

# 악성코드 분석 보고서

(sand-reversingwithlana-tutorials)

2025.09.07

## - Import Table의 구조

- Import Table = **IMAGE\_IMPORT\_DESCRIPTOR** 배열
- 각 디스크립터 = **DLL 하나**에 해당

(마지막은 전부 0인 **NULL 디스크립터**로 종료)

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {  
    DWORD OriginalFirstThunk; // INT(이름 테이블) RVA - 백업/참조용  
    DWORD TimeDateStamp;      // 보통 0  
    DWORD ForwarderChain;     // 보통 0  
    DWORD Name;               // "KERNEL32.DLL" 같은 DLL 이름 문자열 RVA  
    DWORD FirstThunk;         // IAT 시작 RVA (실행 후 실제 주소로 덮임)  
} IMAGE_IMPORT_DESCRIPTOR;
```

- **INT(Import Name Table)** = OriginalFirstThunk가 가리키는 **이름 목록**
  - 각 엔트리는 **IMAGE\_IMPORT\_BY\_NAME{Hint, Name[]}** 를 가리킴
  - **DWORD 0**로 목록 종료

## - IAT(Import Address Table)

- **FirstThunk가 가리키는 배열**
- 실행 전: 보통 이름/ordinal을 간접으로 가리킴("빈 자리")
- 실행 후: 로더가 **각 API의 실제 주소로 덮어써서 포인터 배열**이 됨
- DLL 별 그룹 사이, 그리고 마지막은 **DWORD 0**으로 구분/종료

예)

IAT: [KERNEL32!GetModuleHandleA]

[KERNEL32!ExitProcess]

00000000 ← KERNEL32 그룹 종료

[USER32!BeginPaint]

[USER32!MessageBoxA]

00000000 ← USER32 그룹 종료

## 코드와의 관계

- 프로그램 코드(.text)에서는
- CALL DWORD PTR [00493840]
- JMP DWORD PTR [00493848]

이런 식으로 이 테이블(IAT)의 슬롯을 간접 참조해서 API를 호출합니다.

- 이 테이블 자체는 .idata 섹션에 속하는 데이터 구조이지 코드가 아닙니다.

## - 점프 스텝(Thunk / Import Stub)

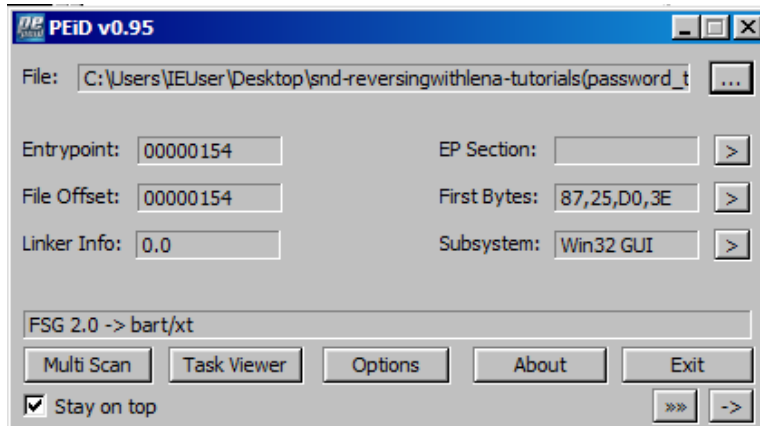
0040120C	\$-	FF25 0C324900	JMP	DWORD PTR DS:[49320C]
00401212		8BC0	MOV	EAX,EAX
00401214	\$-	FF25 08324900	JMP	DWORD PTR DS:[493208]
0040121A		8BC0	MOV	EAX,EAX
0040121C	\$-	FF25 04324900	JMP	DWORD PTR DS:[493204]
00401222		8BC0	MOV	EAX,EAX
00401224	\$-	FF25 00324900	JMP	DWORD PTR DS:[493200]
0040122A		8BC0	MOV	EAX,EAX
0040122C	\$-	FF25 FC314900	JMP	DWORD PTR DS:[4931FC]
00401232		8BC0	MOV	EAX,EAX
00401234	.-	FF25 F8314900	JMP	DWORD PTR DS:[4931F8]
0040123A		8BC0	MOV	EAX,EAX
0040123C	\$-	FF25 F4314900	JMP	DWORD PTR DS:[4931F4]
00401242		8BC0	MOV	EAX,EAX
00401244	.-	FF25 F0314900	JMP	DWORD PTR DS:[4931F0]
0040124A		8BC0	MOV	EAX,EAX
0040124C	\$-	FF25 EC314900	JMP	DWORD PTR DS:[4931EC]
00401252		8BC0	MOV	EAX,EAX
00401254	\$-	FF25 E8314900	JMP	DWORD PTR DS:[4931E8]
0040125A		8BC0	MOV	EAX,EAX
0040125C	\$-	FF25 E4314900	JMP	DWORD PTR DS:[4931E4]
00401262		8BC0	MOV	EAX,EAX
00401264	\$-	FF25 E0314900	JMP	DWORD PTR DS:[4931E0]
0040126A		8BC0	MOV	EAX,EAX

Import Table을 코드에서 참조하기 위해 만들어 둔 "점프 스텝(Thunk / Import Stub)" 들입니다.

즉, IAT를 이용해 API로 점프하는 코드 영역이에요.

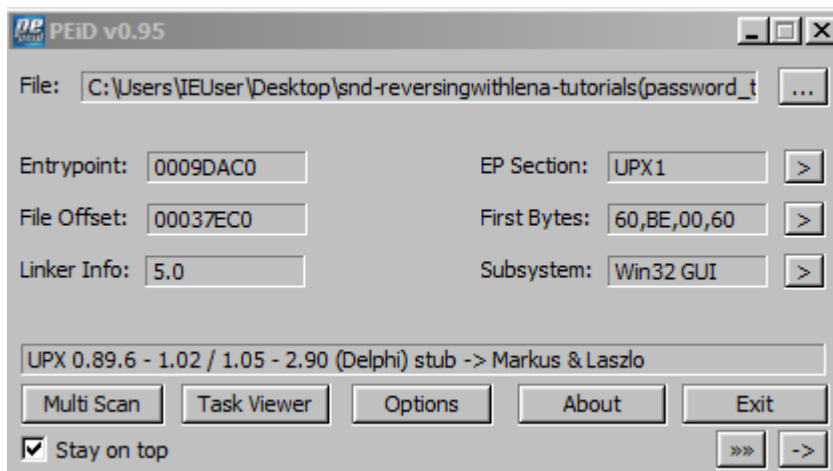
## 1. 문제

### 1-1) UnPackMe\_FSG2.0.exe



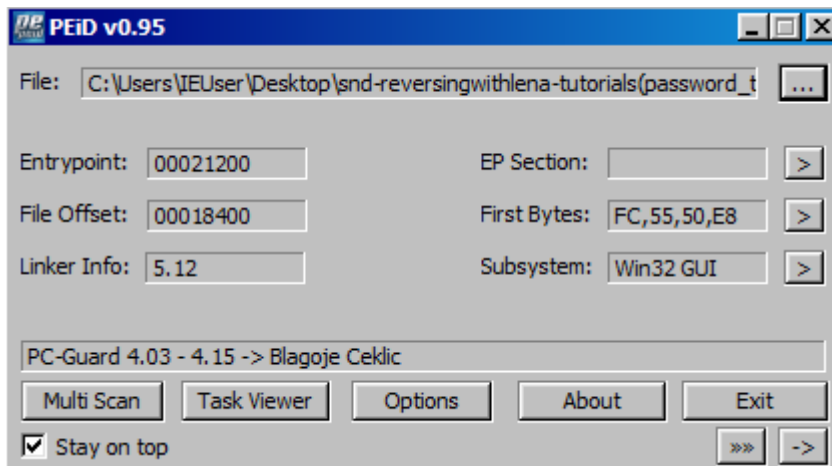
FSG로 패킹한 걸 알 수 있다.

### 1-2) UnPackMe\_UPX.exe



UPX로 패킹한 걸 알 수 있다.

