

# Windows Reversing Training

Written by superdk(superdk@s0f.org)

Reversing 을 어떻게 쉽게 설명해야 할지 고민하던 중, Windows 는 분석툴들이 가식적(Visual)이어서 개념을 이해하기에 충분히 도움이 된다는 결론을 얻었다. 그래서, Windows Reversing 을 설명할 것이며, LenAs Reversing 자료가 난이도도 문안하면서 실제 배포되는 프로그램(구버전이긴 하지만)을 직접 크랙해 보는 재미(?)도 느낄 수 있기 때문에 흥미 유발도 할 수 있는 좋은 자료라 판단되어, 시리즈를 풀이하면서 이런 저런 설명을 하겠다.

## \* 실행환경

- OS : Windows 7 Ultimate K 32 bit(x86)
- CPU : Intel Pentium Dual CPU 1.6GHz
- Memory : 3G
- Reversing Tool : Ollydbg 1.0.9d
  - ( Ollydbg 2.0.0d 의 기능이 전반적으로 뛰어나지만, Call Stack 기능이 없어서 1.0.9d 를 사용함. )

## 1~2. Reversing for Newbies Part 1 ~ 2

- site : <http://www.tuts4you.com/download.php?view.122> (Part 1)

<http://www.tuts4you.com/download.php?view.123> (Part 2)

- contents :



\* Assembly Language를 모르는 독자라면 반드시 첨부된 "Basics of Assembler.doc"를 읽어야 한다.



프로그램의 동작 유형을 분석하기 위해서 reverseMe.exe 를 실행해 보면, 아래의 팝업이 뜬다.



(그림 1-1)

Part1~2 문제는 실제 프로그램이 아니라 Reversing 에 대한 개념을 설명하기 위한 훈련용 프로그램이므로 실제 실행되는 내용이 없으니, 독자 여러분의 혼란이 없기를 바라며 이러한 이유로 실행 환경에서의 힌트가 너무 없기 때문에 디버깅을 하면서 프로그램을 추적해야 한다.



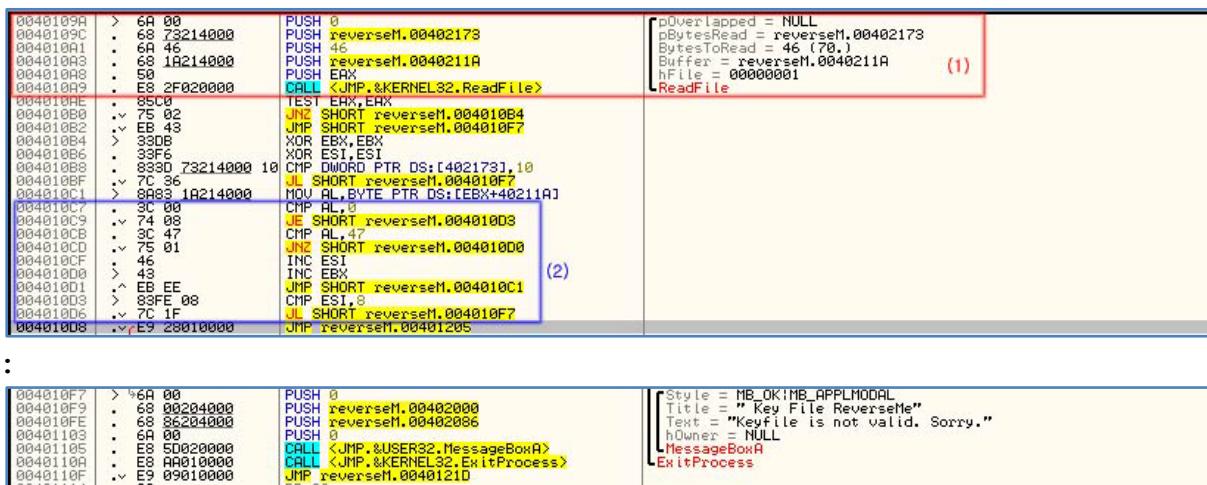
(그림 1-2)

(1)번 영역에서 보면, "Keyfile.dat" 파일을 "Read Only" 외 다양한 옵션으로 열고, 파일이 없으면, (2)번 영역에서 (그림 1-1)을 띄운다는 것을 알 수 있다. "Keyfile.dat" 파일을 만든 후 다시 reverseMe.exe 를 실행해 보자.



