

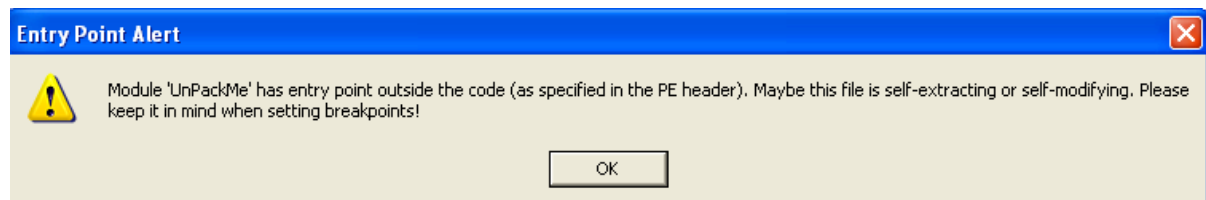
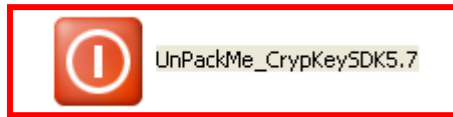
악성코드 분석 보고서

(sand-reversingwithlana-tutorials)

2025.08.14

1. 문제

1-1) UnPackMe_CrpKeySDK5.7

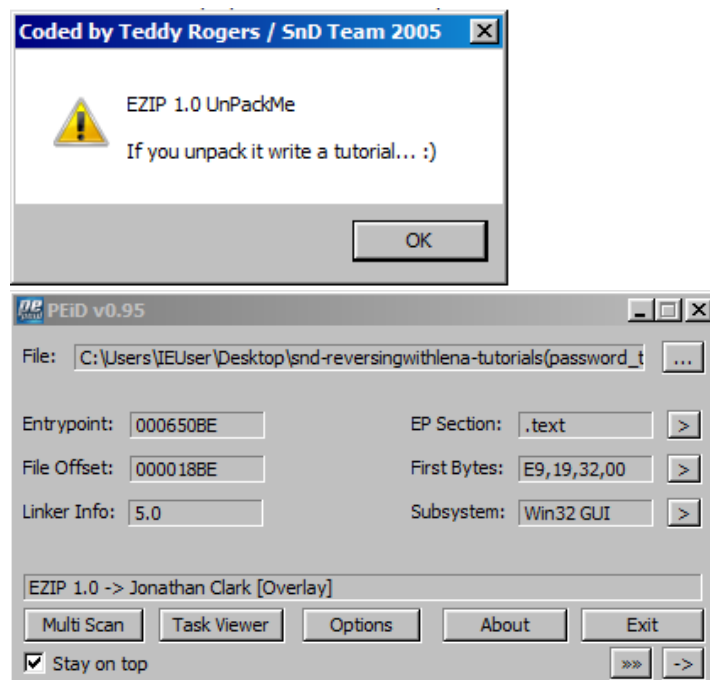


해당 파일 올리디버그로 실행 시 경고 창이 나오는 걸 볼 수 있다.

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!

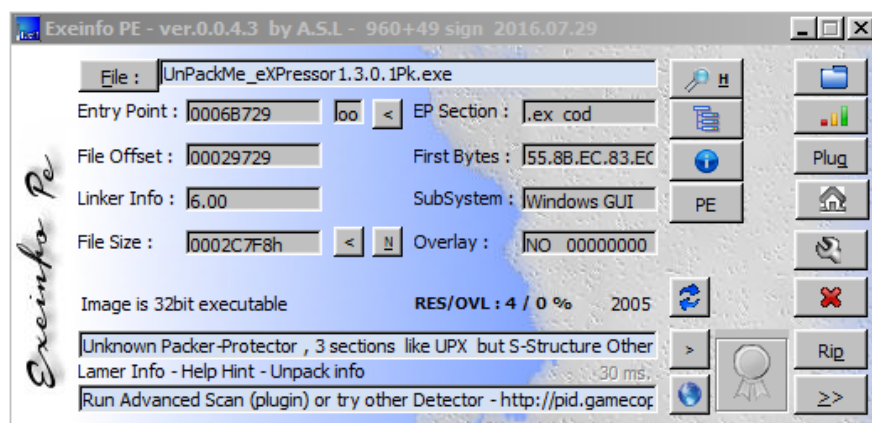
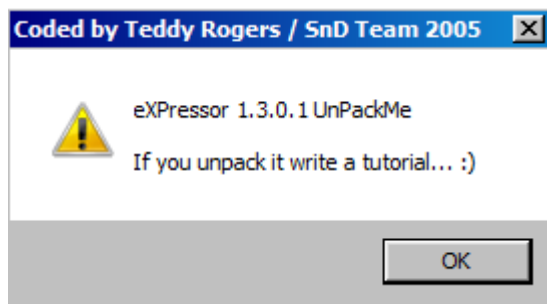
-> Entry Point(EP)**는 .text 섹션 안에 있음. 근데 이런 경고창이 발생했다는 거는 스텝 코드일 가능성이 높음.

1-2) UnPackMe_EZIP1.0



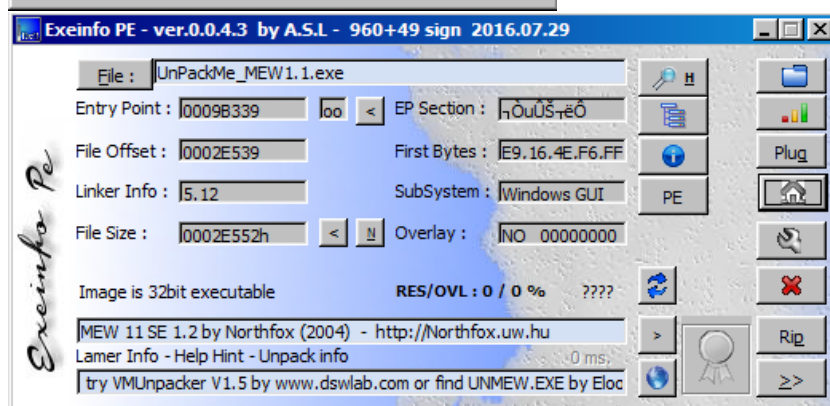
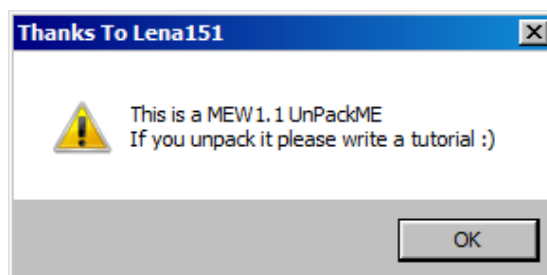
압축된 걸 알 수 있고 경고 창이 안나오게 만들어야한다.

1-3) UnPackMe_eXPressor1.3.0.1PK.exe



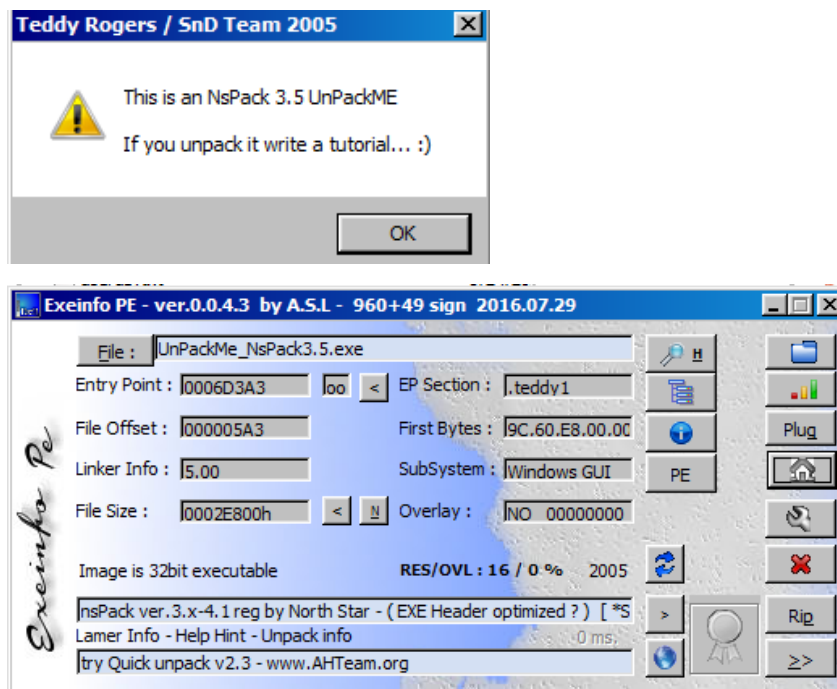
압축된 걸 알 수 있고 경고 창이 안나오게 만들어야한다.

