# 악성코드 분석 보고서

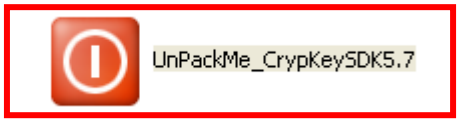
(sand-reversingwithlena-tutorials)

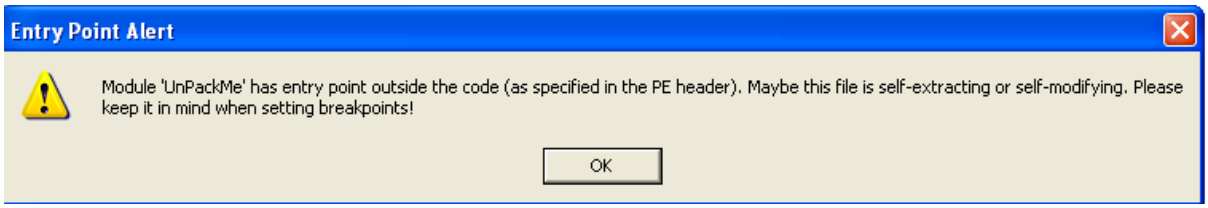
2025.08.14

20

# 1. 문제

1-1) UnPackMe_CrpKeySDK5.7



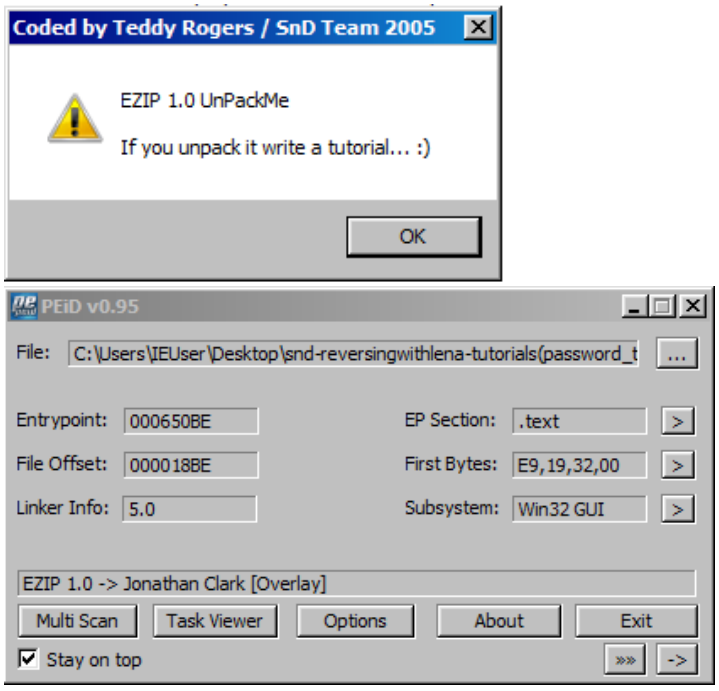




해당 파일 올리디버그로 실행 시 경고 창이 나오는 걸 볼 수 있다.

해석 : 모듈 'UnPackMe'의 엔트리 포인트가 PE 헤더에 지정된 코드 섹션(.text) 밖에 있습니다. 이 파일은 자기 추출(Self-extracting) 또는 자기 수정(Self-modifying)하는 프로그램일 수 있습니다. 브레이크포인트를 걸 때 유의하세요!

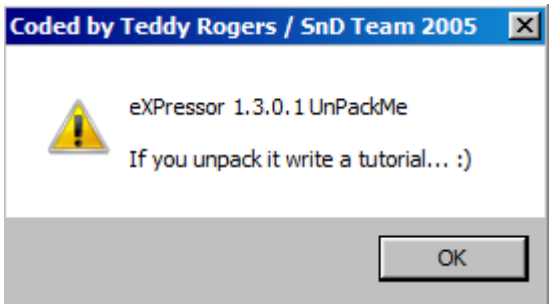
-> Entry Point(EP)**는 .text 섹션 안에 있음. 근데 이런 경고창이 발생했다는 거는 스텁 코드일 가능성이 높음.

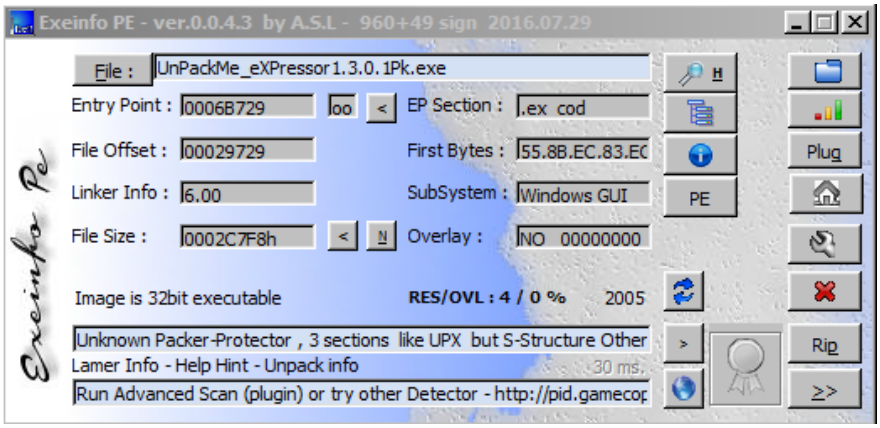
1-2) UnPackMe_EZIP1.0




압축된 걸 알 수 있고 경고 창이 안나오게 만들어야한다.

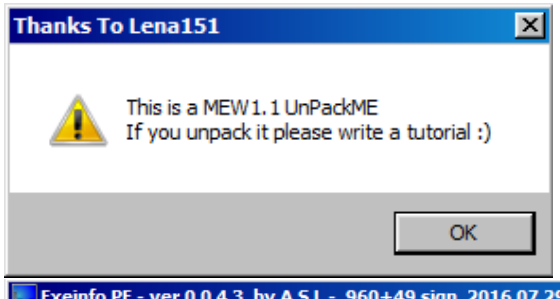
1-3) UnPackMe_eXPressor1.3.0.1PK.exe

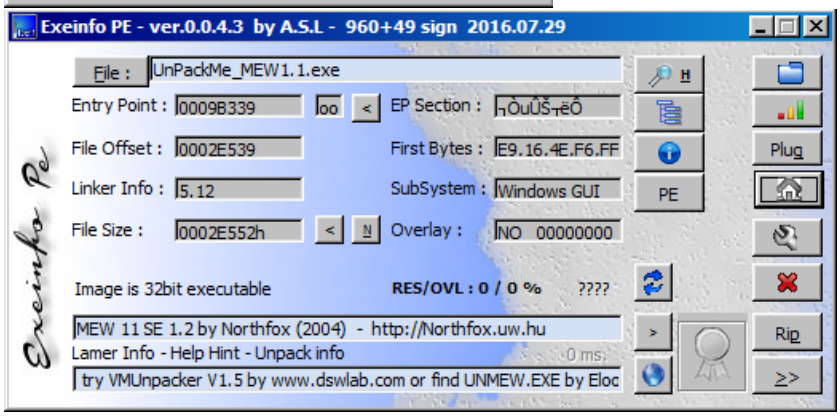



압축된 걸 알 수 있고 경고 창이 안나오게 만들어야한다.

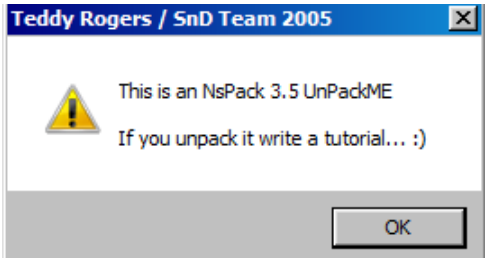
1-4) UnPackMe_MEW1.1.exe

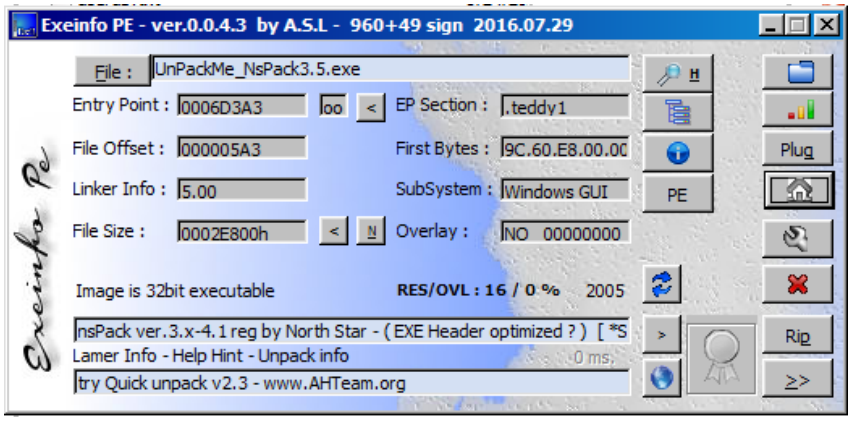



압축된 걸 알 수 있고 경고 창이 안나오게 만들어야한다.

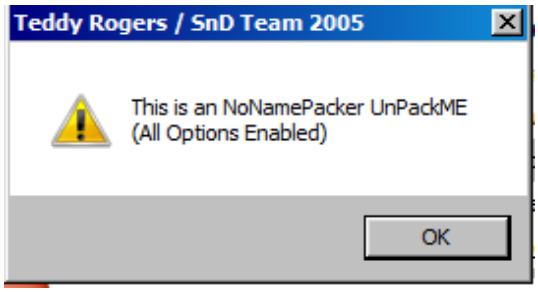
1-5) UnPackMe_NsPack3.5.exe

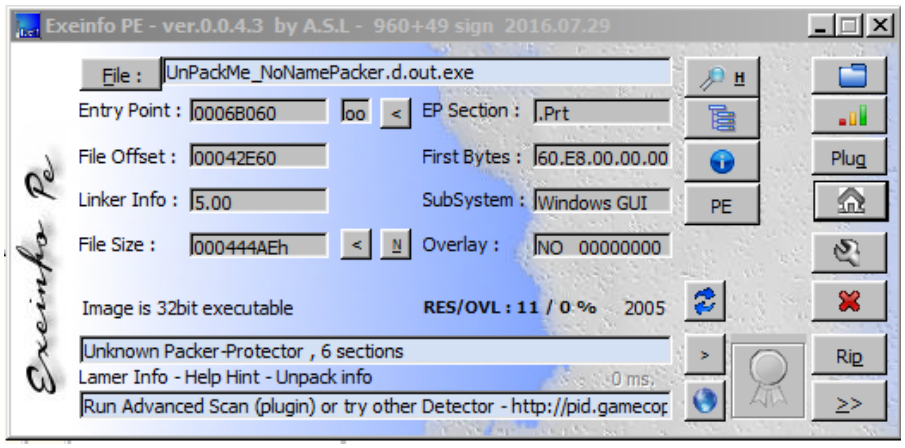



압축된 걸 알 수 있고 경고 창이 안나오게 만들어야한다.

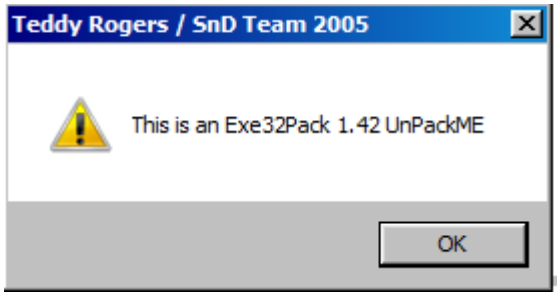
1-6) UnPackMe_NoNamePacker.d.out.exe

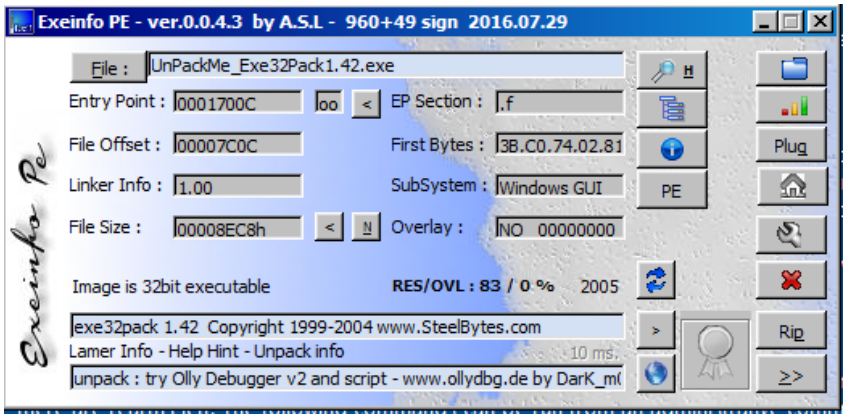



압축되어 있다는 걸 확인할 수 있는데 무슨 압축 프로그램인지 알 수 없다.

1-7) UnPackMe_Exe32Pack1.42.exe

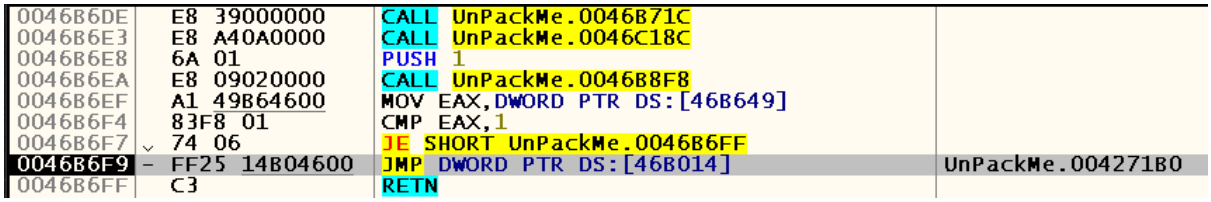



압축된 걸 알 수 있고 경고 창이 안나오게 만들어야한다.