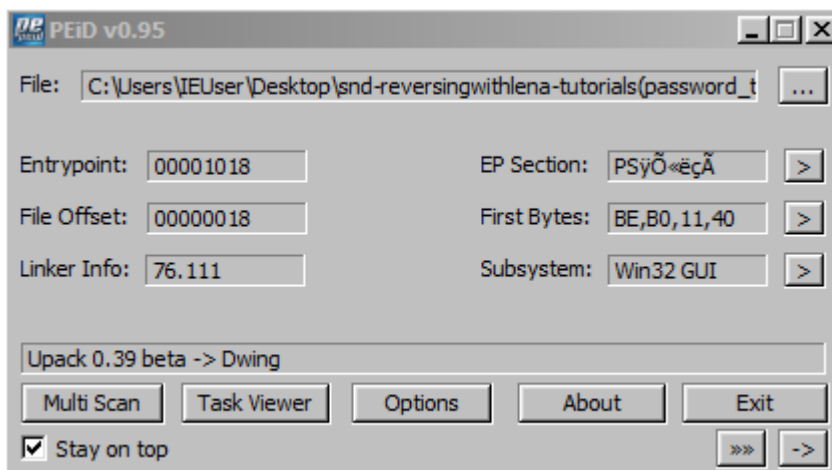
### 1-3) PCGuard4.06C\_UnpackmeAll.exe



) PCGuard4.0로 패킹한 걸 알 수 있다.

-> 해결 실패

1-4) UnPackMe\_WinUpack0.39.exe



WinUpack0.39로 패킹한 걸 알 수 있다.

## 2. 해결 방법

### 2-1) UnPackMe\_FSG2.0.exe

※ FSG 패커

- JMP 코드를 연속으로 3개 사용한다.

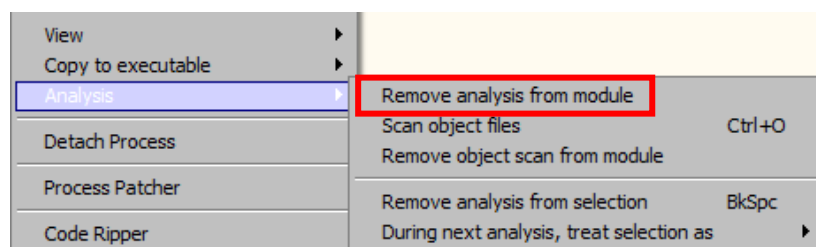
004001CD	^ 78 F3	JS SHORT UnPackMe.004001C2
004001CF	∨ 75 03	JNZ SHORT UnPackMe.004001D4
004001D1	FF63 0C	JMP DWORD PTR DS:[EBX+C]
004001D4	50	PUSH EAX
004001D5	55	PUSH EBP

이렇게 연속으로 사용하고 다음이 OEP인걸 알 수 있다.

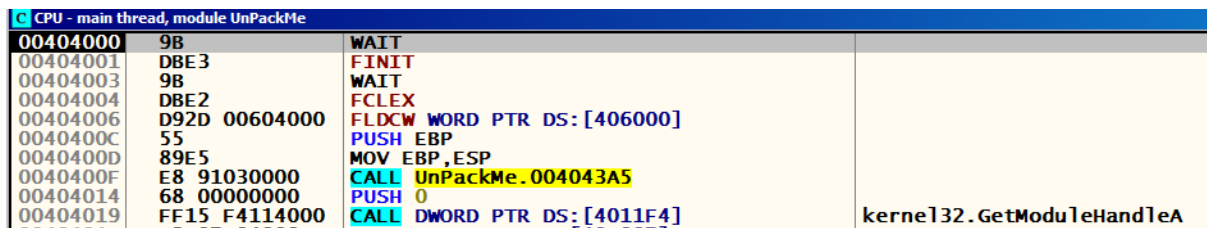
004001CA	8B07	MOV EAX,DWORD PTR DS:[EDI]
004001CC	40	INC EAX
004001CD	^ 78 F3	JS SHORT UnPackMe.004001C2
004001CF	∨ 75 03	JNZ SHORT UnPackMe.004001D4
004001D1	FF63 0C	JMP DWORD PTR DS:[EBX+C]
004001D4	50	PUSH EAX
004001D5	55	PUSH EBP

밑으로 내리다 보면 JMP를 연속으로 3개 사용한 부분을 찾을 수 있다. 그 부분에 BP를 걸고 'F9'으로 실행한다.

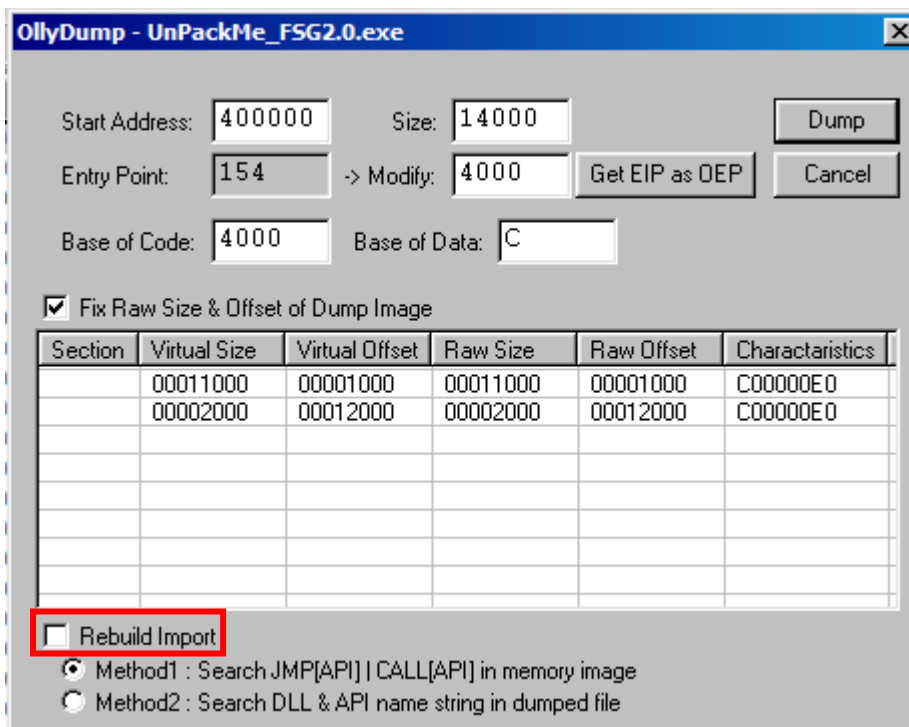
CPU - main thread, module UnPackMe			
00404000	9B	DB 9B	
00404001	DB	DB DB	
00404002	E3	DB E3	
00404003	9B	DB 9B	
00404004	DB	DB DB	
00404005	E2	DB E2	
00404006	D9	DB D9	
00404007	2D	DB 2D	
00404008	00	DB 00	
00404009	60	DB 60	



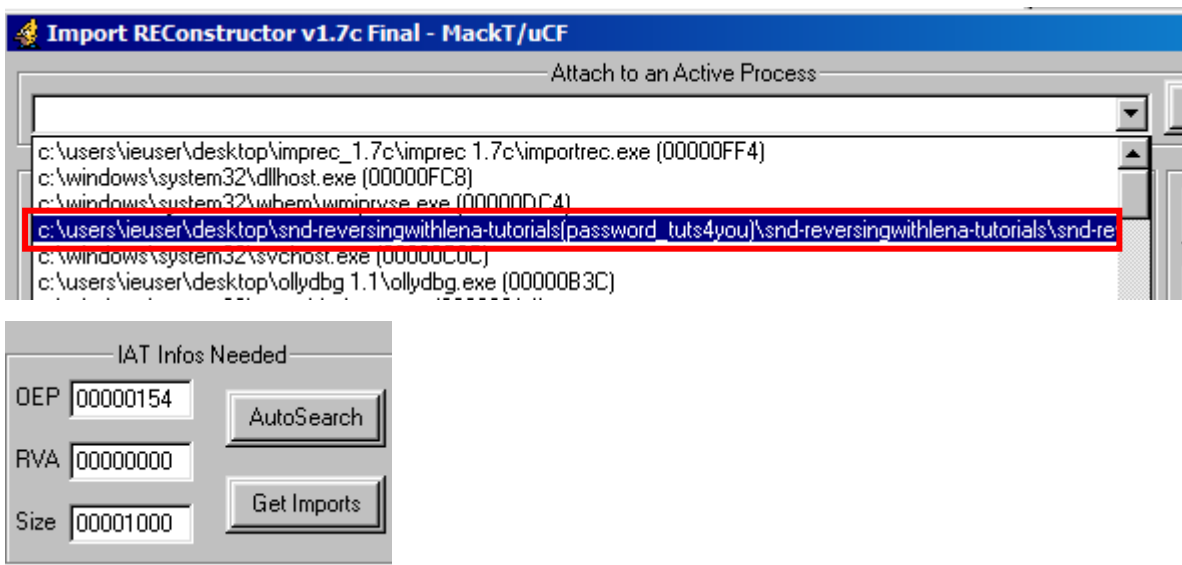
그러면 이 부분이 나오는 걸 알 수 있는데 Remove analysis from module을 누르면 변하는 걸 알 수 있다.