1-4) UnPackMe_MEW1.1.exe



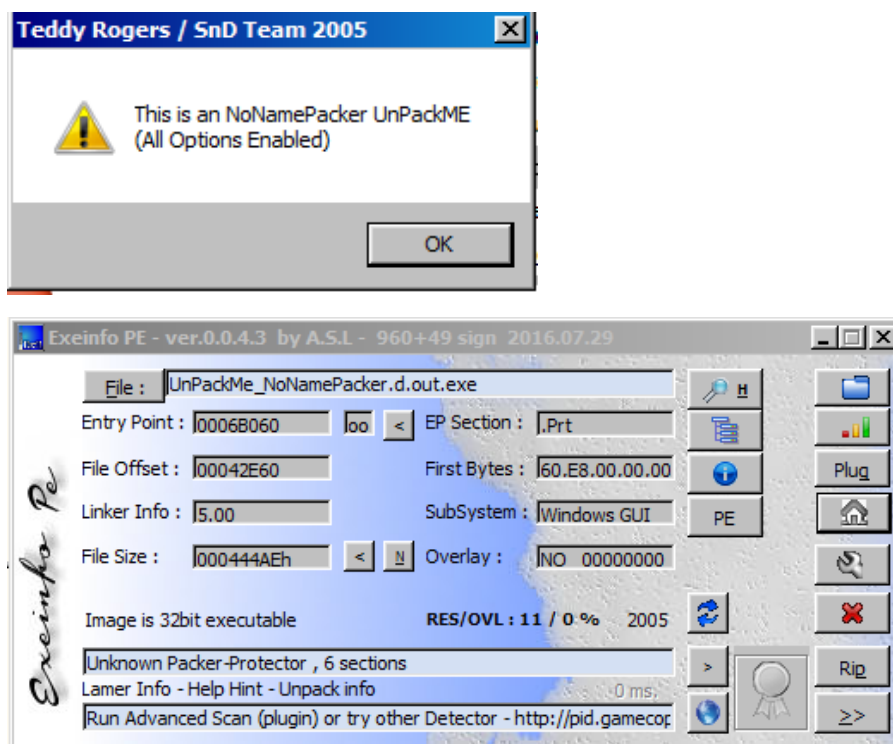
압축된 걸 알 수 있고 경고 창이 안나오게 만들어야한다.

1-5) UnPackMe_NsPack3.5.exe



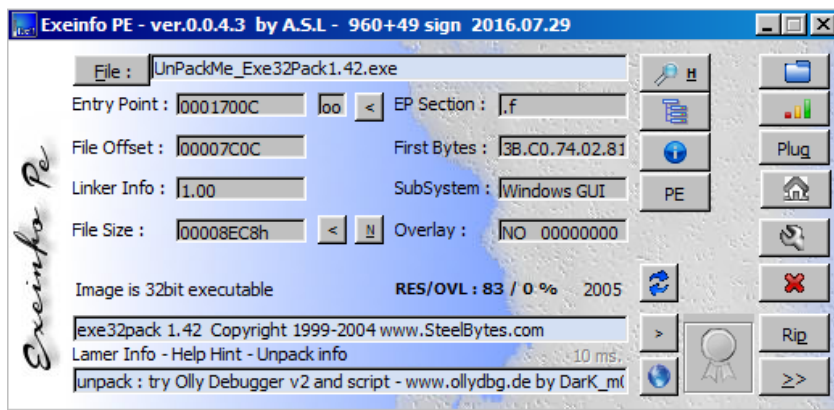
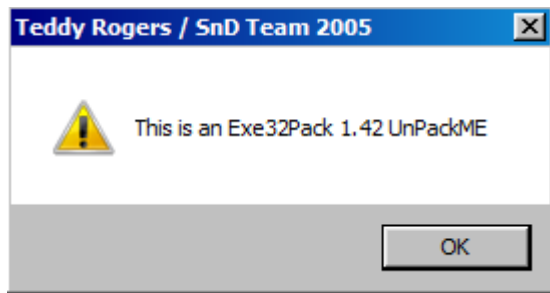
압축된 걸 알 수 있고 경고 창이 나타나게 만들어야한다.

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

0046B6DE	E8 39000000	CALL UnPackMe.0046B71C	
0046B6E3	E8 A40A0000	CALL UnPackMe.0046C18C	
0046B6E8	6A 01	PUSH 1	
0046B6EA	E8 09020000	CALL UnPackMe.0046B8F8	
0046B6EF	A1 49B64600	MOV EAX,DWORD PTR DS:[46B649]	
0046B6F4	83F8 01	CMP EAX,1	
0046B6F7	74 06	JE SHORT UnPackMe.0046B6FF	
0046B6F9	FF25 14B04600	JMP DWORD PTR DS:[46B014]	UnPackMe.00427180
0046B6FF	C3	RETN	

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

00427180	55	DB 55	CHAR 'U'
00427181	8B	DB 8B	
00427182	EC	DB EC	
00427183	6A	DB 6A	CHAR 'j'
00427184	FF	DB FF	
00427185	68	DB 68	CHAR 'h'
00427186	60	DB 60	CHAR '.'
00427187	0E	DB 0E	
00427188	45	DB 45	CHAR 'E'
00427189	00	DB 00	

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

00427180	. 55	PUSH EBP
00427181	. 8BEC	MOV EBP,ESP
00427183	. 6A FF	PUSH -1
00427185	. 68 600E4500	PUSH UnPackMe.00450E60
0042718A	. 68 C8924200	PUSH UnPackMe.004292C8
004271BF	. 64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	. 50	PUSH EAX
004271C6	. 64:8925 00000000	MOV DWORD PTR FS:[0],ESP
004271CD	. 83C4 A8	ADD ESP,-58
004271D0	. 53	PUSH EBX
004271D1	. 56	PUSH ESI
004271D2	. 57	PUSH EDI
004271D3	. 8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
004271D6	. FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
004271DC	. 33D2	XOR EDX,EDX
004271DE	. 8AD4	MOV DI,AX

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
0012C000	00004000			stack of ma	Priv	RW	Guar
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 패턴이 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 호출이 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 래더가 비교적 초반에 보이는 편
Delphi/Object Pascal	push ebp → mov ebp, esp → xor eax, eax	xor eax, eax가 초반에 눈에 띄, 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.

The screenshot shows a debugger's assembly view. The left pane displays addresses from 004271B0 to 00427217. The middle pane shows the corresponding assembly instructions, such as `PUSH EBP`, `MOV EBP, ESP`, `PUSH -1`, `PUSH UnPackM`, `MOV EAX, DWORD PTR [EBP+0]`, `PUSH EAX`, `MOV DWORD PTR [EBP+4], 0`, `ADD ESP, -58`, `PUSH EBX`, `PUSH ESI`, `PUSH EDI`, `MOV DWORD PTR [EBP+8], 8965E8`, `CALL DWORD PTR [EBP+4]`, `XOR EDX, EDX`, `MOV DL, AH`, `MOV DWORD PTR [EBP+12], 891534E64500`, `MOV ECX, EAX`, `AND ECX, 0FF`, `MOV DWORD PTR [EBP+16], 890D30E64500`, `SHL ECX, 8`, `ADD ECX, EDX`, `MOV DWORD PTR [EBP+20], 890D2CE64500`, `SHR EAX, 10`, `MOV DWORD PTR [EBP+24], A328E64500`, `CALL UnPackM`, `TEST EAX, EAX`, `JNZ SHORT UnPackM`, `PUSH 1C`, `CALL UnPackM`, and `ADD ESP, 4`. The right-hand pane shows a menu with various options, and 'Dump debugged process' is highlighted with a red rectangular box.

0x00427180 해당 부분에서 덤프를 해준다.

OllyDump - UnPackMe_CrypKeySDK5.7.exe

Start Address: 400000 Size: 6F000 Dump

Entry Point: 6B6DE Modify: 271B0 Get EIP as OEP Cancel

Base of Code: 1000 Base of Data: 4B000

☒ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	0004917F	00001000	0004917F	00001000	E0000020
.rdata	0000BED8	0004B000	0000BED8	0004B000	C0000040
.data	00008BE4	00057000	00008BE4	00057000	C0000040
.idata	00002BEC	00060000	00002BEC	00060000	C0000040
.rsrc	00007870	00063000	00007870	00063000	C0000040
Have	00002000	0006B000	00002000	0006B000	E0000020
a nice	00001000	0006D000	00001000	0006D000	50000040
day!	00001000	0006E000	00001000	0006E000	C0000040

☐ Rebuild Import

- Method1: Search JMP[API] | CALL[API] in memory image
- Method2: Search DLL & API name string in dumped file

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



[CPU - main thread, module UnPackMe]

File View Debug Plugins Options Window Help

004271B0 . 55 PUSH EBP

004271B1 . 8BEC MOV EBP, ESP

004271B3 . 6A FF PUSH -1

004271B5 . 68 600E4500 PUSH UnPackMe.00450E60

004271B8 . 68 C8924200 PUSH UnPackMe.004292C8

004271BF . 64:A1 000000 MOV EAX, DWORD PTR FS:[0]

004271C5 . 50 PUSH EAX

004271C6 . 64:8925 0000 MOV DWORD PTR FS:[0], ESP

004271CD . 83C4 A8 ADD ESP, -58

004271D0 . 53 PUSH EBX

004271D1 . 56 PUSH ESI

004271D2 . 57 PUSH EDI

004271D3 . 8965 E8 MOV DWORD PTR SS:[EBP-18], ESP

004271D6 . FF15 DC0A4600 CALL DWORD PTR DS:[460ADC]

004271DC . 33D2 XOR EDX, EDX

004271DE . 8AD4 MOV DL, AH

004271E0 . 8915 34E64500 MOV DWORD PTR DS:[45E634], EDX

004271E6 . 8BC8 MOV ECX, EAX

Process still active

Process 'UnPackMe_CrypKeySDK5.7' is active. If you terminate it now, process will be unable to clean up and write unsaved data to disk. Do you really want to terminate active process?

