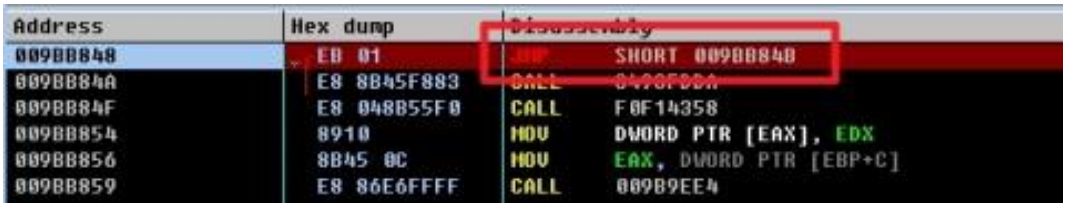
CALL 00C00000 주소값 옆에 E8 3ED17F00의 값이 보일 것이다. 바로 이 주소에 Hardware BP를 걸어 보도록 하겠습니다. BP를 거는 방법은 다음과 같습니다.



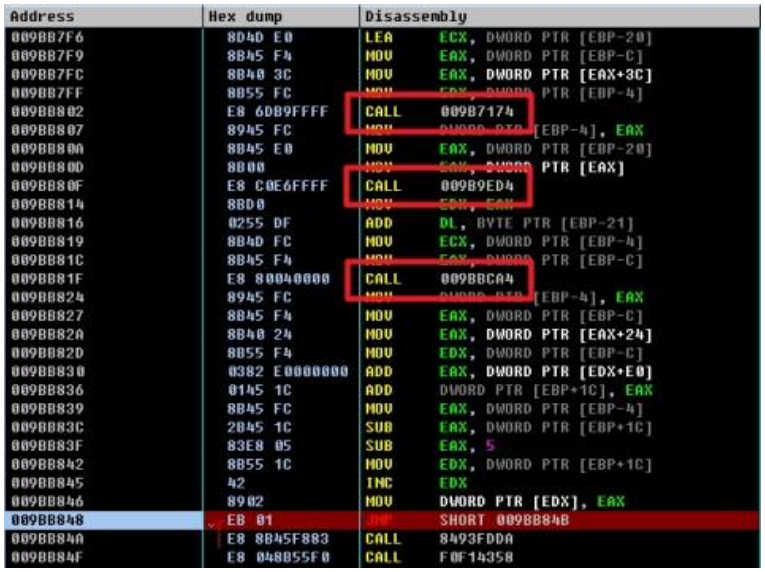
E8부터가 아닌 50부터 걸게 되는데, 이 이유는 cpu는 4바이트씩 값이 움직이기 때문입니다. 부득이하게 E8 3ED17F00의 값이 변경되는 것을 확인 하려면은 4바이트의 첫번째 부분에 BP를 주는 것 입니다.

즉 시나리오는 이렇습니다. AsProtect가 전부 진행되고, 마지막으로 프로그램의 IAT를 에뮬레이팅 해줄 것인데, 그 에뮬레이팅 해주는곳에 Wirte로 Hardware BP를 걸어 놓는다면은 분명 IAT가 에뮬레이팅 될때 멈추게 될 것 입니다. 확인 해 보도록 하겠습니다.

Cltr + F2로 프로그램을 다시 실행 시켜보도록 하겠습니다.



30번째의 Shift + F9의 진행으로 다음과 같은 주소에 BP가 걸림을 확인 할 수 있었습니다. 그리고 이 주소 바로 위로 가게 된다면은 다음과 같은 화면을 볼 수 있을 것입니다.



CALL문이 3개 있고, 아래에 JMP문이 있는 이 코드 부분은 AsProtect의 전형 적인 IAT 에뮬레이팅 부분이다. 버전마다 조금씩 다르겠지만 대부분 이런 형태의 코드를 따르고 있습니다.