(그림 1-3)

문제의 요지를 정확하게 찾았으므로, 시리얼 키가 어떻게 구성되는지 찾아보자.



:

```

00401205 > b6A 00 PUSH 0
00401207 . 68 00204000 PUSH reverseMe.00402000
0040120C . 68 E204000 PUSH reverseMe.004020DE
00401211 . 68 00 PUSH 0
00401213 . 68 4F010000 CALL <JMP.&USER32.MessageBoxA>
00401218 . 68 9C000000 CALL <JMP.&KERNEL32.ExitProcess>
0040121D > C3 RETN

```

Style = MB\_OK|MB\_APPLMODAL  
Title = "Key File ReverseMe"  
Text = "You really did it! Congratz !!!"  
hOwner = NULL  
MessageBox  
ExitProcess

(그림 1-4)

(1)번 영역에서 "Keyfile.dat" 파일을 읽은 후, (2)번 영역에서 파일의 내용을 비교한다.

"CMP DWORD PTR DS:[402173],10"에서 문자열의 길이가 0x10(16)자리인지를 확인한다. 즉, 시리얼의 길이는 16자리가 된다. "MOV AL,BYTE PTR DS:[EBX+40211A]"는 "INC EBX"를 이용하여 읽어들인 문자열을 하나씩 뽑고, "CMP AL, 0" 와 "CMP AL, 47"이 "0"과 "G" 문자를 검사하면서 "INC ESI"를 체크하여 "G" 문자의 개수를 카운트하고, "CMP ESI,8"에서 "G" 문자의 개수가 8개인지 확인한다. 즉, 순서상 "0"이 먼저 나오고 다음으로 "G"가 나오는 순서로 16자리의 문자열이 되도록 반복하면, 시리얼 키 문자열을 다음과 같이 조합할 수 있다. (시리얼 키 : "0G0G0G0G0G0G0G0G") 확인해 보자.



(그림 1-5)

(그림 1-5)와 같이 시리얼 키를 정상적으로 찾았다.

### 3. Reversing for Newbies Part 3

- site : <http://www.tuts4you.com/download.php?view.124>

- contents :



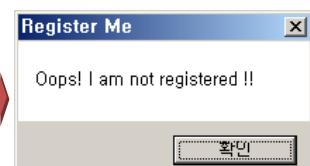
"RegisterMe.exe"를 실행해 보자.



(그림 3-1-1)



(그림 3-1-2)



(그림 3-1-3)

Part3의 프로그램은 (그림 3-1-1) ~ (그림 3-1-3)의 순서로 실행된다.

위의 실행 화면은 "Nag 창"의 개념을 설명하기 위한 프로그램으로 정상적인 프로그램은 아니다. 처음 본 독자는 당황할 수도 있겠지만, 그냥 Nag의 개념을 알고 넘어가면 된다. (그림 3-1-2)가 우리가 사용할 정상적인 프로그램이라고 할 때, (그림 3-1-1)과 (그림 3-1-3)처럼 쉐어웨어의 경우 구매를 제안(?)하는 창을 띄우는 것이 일반적인데, 이러한 귀찮은 창을 "Nag Screen"이라 한다.

이제 이 Nag를 제거해 볼텐데, 우선 가장 쉬운 기본적인 방법으로 제거해 보자.

```

00401000 FS 6A 00 PUSH 0
00401002 E8 00200000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 A3 1C314000 MOU DWORD PTR DS:[40311C],EAX
00401008 83F8 00 CMP EAX,0
0040100F v 74 13 JE SHORT 00401024
00401011 E8 00 PUSH 0
00401012 68 0020040000 PUSH 00403070
00401013 88 3430400000 PUSH 88483034
00401014 E8 00 CALL <JMP.&USER32.MessageBoxA>
00401015 E8 C0100000 CALL <JMP.&KERNEL32.ExitProcess>
00401024 > 6A 0H PUSH DH
00401026 FF35 20314000 PUSH DWORD PTR DS:[403120]
00401027 . 6A 0H PUSH 0
00401028 FF35 1C314000 PUSH DWORD PTR DS:[40311C]
00401029 E8 1900000000 CALL 00401052
0040102A E8 00 PUSH 0
0040102B 68 0020040000 PUSH 00403070
0040102C 68 8930400000 PUSH 88483089
0040102D E8 00 PUSH 0
0040102E E8 9E010000 CALL <JMP.&USER32.MessageBoxA>
00401047 I . 50 PUSH EAX
0040104D E8 BC010000 CALL <JMP.&KERNEL32.ExitProcess>