0x00404000 이 부분이 OEP인 걸 알 수 있다.



해당 부분에서 덤프를 해주는데 Rebuild Import의 체크를 풀어주고 저장한다.



ImpRec를 이용할 건데 해당 파일을 클릭하면 IAT정보가 나오는데 아까 OEP는 4000이었으므로 바꿔준다.

IAT Infos Needed

DEP 00004000 AutoSearch

RVA 00000000

Size 00001000 Get Imports

AutoSearch는 IAT위치를 자동으로 찾아주는 버튼인데 눌러서 찾아준다.

IAT Infos Needed

DEP 00004000 AutoSearch

RVA 000011E8

Size 0000002C Get Imports

이렇게 바뀌는 걸 알 수 있다. 올리디버그를 이용해서 해당 위치가 정확한지 확인을 해 본다.

Address	Hex dump	ASCII
004011C8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004011D8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004011E8	00 00 00 00 52 BE 38 76 30 C6 37 76 13 DB 37 76	....R%8v0A7v!07v
004011F8	7D A3 37 76 95 A3 37 76 EE 29 45 77 C3 81 36 76	}f7v•f7v!)EwA6v
00401208	94 F1 37 76 E5 2D 37 76 E0 C5 37 76 81 A8 37 76	"ñ7vâ-7vâA7v"7v
00401218	FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00	yyyy.....
00401228	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

해당 FSG2.0 파일은 총 2개의 dll을 쓰는데 밑에 내려봐도 다음 dll을 찾아 볼 수 없다. 그럼 해당 4011e8에 위치한 dll은 마지막 dll인 걸 알 수 있다.

Address	Hex dump	ASCII
00401168	33 32 2E 44 4C 4C 00 4B 45 52 4E 45 4C 33 32 2E	32.DLL.KERNEL32.
00401178	44 4C 4C 00 00 00 00 00 00 00 00 00 00 00 00 00	DLL.....
00401188	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00401198	00 00 00 00 A9 AB B4 75 71 01 B5 75 5C 55 B5 75	....0< uq µu\0µu
004011A8	99 EA B9 75 DC 6C B7 75 C8 80 B4 75 C1 AD B4 75	■è'u0l·uE€ uA- u
004011B8	FF FF FF 7F 00 00 00 00 00 00 00 00 00 00 00 00	yyy.....
004011C8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004011D8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004011E8	00 00 00 00 52 BE 38 76 30 C6 37 76 13 DB 37 76	....R%8v0A7v!07v
004011F8	7D A3 37 76 95 A3 37 76 EE 29 45 77 C3 81 36 76	}f7v•f7v!)EwA6v
00401208	94 F1 37 76 E5 2D 37 76 E0 C5 37 76 81 A8 37 76	"ñ7vâ-7vâA7v"7v
00401218	FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00	yyyy.....

위에 올려보면 진짜 IAT 주소를 알 수 있다.

(IAT는 dll의 구분자로 00000000 로 세팅하고 종료를 나타내기 위해 또 다른 00000000 으로 나타낸다. -> 원래는 중간에 00000000 하나만 사용하는데 FSG는 쓰레기 값을 넣어서 헛갈리게 한다. 그러니 주변을 잘 살펴봐야한다.)

\* 안전 마진 때문에 처음 00000000을 포함해서 RVA로 잡았는데 00000000포함하지 않고 0x004011AC부터 해도 상관없음.



IAT Infos Needed

OEP

RVA

Size

시작 주소를 바꿔주고 Get Imports를 눌러준다.

**Import REConstructor v1.7c Final - MackT/uCF**

Attach to an Active Process

Imported Functions Found

☒ ? FTunk:0000119C NbFunc:8 (decimal:8) valid:NO  
☒ ? FTunk:000011EC NbFunc:C (decimal:12) valid:NO

IAT Infos Needed

OEP

RVA

Size

New Import Infos

((ID+ASCII+LOADER))  
 RVA  Size   
☒ Add New Section

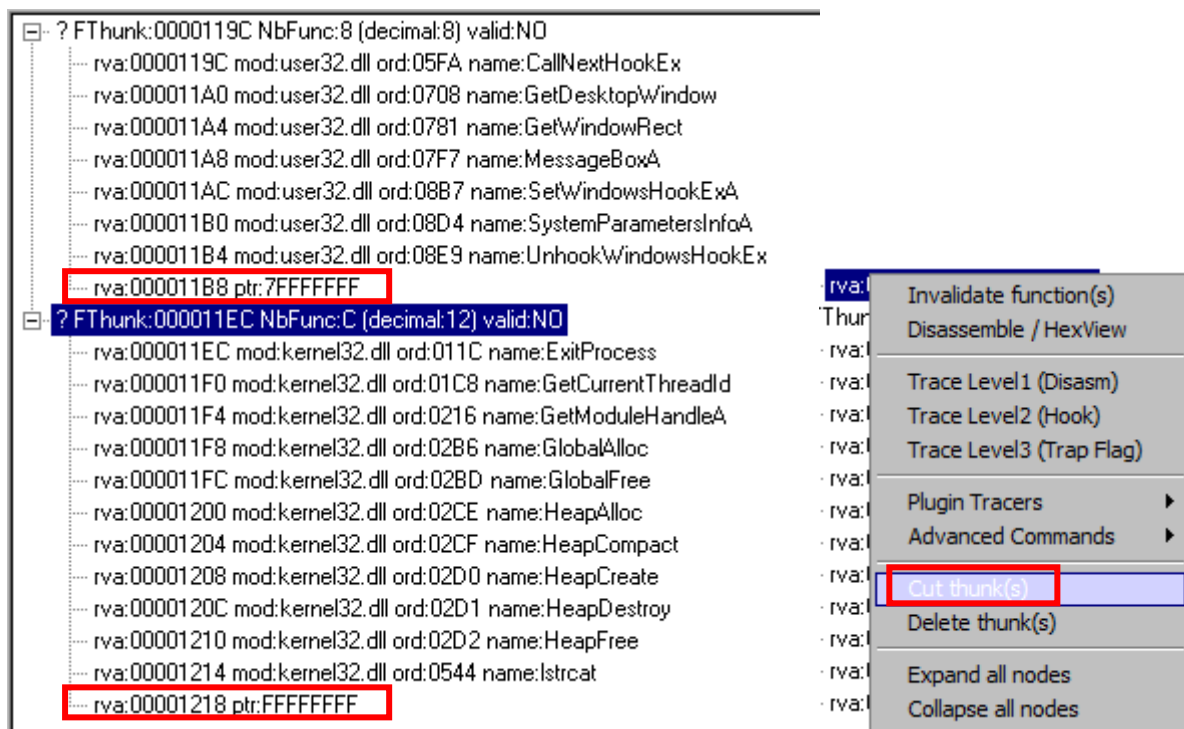
Log

Module loaded: c:\windows\system32\imm32.dll  
 Module loaded: c:\windows\system32\msctf.dll  
 Getting associated modules done.  