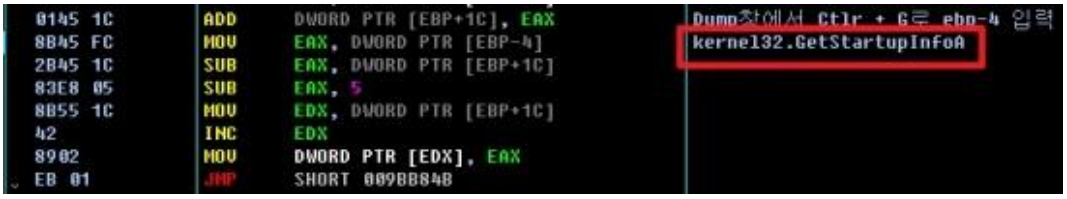
첫번째 CALL 부분의 바로 위 MOV EDX, DWORD PTR [EBP-4] 부분을 기억 합니다.

다음과 같이 Kernel32.GetStartupInfoA를 확인 할 수 있다. 바로 첫번째 IAT를 에뮬레이션 해 주는 부분인 것 입니다.

위의 명령이 실행되고 009BB848 JMP문으로 오게 된다면은 해당 HIMEM CALL은
00402EBD E8 3ED17F00 CALL 00C00000 이렇게 됩니다. 이렇게 되었을 때 Dump를 뜨게 되었을 때 에러가 나게 되는 것 입니다
덤프 파일에서 00402EBD E8 3ED17F00 Call kernel32.GetStartupInfoA 되어야지 에러가 나질 않습니다.

바로 이곳에서 AsProtect가 IAT를 에뮬레이션 해주는 부분이고 이 부분을 수정 한다면은 모든 HIMEM CALL에 함수를 직접 입력 해주고, 덤프를 뜰 수 있게 됩니다.

이 프로그램에서 처음 에뮬레이션 되는 IAT는 7C801EF2 (kernel32.GetStartupInfoA) 입니다.
이것을 바로 저 부분에 적용 시켜 보도록 하겠습니다.



MOV EAX, DWORD PTR [EBP-4] 에서 7C801EF2를 넣으니 제대로 잡히는 것을 볼 수 있습니다.

그리고 F9로 진행 했을 때 00C00000의 값이 제대로 IAT 잡히는것을 확인 할 수 있습니다.
이 모든 작업을 수동으로 할 수는 없을 것 입니다. 자동화를 시켜보도록 하겠습니다.
이제 수동으로 이렇게 잡은 것을 자동화 시키는 작업에 대해 알아 보도록 하겠습니다.

4. 자동화 기법

0x01. 빈 공간 찾기

F2로 재실행 후 다음 주소에 BP를 걸도록 합니다.

첫번째 CALL문 바로 위 MOV EAX, DWORD PTR [EAX+3C]에 BP를 걸고 빈 공간을 찾아보도록 하겠습니다. 빈 공간을 찾는 이유는 자동화 소스를 직접 '삽입'하기 위해서 입니다.
쉽게 예를 들어 새로운 소스를 직접 삽입 한 후 JMP(강제점프)로 그 소스를 실행 하는 것 입니다.

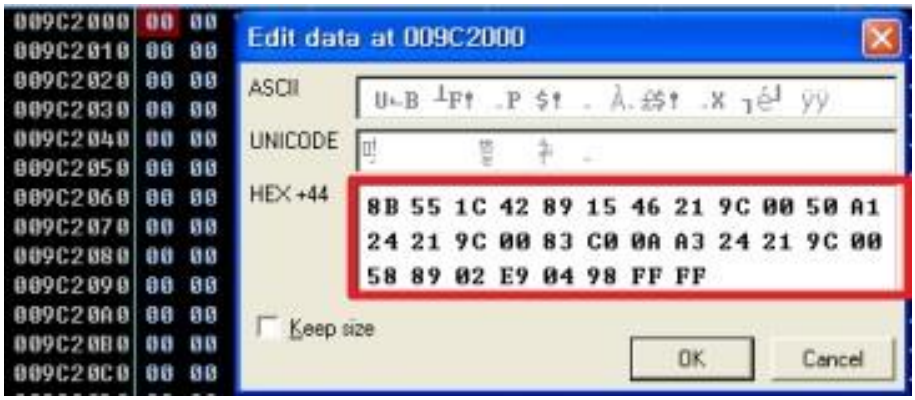
가독성을 높이기 위해 9c2000 주소로 소스를 삽입 하겠습니다.