```

Style = MB\_OK|MB\_APPLMODAL  
Title = "Register Me"  
Text = "Remove the nags to register\nThis will make program fully registered  
hOwner = NULL  
MessageBoxA  
Args4 = 00000000  
Args3 = 00000000  
Args2 = 00000000  
Args1 = 00400000  
Register = 00401052

Style = MB\_OK|MB\_APPLMODAL  
Title = "Register Me"  
Text = "Owner, I am not registered !!"  
hOwner = NULL  
MessageBoxA  
Args4 = 00000000  
Args3 = 00000000  
Args2 = 00000000  
Args1 = 00400000

Style = MB\_OK|MB\_APPLMODAL  
Title = "Register Me"  
Text = "Owner, I am not registered !!"  
hOwner = NULL  
MessageBoxA  
Args4 = 00000000  
Args3 = 00000000  
Args2 = 00000000  
Args1 = 00400000

ExitCode = 110544  
ExitProcess

(그림 3-2)

(1)번 영역에서 (그림 3-1-1)을, (2)번 영역에서 본 프로그램인 (그림 3-1-2)를, (3)번 영역에서 (그림 3-1-3)을 띄운다. 그러므로, (1)번과 (3)번 영역의 "CALL <JMP.&USER32.MessageBoxA>"를 (그림 3-3)과 같이 제거하면 된다.

```

00401000 FS 6A 00 PUSH 0
00401002 E8 00200000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 A3 1C314000 MOU DWORD PTR DS:[40311C],EAX
00401008 83F8 00 CMP EAX,0
0040100F v 74 13 JE SHORT 00401024
00401011 E8 00 PUSH 0
00401012 68 0020040000 PUSH 00403070
00401013 88 3430400000 PUSH 88483034
00401014 E8 00 CALL <JMP.&USER32.MessageBoxA>
00401015 E8 C0100000 CALL <JMP.&KERNEL32.ExitProcess>
00401024 > 6A 0H PUSH DH
00401026 FF35 20314000 PUSH DWORD PTR DS:[403120]
00401027 . 6A 0H PUSH 0
00401028 FF35 1C314000 PUSH DWORD PTR DS:[40311C]
00401029 E8 1900000000 CALL 00401052
0040102A E8 00 PUSH 0
0040102B 68 0020040000 PUSH 00403070
0040102C 68 8930400000 PUSH 88483089
0040102D E8 00 PUSH 0
0040102E E8 9E010000 CALL <JMP.&USER32.MessageBoxA>
00401047 I . 50 PUSH EAX
0040104D E8 BC010000 CALL <JMP.&KERNEL32.ExitProcess>

```

Style = MB\_OK|MB\_APPLMODAL  
Title = "Register Me"  
Text = "Remove the nags to register\nThis will make program fully registered  
hOwner = NULL  
MessageBoxA  
(1)

Style = MB\_OK|MB\_APPLMODAL  
Title = "Register Me"  
Text = "Owner, I am not registered !!"  
hOwner = NULL  
MessageBoxA  
(3)

Style = MB\_OK|MB\_APPLMODAL  
Title = "Register Me"  
Text = "Owner, I am not registered !!"  
hOwner = NULL  
MessageBoxA  
Args4 = 00000000  
Args3 = 00000000  
Args2 = 00000000  
Args1 = 00400000

ExitCode = 0  
ExitProcess

(그림 3-3)

(그림 3-3)의 (1)번과 (3)번처럼 Nag를 호출하는 호출문을 제거하면, (그림 3-1-1)과 (그림 3-1-2)가 없는 본 프로그램인 (그림 3-1-2)만 뜨기 때문에 마치 정품(등록 버전)을 사용하는 듯한 효과를 낼 수 있다.

"RegisterMe.exe"를 해결하였으므로, "RegisterMe.Oops.exe"를 실행해 보자. 그런데, 실행 결과는 "RegisterMe.exe"와 동일하다. 분석을 위해서 Ollydbg로 열어보자. 그런데, 아래의 에러창이 뜬다.



(그림 3-4)

"executable file format" 에러다. 즉, PE(Portable Executable) Format에 관한 내용이다.