Note: you can permanently disable this warning in Options | Security.

Yes No

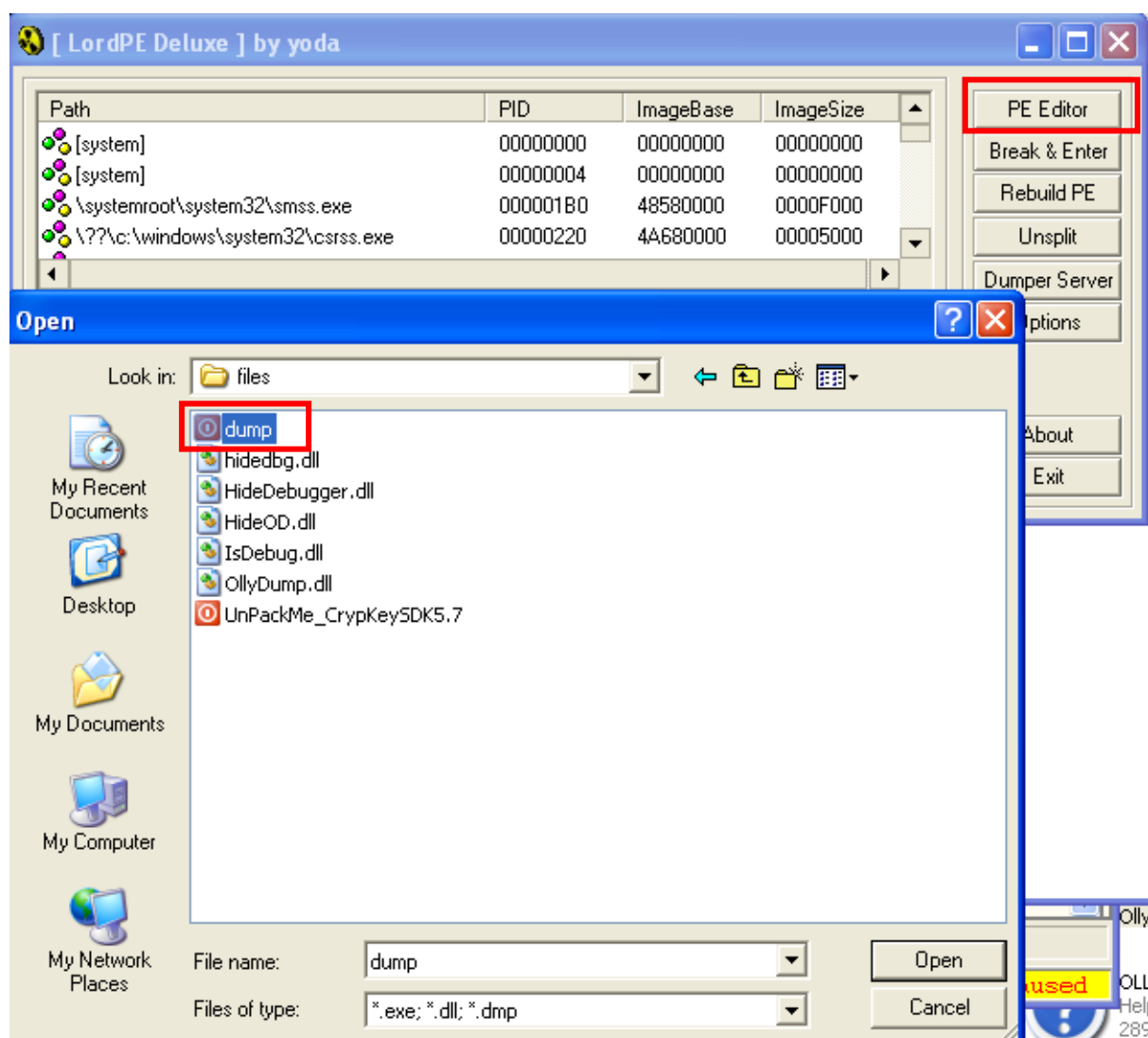
이름을 dump.exe로 저장 후 실행해주면 아까 뜨는 창이 안뜨는 걸 볼 수 있다.

문제 해결!

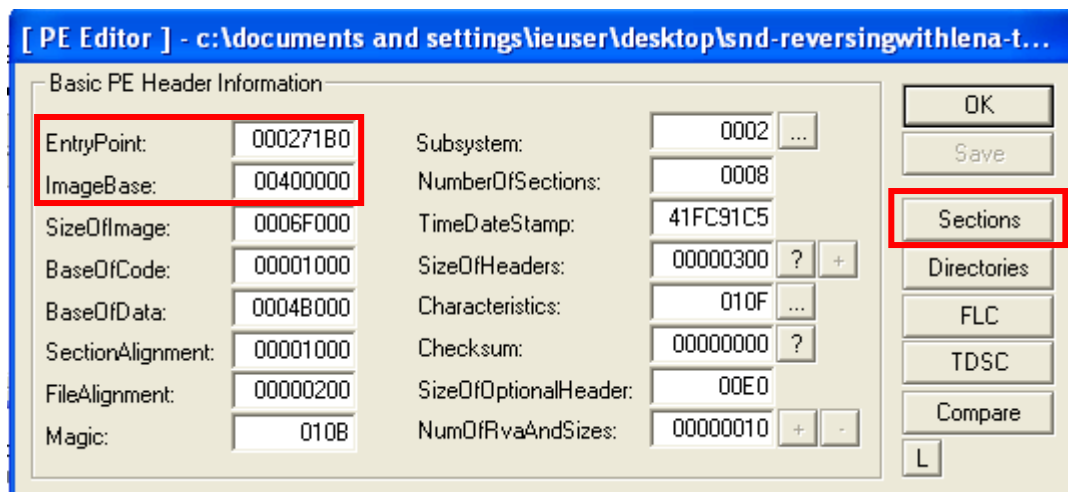
추가적으로 Lena는 LordPE라는 프로그램을 사용했는데 올리디버그 덤프 플러그로 수정이 되었기 때문에 해당 프로그램은 안사용해도 된다.

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 읽어들이어서 분석하고, 손상된 PE가 발견되었을 경우 Rebuild해주는 막강한 기능을 가지고있는 프로그램이다.



PEeditor을 눌러서 dump파일을 열어준다.



EntryPoint와 ImageBase가 제대로 설정되었나 확인 후 Sections에 들어간다.

[Section Table]					
Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	0004917F	00001000	0004917F	E0000020
.rdata	0004B000	0000BED8	0004B000	0000BED8	C0000040
.data	00057000	00008BE4	00057000	00008BE4	C0000040
.idata	00060000	00002BEC	00060000	00002BEC	C0000040
.rsrc	00063000	00007870	00063000	00007870	C0000040
Have	00068000	00002000	00068000	00002000	E0000020
a nice	0006D000	00001000	0006D000	00001000	50000040
day!	0006E000	00001000	0006E000	00001000	C0000040

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		PE header	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.