0x02. 자동화 코드 설명

MOV EDX, DWORD PTR [EBP-4] 기억 나시나요? IAT가 저장되어있는 부분 입니다.
때문에 이곳에 빈 공간으로 JMP해주는 부분으로 변경 해줍니다.

HIMEM CALL을 해주는 부분 역시 강제로 JMP문을 변경 해 줍니다.

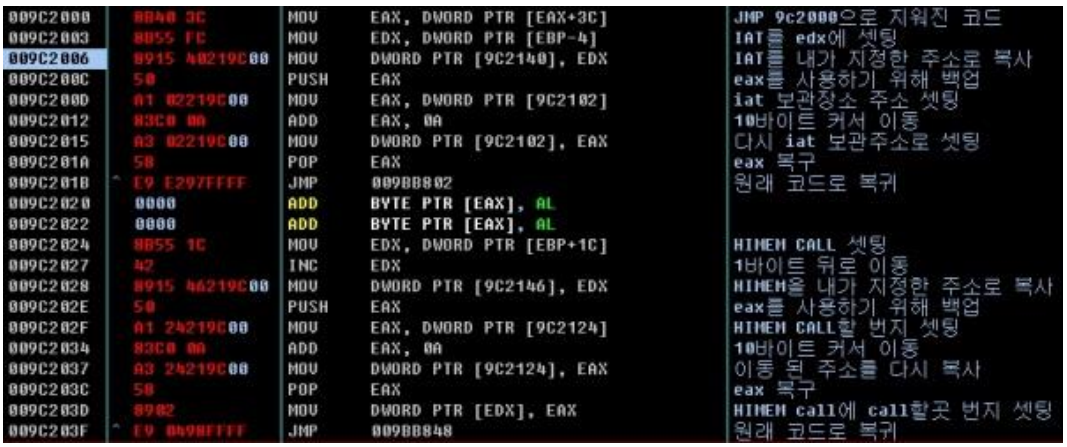
그 뒤에 9c2000에 코드를 삽입 해 줍니다.



덤프창에서 더블 클릭 후 오른쪽버튼으로 복사를 해주면 됩니다. 복사해줄 소스는 다음과 같습니다.

8B 40 3C 8B 55 FC 89 15 40 21 9C 00 50 A1 02 21 9C 00 83 C0 0A A3 02 219C 00 58 E9 E2 97 FF FF 00 00 00 00 8B 55 1C 42 89 15 46 21 9C 00 50 A1 24 21 9C 00 83 C0 0A A3 24 21 9C 00 58 89 02 E9 04 98 FF FF

*주의 이 명령어의 경우 작성자 본인의 환경에서만 적용 가능한 것 으로 이 문서를 보시는 분들은 약간의 수정을 하셔야 합니다



삽입된 코드 입니다. 옆에 주석으로 짧게 달았습니다.
EBP-4에 IAT 값이 저장되어있다는 것을 알고 있기에 IAT 값을 edx에 저장 후 그 값을 빈 공간인 9C2140에 저장 합니다. 그리고 원래 주소로 복귀하고, 마찬가지로 HIMEM Call 오프셋 주소도 빈 공간에 저장 합니다. 그 다음에 이뤄지는 작업들은 다음 IAT를 준비 하기 위한 코드 입니다.