### \* Concept of PE

PE는 하나의 바이너리가 Windows Platform Series 의 모든 GUI 와 다양한 CPU에서 돌아가도록, Unix의 Coff(Common object file format)에서 가져온 개념이다.

먼저, PE의 구조를 보자.

<b>DOS MZ Header</b>	PE loader가 여기를 검색해서 PE Header까지의 offset을 찾음. Offset 이 있으면, DOS Stub을 지나쳐서, PE Header로 곧바로 이동함.
<b>DOS Stub</b>	GUI 모드로 개발된 프로그램을 "DOS Mode"에서 실행했을 경우 "This program cannot run in DOS mode" 라는 메시지를 표시하고 종료함.
<b>PE Header</b>	PE loader에 쓰일 기본 항목들이 정의됨. PE Header의 유효성 검사를 위한 구조체 항목을 가지고 있음.
<b>Section table</b>	메모리에 mapping 할 sections 와 maps의 정보를 가지고 있음.
<b>Section 1</b>	구체 내용
<b>Section ...</b>	구체 내용 ...
<b>Section n</b>	구체 내용

(표 2-1)

(표 2-1)의 내용이 바이너리/메모리 상에 어떻게 표현되어 있는지 확인해 보자.

Address	Hex dump	ASCII
00400000	4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00	MZP.....♦.%
00400010	B8 00 00 00 00 00 00 40 00 1A 00 00 00 00 00 00	?.....@.+..
00400020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00400030	00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00	.....\$
00400040	B8 10 00 0E 1F B4 09 C0 21 B8 01 4C C0 21 90 90	?..AY???.L?B4
00400050	54 68 69 73 20 70 72 6F 67 72 61 60 20 60 75 73	This program mus
00400060	74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57	t be run under W
00400070	69 6E 33 32 00 0A 24 37 00 00 00 00 00 00 00 00	in32..\$7.....
00400080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00400090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004000A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004000B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004000C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004000D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004000E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004000F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00400100	50 45 00 00 4C 01 05 00 F0 26 6C 84 00 00 00 00	PE..L04..?I?..
00400110	00 00 E0 00 8E 81 0B 01 02 19 00 06 00 00 00	?=j0e4..+
00400120	00 18 00 00 00 00 00 00 10 00 00 00 10 00 00 00	↑.....
00400130	00 20 00 00 00 40 00 00 10 00 00 00 02 00 00 00	.....@.►..\$
00400140	01 00 00 00 00 00 00 03 00 0A 00 00 00 00 00 00	0.....^..
00400150	00 60 00 00 00 04 00 00 00 00 00 00 02 00 00 00	.....♦..\$
00400160	00 00 10 00 00 20 00 00 00 00 10 00 00 10 00 00	.....►..►..
00400170	00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 00	.....►..►..
00400180	00 30 00 00 DE 0A 00 00 50 00 00 1E 06 00 00 00	0...?...P..▲..

(그림 3-5)

DOS MZ Header는 Address가 "00400000"이고, 문자열 MZ로 시작되며, Offset 이 "00000100"인 것을 알 수 있다. 실제로 PE Header는 Address가 "00400100"이고 문자열 PE로 시작되는 것을 볼 수 있다.

메모리 맵을 보자. PE header 라인 아래에 관련 정보(code, imports, data...)가 함께 있다.

Memory map								
Address	Size	Owner	Section	Contains	Type	Access	Initial access	...
00400000 00001000	Register			PE header	Image	R	RWE	
00401000 00001000	Register	.text		code	Image	R	RWE	
00402000 00001000	Register	.rdata		imports	Image	R	RWE	
00403000 00001000	Register	.data		data	Image	R	RWE	
00404000 00001000	Register	.rsrc		resources	Image	R	RWE	
00410000 00009000					Map	R	R	

(그림 3-6)

D Dump - reverseM 00400000..00400FFF

00400000	4D 5A	ASCII "MZ"	DOS EXE Signature
00400002	5000	DW 0050	DOS_PartPag = 50 (80.)
00400004	0200	DW 0002	DOS_PageCnt = 2
00400006	0000	DW 0000	DOS_ReloCnt = 0
00400008	0400	DW 0004	DOS_HdrSize = 4
0040000A	0F00	DW 000F	DOS_MinMem = F (15.)
0040000C	FFFF	DW FFFF	DOS_MaxMem = FFFF (65535.)
0040000E	0000	DW 0000	DOS_ReloSS = 0
00400010	B800	DW 00B8	DOS_ExeSP = B8
00400012	0000	DW 0000	DOS_ChkSum = 0
00400014	0000	DW 0000	DOS_ExeIP = 0
00400016	0000	DW 0000	DOS_ReloCS = 0
00400018	4000	DW 0040	DOS_TableOff = 40
0040001A	1A00	DW 001A	DOS_Overlay = 1A
0040001C	00	DR 0A	