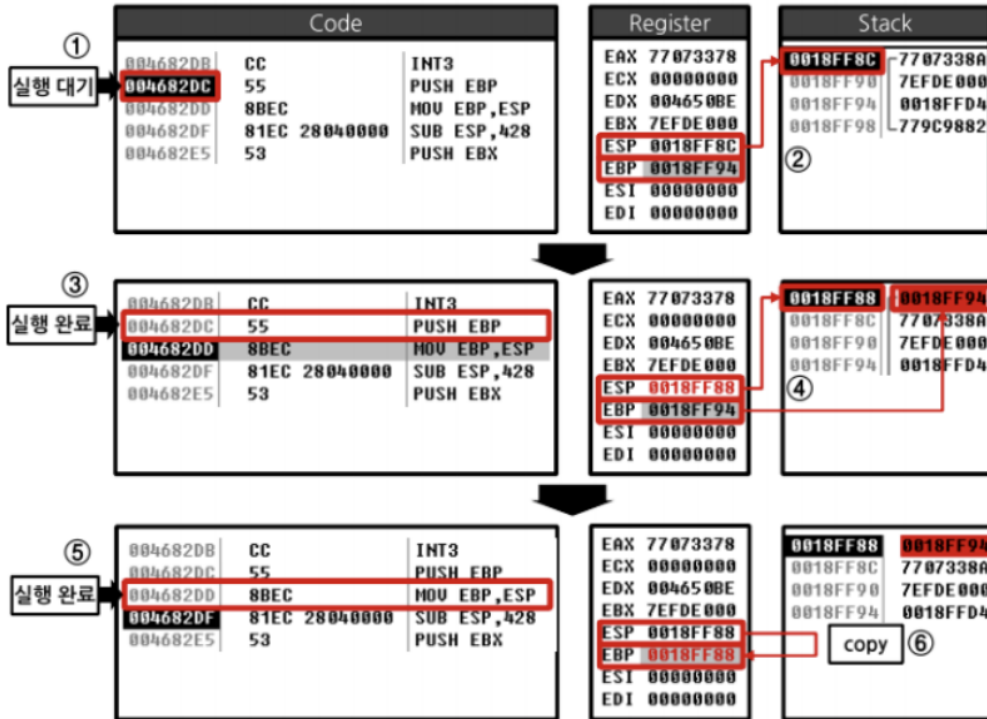
따라서 EBP 레지스터는 항상 현재 스택에서의 가장 높은 주소 값을 가지고 있으며, ESP 레지스터는 가장 낮은 주소 값을 가지고 있다.

```
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

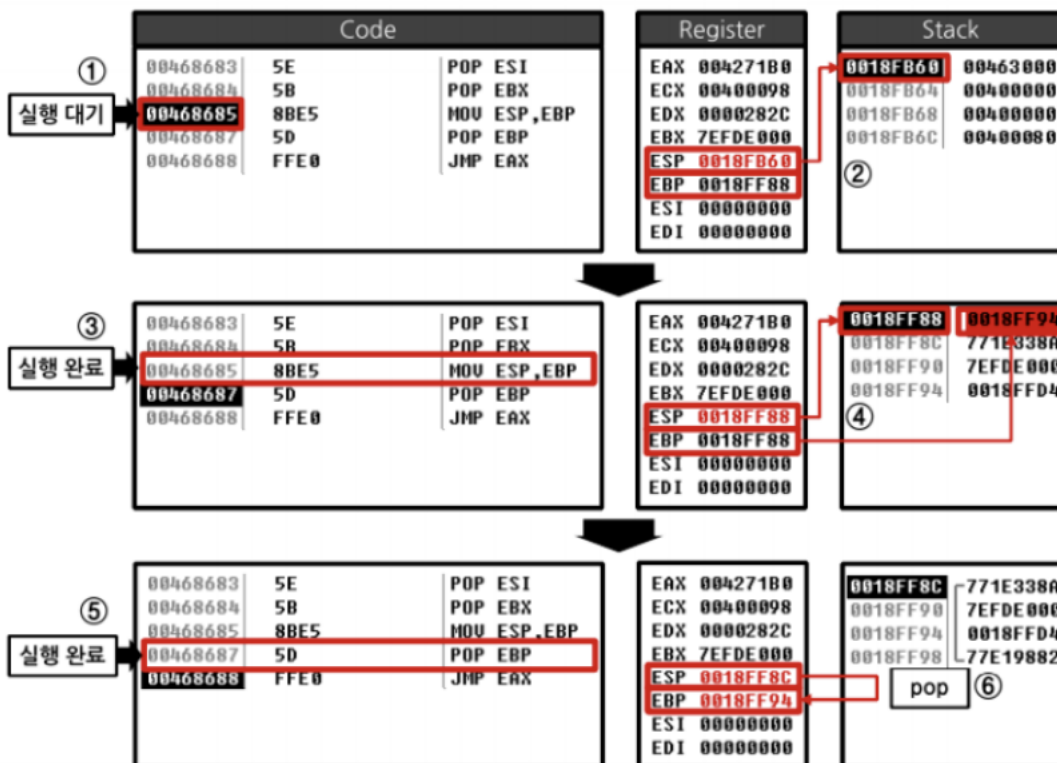
서브루틴 시작/종료할 때의 레지스터 변화

서브루틴 시작



서브루틴 시작

서브루틴 종료



서브루틴 종료

※ ESP Trick 이란?

정상 RET 대신 ESP를 직접 조작해서 흐름을 바꿔버리는 기법

asm

복사

```
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.
- ↳ jmp 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/셸코드 시작 주소를 미리 EDI 등에 준비해 둬
```

```
*/
```

```
.....
```

```
; === 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>

CPU - main thread, module UnPackMe				
004650BE	\$	E9 19320000	JMP	UnPackMe.004682DC
004650C3	.	E9 7C2A0000	JMP	UnPackMe.00467B44
004650C8	\$	E9 19240000	JMP	UnPackMe.004674E6
004650CD	\$	E9 FF230000	JMP	UnPackMe.004674D1
004650D2	.	E9 1E2E0000	JMP	UnPackMe.00467EF5
004650D7	\$	E9 882E0000	JMP	UnPackMe.00467F64
004650DC	\$	E9 2C250000	JMP	UnPackMe.0046760D
004650E1	\$	E9 AE150000	JMP	UnPackMe.00466694
004650E6	\$	E9 772B0000	JMP	UnPackMe.00467C62
004650EB	\$	E9 87020000	JMP	UnPackMe.00465377
004650F0	\$	E9 702E0000	JMP	UnPackMe.00467F65
004650F5		CC	INT3	
004650F6		CC	INT3	

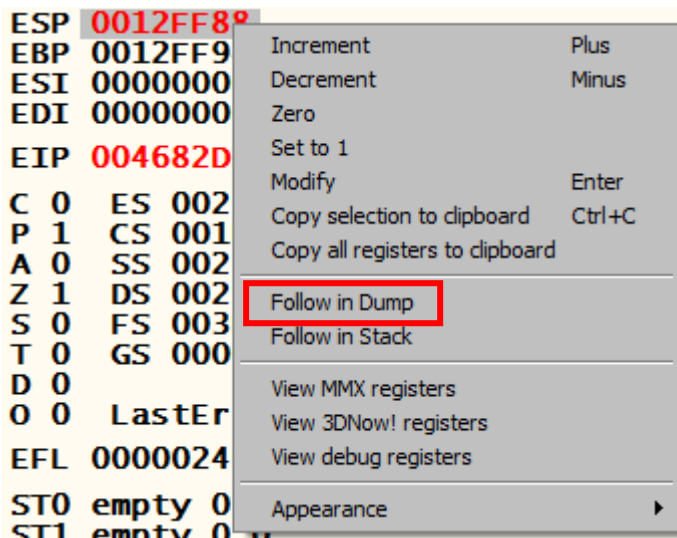
JMP 코드가 많은데 'F8'을 이용해서 실행해본다.

CPU - main thread, module UnPackMe				
004682DC	>	55	PUSH EBP	
004682DD	.	8BEC	MOV EBP,ESP	
004682DF	.	81EC 28040000	SUB ESP,428	
004682E5	.	53	PUSH EBX	
004682E6	.	56	PUSH ESI	
004682E7	.	57	PUSH EDI	
004682E8	.	8D85 94FCFFFF	LEA EAX,DWORD PTR SS:[EBP-36C]	
004682EE	.	50	PUSH EAX	
004682EF	.	E8 FCCDFFFF	CALL UnPackMe.004650F0	
004682F4	.	59	POP ECX	
004682F5	.	85C0	TEST EAX,EAX	
004682F7	.	75 05	JNZ SHORT UnPackMe.004682FE	
004682F9	.	E9 8C030000	JMP UnPackMe.0046868A	
004682FE	>	68 00800000	PUSH 8000	
00468303	.	6A 00	PUSH 0	
00468305	.	FF95 DCFCEFFF	CALL DWORD PTR SS:[EBP-324]	
0046830B	.	8985 3CFCEFFF	MOV DWORD PTR SS:[EBP-3C4],EAX	
00468311	.	C785 64FCFFFF	MOV DWORD PTR SS:[EBP-39C],UnPackMe.00468000	
0046831B	.	C785 68FCFFFF	MOV DWORD PTR SS:[EBP-398],UnPackMe.00468000	
00468325	.	8D85 94FCFFFF	LEA EAX,DWORD PTR SS:[EBP-36C]	
0046832B	.	8985 6CFCEFFF	MOV DWORD PTR SS:[EBP-394],EAX	
00468331	.	83A5 48FCFFFF	AND DWORD PTR SS:[EBP-3B8],0	
00468338	.	6A 38	PUSH 38	
0046833A	.	68 00B04600	PUSH UnPackMe.0046B000	
0046833F	.	8D85 44FCFFFF	LEA EAX,DWORD PTR SS:[EBP-3BC]	
00468345	.	50	PUSH EAX	
00468346	.	E8 E7CCEFFF	CALL UnPackMe.00465032	