0x03. IAT 주소

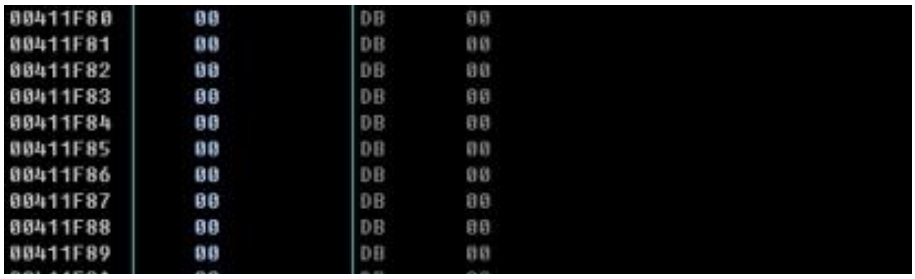
이렇게 보기 좋게 저장됨을 확인 할 수 있습니다.

```
F2 1E 80 7C 00 00 BE 2E 40 00 C9 2F 81 7C 00 00 B0 30 40 00 E1 0E 81 7C 00 00 C3 30 40 00 27 CD 80 7C 00 00 42 31 40
00 AD 2F 81 7C 00 00 51 18 40 00 A5 99 80 7C 00 00 96 6A 40 00 06 2F 81 7C 00 00 D8 6B 40 00 5F B5 80 7C 00 00 31 27
40 00 91 9E 80 7C 00 00 53 5B 40 00 09 9F 80 7C 00 00 6B 5B 40 00 C3 F8 85 7C 00 00 AB 5C 40 00 F2 1E 80 7C 00 00 80
18 40 00 31 B7 80 7C 00 00 B8 18 40 00 F6 E8 D0 77 00 00 DA 11 40 00 F6 E8 D0 77 00 00 20 12 40 00 39 7C D0 77 00 00
36 12 40 00
```

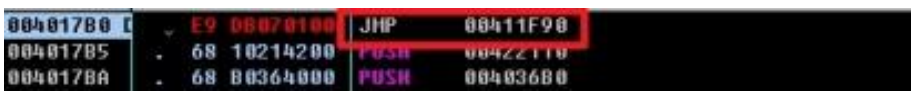
최종적으로 뽑아낸 IAT 주소 값들 입니다. 이 값을 가지고 Dump 한 파일에 적용 시키면 되는 거라고 여기까지 오신 분들은 충분히 예상 하실 수 있습니다.

0x04. Dump 파일에 적용

ASProtect 2.1x SKE가 모든 해당 HIMEM Call에 에뮬레이션 해주었던 것을
덤프 한 파일에 우리가 만든 코드를 이용 하여 자동으로 에뮬레이션 해주도록 하겠습니다.
올리로 덤프한 dump.exe를 불러 옵니다.



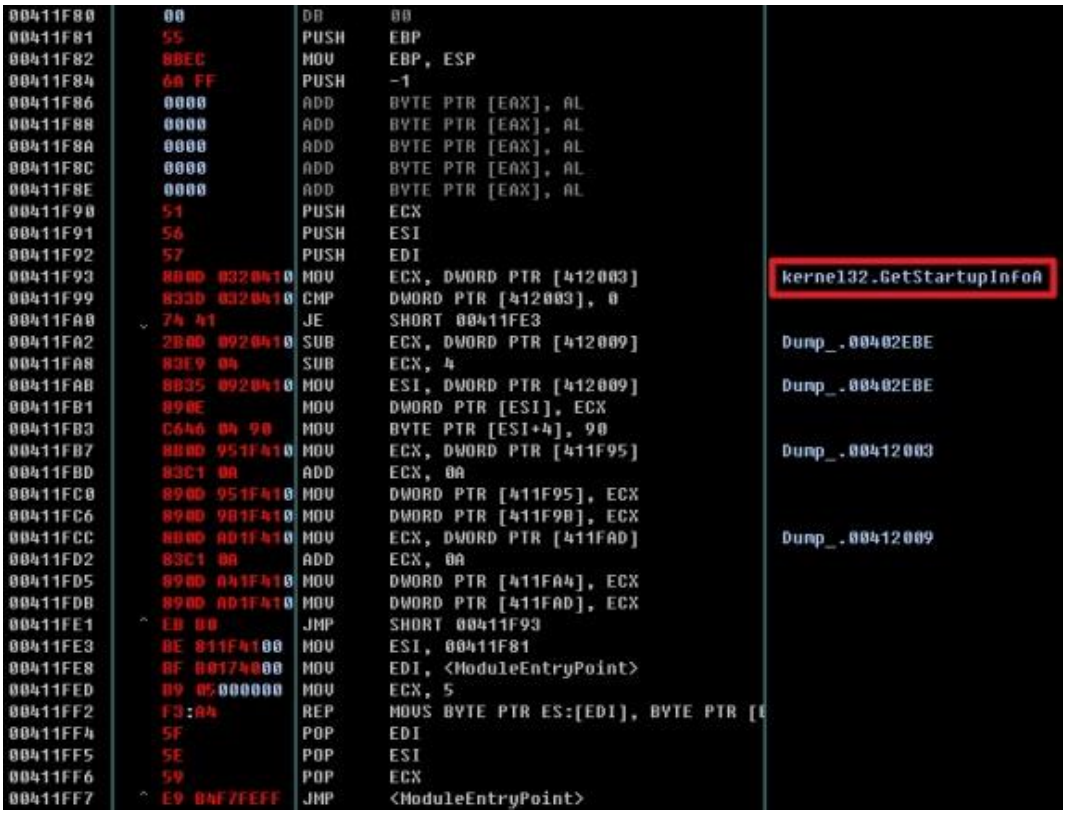
그리고 또 코드를 삽입해 줄 것이니 다음과 같은 빈 공간을 찾도록 합니다.