D Dump - reverseM 00400000..00400FFF

00400038	00	DB 00	
00400039	00	DB 00	
0040003A	00	DB 00	
0040003B	00	DB 00	
0040003C	00010000	DD 000000100	Offset to PE signature
00400040	BA	DB BA	
00400041	10	DB 10	
00400042	00	DB 00	
00400043	0E	DB 0E	
00400044	1F	DB 1F	
00400045	B4	DB B4	
00400046	09	DB 09	
00400047	CD	DB CD	
00400048	21	DB 21	
00400049	RR	DR RR	

D Dump - reverseM 00400000..00400FFF

00400100	50 45 00 01	ASCII "PE"	PE signature (PE)
00400104	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
00400106	0500	DW 0005	NumberOfSections = 5
00400108	F0266C84	DD 846C26F0	TimeDateStamp = 846C26F0
0040010C	00000000	DD 00000000	PointerToSymbolTable = 0
00400110	00000000	DD 00000000	NumberOfSymbols = 0
00400114	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
00400116	8E81	DW 818E	Characteristics = EXECUTABLE_IMAGE 32BIT_MACHINE LIM
00400118	0B01	DW 010B	MagicNumber = PE32
0040011A	02	DB 02	MajorLinkerVersion = 2
0040011B	19	DB 19	MinorLinkerVersion = 19 (25.)
0040011C	000060000	DD 00000600	SizeOfCode = 600 (1536.)
00400120	00180000	DD 00001800	SizeOfInitializedData = 1800 (6144.)
00400124	00000000	DD 00000000	SizeOfUninitializedData = 0
00400128	00100000	DD 00001000	AddressOfEntryPoint = 1000
0040012C	00100000	DD 00001000	BaseOfCode = 1000
00400130	00200000	DD 00002000	BaseOfData = 2000
00400134	00004000	DD 00400000	ImageBase = 400000
00400138	00100000	DD 00001000	SectionAlignment = 1000
0040013C	00020000	DD 00000200	FileAlignment = 200
00400140	0100	DW 0001	MajorOSVersion = 1

(그림 3-7)

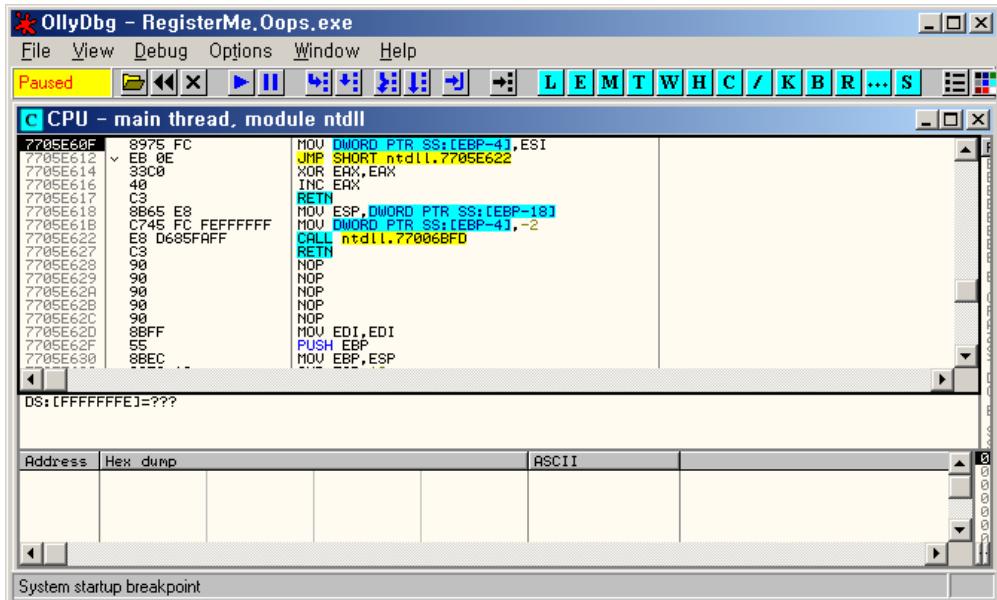
(그림 3-7)과 같이 메모리 덤프를 보면 ImageBase 와 AddressOfEntryPoint 값은 알 수 있다. 그런데, 이 두 값을 더하면 “401000”이 되고, 이 값은 (그림 3-2)와 (그림 3-3)의 왼쪽 상단에 있는 Address 임을 알 수 있다. 즉, 프로그램이 시작되는 지점이다.