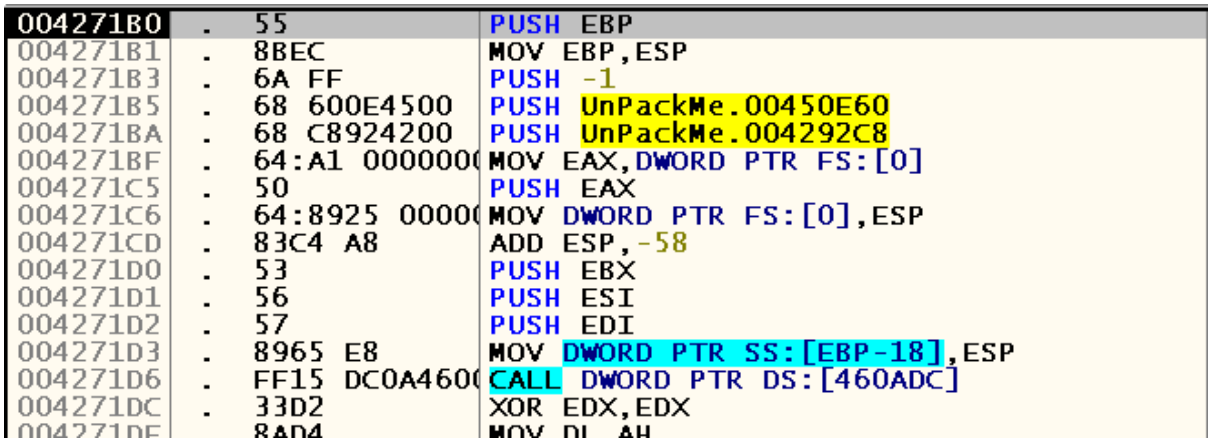
# 2. 해결 방법

2-1) UnPackMe_CrpKeySDK5.7




'OK' 버튼을 누르고 'F8'을 눌러주면 0x0046B6F9에서 0x00427180으로 넘어가는 걸 볼 수 있다.


이렇게 나오는데 'ctrl+a'를 눌러주면 실행 가능 코드 분석로 보여진다.




이런식으로 바뀌는 걸 볼 수 있다.

1. OEP를 알 수 있는 첫 번째 방법

| Address | Size | Owner | Section | Contains | Type | Access | Initial |
|---|---|---|---|---|---|---|---|
| 00010000 | 00001000 | | | | Priv | RW | RW |
| 00020000 | 00001000 | | | | Priv | RW | RW |
| 0012B000 | 00001000 | | | | Priv | RW Guar | RW |
| 0012C000 | 00004000 | | | stack of ma | Priv | RW Guar | RW |
| 00130000 | 00003000 | | | | Map | R | R |
| 00140000 | 00028000 | | | | Priv | RW | RW |
| 00240000 | 00006000 | | | | Priv | RW | RW |
| 00250000 | 00003000 | | | | Map | RW | RW |
| 00260000 | 00016000 | | | | Map | R | R |
| 00280000 | 00041000 | | | | Map | R | R |
| 002D0000 | 00041000 | | | | Map | R | R |
| 00320000 | 00006000 | | | | Map | R | R |
| 00330000 | 00003000 | | | | Map | R E | R E |
| 003F0000 | 00002000 | | | | Map | R E | R E |
| 00400000 | 00001000 | UnPackMe | | PE header | Imag | R | RWE |
| 00401000 | 0004A000 | UnPackMe | .text | code | Imag | R | RWE |
| 0044B000 | 0000C000 | UnPackMe | .rdata | | Imag | R | RWE |
| 00457000 | 00009000 | UnPackMe | .data | data | Imag | R | RWE |
| 00460000 | 00003000 | UnPackMe | .idata | imports | Imag | R | RWE |
| 00463000 | 00008000 | UnPackMe | .rsrc | resources | Imag | R | RWE |
| 0046B000 | 00002000 | UnPackMe | Have | SFX | Imag | R | RWE |
| 0046D000 | 00001000 | UnPackMe | a nice | relocations | Imag | R | RWE |
| 0046E000 | 00001000 | UnPackMe | day! | | Imag | R | RWE |

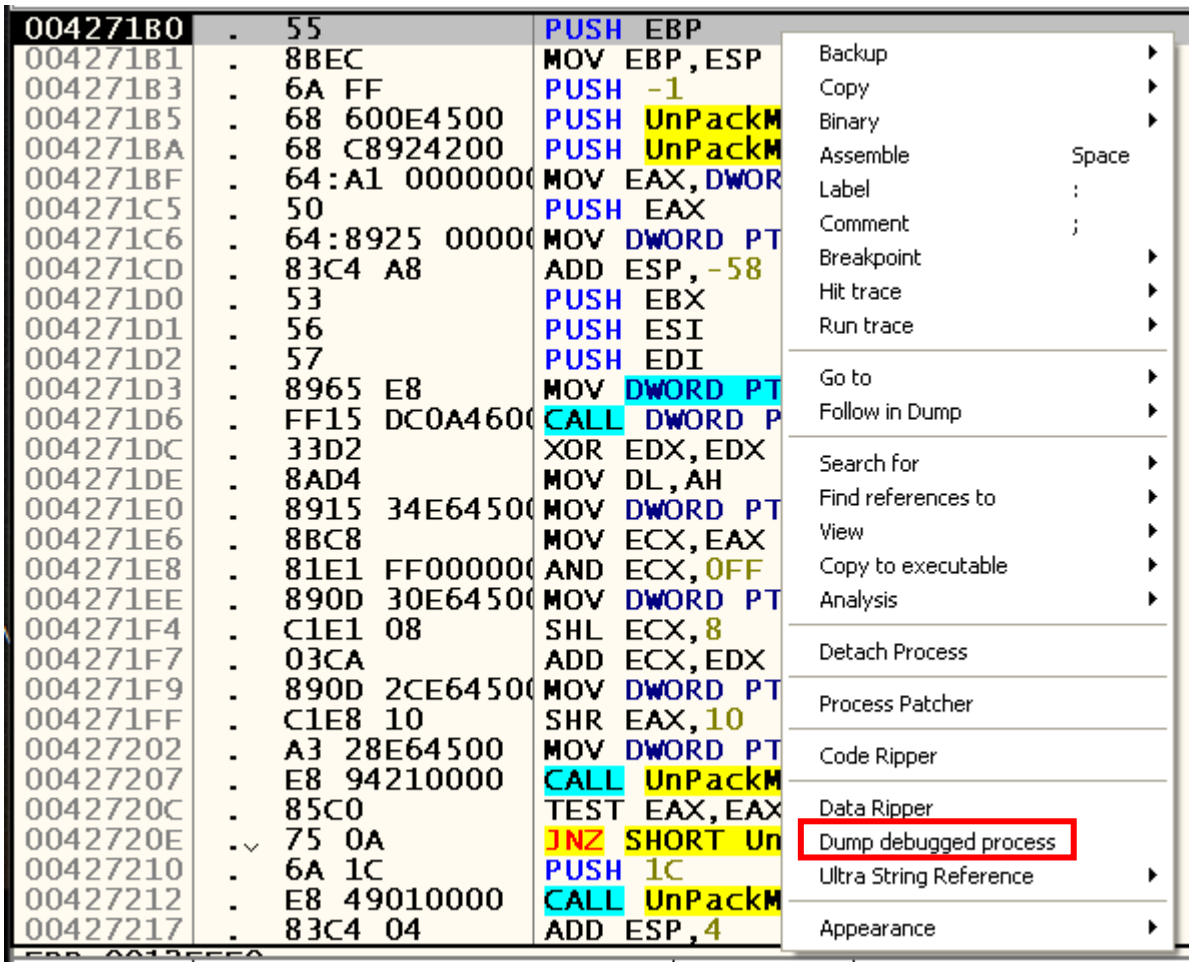
Memory Dump에서 보면 OEP는 .text에 있는 걸 알 수 있는데 현재 코드 위치가 0x00427180이므로 해당 주소가 OEP인 걸 알 수 있다.

2. OEP를 알 수 있는 두 번째 방법

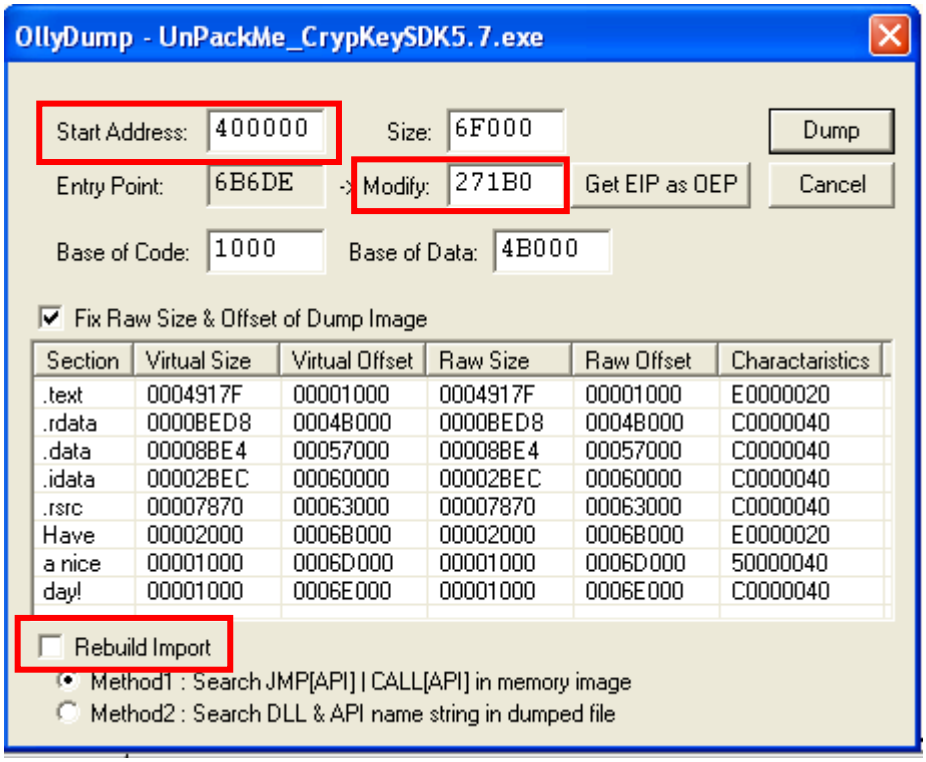
| 컴파일러/툴체인 | 전형적 OEP/프롤로그 스니펫 | 눈에 띄는 특징/힌트 |
|---|---|---|
| **MSVC (Visual C++)** | push ebp → mov ebp, esp → sub esp, imm → (필요 시) push ebx/esi/edi | 매우 "정석" 함수 프롤로그. CRT 진입점(WinMainCRTStartup 등)에서는 곧바로 GetCommandLineA/W, GetStartupInfoA/W, GetModuleHandleA 같은 **커널32/유저32 호출 패턴**이 이어짐 |
| **MinGW/GCC (Windows)** | push ebp → mov ebp, esp → sub esp, imm → (PIC일 때) call __x86.get_pc_thunk.bx → add ebx, imm | **PIC 레지스터 설정**( get_pc_thunk )과 함께 **__main 또는 ___main 호출**이 자주 보임. 초기화 루틴 후 main/WinMain 진입 |
| **Clang/LLVM (MSVC CRT 링크)** | MSVC와 매우 유사: push ebp → mov ebp, esp → sub esp, imm | 링커·CRT가 MSVC면 **패턴이 사실상 MSVC와 동일**. 코드 생성 스타일만 약간 상이 |
| **Borland/Embarcadero C++** | push ebp → mov ebp, esp → sub esp, imm → push ebx/esi/edi… | **RTL 초기화 호출**(예: @StartExe 류), **SEH 비슷한 래더**가 비교적 초반에 보이는 편 |
| **Delphi/Object Pascal** | push ebp → mov ebp, esp → xor eax, | **xor eax, eax가 초반에 눈에 띔**, 그리고 |

| | | |
|---|---|---|
| | eax → push ebx push esi push edi → (곧바로) @InitExe/@GetMem 등 **RTL 심볼 호출** | **@로 시작하는 런타임 루틴**( @InitTables, @ClassCreate 등 ) 호출이 연달아 나옴 |
| **Intel C/C++ (ICC, MS CRT)** | MSVC와 거의 동일 | 최적화가 강해 프롤로그가 **짧거나 재배치** 될 수 있음 |
| **LCC/Watcom 등 레거시** | push ebp → mov ebp, esp 중심의 보수적 프롤로그 | **레거시 CRT 호출**(고유한 초기화 심볼)이 조기 등장 |

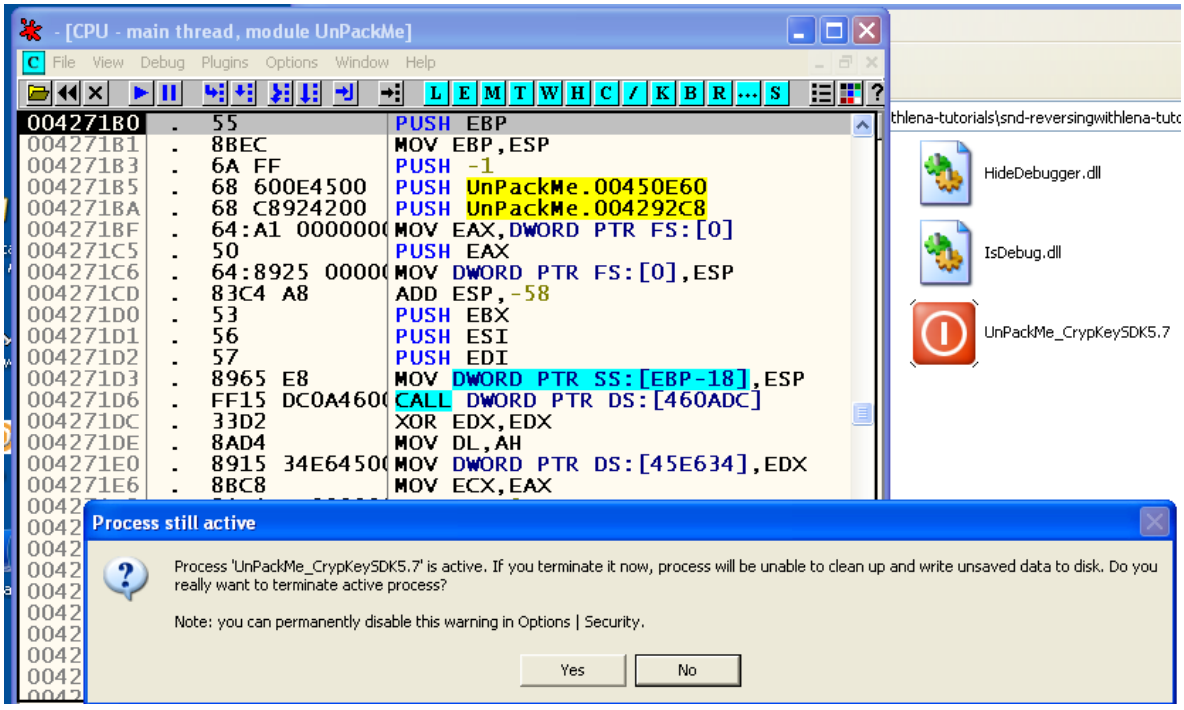
MSVC CRT는 프로그램 시작 시 거의 항상 SEH 프레임을 등록하므로 MSVC계열인 걸 알 수 있고 추가적으로 이 부분이 OEP인 걸 알 수 있다.




0x00427180 해당 부분에서 덤프를 해준다.