 Image Base:00400000 Size:00014000  
 Original IAT RVA found at: 00001200 in Section RVA: 00001000 Size:00011000  
 IAT read successfully.  
 rva:00001200 forwarded from mod:ntdll.dll ord:02BD name:RtlAllocateHeap  
 -----  
 Current imports:  
 0 (decimal:0) valid module(s)  
 14 (decimal:20) imported function(s) (added: +14 (decimal:20))  
 2 (decimal:2) unresolved pointer(s) (added: +2 (decimal:2))

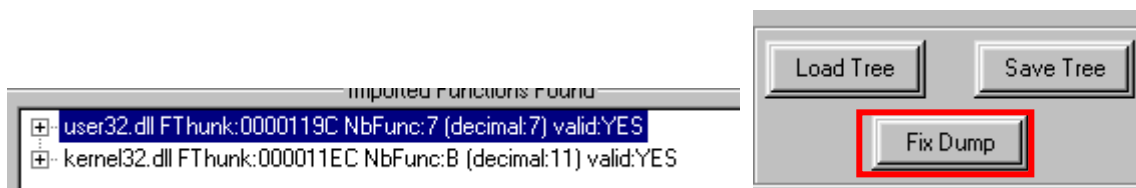
If you wish to help continue developing and bug fixing ImpREC please visit our forum at Tuts 4 You:  
<http://www.tuts4you.com>

2008.03.10

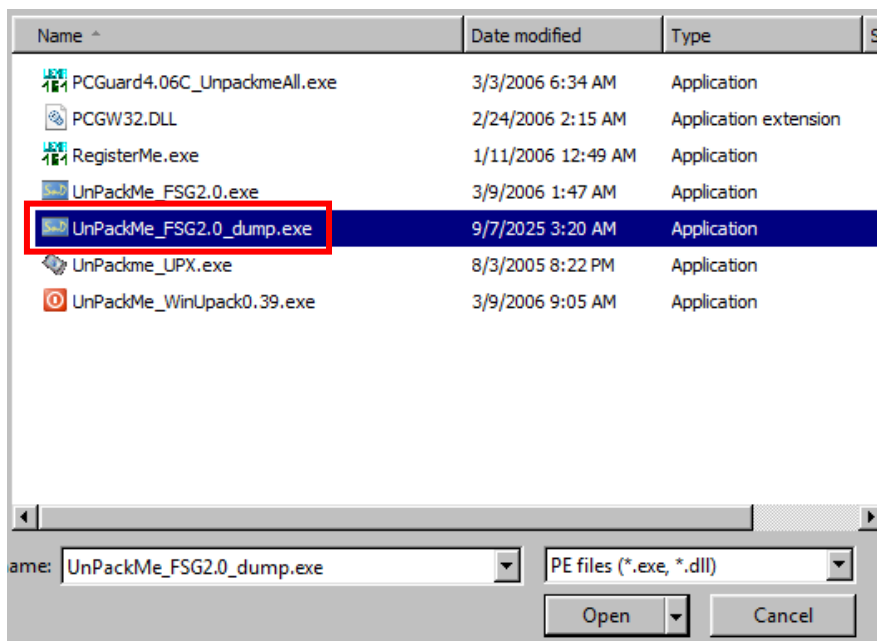
2군데가 잘못되었다고 나오고 Valid:NO로 나오는 걸 볼 수 있다.






이렇게 2개가 잘못되었기 때문에 그렇다. 2개를 Cut thunk를 이용하여 없애준다.



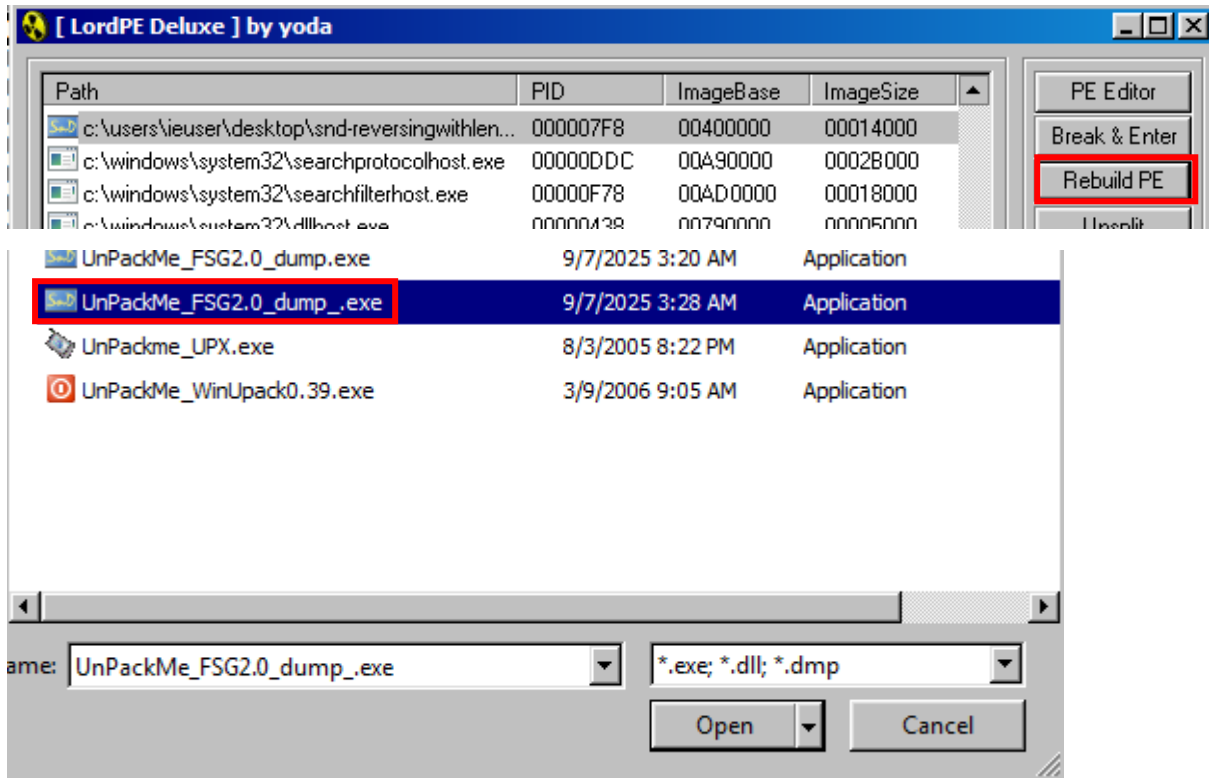
이렇게 바뀌는 걸 확인 할 수 있다. Fix Dump를 이용해서 저장해준다.



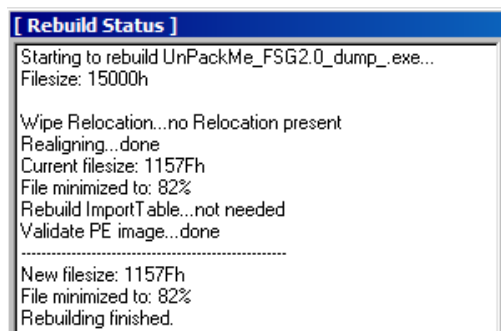
이전에 덤프한 걸 선택하여 해당 파일의 임포트 부분을 수정해준다.

 UnPackMe_FSG2.0.exe	3/9/2006 1:47 AM	Application	9 KB
 UnPackMe_FSG2.0_dump.exe	9/7/2025 3:20 AM	Application	80 KB
 UnPackMe_FSG2.0_dump_.exe	9/7/2025 3:28 AM	Application	84 KB

수정된 파일은 위에서 선택한 파일명 + "\_"로 저장된다.



임포트 부분도 수정했으니까 PE헤더 부분도 리빌드 해줘야한다.



완료된 걸 확인 할 수 있고 올리디버그로 열어본다.