PE Explorer로 보면 해당 값을 더 편하게 확인할 수 있다.

HEADERS INFO		
Address of Entry Point:	00401000	Real Image Checksum: 000111A8h
Field Name	Data Value	Description
Machine	014Ch	i386
Number of Sections	0004h	
Time Date Stamp	3801291Eh	16/03/2000 18:34:06
Pointer to Symbol Table	00000000h	
Number of Symbols	00000000h	
Size of Optional Header	00E0h	
Characteristics	010Fh	
Magic	0108h	PE32
Linker Version	0C05h	5.12
Size of Code	40000400h	
Size of Initialized Data	40000400h	
Size of Uninitialized Data	00000000h	
Address of Entry Point	00401000h	
Base of Code	40001000h	
Base of Data	40002000h	
Image Base	00400000h	
Section Alignment	00001000h	
File Alignment	00000200h	
Operating System Version	00000004h	4.0
Image Version	0000000h	0.0
Subsystem Version	00000004h	4.0
Win32 Version Value	0000000h	Reserved
Size of Image	00005000h	20480 bytes
Size of Headers	00000400h	
Checksum	00008499h	
Subsystem	002h	Win32 GUI
Dll Characteristics	0000h	
Size of Stack Reserve	00100000h	
Size of Stack Commit	00001000h	
Size of Heap Reserve	00100000h	
Size of Heap Commit	00001000h	
Loader Flags	00000000h	Obsolete
Number of Data Directories	40000004h	

(그림 3-8)

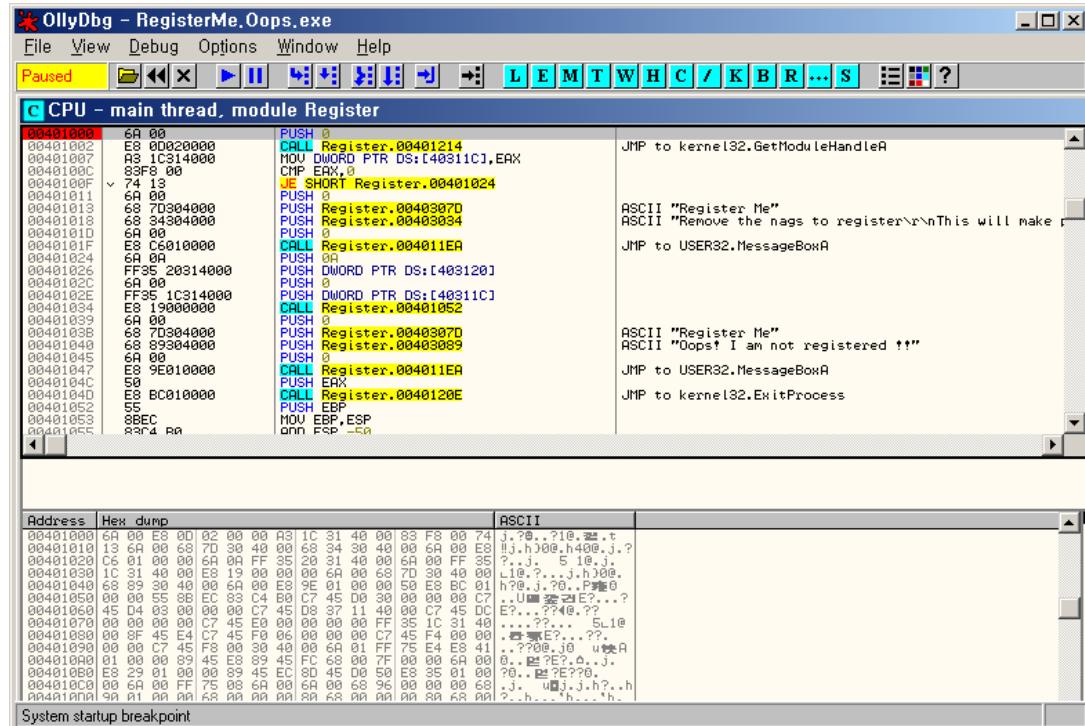
다시 Ollydbg로 돌아가 보자.