Start Address와 Modify가 0x00427180로 설정되었는지 확인 하고 Rebuild Import 체크를 해제해준다,

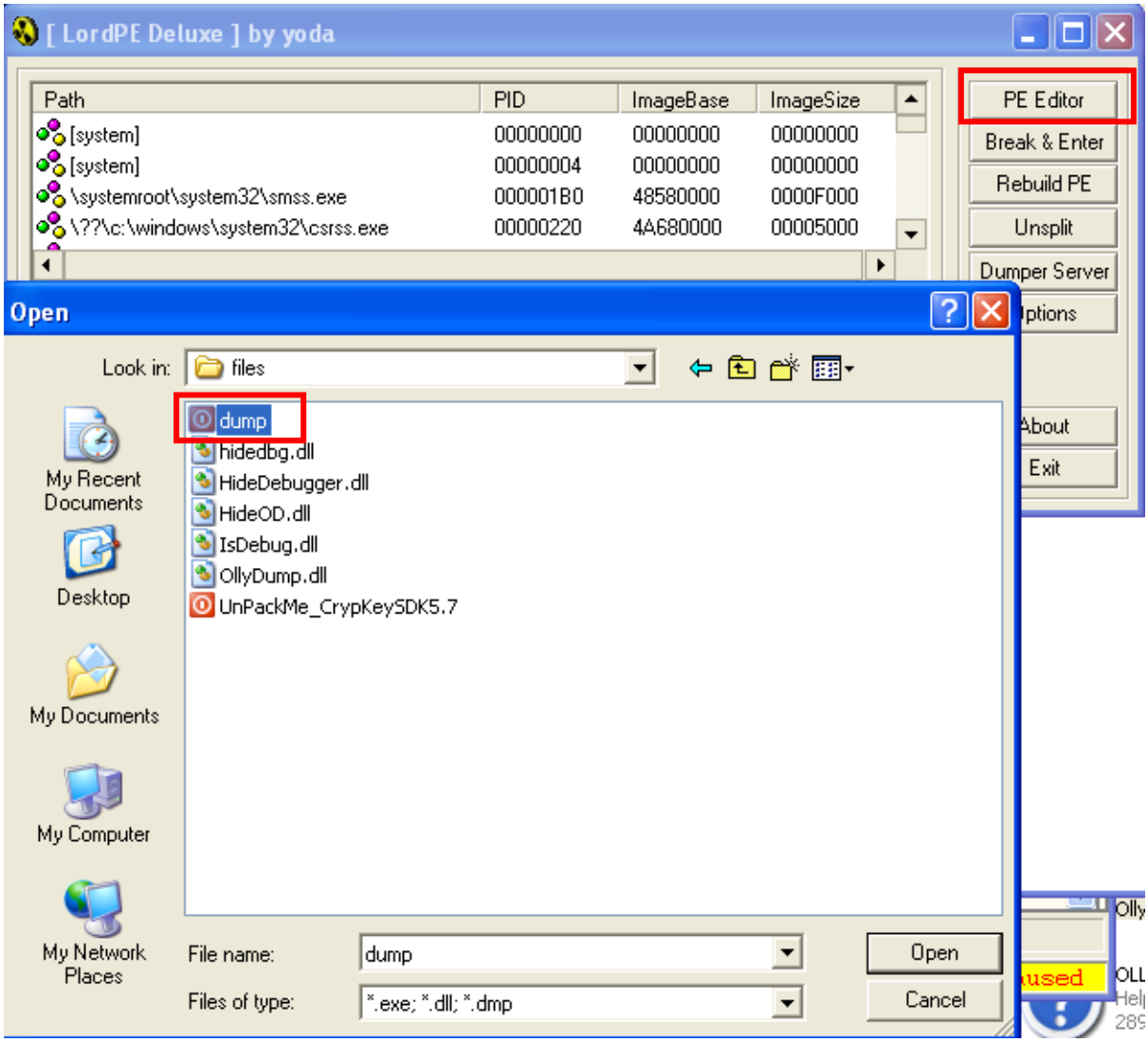

이름을 dump.exe로 저장 후 실행해주면 아까 뜨는 창이 안뜨는 걸 볼 수 있다.

문제 해결!

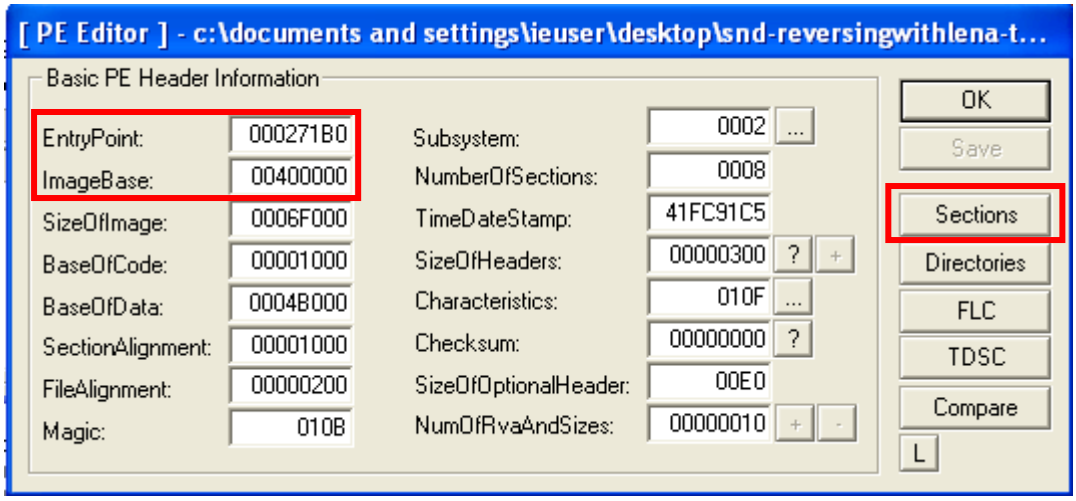
추가적으로 Lena는 LordPE라는 프로그램을 사용했는데 올리디버그 덤프 플러그로 수정이 되었기 때문에 해당 프로그램은 안사용해도 된다.

하지만 해당 프로그램 사용방법을 소개해보자면

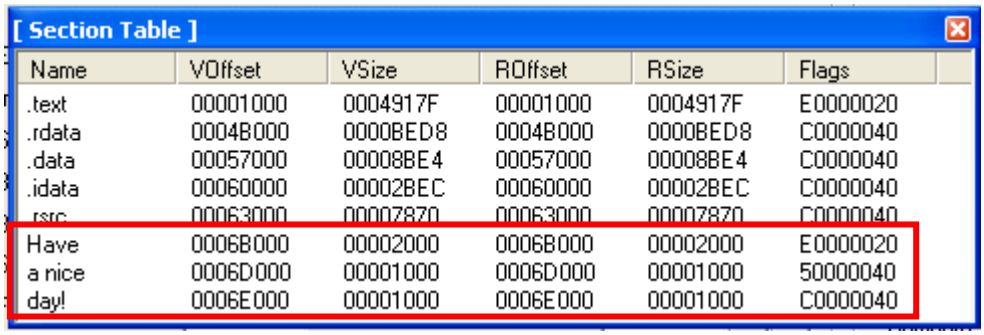
LordPE : PE라고 하는 윈도우 하에 실행되는 모든 실행파일의 헤더(프로그램이 실행될때 전체 프로그램 구조의 미니맵과 같은 역할을 하는 것을 PE라고 한다.) 를 읽어들여서 분석하고, 손상된 PE가 발견되었을 경우 Rebuild해주는 막강한 기능을 가지고있는 프로그램이다.




PEediter을 눌러서 dump파일을 열어준다.




EntryPoint와 ImageBase가 제대로 설정되었나 확인 후 Sections에 들어간다.




해당 부분은 원래 PE파일에 없는 부분이다. 삭제(wipe section header) 후 저장(Rebuild)을 해준다.

```
003F0000 00002000                          Map  R E    R E
00400000 00001000 dump                     Imag R      RWE
00401000 0004A000 dump     .text    code   Imag R      RWE
0044B000 0000C000 dump     .rdata         Imag R      RWE
00457000 00009000 dump     .data    data  Imag R      RWE
00460000 00003000 dump     .idata   imports Imag R      RWE
00463000 00008000 dump     .rsrc    resources Imag R    RWE
00470000 00103000                          Map  R      R
00580000 00001000                          Priv RW     RW
00590000 00066000                          Map  R E    R E
00890000 00001000                          Priv RW     RW
```

올리디버그에서 실행하면 없어진 걸 볼 수 있다.

2-2) UnPackMe_EZIP1.0 (ESP Trick)

- 스택에 대한 간단한 설명

- EBP 레지스터 : 현재 스택 프레임의 베이스 주소(최하단 주소)를 저장하고 있는 레지스터
- ESP 레지스터 : 현재 스택의 최상단 주소를 저장하고 있는 레지스터
- 스택 프레임: 함수가 사용하는 독립적인 메모리 영역 >> 함수가 실행 중일 때 존재하고 실행이 끝나면 사라짐

**메모리 상에서의 스택은 높은 주소에서 낮은 주소로 자라기 때문에**, 낮은 주소일수록 최상단의 주소이다.
따라서 EBP 레지스터는 항상 현재 스택에서의 가장 높은 주소 값을 가지고 있으며,
ESP 레지스터는 가장 낮은 주소 값을 가지고 있다.

```
PUSH EBP
MOV EBP, ESP
// 원래 루틴의 ebp값을 스택에 push = esp가 가리키는 주소에 ebp 값 저장
.....
// 서브루틴
// 서브루틴 중 ebp에는 이전 루틴의 ebp를 가리키는 스택의 주소 저장
.....
MOV ESP, EBP
POP EBP
// esp에 들어간 데이터(이전 루틴의 프레임 포인)를 ebp로 옮김
JMP EAX
```

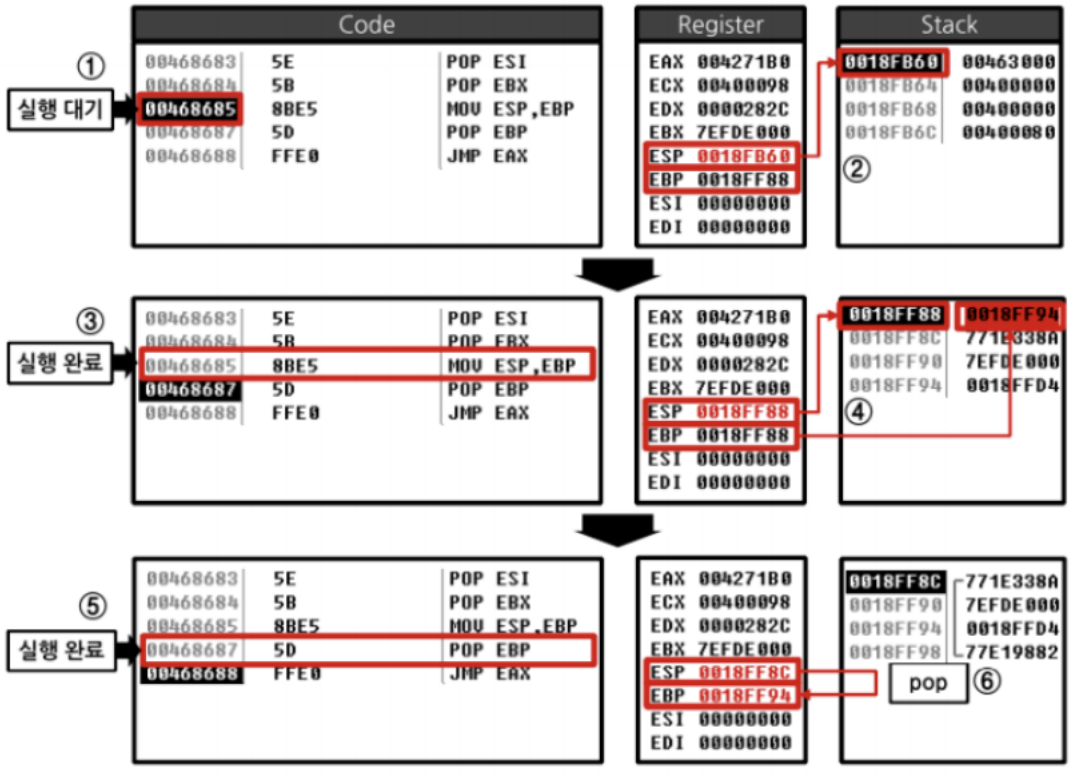
JMP EAX or RET 이렇게 쓰임

서브루틴 시작/종료할 때의 레지스터 변화

서브루틴 시작


서브루틴 시작

서브루틴 종료




서브루틴 종료

※ ESP Trick 이란?

정상 RET 대신 ESP를 직접 조작해서 흐름을 바꿔버리는 기법

```asm
add esp, 0x20   ; ESP를 강제로 이동 (리턴 주소 무시)
jmp esp         ; ESP가 가리키는 메모리로 점프 (스택 실행)
```

**핵심**

- RET를 쓰면 스택의 [리턴 주소]로 돌아가야 하는데,
  jmp esp로 강제 점프 → 디버거의 콜스택 추적이 무너짐.

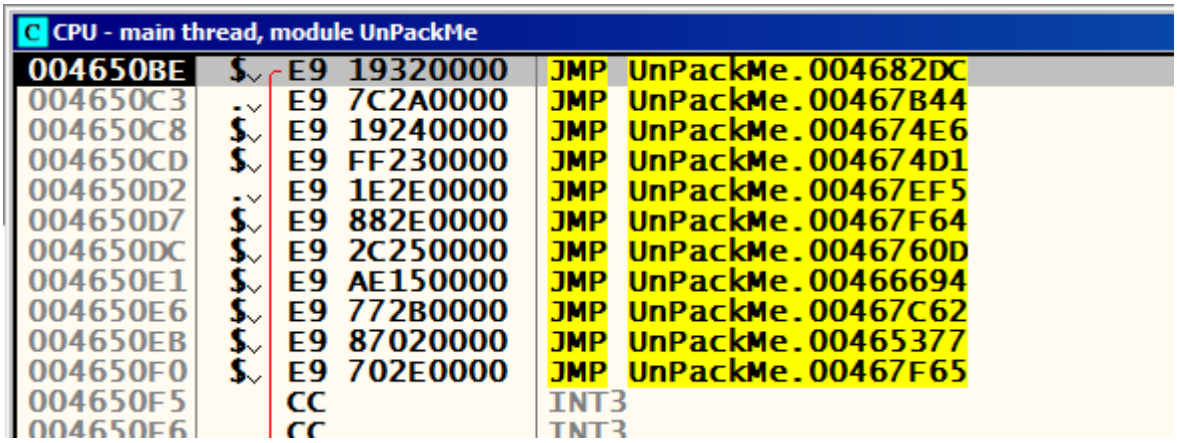
ㄴ jum esp 는 무조건 있어야함 (or CALL ESP가 있음)

- 때때로 MOV ESP, [메모리]로 **스택 피벗(stack pivot)**도 함.