찾은 빈 공간으로 JMP문을 처음 EP 부분에 넣어 줍니다.

이렇게 해주면은 총 5바이트를 잡아먹어서 그 밑의 부분의 코드까지 지워 집니다.
다음은 원본코드 5바이트 어셈 코드 입니다.
004017B0 55 push ebp <=== 이곳을 jmp 00411F90 로 수정하면
004017B1 8BEC mov ebp, esp

004017B3 6A FF push -1 <===이곳까지 5바이트 잡아 먹습니다.
그렇기에 411F81 주소에 붙여 붙여 넣습니다.



00411F80 0055 8B ; 엔트리 포인터를 점프로 수정 하기 전 값들
00411F83 EC ; 엔트리 포인터를 점프로 수정 하기 전 값들
00411F84 6A FF ; 엔트리 포인터를 점프로 수정 하기 전 값들
00411F86 0000 add byte ptr ds:[eax], al
00411F88 0000 add byte ptr ds:[eax], al
00411F8A 0000 add byte ptr ds:[eax], al
00411F8C 0000 add byte ptr ds:[eax], al
00411F8E 0000 add byte ptr ds:[eax], al
00411F90 51 push ecx ; 레지스터를 사용하기 위해 원래 값 스택에 보관
00411F91 56 push esi ; 레지스터를 사용하기 위해 원래 값 스택에 보관
00411F92 57 push edi ; 레지스터를 사용하기 위해 원래 값 스택에 보관
00411F93 mov ecx, dword ptr ds:[412003] ; 에뮬레이션 해줄 IAT불러오기 시작
00411F99 cmp dword ptr ds:[412003], 0 ; 에뮬레이션 해줄 IAT가 더 없는가?
00411FA0 je short dump_.00411FE3 ; 끝이면 NP 복원 코드로 점프
00411FA2 sub ecx, dword ptr ds:[412009] ; 현재 IAT값을 오프셋 주소 값을 얻기 위해 변환
00411FA8 sub ecx, 4 ; IAT호출할 주소를 정확히 얻기 위해 한번 더 빼줌
00411FAB mov esi, dword ptr ds:[412009] ; HIMEM Call 오프셋 주소를 ESI로 이동