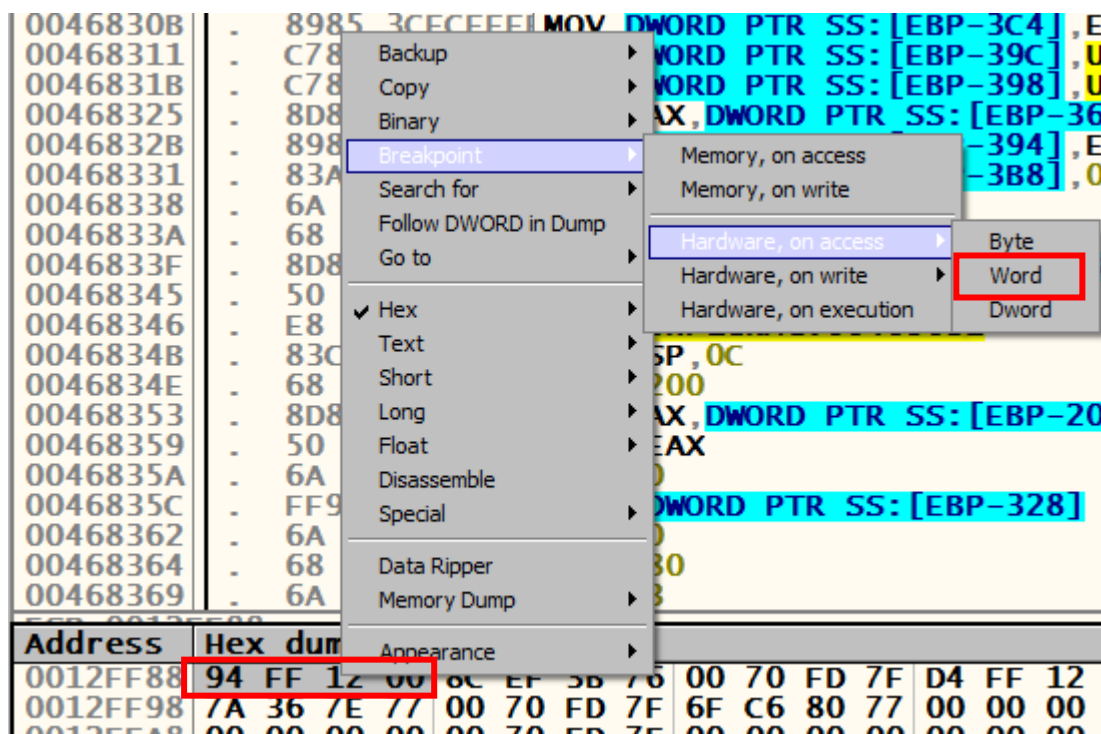
해당 코드가 나오는데 서브루틴이 시작되는 부분으로 볼 수 있다. 원래의 OEP를 찾기 위해서는 ESP는 서브루틴이 시작할 때 원래 PUSH EBP를 하면서 원래의 EBP의 값을 처음에 저장하고 그때 ESP는 원래의 EBP를 저장한 것을 가르키고 있다.

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

004682DC	>	55	PUSH EBP		
004682DD	.	8BEC	MOV EBP,ESP		
004682DF	.	81EC 28040000	SUB ESP,428		
004682E5	.	53	PUSH EBX		
004682E6	.	56	PUSH ESI		
004682E7	.	57	PUSH EDI		
004682E8	.	8D85 94FCFFFF	LEA EAX,DWORD PTR SS:[EBP-36C]		
004682EE	.	50	PUSH EAX		
004682EF	.	E8 FCCDFFFF	CALL UnPackMe.004650F0		
004682F4	.	59	POP ECX		
				Registers (FPU)	
				EAX	763BEF7A k
				ECX	00000000
				EDX	004650BE U
				EBX	77FD7000
				ESP	0012FF88
				EBP	0012FF94
				ESI	00000000
				EDI	00000000



ESP의 값에서 Follow in Dump를 클릭하면 아래로 이동한다.



ESP가 가르키는 스택 주소에 저장하고 있는 원래의 EBP 주소를 BP를 걸어주고 'F9'을 이용하여 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.

CPU - main thread, module UnPackMe						
004271B0	55					PUSH EBP
004271B1	8BEC					MOV EBP,ESP
004271B3	6A FF					PUSH -1
004271B5	68 600E4500					PUSH UnPackMe.00450E60
004271BA	68 C8924200					PUSH UnPackMe.004292C8
004271BF	64:A1 00000000					MOV EAX,DWORD PTR FS:[0]
004271C5	50					PUSH EAX
004271C6	64:8925 00000000					MOV DWORD PTR FS:[0],ESP
004271CD	83C4 A8					ADD ESP,-58
004271D0	53					PUSH EBX
004271D1	56					PUSH ESI
004271D2	57					PUSH EDI
004271D3	8965 E8					MOV DWORD PTR SS:[EBP-18],ESP
004271D6	FF15 DC0A4600					CALL DWORD PTR DS:[460ADC]
004271DC	33D2					XOR EDX,EDX
004271DE	8AD4					MOV DL,AH
004271E0	8915 34E64500					MOV DWORD PTR DS:[45E634],EDX
004271E6	8BC8					MOV ECX,EAX
004271E8	81E1 FF000000					AND ECX,0FF
004271EF	8905 34E64500					MOV DWORD PTR DS:[45E634],ECX

00400000	00001000	UnPackMe		PE header	Image	RWE	RWE
00401000	0004A000	UnPackMe	.text		Image	RWE	RWE
0044B000	0000C000	UnPackMe	.rdata		Image	RWE	RWE
00457000	00009000	UnPackMe	.data		Image	RWE	RWE

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

OllyDump - UnPackMe_EZIP1.0.exe

Start Address: 400000 Size: 6F000 **Dump**

Entry Point: 650BE -> Modify: 271B0 Get EIP as OEP Cancel

Base of Code: 65000 Base of Data: 6A000

☒ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	0004917F	00001000	0004917F	00001000	60000020
.rdata	0000BED8	0004B000	0000BED8	0004B000	40000040
.data	00008BE4	00057000	00008BE4	00057000	C0000040
.idata	00002BEC	00060000	00002BEC	00060000	C0000040
.rsrc	000013E8	00063000	000013E8	00063000	40000040
.text	00004706	00065000	00004706	00065000	60000020
.rdata	000003CC	0006A000	000003CC	0006A000	40000040
.data	00004000	0006B000	00004000	0006B000	C0000040

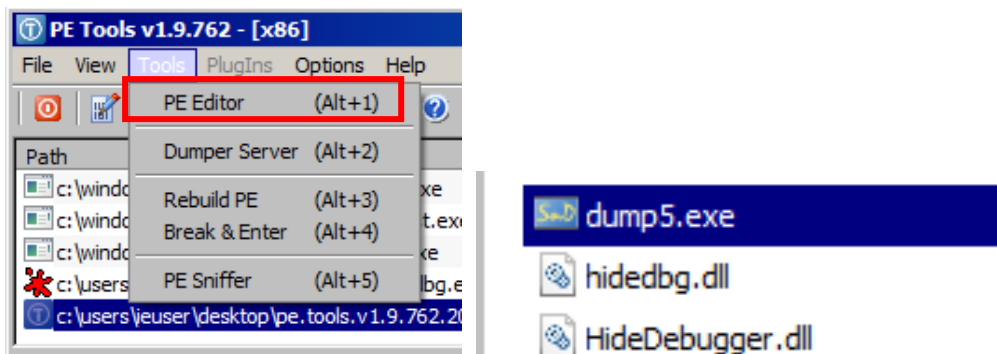
☐ Rebuild Import

☒ Method1 : Search JMP[API] | CALL[API] in memory image

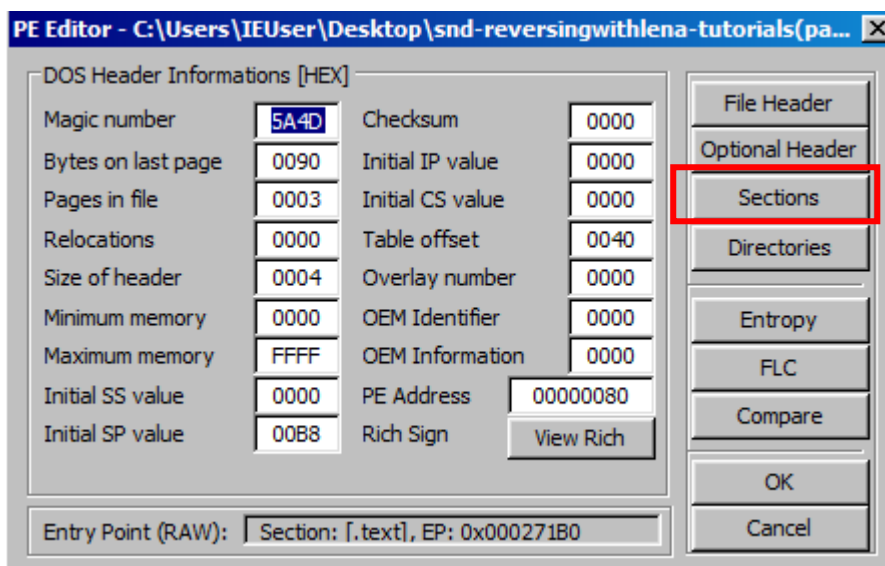
☐ Method2 : Search DLL & API name string in dumped file

이 부분에서 dump를 해주는데 'Rebuild Import' 해제해주고 Dump해준다.