CPU - main thread, module UnPackMe						
00404000	\$ 9B					WAIT
00404001	. DBE3					FINIT
00404003	. 9B					WAIT
00404004	. DBE2					FCLEX
00404006	. D92D 00604000					FLDCW WORD PTR DS:[406000]
0040400C	. 55					PUSH EBP
0040400D	. 89E5					MOV EBP,ESP
0040400F	. E8 91030000					CALL UnPackMe.004043A5
00404014	. 68 00000000					PUSH 0
00404019	. FF15 F4114000					CALL DWORD PTR DS:[<&kernel32.GetModule
0040401E	. A3 07E04000					MOV DWORD PTR DS:[40E007] EAX

00210000	00003000				Priv	RW	RW
002A0000	00006000				Priv	RW	RW
00400000	00001000	UnPackMe		PE header	Imag	R	RWE
00401000	00011000	UnPackMe			Imag	R	RWE
00412000	00002000	UnPackMe		resources	Imag	R	RWE
00414000	00001000	UnPackMe	.mact	imports	Imag	R	RWE
00420000	00004000				Map	R	R

OEP 제대로 찾은 걸 볼 수 있고 추가한 임포트가 제대로 저장된 걸 확인할 수 있다.

## 2-2) UnPackMe\_UPX.exe

※ UPX 패커

- 언패킹하기 쉬운 대표적인 패커
- 언패킹 스텝 코드 끝에서 **JMP OEP**로 원래 프로그램 진입.

↳ 마지막 JMP에 BP 걸면 바로 OEP 확인 가능.

- UPX0 = 압축된 원래의 원본 코드 (여기에 OEP있음) , UPX1 = 스텝 코드

CPU - main thread, module UnPackme						
0049DAC0	\$ 60					PUSHAD
0049DAC1	. BE 00604600					MOV ESI,UnPackme.00466000
0049DAC6	. 8DBE 00B0F9F1					LEA EDI,DWORD PTR DS:[ESI+FFF9B000]
0049DACC	. C787 18B70700					MOV DWORD PTR DS:[EDI+7B718],29023006
0049DAD6	. 57					PUSH EDI
0049DAD7	. 83CD FF					OR EBP,FFFFFFFF
0049DADA	. EB 0E					JMP SHORT UnPackme.0049DAEA
0049DADC	. 90					NOP
0049DADD	. 90					NOP
0049DADE	. 90					NOP
0049DADE	. 90					NOP
0049DAE0	> 8A06					MOV AL,BYTE PTR DS:[ESI]
0049DAE2	. 46					INC ESI
0049DAE3	. 8807					MOV BYTE PTR DS:[EDI],AL
0049DAE5	. 47					INC EDI

UPX패커는 마지막 JMP에 BP를 걸면 바로 OEP 확인이 가능하기 때문에 내려가서 찾아 본다.

0049DC32	.	8903	MOV DWORD PTR DS:[EBX],EAX
0049DC34	.	83C3 04	ADD EBX,4
0049DC37	.	EB D8	JMP SHORT UnPackme.0049DC11
0049DC39	>	FF96 08DB0900	CALL DWORD PTR DS:[ESI+9DB08]
0049DC3F	>	61	POPAD
0049DC40	-	E9 BB33F6FF	JMP UnPackme.00401000

밑으로 내려가면 JMP 명령어로 넘어가는 걸 알 수 있다.

CPU - main thread, module UnPackme						
00401000	EB 10	JMP SHORT UnPackme.00401012				
00401002	66:623A	BOUND DI,DWORD PTR DS:[EDX]				
00401005	43	INC EBX				
00401006	2B2B	SUB EBP,DWORD PTR DS:[EBX]				
00401008	48	DEC EAX				
00401009	4F	DEC EDI				
0040100A	4F	DEC EDI				
0040100B	4B	DEC EBX				
0040100C	90	NOP				
0040100D	- E9 E8334700	JMP 008743FA				
00401012	A1 DB334700	MOV EAX,DWORD PTR DS:[4733DB]				
00401017	C1E0 02	SHL EAX,2				
0040101A	A3 DF334700	MOV DWORD PTR DS:[4733DF],EAX				

00400000	00001000	UnPackme		PE header	Imag	R	RWE
00401000	00065000	UnPackme	UPX0		Imag	R	RWE
00466000	00038000	UnPackme	UPX1	code,exports	Imag	R	RWE
0049E000	00001000	UnPackme	.rsrc	data,imports	Imag	R	RWE
004A0000	00101000			Man	R		R

0x00401000이 OEP인 걸 알 수 있다.

OllyDump - UnPackme\_UPX.exe

Start Address: 400000

Size: 9F000

Dump

Entry Point: 9D&C0

-> Modify: 1000

Get EIP as OEP

Cancel

Base of Code: 66000

Base of Data: 9E000

☒ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
UPX0	00065000	00001000	00065000	00001000	E0000080
UPX1	00038000	00066000	00038000	00066000	E0000040
.rsrc	00001000	0009E000	00001000	0009E000	C0000040

☐ Rebuild Import

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

해당 부분에서 덤프를 생성한다.

IAT Infos Needed

OEP

RVA

Size

ImpRec를 이용해서 OEP를 1000으로 바꿔주고 AutoSearch를 눌러 IAT를 자동으로 찾을 것이다.

IAT Infos Needed

OEP

RVA

Size

0x00485278 주소가 나오는데 올리디버그로 확인해보면 맞지 않은 걸 알 수 있다.

00485268	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....~ëöu.  öu° öu
00485278	00 00 00 00	88 EA F6 75	0C 19 F6 75	BA 18 F6 75	.....	.....	.....	.....
00485288	81 EC F6 75	E2 DE F6 75	41 99 6F 77	DA 45 F6 75	.....	.....	.....	.....
00485298	10 73 6E 77	EA 7C F8 75	40 25 FB 75	52 BE F7 75	.....	.....	.....	.....
004852A8	4C 4E F7 75	AB C1 F6 75	CD A6 F6 75	90 A2 F8 75	.....	.....	.....	.....
004852B8	87 F1 F6 75	C1 F9 F5 75	2B DB F6 75	CF 90 F7 75	.....	.....	.....	.....
004852C8	FC 27 FB 75	FF 90 F7 75	D5 D9 F6 75	30 C6 F6 75	.....	.....	.....	.....

이 부분이 나오는데 위에 올려서 처음 DLL이 맞는지 확인해본다.

004850C8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
004850D8	ED 45 60 77	5B 48 60 77	43 48 60 77	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
004850E8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
004850F8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00485108	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....
00485118	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....