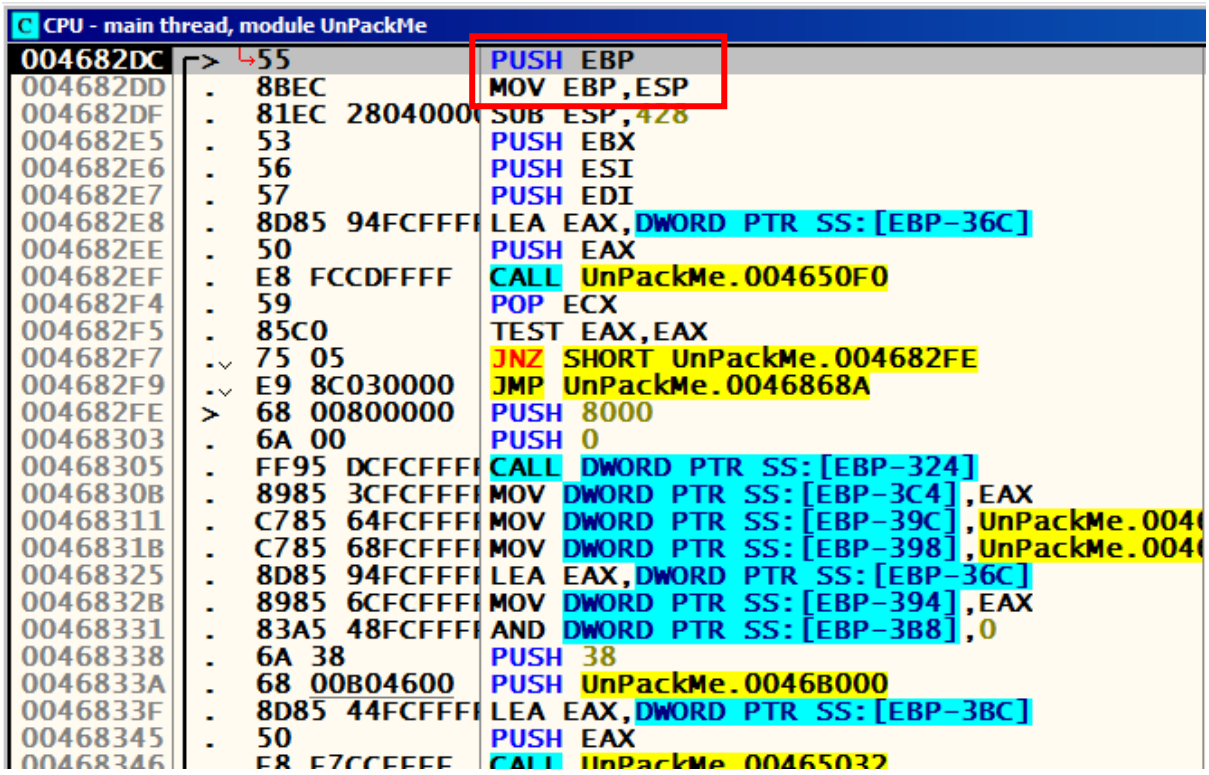
```
PUSH EBP
MOV  EBP, ESP
// 원래 루틴의 ebp값을 스택에 저장 (push = esp가 가리키는 주소에 ebp 값 저장)
// 현재 함수의 프레임 기준점(EBP)을 ESP로 설정
......
/* 서브루틴
   (언팩/복호화 루틴 등 필요한 작업 수행)
   필요하다면 여기서 새 스택/쉘코드 시작 주소를 미리 EDI 등에 준비해 둠
*/
......

; === 아래(에필로그 자리)를 ESP Trick으로 교체 ===
; EDI = 언팩된 코드(또는 실행할 버퍼)의 시작 주소라고 가정
MOV  ESP, EDI         ; 스택 피벗: ESP를 새 실행 버퍼로 이동
ADD  ESP, 20h         ; (선택) 시작 지점 미세 조정
JMP  ESP              ; ★ 정상 RET 대신, 스택이 가리키는 코드로 즉시 점프
```

이런 식으로 있을 때 빨간 칸에 코드가 도착했을 때 ESP에 있는 값에 브레이크를 거는 이유는 ESP에 이전 EBP 값이 들어가는 데 모든 함수는 끝날 때 이전 EBP로 무조건 돌아가기 때문에 ESP에 있는 값에 브레이크를 건다.

* 참고 : https://doongdangdoongdangdong.tistory.com/201

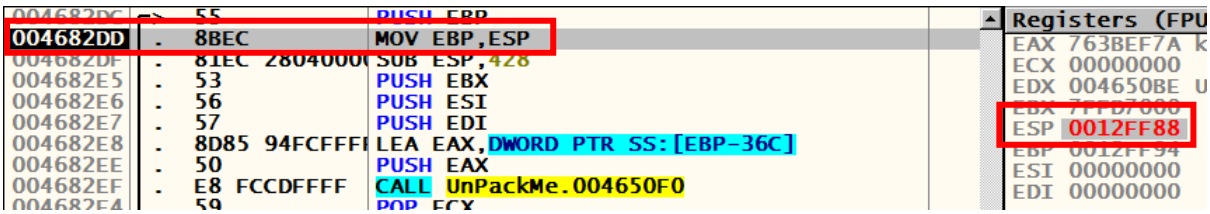


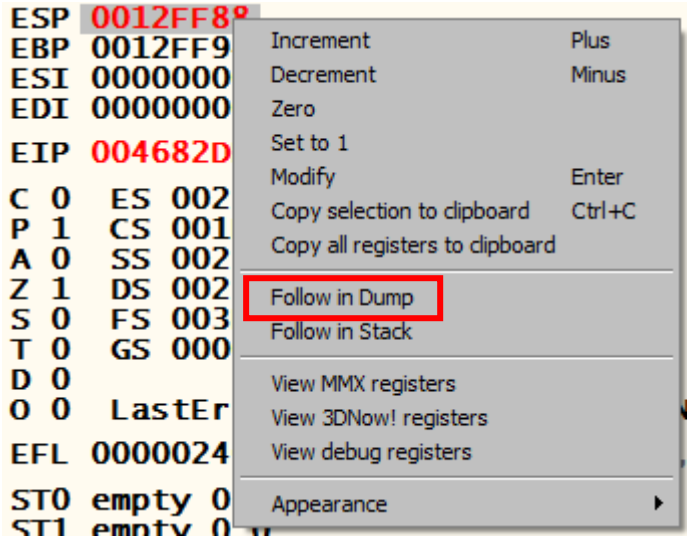

JMP 코드가 많은데 'F8'을 이용해서 실행해본다.



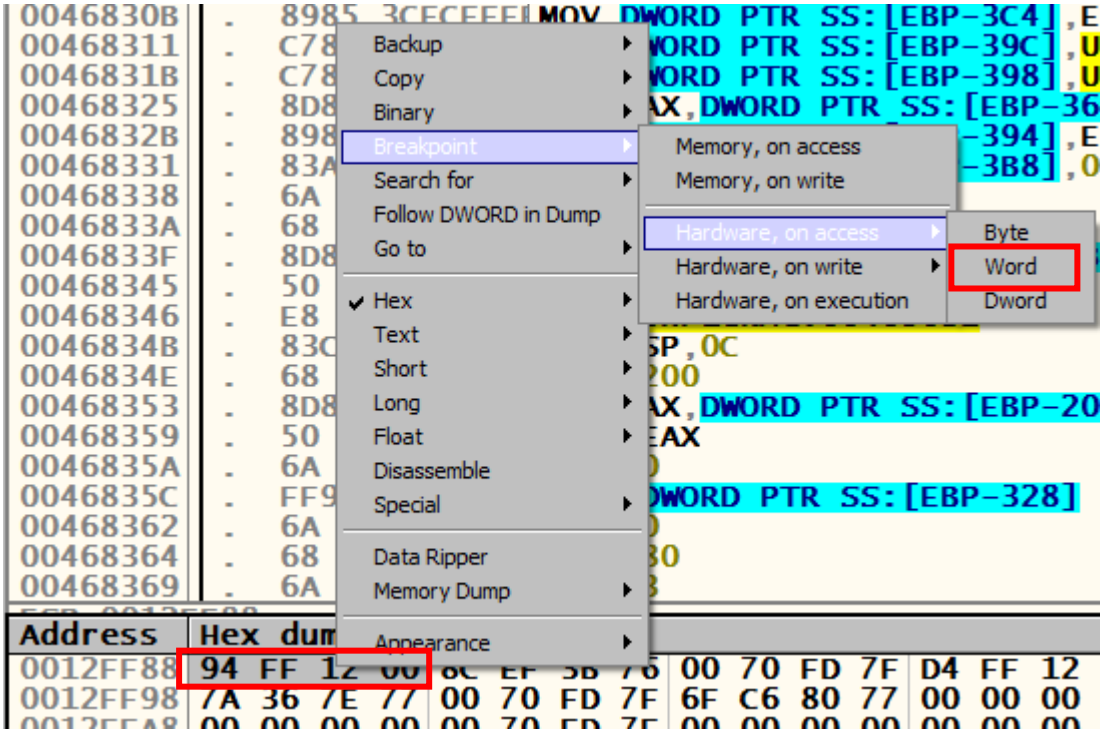

해당 코드가 나오는데 서브루틴이 시작되는 부분으로 볼 수 있다. 원래의 OEP를 찾기 위해서는 ESP는 서브루틴이 시작할 때 원래 PUSH EBP를 하면서 원래의 EBP의 값을 처음에 저장하고 그때 ESP는 원래의 EBP를 저장한 것을 가르키고 있다.

근데 서브루틴이 끝나고 EBP가 원래로 돌아와야하기 때문에 처음 ESP가 저장하고 있는 EBP의 값에 BP를 걸어줘야한다.



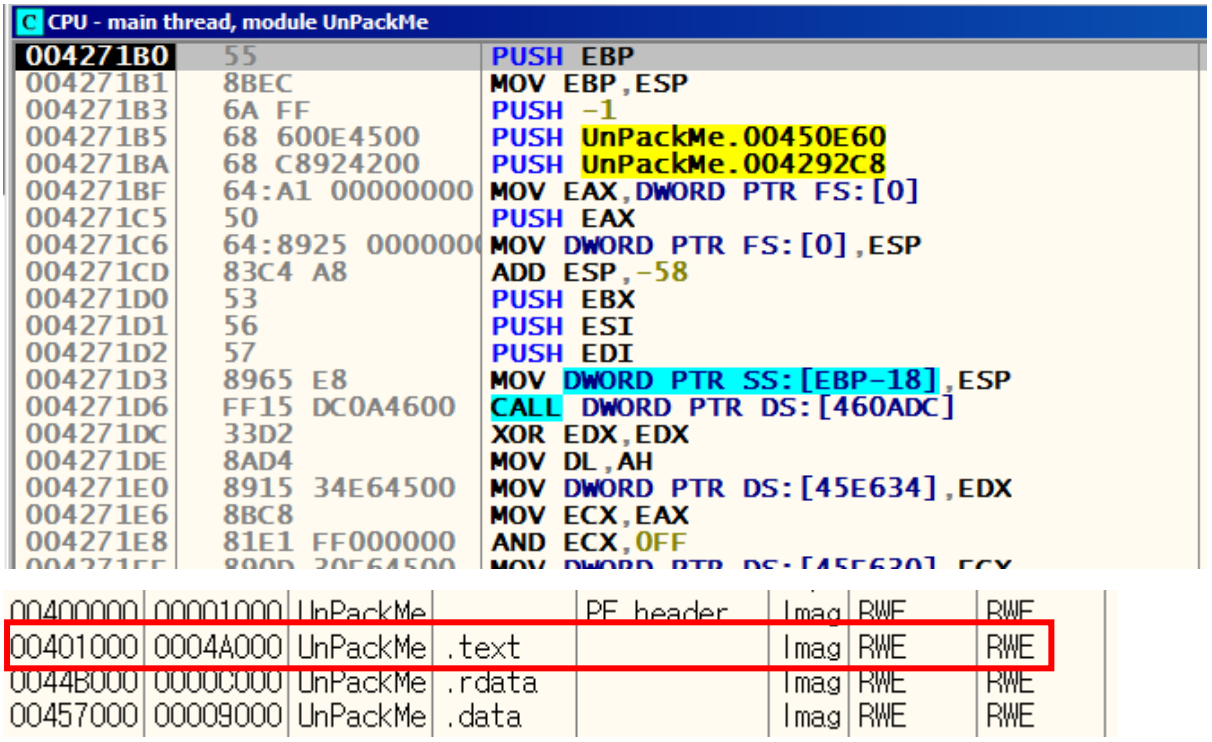



ESP의 값에서 Follow in Dump를 클릭하면 아래로 이동한다.

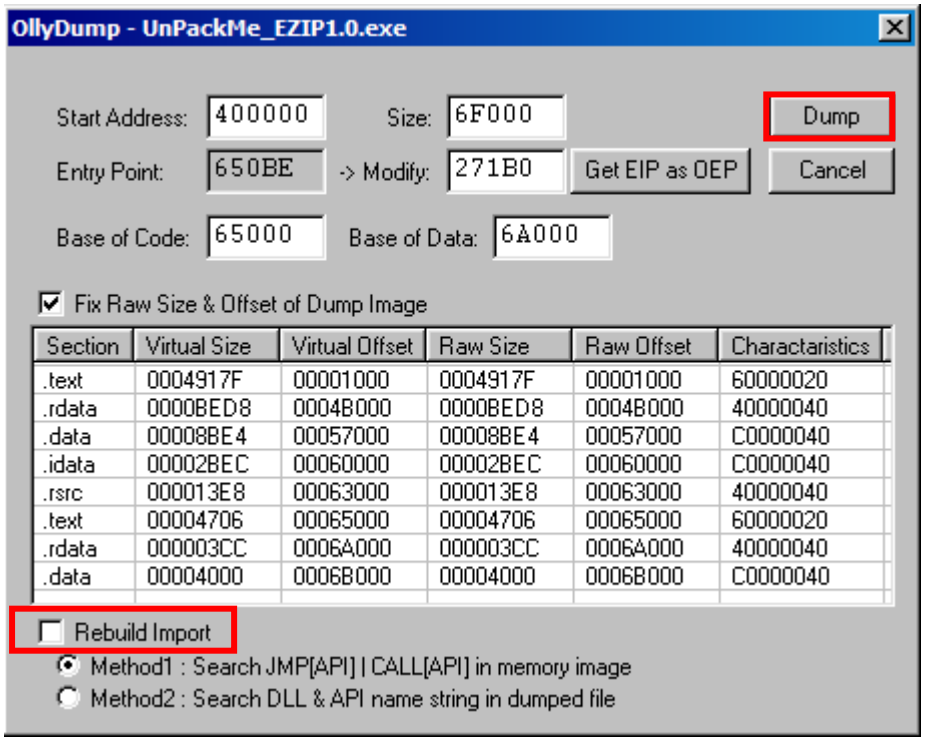



ESP가 가르키는 스택 주소에 저장하고 있는 원래의 EBP 주소를 BP를 걸어주고 'F9'을 이용해서 실행한다. (HBP는 메모리든 명령어가 0012FF94일 때 멈춘다는 뜻)


여기서 멈추는 걸 볼 수 있는데 이 부분이 다른 루틴인 원래의 OEP로 넘어갈 수 있는 코드인 걸 알 수 있다. 'F8'눌러서 실행해본다.