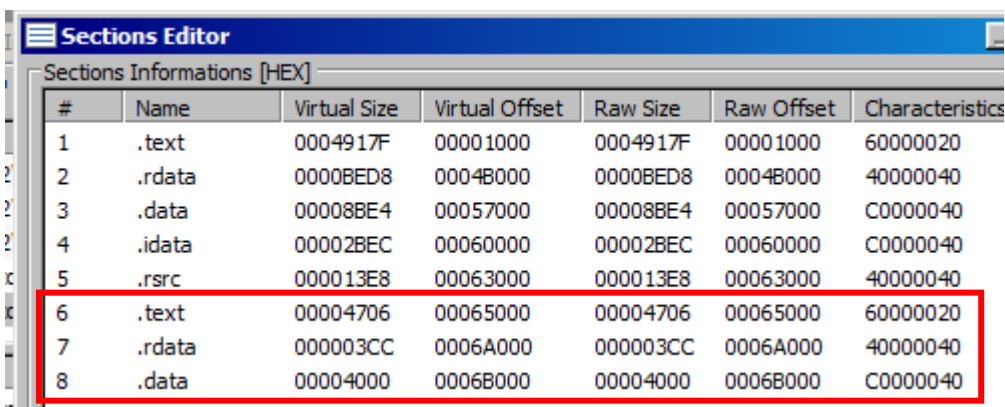
좀 더 깔끔한 덤프를 하기 위해서 PE Tools를 사용해서 Rebuild를 해준다.

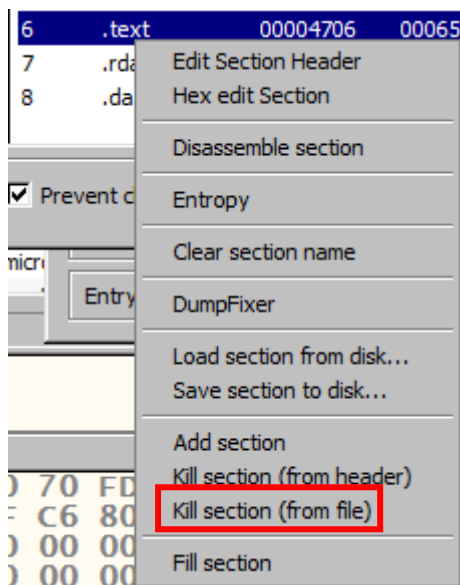


PE Tools에서 PE Editor를 이용해서 dump5.exe를 열어준다.

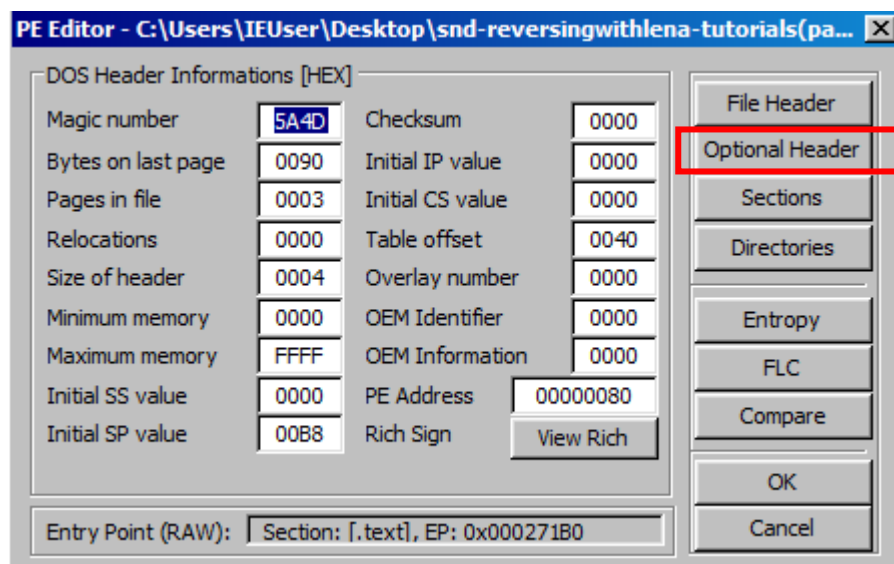


Sections 들어가서 필요없는 부분을 삭제해준다.





.text, .rdata, .data 부분을 삭제하고 저장해준다.



Size Of Code	00049200	Size Of Image	0006F000	?
Size Of Init Data	00017A00	Size Of Headers	00000400	?
Size of UnInit Data	00000000	Checksum	00000000	?
Entry Point	000271B0	Subsystem	0002	...
Base Of Code	00065000	DLL Characteristics	0000	...
Base Of Data	0006A000	Size Of Stack Reserve	00100000	
Image Base	00400000	Size Of Stack Commit	00001000	

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

* BaseOfCode 랑 BaseOfData 차이

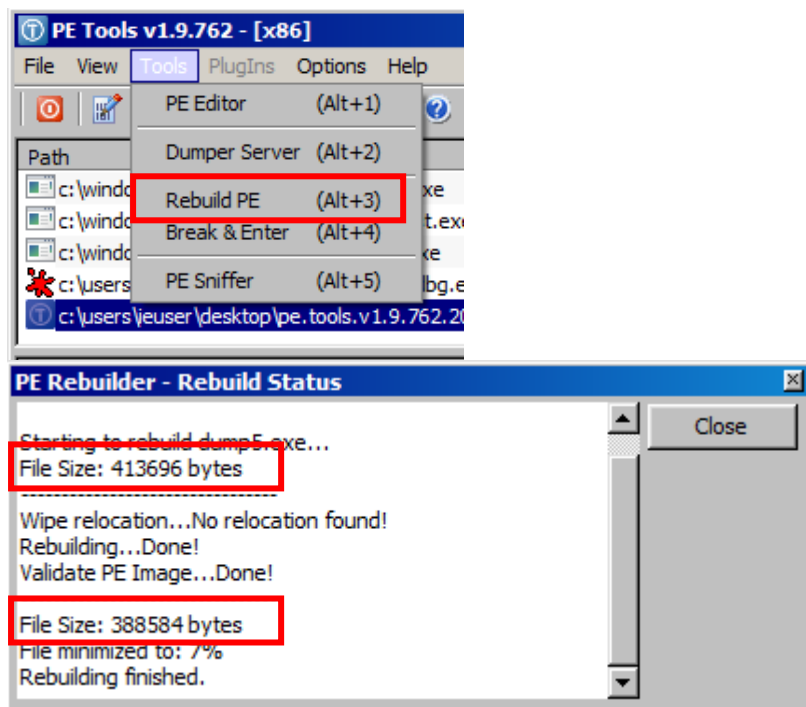
Base Of Code : 실행 가능한 코드 섹션(보통 .text)의 시작 RVA

Base Of Data : 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

CPU - main thread, module UnPackMe		
0046B729	55	PUSH EBP
0046B72A	8BEC	MOV EBP,ESP
0046B72C	83EC 64	SUB ESP,64
0046B72F	53	PUSH EBX
0046B730	56	PUSH ESI
0046B731	57	PUSH EDI
0046B732	EB 0C	JMP SHORT UnPackMe.0046B740
0046B734	45	DB 45
0046B735	78	DB 78
0046B736	50	DB 50
0046B737	72	DB 72
0046B738	35	DB 35

서브루틴이 시작되는 부분이라서 ESP 에 BP 걸어준다.

Registers (FPU)	
EAX	7613EF7A kernel32.BaseThre
ECX	00000000
EDX	0046B729 UnPackMe.<Module
EBX	7FFDF000
ESP	0012FF88
EBP	0012FF88
ESI	00000000
EDI	00000000
EIP	0046B732
C 0	ES 0
P 1	CS 0
A 0	SS 0
Z 1	DS 0
S 0	FS 0
T 0	GS 0
D 0	
O 0	Last
EFL	00000000

0046B73E	2E	Backup	
0046B73F	2E	Copy	
0046B740	B8 29	Binary	UnPackMe.<ModuleEntry
0046B745	2B05	Breakpoint	Memory, on access
0046B74B	A3 88	Search for	Memory, on write
0046B750	833D	Follow DWORD in Dump	Hardware, on access
0046B757	74 13	Go to	Hardware, on write
0046B759	A1 88		Hardware, on execution
0046B75E	0305	Hex	Byte
0046B764	8945	Text	Word
0046B767	E9 41	Short	Dword
0046B76C	C705	Long	
0046B776	837D	Float	
0046B77A	74 04	Disassemble	
0046B77C	8365	Special	
0046B780	6A 04		
0046B782	68 00	Data Ripper	
0046B787	68 04	Memory Dump	
0046B78C	6A 00	Appearance	
Address	Hex	dump	
0012FF88	94 FF 12 00	8C EF 13 76	00 F0 FD 7F D4 FF 12 0

BP걸어주고 'F9'눌러서 진행시켜준다.

CPU - main thread, module UnPackMe			
0046B7C6	> 8B45 AC	MOV EAX, DWORD PTR SS:[EBP-54]	
0046B7C9	- 40	INC EAX	
0046B7CA	- 8945 AC	MOV DWORD PTR SS:[EBP-54], EAX	
0046B7CD	- 8B45 AC	MOV EAX, DWORD PTR SS:[EBP-54]	
0046B7D0	- 2B45 EC	SUB EAX, DWORD PTR SS:[EBP-14]	

처음에 여기서 걸리는데 여기는 아니니까 계속 진행 시켜준다.