위에 올려보면 0x004850D8부터가 IAT의 시작인 걸 알 수 있다. (이번에는 안전마진 미포함해서 설정)

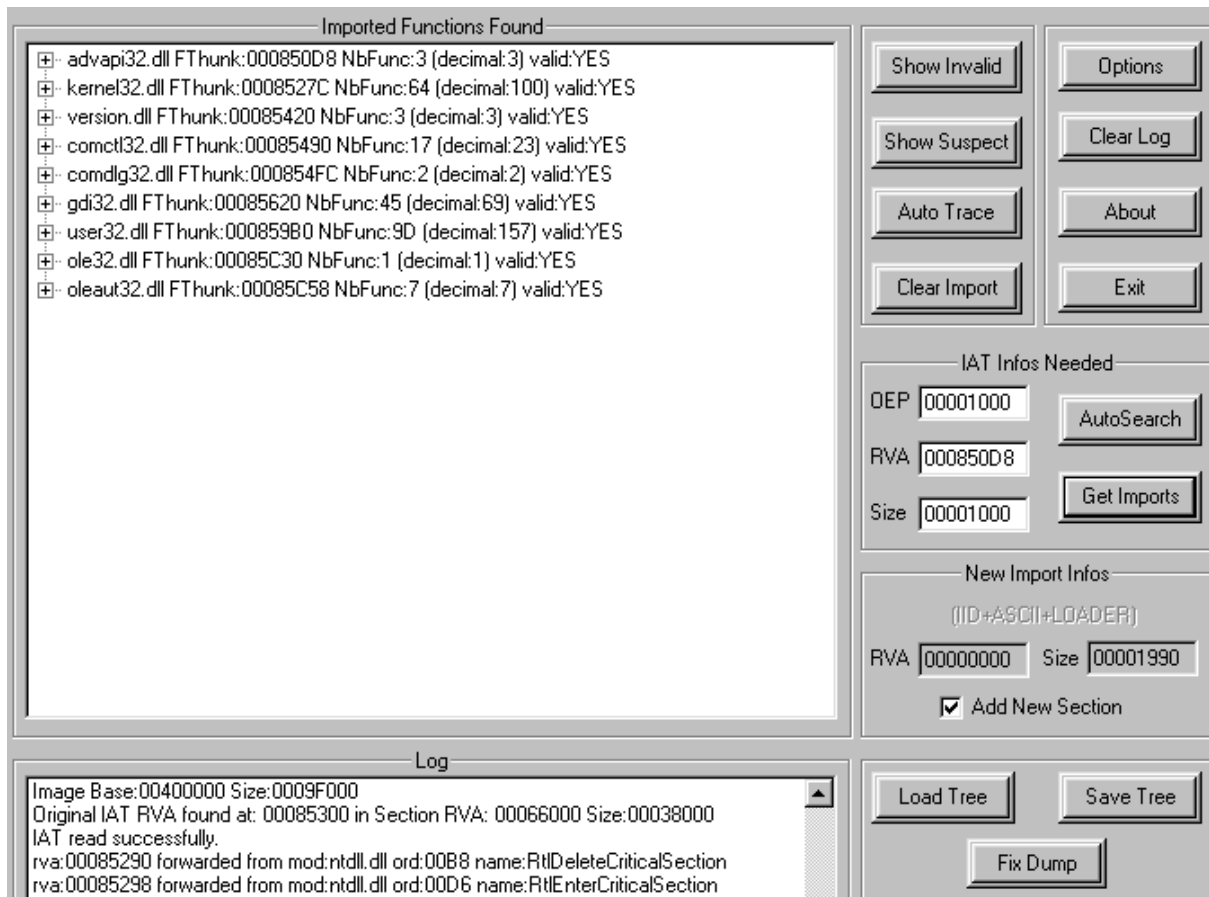
IAT Infos Needed

OEP

RVA

Size

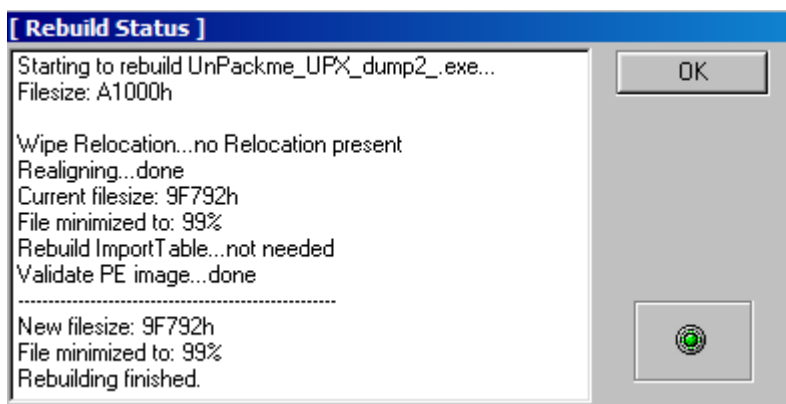
RVA와 Size를 고쳐주고 Import를 얻는다.



전부 Valid:YES인 걸 확인할 수 있다. Fix Dump를 눌러서 import를 생성해준다.

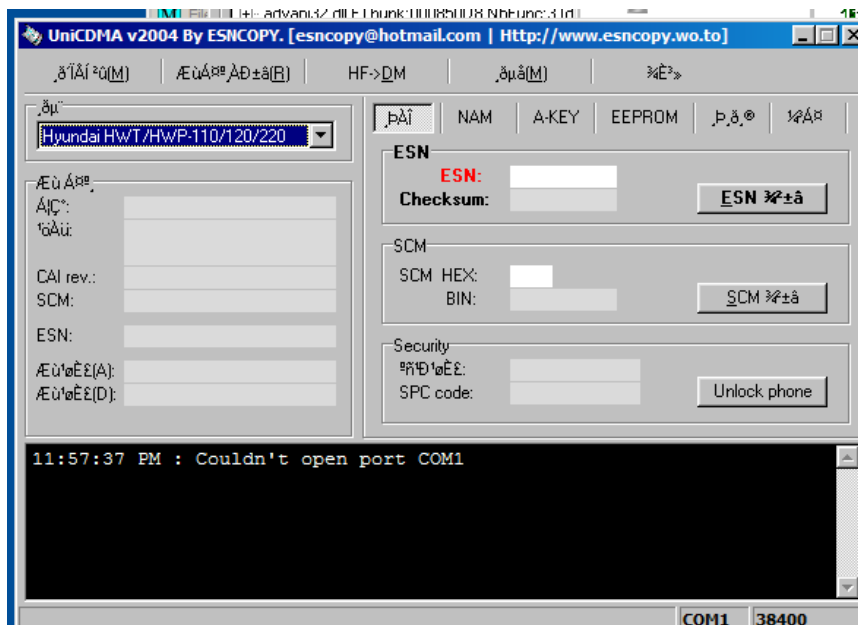
UnPackme_UPX_dump1.exe	9/10/2025 10:13 PM	Application
UnPackme_UPX_dump2.exe	9/10/2025 11:02 PM	Application
UnPackme_UPX_dump2_.exe	9/10/2025 11:55 PM	Application
UnPackMe_WinUpack01.39.exe	3/9/2006 9:05 AM	Application

LordPE를 이용해서 리빌드를 해준다.



리빌드 해준 후 올리디버그로 열어본다.

00400000	00001000	UnPackme		PE header	Imag	R	RWE
00401000	00065000	UnPackme	UPX0		Imag	R	RWE
00466000	00038000	UnPackme	UPX1	code, exports	Imag	R	RWE
0049E000	00001000	UnPackme	.rsrc	data, resource	Imag	R	RWE
0049F000	00002000	UnPackme	.mact	imports	Imag	R	RWE
004B0000	00101000				Mem	D	D



Imports 생성된 걸 확인할 수 있고 잘 실행되는 걸 확인 할 수 있다.

### 2-3) PCGuard4.06C\_UnpackmeAll.exe

#### ※ PCGuard 패커

- PC Guard 특징: DLL 기반 보호 + OEP 은폐 기법.
- 분석 난이도 포인트: OEP를 여러 번 바꿔가며 숨김 → 디버거로 OEP 잡기가 어려움.
- 우회법: ESP 기반 OEP 트릭, 또는 반복되는 JMP 패턴 추적.

-> 중간에 BP가 걸려야하는데 걸리지 않는다. 레나를 요약해보면

PC Guard 패커에는 바이너리의 실행 회수를 제한할 수 있는 옵션이 있다. 분석하고자 하는 바이너리는 해당 옵션이 체크된 상태로 패킹 된 것이기 때문에 완벽한 언패킹이 일어나기 전까지 자꾸 실행하면 안된다.

PC Guard 패커는 패킹/언패킹을 위하여 DLL를 사용한다. 따라서 해당 바이너리와 같은 폴더 내에 dll 파일이 없으면 실행되지 않는다. 그리고 PC Guard 패커는 OEP를 가지고



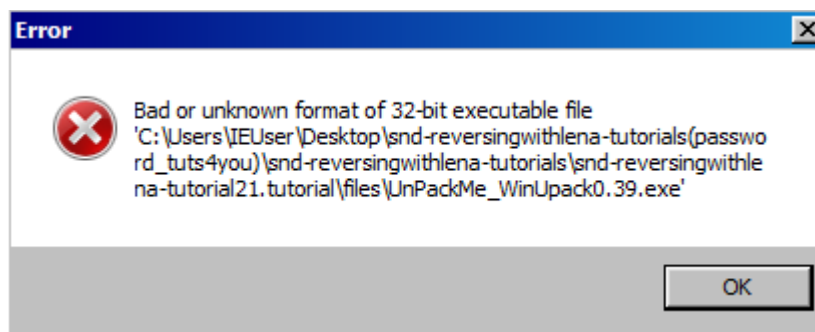
놈으로써 언패킹 루틴 중에 OEP를 숨길려고 한다. 분석을 하다 보면 실제 프로그램이 실행되기 전에 약 20번 정도 "break"가 발생한다.

근데 난 OEP에 도달하지 않고 계속 멈춘다.. 실패,,

## 2-4) UnPackMe\_WinUpack0.39.exe

※ WinUpack 패커

- Windows 32비트 PE(EXE/DLL) 대상
- UPX처럼 런타임에 압축을 풀고 실행하는 구조 → 분석을 어렵게 만드는 보호 기능으로도 사용 가능.