00411FB1 mov dword ptr ds:[esi], ecx ; HIMEM Call에 IAT 복사
00411FB3 mov byte ptr ds:[esi+4], 90 ; HIMEM Call 오프셋 주소르부터 5번째 바이트 NOP으로
00411FB7 mov ecx, dword ptr ds:[411F95] ; 411F95 주소 4바이트 값을 ECX 에 넣고
00411FBD add ecx, 0A ; 다음 IAT를 복사하기위해 10바이트 뒤로 이동
00411FC0 mov dword ptr ds:[411F95], ecx ; 10바이트 이동된값을 411F95로 복사
00411FC6 mov dword ptr ds:[411F9B], ecx ; 10바이트 이동된값을 비교하는 루틴 411F9B로도 복사
00411FCC mov ecx, dword ptr ds:[411FAD] ; 411FAD가 가르키는 HIMEM Call 오프셋 주소를 담고
00411FD2 add ecx, 0A ; 다음 HIMEM Call 오프셋 주소 얻기위해 10바이트 뒤로 이동
00411FD5 mov dword ptr ds:[411FA4], ecx ; 그런후 411FA4 주소에 저장
00411FDB mov dword ptr ds:[411FAD], ecx ; 411FAD 주소에도 복사
00411FE1 jmp short dump_.00411F93 ; 찾아낸 모든 IAT 에뮬레이션 끝날때까지 계속루프
00411FE3 mov esi, dump_.00411F81 ; NP 의 원레값을 ESI에 넣고
00411FE8 mov edi, dump_.<ModuleEntryPoint> ; NP 주소를 EDI에 담은후
00411FED mov ecx, 5 ; ESI가 가르키는 주소부터 5바이트 까지만
00411FF2 rep movs byte ptr es:[edi], byte ptr ds:[esi] ; 복원을 시작한다.
00411FF4 pop edi ; 레지스터 사용을 다 햇으므로 원래값 복원
00411FF5 pop esi ; 레지스터 사용을 다 햇으므로 원래값 복원
00411FF6 pop ecx ; 레지스터 사용을 다 햇으므로 원래값 복원
00411FF7 jmp dump_.<ModuleEntryPoint> ; NP로 점프하면 모든 작업 끝

다음은 우리가 만든 전체 코드를 헥사로 복사 한 것입니다.
이것을 전부 411F81 번지부터 복사 해 넣으시면 됩니다.

55 8B EC 6A FF 00 00 00 00 00 00 00 00 00 00 51 56 57 8B 0D 03 20 41 00 83 3D 03 20 41 00 00
74 41 2B 0D 09 20 41 00 83 E9 04 8B 35 09 20 41 00 89 0E C6 46 04 90 8B 0D 95 1F 41 00 83 C1 0A
89 0D 95 1F 41 00 89 0D 9B 1F 41 00 8B 0D AD 1F 41 00 83 C1 0A 89 0D A4 1F 41 00 89 0D AD 1F
41 00 EB B0 BE 81 1F 41 00 BF B0 17 40 00 B9 05 00 00 00 F3 A4 5F 5E 59 E9 B4 F7 FE FF

마지막으로 최종적으로 412003주소에 우리가 얻은 IAT와 HIMEM Call값들을 넣어주면은 정상적으로 코드가 작동하게 되어서, 정상실행이 됩니다.

전 잘 실행되네요. 여기까지 AsProtect 2.1 SKE 언패킹을 마치도록 하겠습니다.

5. 마무리

0x01. 끝으로..

Asprotect 2.1을 언팩하면서 많은 외국 문서들과, 외국 자동스크립트를 접했었는데 제대로 설명이 안되어 있다가 Cool soft에서 어느 분께서 올려주신 문서로 겨우 언팩 할 수 있었습니다.
이 자리를 빌어 그분께 감사 드린다고 말씀 드리고 싶습니다. 끝으로 서두에 밝혔지만 이 문서의 저작권(?)은 그분께 있음을 다시 한번 말씀 드립니다. 고작 전 문서를 보기 좋게 한 것 뿐이 없습니다.
고마운 분의 닉네임도 몰라 죄송합니다. ^^a 하하 연락 주시면 제가 고기 사드리겠습니다!!

문서를 끝까지 읽어주셔서 감사합니다~!