CPU - main thread, module UnPackMe			
0046BDE7	- FFE0	JMP EAX	UnPackMe.004271B0
0046BDE9	> 5F	POP EDI	
0046BDEA	- 5E	POP ESI	
0046BDEB	- 5B	POP EBX	
0046BDEC	- C9	LEAVE	
0046BDED	- C3	RETN	

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

CPU - main thread, module UnPackMe			
004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP, ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH UnPackMe.00450E60	
004271BA	68 C8924200	PUSH UnPackMe.004292C8	
004271BF	64:A1 00000000	MOV EAX, DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	
004271C6	64:8925 00000000	MOV DWORD PTR FS:[0], ESP	
004271CD	83C4 A8	ADD ESP, -58	

002D0000	00001000				Map	RW	Cop	RW
002E0000	00035000				Priv	RW		RW
00400000	00001000	UnPackMe		PE header	Imag	R		RWE
00401000	0006A000	UnPackMe	.data		Imag	R		RWE
0046B000	00002000	UnPackMe	.ex_cod	code, data, in	Imag	R		RWE
0046D000	00002000	UnPackMe	.ex_rsc	resources	Imag	R		RWE
00470000	00101000				Map	R		R

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

OllyDump - UnPackMe_eXPressor1.3.0.1Pk.exe

Start Address: Size:

Entry Point: -> Modify:

Base of Code: Base of Data:

☒ Fix Raw Size & Offset of Dump Image

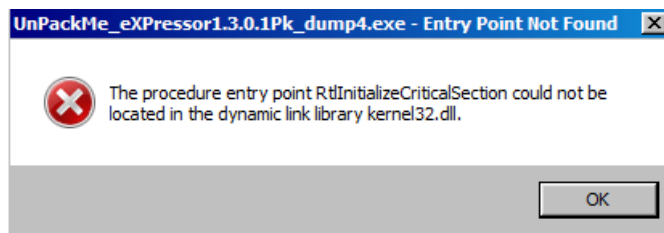
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.data	0006A000	00001000	0006A000	00001000	C0000020
.ex_cod	000017E4	0006B000	000017E4	0006B000	C0000040
.ex_rsc	00001FF8	0006D000	00001FF8	0006D000	40000040

☒ Rebuild Import

☒ Method1 : Search JMP[API] | CALL[API] in memory image

☐ Method2 : Search DLL & API name string in dumped file

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·소형화하고 단순한 보호를 겸함.
- 특징 : 새 섹션을 거의 만들지 않고, PE 헤더 빈 공간에 언패킹 스텝을 넣어 크기를 최소화. (일반적인 실행 파일은 언패킹 스텝을 코드 뒤에 따로 만들어서 포함시킨다.)
- > OEP를 찾기 위해서는 헤더에서 RET를 찾아주면된다.

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	AA	MOV 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	1??T我	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 스텝이라는 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 'h'
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.

CPU - main thread, module UnPackMe					
004271B0	.	55		PUSH	EBP
004271B1	.	8BEC		MOV	EBP,ESP
004271B3	.	6A FF		PUSH	-1
004271B5	.	68 600E4500		PUSH	UnPackMe.00450E60
004271BA	.	68 C8924200		PUSH	UnPackMe.004292C8
004271BF	.	64:A1 00000000		MOV	EAX,DWORD PTR FS:[0]
004271C5	.	50		PUSH	EAX
004271C6	.	64:8925 00000000		MOV	DWORD PTR FS:[0],ESP
004271CD	.	83C4 A8		ADD	ESP,-58
004271D0	.	53		PUSH	EBX
004271D1	.	56		PUSH	ESI
004271D2	.	57		PUSH	EDI
004271D3	.	8965 F8		MOV	DWORD PTR SS:[EBP-18],ESP

00400000	00001000	UnPackMe		PE header	Image	F
00401000	0006C000	UnPackMe	MEW	code	Image	F
0046D000	0003D000	UnPackMe	???	SFX, imports	Image	F
004B0000	00003000				Map	F
00570000	00003000				Map	F

여기가 OEP인걸 알 수 있고 이 부분에서 덤프를 떠준다.

OllyDump - UnPackMe_MEW1.1.exe

Start Address: 400000 Size: AA000 Dump

Entry Point: 9B339 -> Modify: 271B0 Get EIP as OEP Cancel

Base of Code: 1000 Base of Data: C

☒ Fix Raw Size & Offset of Dump Image

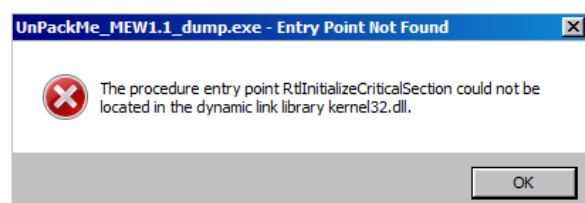
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
MEW	0006C000	00001000	0006C000	00001000	C00000E0
???	0003D000	0006D000	0003D000	0006D000	C00000E0

☒ Rebuild Import

☒ Method1 : Search JMP[API] | CALL[API] in memory image

☐ Method2 : Search DLL & API name string in dumped file

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.

0046D3A3	9C	PUSHFD
0046D3A4	60	PUSHAD
0046D3A5	E8 00000000	CALL UnPackMe.0046D3AA
0046D3AA	5D	POP EBP
0046D3AB	83ED 07	SUB EBP,7
0046D3AE	8D85 D9FCFFFF	LEA EAX,DWORD PTR SS:[EBP-327]
0046D3B4	8038 01	CMP BYTE PTR DS:[EAX],1

Registers (FP)	
EAX	7799EF7A
ECX	00000000
EDX	0046D3A3
EBX	7FFD3000
ESP	0012FF88
EBP	0012FF94
ESI	00000000
EDI	00000000
EIP	0046D3A4

'F8' 누르면 ESP가 바뀌는 것을 알 수 있다.

The screenshot shows a debugger's CPU window with the following registers and values:

EAX	7799EF7A	kernel32.BaseTh
ECX	00000000	
EDX	0046D3A3	ASCII "æ`è"
EBX	7FFD3000	
ESP	0012FF88	
EBP	0012FF94	
ESI	00000000	
EDI	00000000	
EIP	0046D3A4	

A context menu is open over the CPU window, showing options like 'Breakpoint', 'Search for', 'Go to', 'Hex', 'Text', 'Short', 'Long', 'Float', 'Disassemble', 'Special', 'Data Ripper', 'Memory Dump', and 'Appearance'. The 'Hex' option is selected, and the 'Word' option is highlighted in red.

BP를 걸어주고 'F9'을 눌러준다.

CPU - main thread, module UnPackMe		
0046D617	- E9 949BFBF	JMP UnPackMe.004271B0
0046D61C	8BB5 65FCFFFF	MOV ESI,DWORD PTR SS:[EBP-39B]
0046D622	0BF6	OR ESI,ESI
0046D624	0F84 97000000	JE UnPackMe.0046D6C1

해당 구간에서 멈추는 것을 확인할 수 있다.

004271B0	. 55	PUSH EBP
004271B1	. 8BEC	MOV EBP,ESP
004271B3	. 6A FF	PUSH -1
004271B5	. 68 600E4500	PUSH UnPackMe.00450E60
004271BA	. 68 C8924200	PUSH UnPackMe.004292C8
004271BF	. 64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	. 50	PUSH EAX
004271C6	. 64:8925 0000	MOV DWORD PTR FS:[0],ESP
004271CD	. 83C4 A8	ADD ESP,-58

00250000	0002C000				Priv	RW		RW
00400000	00001000	UnPackMe		PE header	Imag	R		RWE
00401000	0006C000	UnPackMe	.teddy0	code	Imag	R		RWE
0046D000	00028000	UnPackMe	.teddy1	SFX,data,imp	Imag	R		RWE
00495000	00008000	UnPackMe	.teddy2	resources	Imag	R		RWE
004A0000	00002000				Man	R		R

해당부분이 OEP인 걸 알 수 있어서 덤프를 떠준다.

OllyDump - UnPackMe_NsPack3.5.exe

Start Address: 400000 Size: 9D000 Dump

Entry Point: 6D3A3 -> Modify: 271B0 Get EIP as OEP Cancel

Base of Code: 1000 Base of Data: 6D000

☒ Fix Raw Size & Offset of Dump Image

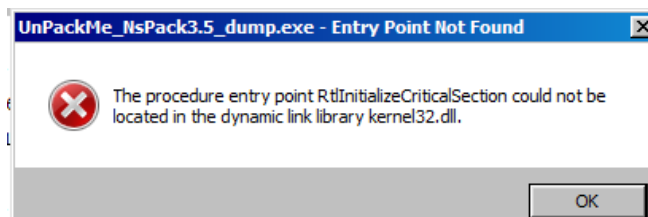
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.teddy0	0006C000	00001000	0006C000	00001000	E0000060
.teddy1	00027FCC	0006D000	00027FCC	0006D000	E0000060
.teddy2	000077DC	00095000	000077DC	00095000	E0000060

☒ Rebuild Import

Method1: Search JMP[API] | CALL[API] in memory image

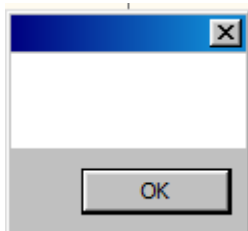
Method2: Search DLL & API name string in dumped file

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.