- 스텝이 단순해 언패킹 난이도는 낮은 편 → OllyDbg + ESP/OEP 브레이크로 쉽게 풀림.



-> 해석 : 디버거(또는 PE 분석 툴)가 이 실행 파일을 정상적인 32비트 PE(Portable Executable) 로 인식하지 못한다.

WinUpack 같은 패커는 실행파일을 압축/암호화하면서 PE 헤더나 섹션 구조를 일반적인 형태와 다르게 만들기 때문에 이런 에러가 발생한다. 하지만 무시하고 올리디버그로 실행해본다.

CPU - main thread, module ntdll		
779805DA	8975 FC	MOV DWORD PTR SS:[EBP-4],ESI
779805DD	EB 0E	JMP SHORT ntdll.779805ED
779805DF	33C0	XOR EAX,EAX
779805E1	40	INC EAX
779805E2	C3	RETN
779805E3	8B65 E8	MOV ESP,DWORD PTR SS:[EBP-18]
779805E6	C745 FC FFFFFFFF	MOV DWORD PTR SS:[EBP-4],-2
779805ED	E8 7722FBFF	CALL ntdll.77932869
779805F2	C3	RETN
779805F3	90	NOP
779805F4	90	NOP

처음 실행해보면 실행파일의 코드가 아니라 dll파일이 처음 뜨는 걸 볼 수 있다.

002A0000	00067000				Map	R	R
00400000	000A3000	UnPackMe		PE header	Imag	R	RWE
75820000	00001000	KERNELBA		PE header	Imag	R	RWE
75821000	00044000	KERNELBA	.text	code, imports	Imag	R	RWE
75865000	00002000	KERNELBA	.data	data	Imag	R	RWE
75867000	00001000	KERNELBA	.rsrc	resources	Imag	R	RWE
75868000	00003000	KERNELBA	.reloc	relocations	Imag	R	RWE

UnPackMe가 헤더만 있는 걸 볼 수 있다. WinUpack이 일반적인 형태와 다르게 만들기 때문이다. 하지만 헤더의 크기가 큰 걸 보면 여기에 코드랑 다 들어있는 걸 볼 수 있다.

0040002A	4C	DB 4C	MajorLinkerVersion = 4C (76.)
0040002B	6F	DB 6F	MinorLinkerVersion = 6F (111.)
0040002C	61644C69	DD 694C6461	SizeOfCode = 694C6461 (1766614113.)
00400030	62726172	DD 72617262	SizeOfInitializedData = 72617262 (1918988898.)
00400034	79410000	DD 00004179	SizeOfUninitializedData = 4179 (16761.)
00400038	18100000	DD 00001018	AddressOfEntryPoint = 1018
0040003C	10000000	DD 00000010	Offset to PE signature
00400040	00800400	DD 0004B000	BaseOfData = 4B000
00400044	00004000	DD 00400000	ImageBase = 400000
00400048	00100000	DD 00001000	SectionAlignment = 1000
0040004C	00020000	DD 00000200	FileAlignment = 200

EP가 0x401018인 걸 알 수 있다.

CPU - main thread, module UnPackMe			
00401018	BE B0114000	MOV ESI, UnPackMe.004011B0	
0040101D	AD	LODS DWORD PTR DS:[ESI]	
0040101E	50	PUSH EAX	
0040101F	FF76 34	PUSH DWORD PTR DS:[ESI+34]	
00401022	EB 7C	JMP SHORT UnPackMe.004010A0	
00401024	48	DEC EAX	
00401025	010F	ADD DWORD PTR DS:[EDI], ECX	
00401027	010B	ADD DWORD PTR DS:[EBX], ECX	
00401029	014C6F 61	ADD DWORD PTR DS:[EDI+EBP*2+61], ECX	
0040102D	64 4C	DEC ESP	
0040102F	6962 72 617279	IMUL ESP, DWORD PTR DS:[EDX+72], 41797261	
00401036	0000	ADD BYTE PTR DS:[EAX], AL	

BP를 걸어주고 'F9'을 이용해서 여기로 온다.

CPU - main thread, module UnPackMe			
00401017	00BE B0114000	ADD BYTE PTR DS:[ESI+4011B0], BH	
0040101D	AD	LODS DWORD PTR DS:[ESI]	
0040101E	50	PUSH EAX	
0040101F	FF76 34	PUSH DWORD PTR DS:[ESI+34]	
00401022	EB 7C	JMP SHORT UnPackMe.004010A0	
00401024	48	DEC EAX	
00401025	010F	ADD DWORD PTR DS:[EDI], ECX	
00401027	010B	ADD DWORD PTR DS:[EBX], ECX	
00401029	014C6F 61	ADD DWORD PTR DS:[EDI+EBP*2+61], ECX	
0040102D	64 4C	DEC ESP	

Registers (FPU)	
EAX	004271B0 UnPackMe.004271B0
ECX	00000000
EDX	00401018 UnPackMe.00401018
EBX	7F610000
ESP	0012FF88 ASCII ""qB"
EBP	0012FF94
ESI	004011B4 UnPackMe.004011B4
EDI	00000000

ESP가 바뀌는 걸 이용해서 OEP를 찾을거다. 현재 부분에서 바뀌는 걸 볼 수 있다.

Address	Hex dump
0012FF88	B0 71 42 00 8C EF 2B 76 00 40 FD 7F D4 FF 12 00
0012FF98	7A 36 94 77 00 40 FD 7F B7 B8 93 77 00 00 00 00
0012FFA8	00 00 00 00 00 40 FD 7F 00 00 00 00 00 00 00 00
0012FFB8	00 00 00 00 A0 FF 12 00 00 00 00 00 FF FF FF FF
0012FFC8	95 E1 8F 77 8B 49 12 00 00 00 00 00 EC FF 12 00
0012FFD8	4D 36 94 77 18 10 40 00 00 40 FD 7F 00 00 00 00
0012FFE8	00 00 00 00 00 00 00 00 00 00 00 00 18 10 40 00

해당 부분에 BP를 걸어주고 실행해준다.

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

OEP에 도달하는 걸 볼 수 있다.

근데 여기서는 이 방식으로 풀면 안될거 같다. 왜냐하면 ESP 바뀌는 것을 이용해서 OEP에 도달하려면 BP때문에 JMP EAX에 도달해야하는데 바로 PUSH EBP에 도달했기 때문이다. 이건 우연으로 걸린 거 같다.

이 부분은 RET를 찾아서 하면 된다.

CPU - main thread, module UnPackMe		
00401018	BE B0114000	MOV ESI,UnPackMe.004011B0
0040101D	AD	LODS DWORD PTR DS:[ESI]
0040101E	50	PUSH EAX
0040101F	FF76 34	PUSH DWORD PTR DS:[ESI+34]
00401022	EB 7C	JMP SHORT UnPackMe.004010A0
00401024	48	DEC EAX
00401025	010F	ADD DWORD PTR DS:[EDI],ECX
00401027	010B	ADD DWORD PTR DS:[EBX],ECX
00401029	014C6F 61	ADD DWORD PTR DS:[EDI+EBP*2+61],ECX
0040102D	64:4C	DEC ESP
004010E9	57	PUSH EDI
004010EA	51	PUSH ECX
004010EB	E9 CF8D0900	JMP UnPackMe.00499EBF
004010F0	56	PUSH ESI
004010F1	10E2	ADC DL,AH
004010F3	E3 B1	JECXZ SHORT UnPackMe.004010A6

0x00401018에서 내려가다 보면 크게 JMP하는 부분이 있다. 이 부분에서 한 번 넘어가 준다.

CPU - main thread, module UnPackMe		
00499EBF	58	POP EAX
00499EC0	8D5483 58	LEA EDX,DWORD PTR DS:[EBX+EAX*4+58]
00499EC4	FF16	CALL DWORD PTR DS:[ESI]
00499EC6	72 4F	JB SHORT UnPackMe.00499F17
00499EC8	04 FD	ADD AL,0FD
00499ECA	1AD2	SBB DL,DL
00499ECC	22C2	AND AL,DL
00499ECE	3C 07	CMP AL,7

그럼 해당 부분으로 넘어오는데 여기서 RET를 찾아서 BP를 걸어준다.