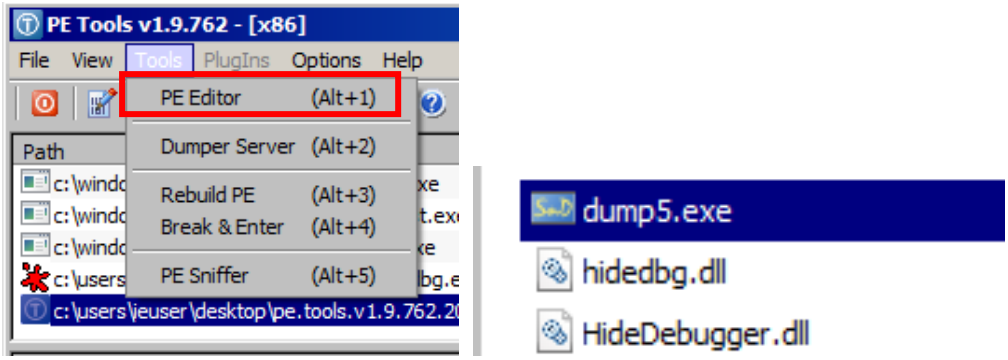



서브루틴으로 넘어가는 코드를 볼 수 있는데 0x004271B0이 코드 시작 부분이고 Memory Map에서도 확인해보면 코드 시작 부분인 걸 알 수 있다.

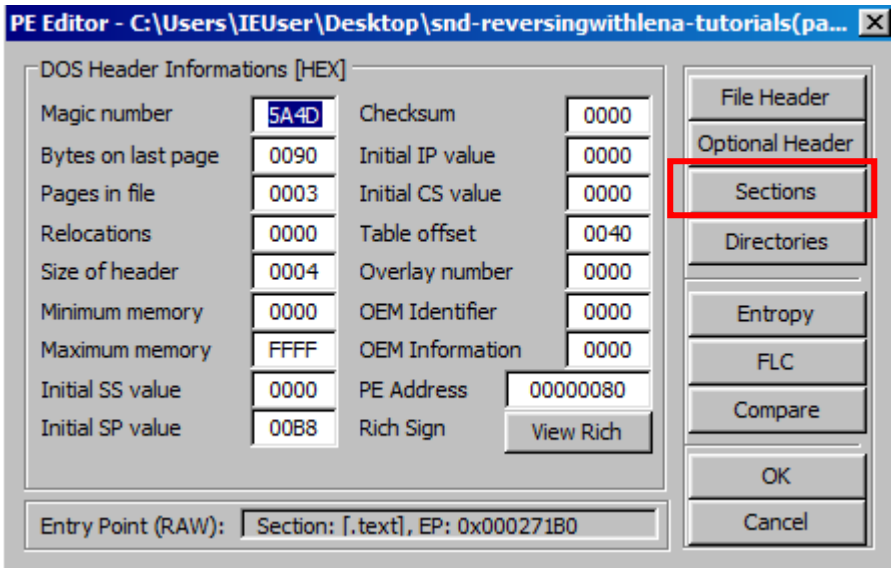



이 부분에서 dump를 해주는데 'Rebuild Import' 해제해주고 Dump해준다.

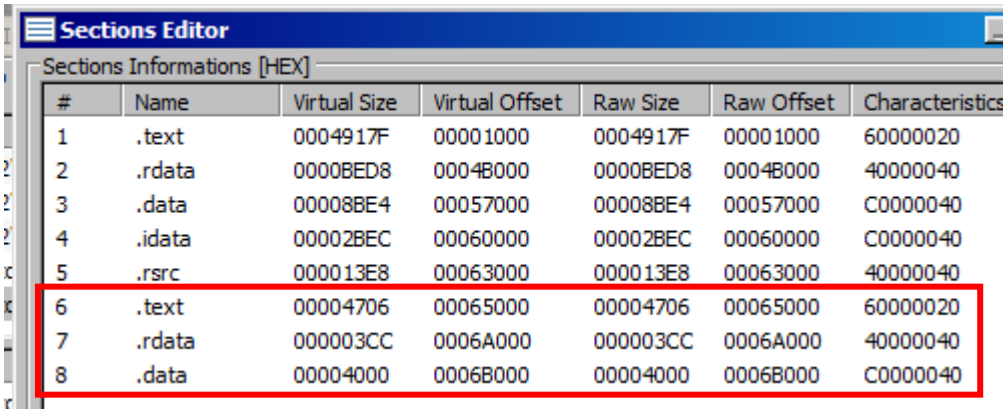
좀 더 깔끔한 덤프를 하기 위해서 PE Tools를 사용해서 Rebuild를 해준다.

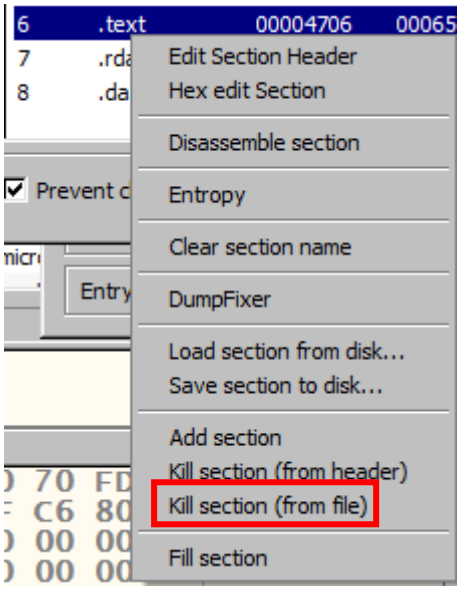



PE Tools에서 PE Editor을 이용해서 dump5.exe를 열어준다.

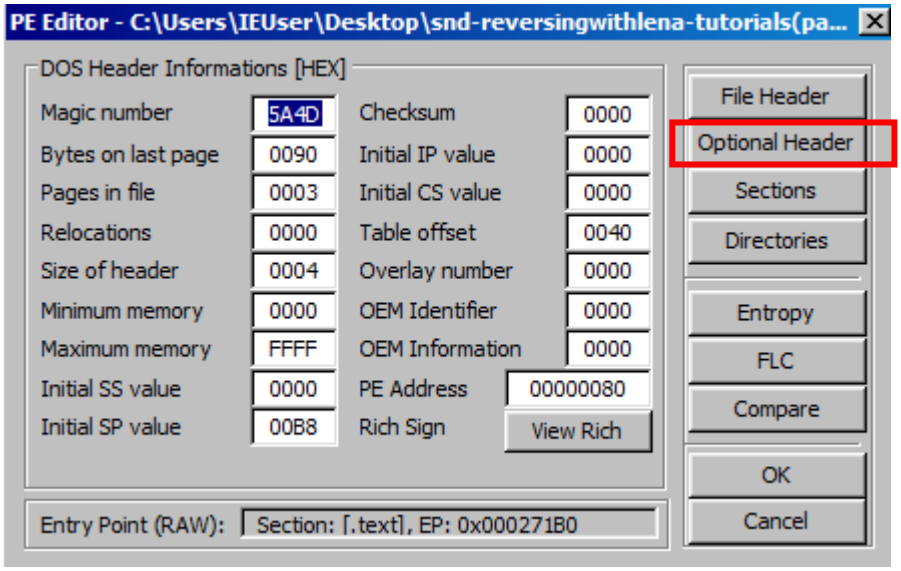


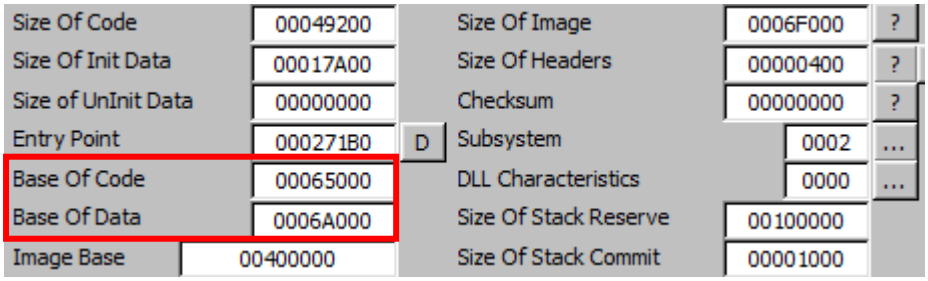

Sections 들어가서 필요없는 부분을 삭제해준다.






.text, .rdata, .data 부분을 삭제하고 저장해준다.




Optional Header에 들어가보면 Base Of Code, Base Of Data가 다른 걸 볼 수 있다.

* BaseOfCode 랑 BaseOfData 차이

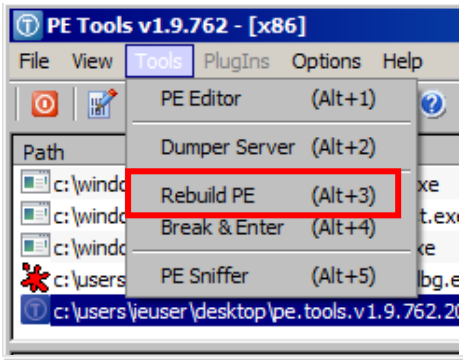
Base Of Code : 실행 가능한 코드 섹션(보통 .text)의 **시작 RVA**

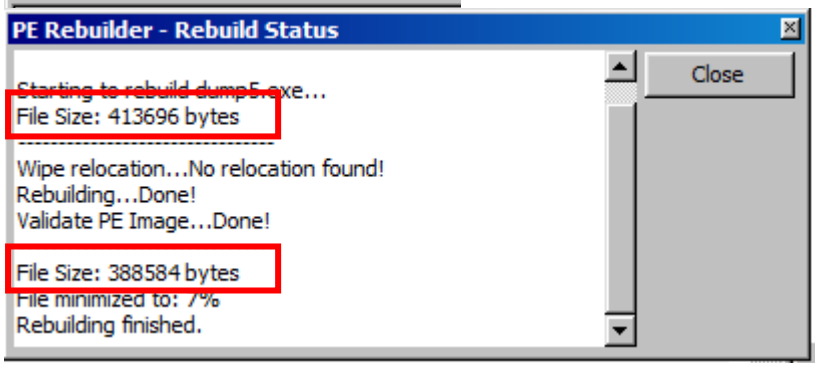
Base O fData : 데이터 섹션들(보통 .data, .rdata 등)의 **시작 RVA**.

| | | | | |
|---|---|---|---|---|
| 00400000 | 00001000 | UnPackMe | | PE header |
| 00401000 | 0004A000 | UnPackMe | .text | |
| 0044B000 | 0000C000 | UnPackMe | .rdata | |
| 00457000 | 00009000 | UnPackMe | .data | |
| 00460000 | 00003000 | UnPackMe | .idata | |

| | |
|---|---|
| Entry Point | 000271B0 |
| Base Of Code | 0001000 |
| Base Of Data | 0004B000 |

원래의 Memory Map을 참고해서 수정해준다.

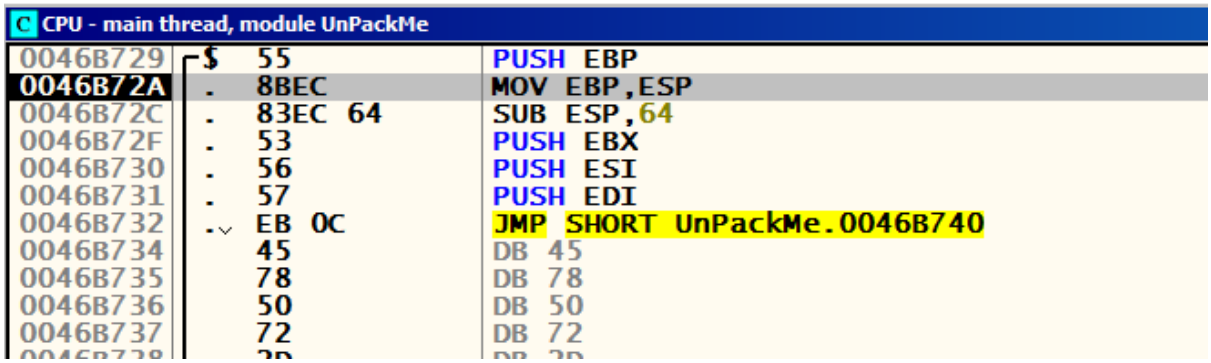


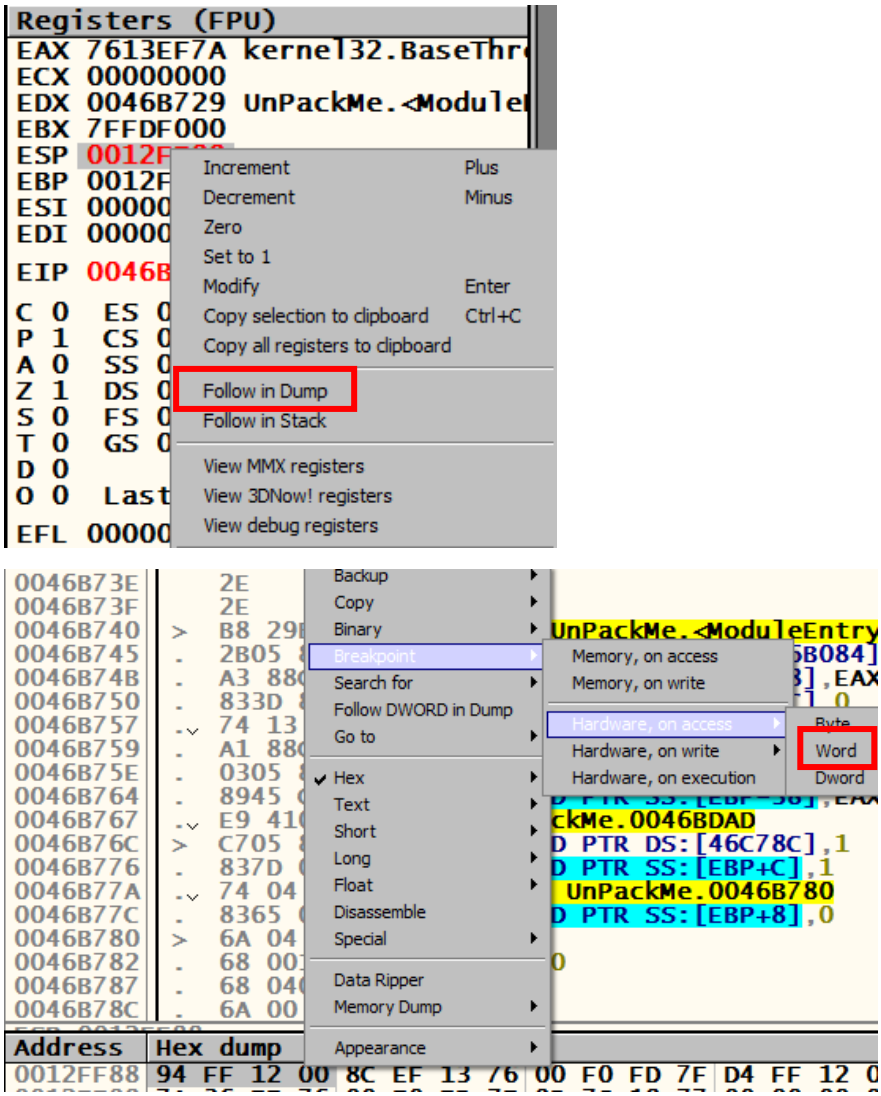

Rebuild PE를 해주면 약 7%정도 줄어든 걸 확인 할 수 있다.