0046B5B8	50	PUSH EAX
0046B5B9	C3	RETN
0046B5BA	50	PUSH EAX
0046B5BB	56	PUSH ESI
0046B5BC	89EA	MOV EDI, EDI

우선 패킹되어 있기 때문에 OEP로 넘어가는 구간을 찾아줘야한다.

해당 부분에 BP를 걸어주고 'F8'로 계속 실행해준다.

0046BB13	09C0	OR EAX, EAX
0046BB15	0F84 B4050000	JE UnPackMe.0046C0CF
0046BB1B	FFD0	CALL EAX
0046BB1D	09C0	OR EAX, EAX
0046BB1F	0F84 AA050000	JE UnPackMe.0046C0CF
0046BB25	6A 00	PUSH 0

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

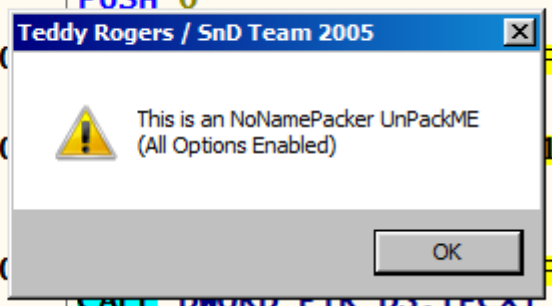
```

C 0  ES 0
P 0  CS 0
A 0  SS 0
Z 1  DS 0
S 0  FS 0
T 0  GS 0
D 0
O 0  Last

```

ZF=1로 바꿔주고 실행한다.

0046BB1F	0F84 AA050000	JE UnPackMe.0046C0CF
0046BB25	6A 00	PUSH 0
0046BB27	89EA	
0046BB29	81C2 F73546C0	
0046BB2F	52	
0046BB30	89EA	
0046BB32	81C2 153646C0	
0046BB38	52	
0046BB39	6A 00	
0046BB3B	89E9	
0046BB3D	81C1 FB4946C0	
0046BB43	FF11	

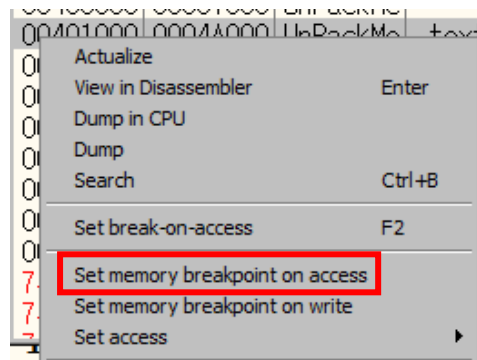


원래의 창이 나오는 걸 알 수 있다. 이제 언패킹을 해야하니 다시 돌아와서 'F8'로 계속 실행시켜준다.

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. 이것로는 찾을 수 없기 때문에 다른 방법을 찾아야한다.

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

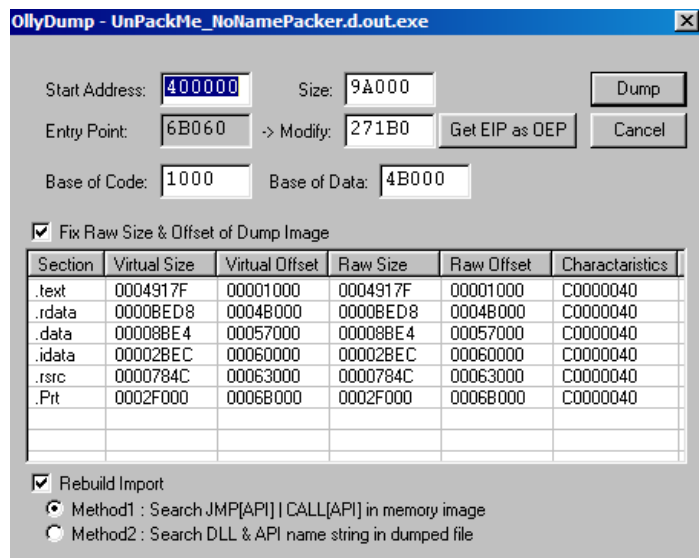
'F9'으로 실행해주면 해당 부분에서 멈추는 걸 알 수 있다.

```

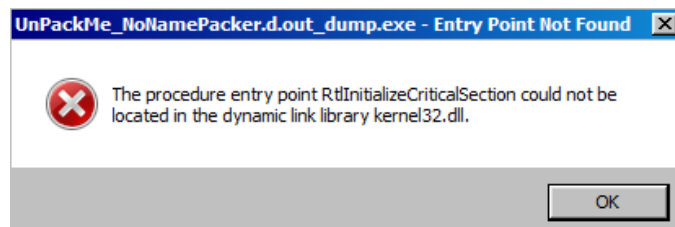
CPU - main thread, module UnPackMe
004271B0 . 55 PUSH EBP
004271B1 . 8BEC MOV EBP,ESP
004271B3 . 6A FF PUSH -1
004271B5 . 68 600E4500 PUSH UnPackMe.00450E60
004271BA . 68 C8924200 PUSH UnPackMe.004292C8
004271BF . 64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
004271C5 . 50 PUSH EAX
004271C6 . 64:8925 00000000 MOV DWORD PTR FS:[0],ESP

```

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

CPU - main thread, module UnPackMe			
00417012	. 55		PUSH EBP
00417013	. 3BC0		CMP EAX,EAX
00417015	. 74 02		JE SHORT UnPackMe.00417019
00417017	. 8183 533BC974		ADD DWORD PTR DS:[EBX+74C93B53],3B56BC01
00417021	. D27402 81		SAL BYTE PTR DS:[EDX+EAX-7F],CL
00417025	. 8557 E8		TEST DWORD PTR DS:[EDI-18],EDX

0x00417013에서 ESP가 변함으로 거기서 BP를 해준다.

Registers (FPU)
EAX 756BEF7A kernel32.
ECX 00000000
EDX 0041700C UnPackMe.
EBX 7FFD9000
ESP 0012FF88
EBP 0012FF88
ESI 00000000
EDI 00000000
EIP 00417013
C 0 ES 0
P 1 CS 0
A 0 SS 0
Z 1 DS 0
S 0 FS 0
T 0 GS 0

Increment Plus
Decrement Minus
Zero
Set to 1
Modify Enter
Copy selection to clipboard Ctrl+C
Copy all registers to clipboard
Follow in Dump
Follow in Stack

0041703F
00417040
00417046
0041704C
00417053
00417055
0041705B
00417061
00417063
00417065
00417066
0041706B
0041706D
0041706F
00417071
EAX=756BEF7A
Address
0012FF88
0012FF89

Backup
Copy
Binary
Breakpoint
Search for
Follow DWORD in Dump
Go to
Hex
Text
Short
Long
Float
Disassemble
Special
Data Ripper
Memory Dump
Appearance

DB 67
SUB EDX,DWORD PTR SS:
SUB EDX,2C
CMB BYTE PTR DS:[EBP
Me.00
PTR SS:
ADD EAX,51B
JMP EAX
CMP ECX,ECX
JE SHORT UnPackMe.00
DB BA
seThreadInitThunk)

BP를 걸어주고 'F9'을 해준다.

CPU - main thread, module UnPackMe			
0040A000	6D	DB 6D	
0040A001	AD	DB AD	
0040A002	B5	DB B5	
0040A003	6D	DB 6D	
0040A004	AD	DB AD	
0040A005	B4	DB B4	
0040A006	AB	DB AB	
0040A007	FF	DB FF	

00400000	00001000	UnPackMe		PE header	Imag	RWE	RWE
00401000	00001000	UnPackMe	.idata		Imag	RWE	RWE
00402000	00008000	UnPackMe	.rsrc	resources	Imag	RWE	RWE
0040A000	00002000	UnPackMe	.text	code	Imag	RWE	RWE
0040C000	00001000	UnPackMe	.data	code,data	Imag	RWE	RWE
0040D000	00009000	UnPackMe	.bss	code	Imag	RWE	RWE
00416000	00001000	UnPackMe	IMPORTS	code	Imag	RWE	RWE
00417000	00002000	UnPackMe	f	code,imports	Imag	RWE	RWE

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

OllyDump - UnPackMe_Exe32Pack1.42.exe

Start Address: 400000 Size: 19000 Dump

Entry Point: 1700C -> Modify: A000 Get EIP as OEP Cancel

Base of Code: A000 Base of Data: 1000

☒ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.idata	00001000	00001000	00001000	00001000	C0000080
.rsrc	00008000	00002000	00008000	00002000	400000C0
.text	00002000	0000A000	00002000	0000A000	C0000080
.data	00001000	0000C000	00001000	0000C000	C0000080
.bss	00009000	0000D000	00009000	0000D000	C0000080
IMPOR...	00001000	00016000	00001000	00016000	C0000080
.f	00001851	00017000	00001851	00017000	600000E0

☐ Rebuild Import

☒ Method1 : Search JMP[API] | CALL[API] in memory image

☐ Method2 : Search DLL & API name string in dumped file

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,,