0049A056	FFD5	CALL EBP
0049A058	AB	STOS DWORD PTR ES:[EDI]
0049A059	EB E7	JMP SHORT UnPackMe.0049A042
0049A05B	C3	RETN
0049A05C	00C0	ADD AL,AL
0049A05E	06	PUSH ES

'F9'를 이용해서 해당 부분까지 내려오고 'F8'을 이용해서 하나만 넘어가준다.

CPU - main thread, module UnPackMe			
004271B0	55	PUSH EBP	kernel32.GetP
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 FAX	

그럼 OEP에 도달하는 걸 볼 수 있다.

IAT Infos Needed

OEP 000271B0

AutoSearch

RVA 00060814

Get Imports

Size 00000718

해당 부분에서 덤프 떠준 후 ImpRec를 이용해서 OEP를 271B0으로 바꿔주고 AutoSearch를 눌러 IAT를 자동으로 찾을 것이다.

IAT Infos Needed

OEP 000271B0

AutoSearch

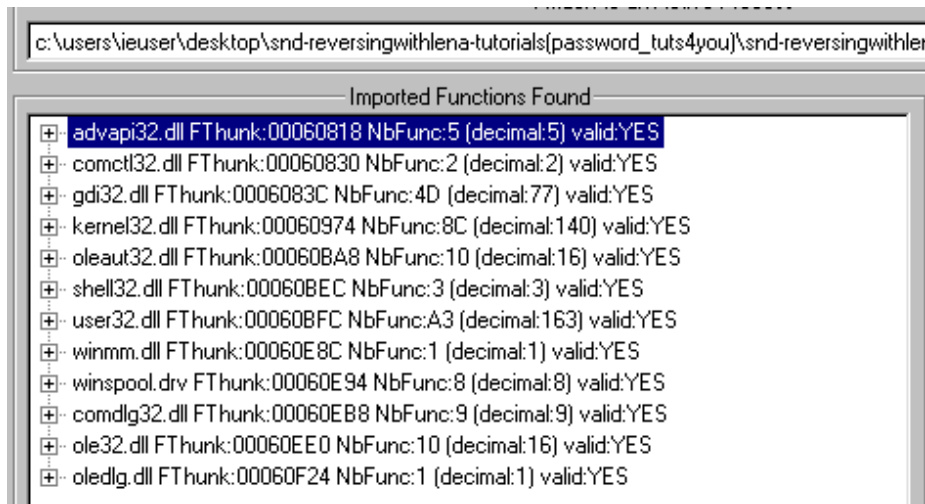
RVA 00060814

Get Imports

Size 00000718

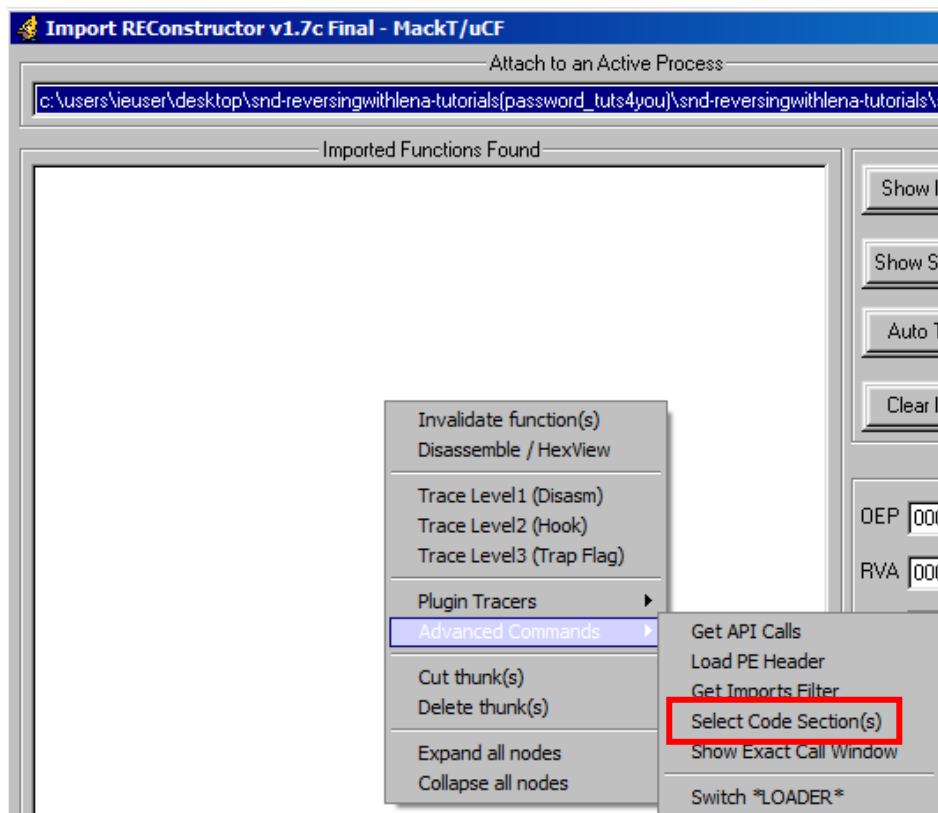
Address	Hex dump	ASCII
00460814	00 00 00 00 ED 45 0F 76 5B 48 0F 76 B1 13 0F 76	...iE%v[H%v±!%v
00460824	FB 13 0F 76 43 48 0F 76 00 00 00 00 39 17 91 63	û!%vCH%v...9! 'c
00460834	F1 7C 91 63 00 00 00 00 25 84 DF 75 89 91 DF 75	ñ  'c...%„Bu% 'Bu
00460844	FE 7C DF 75 21 8B DF 75 86 F4 DF 75 8E 81 DF 75	b Bu!<Bu†ôBužBu
00460854	25 12 E1 75 64 7E DF 75 DB 7E DF 75 B0 72 E2 75	%!âud~Bu0~Bu° râu
00460864	61 B7 E1 75 43 03 E0 75 F9 FF DF 75 4E 7F DF 75	a·áuC!âuüÿBuNıBu
00460874	C4 8B DF 75 CD 9B DF 75 84 83 DF 75 C1 D6 DF 75	Ä<Buİ>Bu„fBuAÖBu

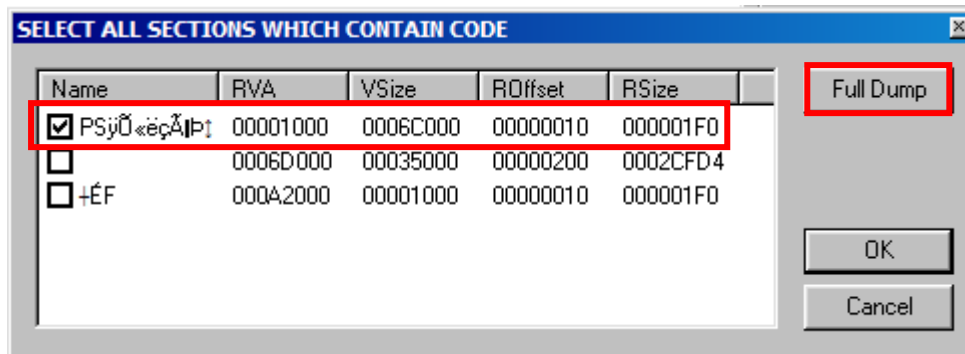
RVA가 60814로 나온다 올리디버그에서 확인해 보면 이 부분이 IAT 시작 부분인걸 알 수 있다.



Get Imports 누르면 Valid: YES 나오는 걸 확인할 수 있고 Fix Dump 눌러서 만들어준다.  
그럼 제대로 작동하는 걸 알 수 있다.

WinUpack처럼 모든 코드가 "header" 섹션에 몰려 있으면 텍스트 섹션으로는 검색이 안  
되기 때문에 "header" 섹션을 직접 선택해야 올바르게 IAT를 찾을 수 있다.





해당 부분을 체크하고 덤프해준다. 해당 부분에서 텍스트 부분을 찾으라는 의미이다.

방금 덤프한거에서 위에 IAT 찾던거랑 똑같이 해주면 된다.