-> 근데 경고창은 그대로 나와서,,, 해결은 못했다,,

2-3) UnPackMe_eXPressor1.3.0.1PK.exe



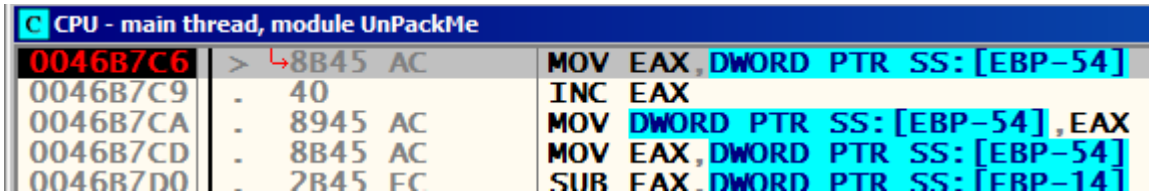

서브루틴이 시작되는 부분이라서 ESP 에 BP 걸어준다.



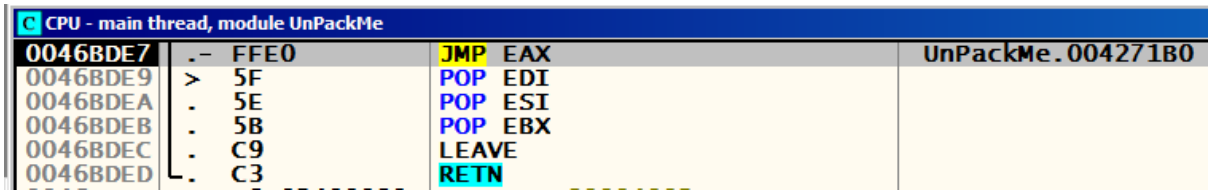




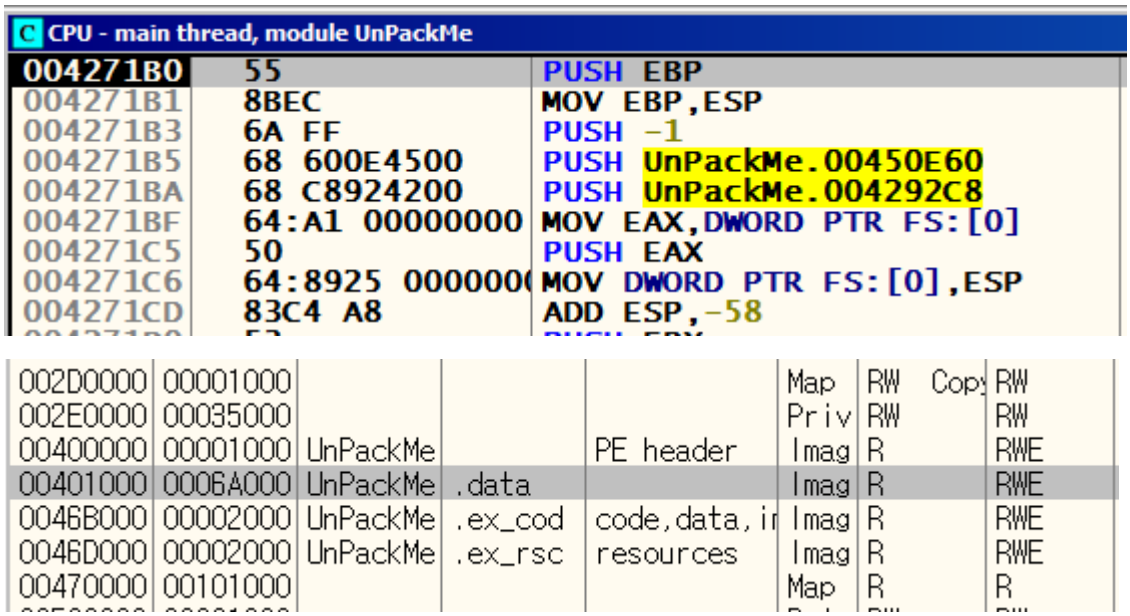
BP걸어주고 'F9'눌러서 진행시켜준다.



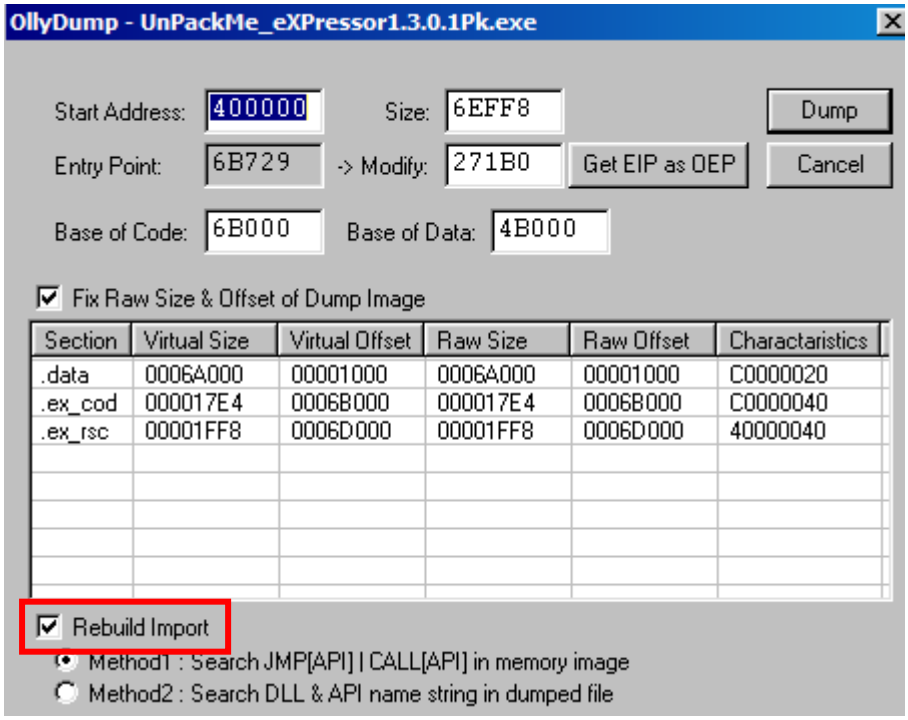

처음에 여기서 걸리는데 여기는 아니니까 계속 진행 시켜준다.



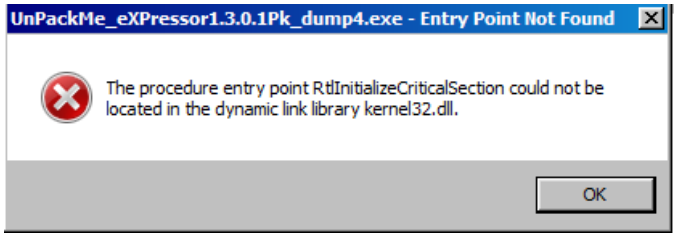

이 부분에서 서브루틴이 끝나고 다른 서브 루틴으로 넘어가는 걸 알 수 있다.




이 부분이 원래의 OEP인 걸 알 수 있다. 이 부분에서 덤프를 떠준다.




이번에는 Rebuild Import를 체크해주고 덤프를 떠준다. -> 이 부분은 레나가 나중에 설명




덤프 떠준 걸 실행해보면 오류가 발생한다. 해결을 못하겠어서 이번에도 실패이다,,

2-4) UnPackMe_MEW1.1.exe

※ MEW1.1란?

- 2000년대 초반에 많이 쓰인 무료(EXE) 패커

- 실행 파일을 **압축·소형화**하고 단순한 보호를 겸함.

- 특징 : 새 섹션을 거의 만들지 않고, PE 헤더 빈 공간에 언패킹 스텁을 넣어 크기를 최소화. (일반적인 실행 파일은 언패킹 스텁을 코드 뒤에 따로 만들어서 포함시킨다.)

-> OEP를 찾기 위해서는 **헤더에서 RET를 찾아**주면된다.

```
CPU - main thread, module UnPackMe
0049B339 - E9 164EF6FF    JMP UnPackMe.00400154
0049B33E   0C D0          OR AL,0D0
0049B340   06             PUSH ES
0049B341   0000           ADD BYTE PTR DS:[EAX],AL
0049B343   0000           ADD BYTE PTR DS:[EAX],AL
0049B345   0000           ADD BYTE PTR DS:[EAX],AL
0049B347   0000           ADD BYTE PTR DS:[EAX],AL
0049B349   0010           ADD BYTE PTR DS:[EAX],DL
0049B34B   B3 09          MOV BL,9
0049B34D   000CD0         ADD BYTE PTR DS:[EAX+EDX*8],CL
0049B350   06             PUSH ES
0049B351   0000           ADD BYTE PTR DS:[EAX],AL
0049B353   0000           ADD BYTE PTR DS:[EAX],AL
0049B355   0000           ADD BYTE PTR DS:[EAX],AL
0049B357   0000           ADD BYTE PTR DS:[EAX],AL
0049B359   0000           ADD BYTE PTR DS:[EAX],AL
0049B35B   0000           ADD BYTE PTR DS:[EAX],AL
0049B35D   0000           ADD BYTE PTR DS:[EAX],AL
0049B35F   0000           ADD BYTE PTR DS:[EAX],AL
```

우선 'F8'을 눌러서 넘어가 준다.

```
CPU - main thread, module UnPackMe
00400154   BE 1CD04600    MOV ESI,UnPackMe.0046D01C
00400159   8BDE           MOV EBX,ESI
0040015B   AD             LODS DWORD PTR DS:[ESI]
0040015C   AD             LODS DWORD PTR DS:[ESI]
0040015D   50             PUSH EAX
0040015E   AD             LODS DWORD PTR DS:[ESI]
0040015F   97             XCHG EAX,EDI
00400160   B2 80          MOV DL,80
00400162   A4             MOVS BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 00260000 | 00003000 | | | | Priv | RW | RW |
| 00400000 | 00001000 | UnPackMe | | PE header | Imag | R | RWE |
| 00401000 | 0006C000 | UnPackMe | MEW | code | Imag | R | RWE |
| 0046D000 | 0003D000 | UnPackMe | ┐??┬戎 | SFX,imports | Imag | R | RWE |
| 75690000 | 00001000 | KERNELBA | | PE header | Imag | R | RWE |
| 75691000 | 00044000 | KERNELBA | .text | code,imports | Imag | R | RWE |
| 756D5000 | 00002000 | KERNELBA | .data | data | Imag | R | RWE |
| 756D7000 | 00001000 | KERNELBA | .rsrc | resources | Imag | R | RWE |
| 756D8000 | 00003000 | KERNELBA | .reloc | relocations | Imag | R | RWE |

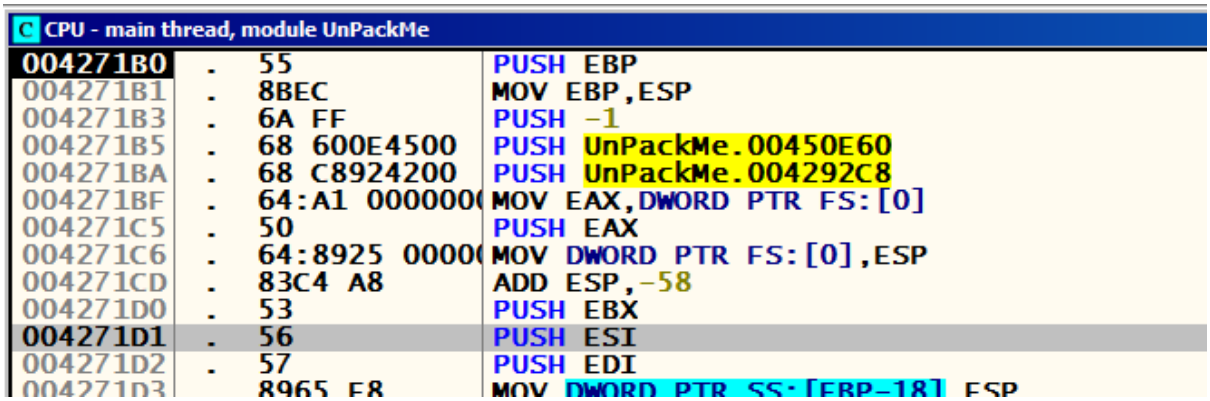
PE헤더이고 스텁이라는 걸 알 수 있다. 스크롤로 내려서 RET를 찾아준다.

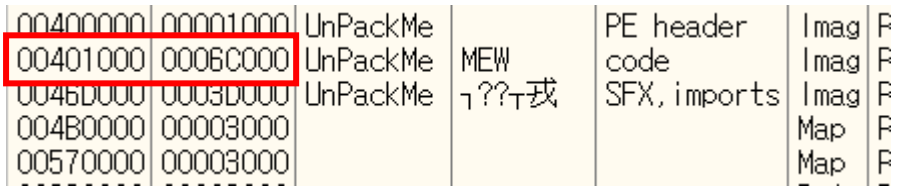
```
004001F8   AB             STOS DWORD PTR ES:[EDI]
004001F9   85C0           TEST EAX,EAX
004001FB ^ 75 E5          JNZ SHORT UnPackMe.004001E2
004001FD   C3             RETN
004001FE   0000           ADD BYTE PTR DS:[EAX],AL
```

찾은 곳 BP를 걸어주고 'F9'을 눌러 EIP가 여기로 오게 해준다.

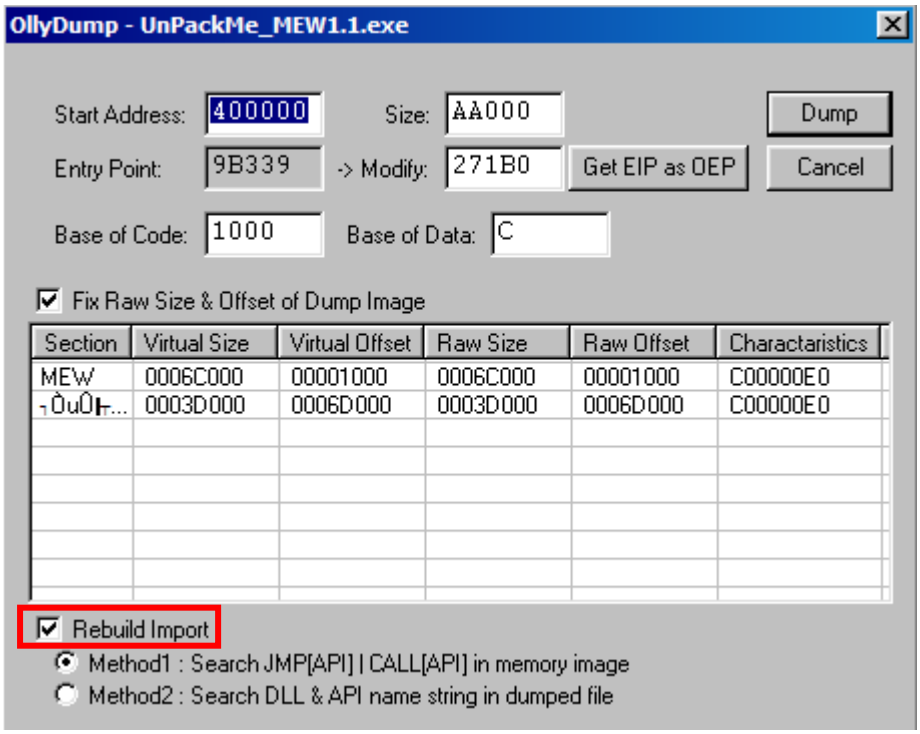
```
CPU - main thread, module UnPackMe
004271B0   55             DB 55                    CHAR 'U'
004271B1   8B             DB 8B
004271B2   EC             DB EC
004271B3   6A             DB 6A                    CHAR 'j'
004271B4   FF             DB FF
004271B5   68             DB 68                    CHAR 'h'
004271B6   60             DB 60                    CHAR '`'
004271B7   0E             DB 0E
004271B8   45             DB 45                    CHAR 'E'
004271B9   00             DB 00
004271BA   68             DB 68                    CHAR 'h'
```

'F8'을 눌러서 넘어가면 디스어셈블된 상태를 볼 수 있다. 'ctrl + a'을 눌러서 바꿔준다.

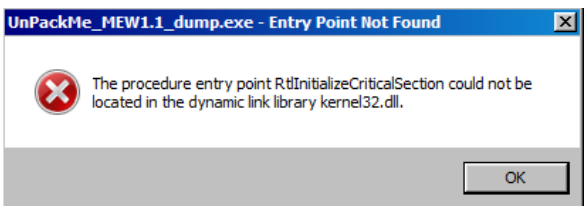



여기가 OEP인걸 알 수 있고 이 부분에서 덤프를 떠준다.




체크해주고 저장해준다.




덤프 떠준 걸 실행해보면 오류가 발생한다. 해결을 못하겠어서 이번에도 실패이다,,

2-5) UnPackMe_NsPack3.5.exe

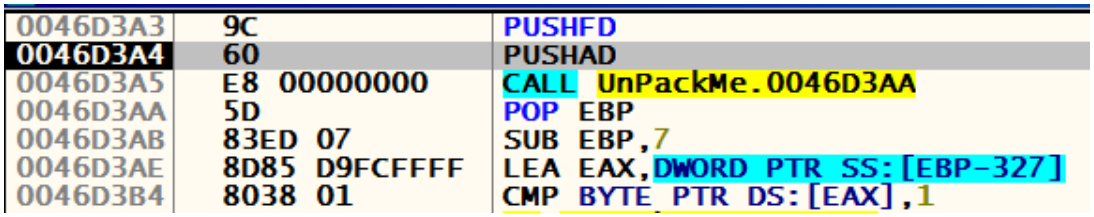
※ NsPack3.5 란?

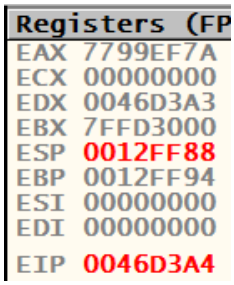
- 2000년대 초반에 많이 쓰이던 실행 파일 압축기(패커)

- 실행 파일 압축 + 간단 보호 기능을 가진 상용 패커

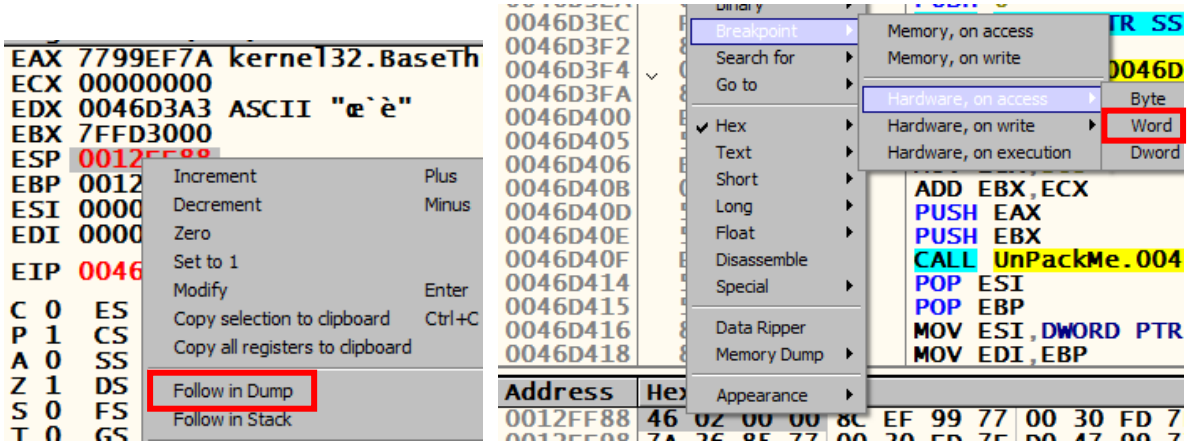
- 크기 절감, 빠른 실행, v4.x부터 IAT 은닉/간단 안티디버깅.

-> 언팩 방법 : 실행 → OEP 포착 → 덤프 + IAT 복구 → Overlay 복사.

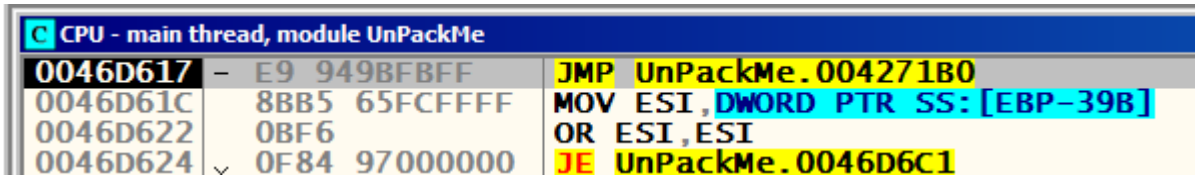


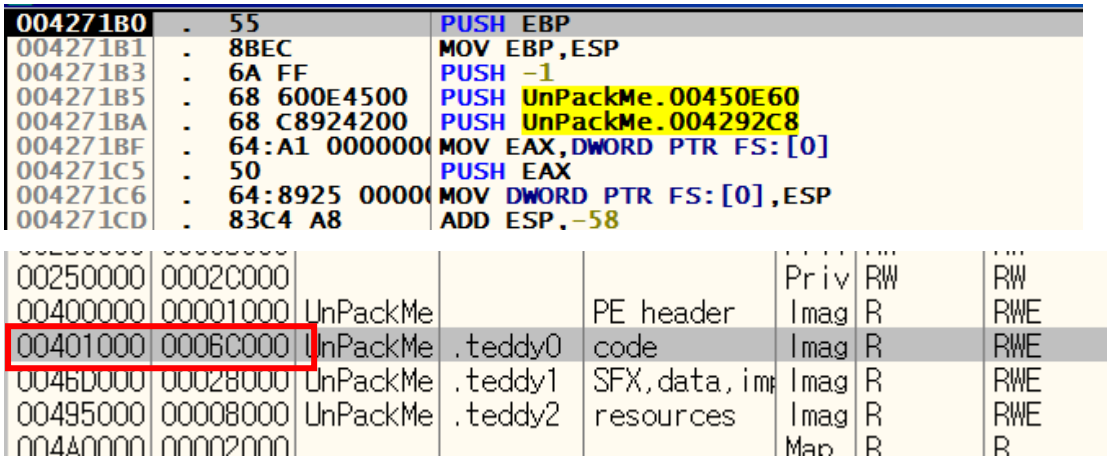

'F8' 누르면 ESP가 바뀌는 것을 알 수 있다.



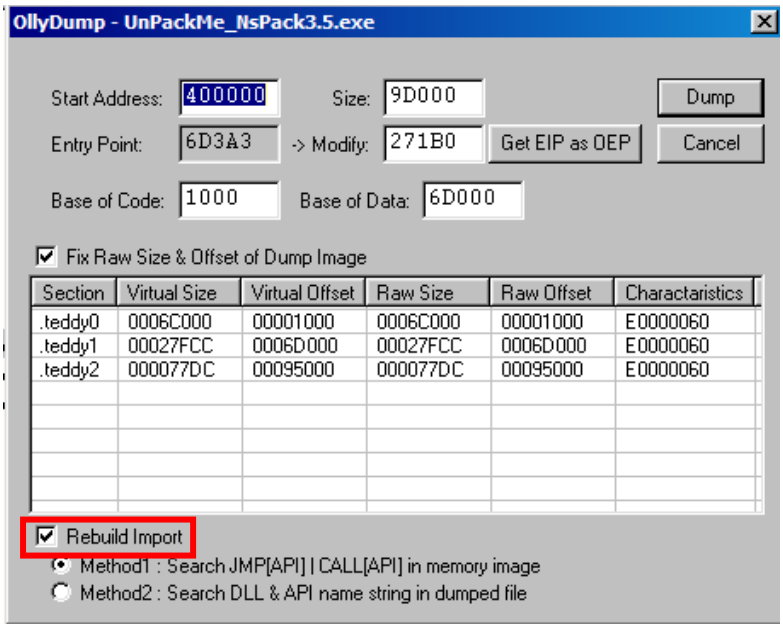

BP를 걸어주고 'F9'을 눌러준다.



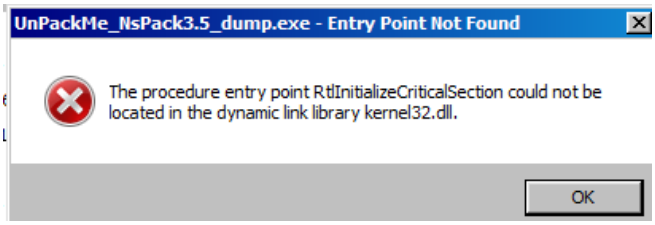

해당 구간에서 멈추는 것을 확인할 수 있다.




해당부분이 OEP인 걸 알 수 있어서 덤프를 떠준다.

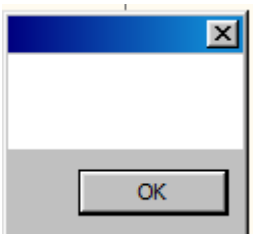



체크해주고 저장해준다.

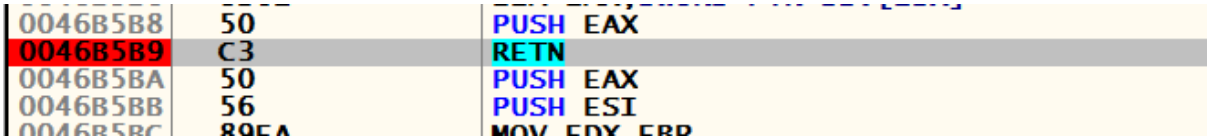



덤프 떠준 걸 실행해보면 오류가 발생한다. 해결을 못하겠어서 이번에도 실패이다,,

2-6) UnPackMe_NoNamePacker.d.out.exe

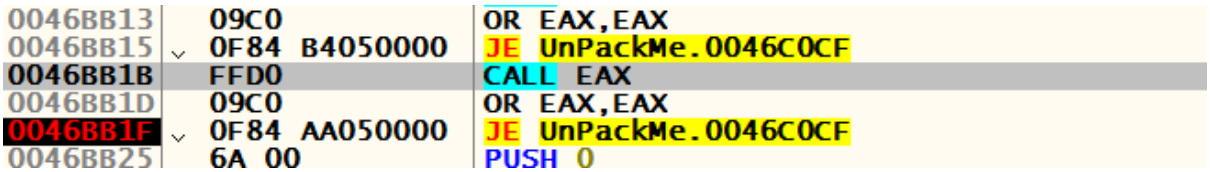



올리디버그로 실행했을 때 다르게 나오는 걸 볼 수 있다. 이런 경우에는 19번에서 사용했던 안티디버깅을 하고 있다는 걸 알 수 있다.

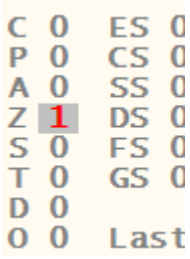



우선 패킹되어 있기 때문에 OEP로 넘어가는 구간을 찾아줘야한다.

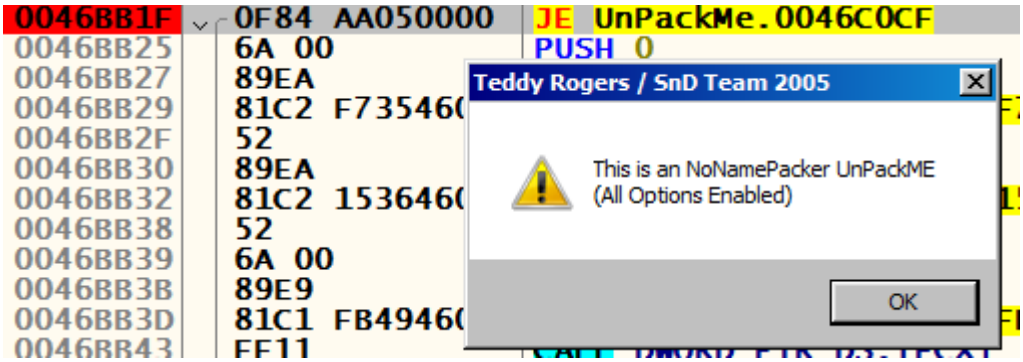
해당 부분에 BP를 걸어주고 'F8'로 계속 실행해준다.




0x0046BB1B에서 IseDeuggerPresent함수를 호출하는걸 알 수 있다. 호출 후 EAX에서 비교하고 JE에서 점프를 해야 빈 창이 나오는 걸 알 수 있다.




ZF=1로 바꿔주고 실행한다.

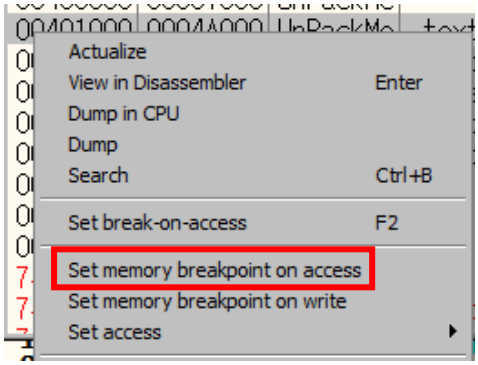



원래의 창이 나오는 걸 알 수 있다. 이제 언패킹을 해야하니 다시 돌아와서 'F8'로 계속 실행시켜준다.

```
0046C34A   EB 05          JMP SHORT UnPackMe.0046C351
0046C34C   3E:C600 00     MOV BYTE PTR DS:[EAX],0
0046C350   40             INC EAX
0046C351   3E:8038 00     CMP BYTE PTR DS:[EAX],0
0046C355   75 F5          JNZ SHORT UnPackMe.0046C34C
0046C357   C3             RETN
0046C358   55             PUSH EBP
0046C359   89E5           MOV EBP,ESP
```

실행하다보면 이 부분에서 무한루프가 도는 걸 알 수 있다. 이걸로는 찾을 수 없기 때문에 다른 방법을 찾아야한다.

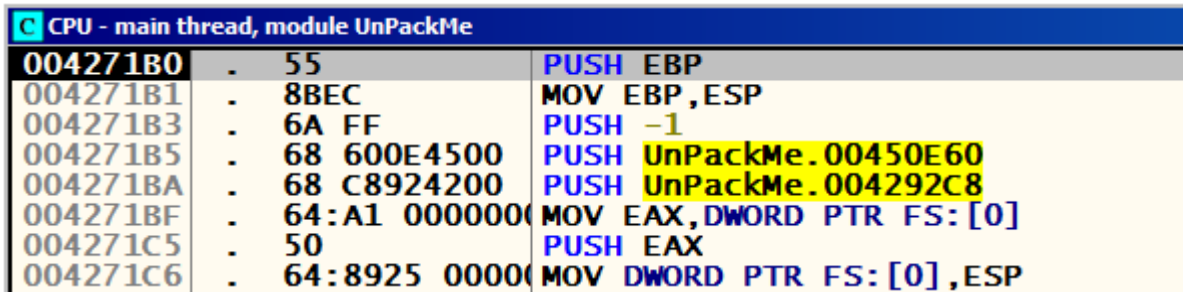
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 00400000 | 00001000 | UnPackMe | | PE header | Imag | RW | Copy | RWE |
| 00401000 | 0004A000 | UnPackMe | .text | code | Imag | RW | Copy | RWE |
| 0044B000 | 0000C000 | UnPackMe | .rdata | | Imag | RW | Copy | RWE |
| 00457000 | 00009000 | UnPackMe | .data | data | Imag | RW | Copy | RWE |
| 00460000 | 00003000 | UnPackMe | .idata | | Imag | RW | Copy | RWE |
| 00463000 | 00008000 | UnPackMe | .rsrc | resources | Imag | RW | Copy | RWE |
| 0046B000 | 0002F000 | UnPackMe | .Prt | SFX,imports | Imag | RW | Copy | RWE |
| 00690000 | 00006000 | | | | Priv | RW | | RW |



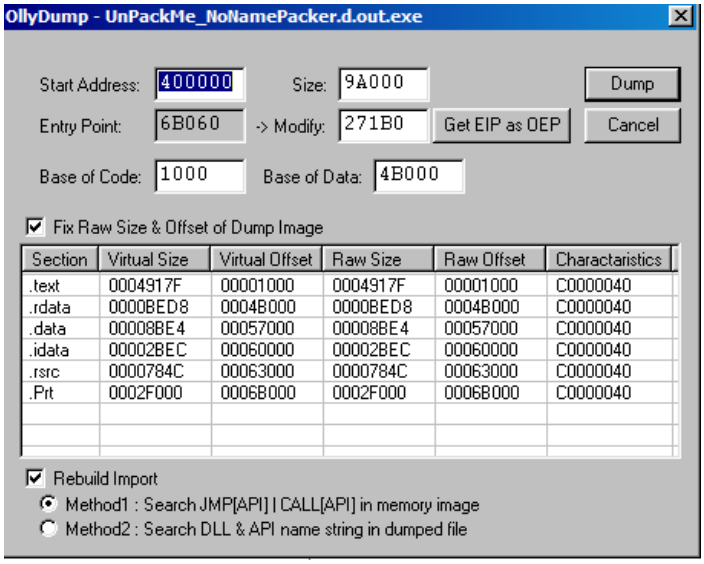

해당 부분에서 Set memory breakpoint on access을 걸어준다. 왜냐하면 OEP가 .text에 있기 때문에 무조건 거치기 때문이다.

```
CPU - main thread, module UnPackMe
004271B0   55    DB 55
004271B1   8B    DB 8B
004271B2   EC    DB EC
004271B3   6A    DB 6A
004271B4   FF    DB FF
004271B5   68    DB 68
004271B6   60    DB 60
004271B7   0E    DB 0E
004271B8   45    DB 45
004271B9   00    DB 00
004271BA   68    DB 68
004271BB   C8    DB C8
004271BC   92    DB 92
004271BD   42    DB 42
004271BE   00    DB 00
004271BF   64    DB 64
004271C0   A1    DB A1
004271C1   00    DB 00
```

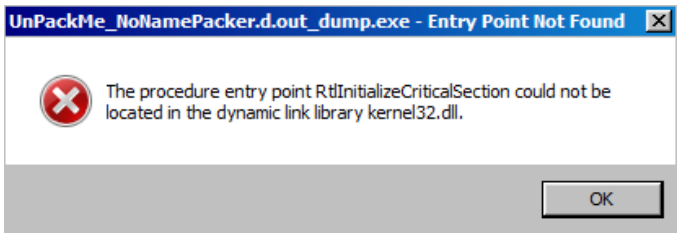
'F9'으로 실행해주면 해당 부분에서 멈추는 걸 알 수 있다.




'ctrl + a'해주면 바뀌는 걸 알 수 있고 해당 부분이 OEP인 걸 알 수 있다.




해당 부분에서 덤프를 떠준다.




덤프 떠준 걸 실행해보면 오류가 발생한다. 해결을 못하겠어서 이번에도 실패이다,,

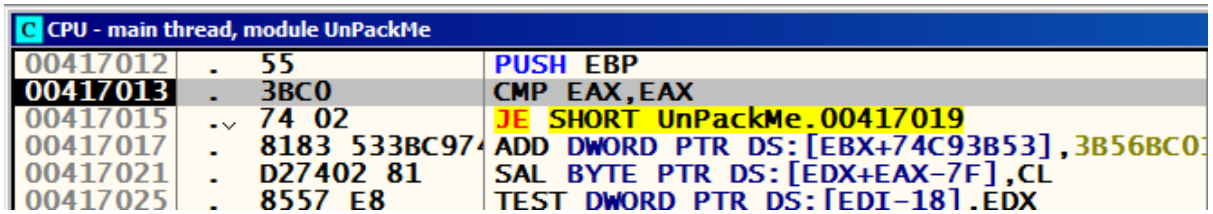
2-7) UnPackMe_Exe32Pack1.42.exe

※ Exe32Pack1.42 란?

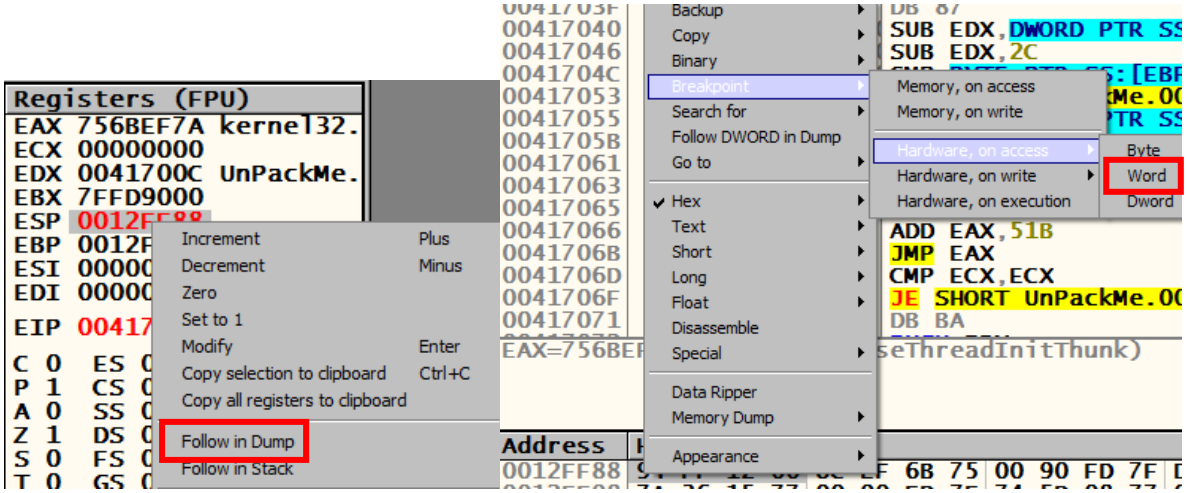
- 2000년대 초반 유행했던 Win32 실행 파일 패커(압축기) 중 하나

- 보통 새로운 섹션(.e32pack 또는 비정상 이름) 안에 언패킹 스텁 삽입.

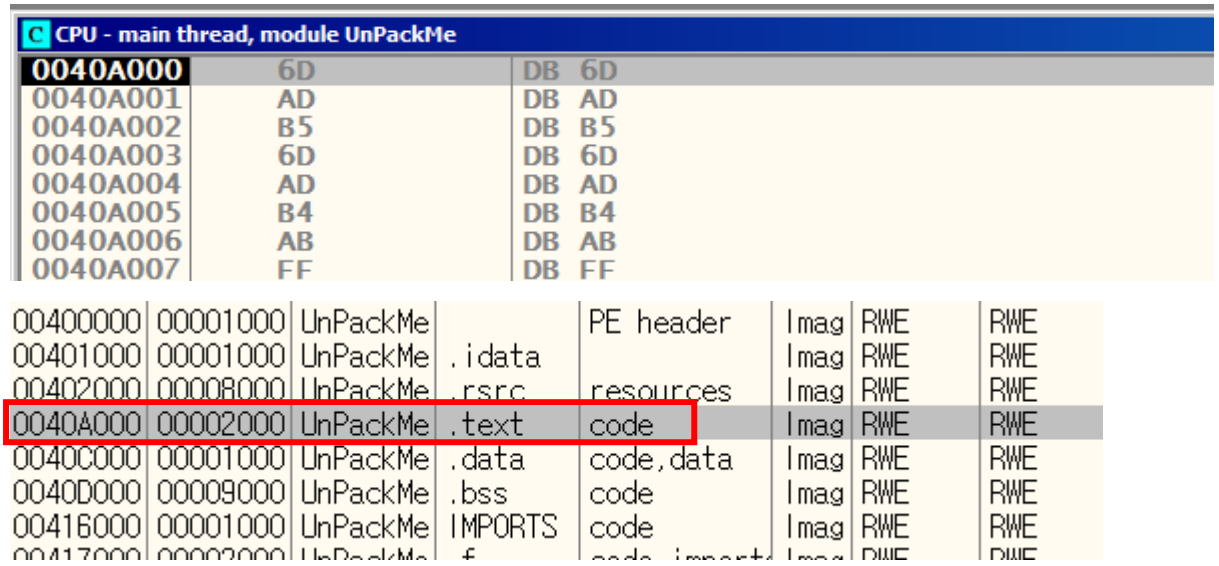
- 기본 버전(1.4x)은 강력한 안티디버깅 기법은 거의 없음. 하지만 Import Table을 줄여놓거나 변조하는 경우가 많아서, 덤프 후 IAT 복구가 필요.



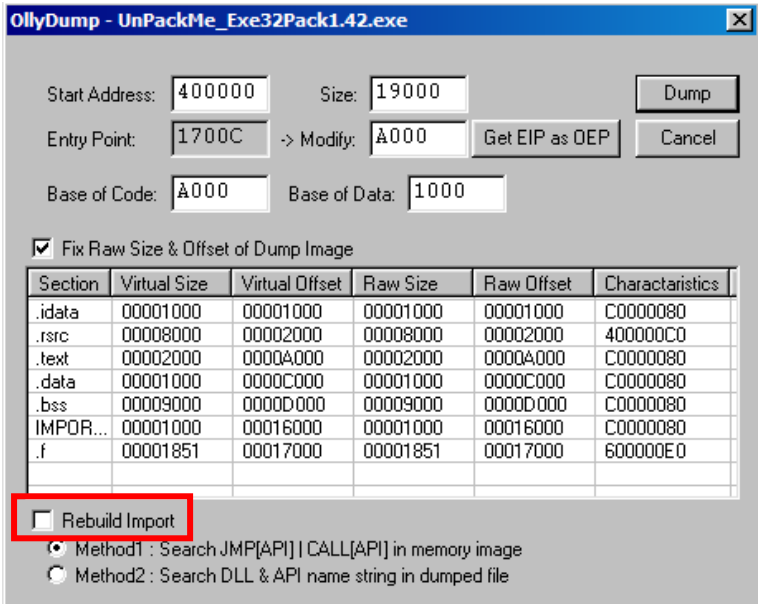

0x00417013에서 ESP가 변함으로 거기서 BP를 해준다.




BP를 걸어주고 'F9'을 해준다.




이 부분이 나오는데 'ctrl + a'가 안된다. 하지만 OEP로 추정할 수 있다. 여기서 덤프를 떠준다.




이번에는 체크를 해제하고 덤프를 떠준다. -> 왜 이번에는 해제하는지 나중에 알려준다 함.

덤프 떠준 걸 실행해보면 언팩하기 전 이랑 똑 같은 창이 나오는데 실패인지 아닌지 모르겠다,,