(그림 3-9)

(그림 3-9)와 같이 Ollydbg로 로딩 중 “executable file format” 에러를 만나고 혼란스럽게 모듈이 로딩되면, 분석이 쉽지 않다. 이런 경우는 일반적으로 분석을 어렵게 하기 위해서 구조를 꼬아놨다고 생각하면 된다. 하지만, (그림 3-5)~(그림 3-7)에서 설명한 것과 같이 PE 구조를 이해할 수 있다면, 이러한 프로그램의 PE 구조를 분석하여 프로그램의 시작 주소가 “401000”이라는 것을 알 수 있고, 이 주소로 이동해서 (그림 3-9)와 같이 프로그램의 디버깅 시작점을 찾을 수 있다.

이번 샘플에서 우리는 PE 구조의 원본을 가지고 있으므로, 원본(RegisterMe.exe)을 이용하여 “RegisterMe.Oops.exe”를 원본 형태로 복원해 보면서 개념을 재정립할 수 있다.



(그림 3-9)

이렇게 “401000”으로 이동한 화면이 (그림 3-2), (그림 3-3)과 같다는 것을 알 수 있다. (즉. 제대로 시작점을 찾았다.)

여기에서 다시 "RegisterMe.exe"로 돌아가서 offset을 1000에서 1024로 바꾸면 어떻게 될까? 당연히 1000~1023 까지의 첫 Nag 를 실행하지 않기 때문에 "본 프로그램" 시작 전에 보이던 Nag 가 뜨지 않는다.

그리고, 이와 같이 PE가 혼란스러운 바이너리는 (그림 3-10)의 (1)번 영역이 (그림 3-6) 과는 다른 메모리 구조를 갖는다. 이 때문에 ollydbg에서 관련정보(code, imports, data...) 없이 이상하게 로드되는 것이다.

Address	Size	Owner	Section	Contains	Type	Access	Initial access
00400000	000005000	Register		PE header	Image	R	RWE
00410000	00101000				Map	R	R
00520000	00123000				Map	R	R
01120000	00016000				Private	RW	RW
01148000	00003000				Private	RW	RW
01160000	00930000				Map	R	R
01AE0000	00001000				Private	RW	RW
01B20000	00003000				Private	RW	RW
01B30000	00001000				Private	RW	RW
01CAF000	00002000				Private	RW	Guarded
01CAE000	00001000			stack of thread 00000C00	Private	RW	Guarded

(그림 3-10)

메모리 덤프를 보고, "RegisterMe.Oops.exe"를 "RegisterMe.exe"와 동일하게 바꾸어 보자.

D Dump - Register 00400000..00404FFF			
00400000	00000000	DD	00000000
00400004	E000	DW	00E0
00400006	0F01	DW	010F
00400008	0B01	DW	010B
0040000A	05	DB	05
0040000B	0C	DB	0C
<b>0040000C</b>	<b>00040040</b>	<b>DD</b>	<b>40000400</b>
0040000E	000AA040	DD	40000A00
004000E4	00000000	DD	00000000
004000E8	00100000	DD	00001000
004000EC	00100040	DD	40001000
004000F0	00200040	DD	40002000
004000F4	00004000	DD	00400000
			NumberOfSymbols = 0
			SizeOfOptionalHeader = E0 (224.)
			Characteristics = EXECUTABLE_IMAGE 32BIT_MACHINE
			MagicNumber = PE32
			MajorLinkerVersion = 5
			MinorLinkerVersion = C (12.)
			SizeOfCode = 40000400 (1073742848.)
			SizeOfInitializedData = 40000A00 (1073744384.)
			SizeOfUninitializedData = 0
			AddressOfEntryPoint = 1000
			BaseOfCode = 40001000
			BaseOfData = 40002000
			ImageBase = 400000

(그림 3-11)

(그림 3-11)의 사각형 부분이 "RegisterMe.exe"와 다른 부분이다. 이 부분(4000DC 이후)의 7개 값을 바꾸어서 두 프로그램의 PE 구조를 동일하게 만들어 보자.

The image shows two side-by-side 'Edit data at 004000DC' dialog boxes from a debugger. Both boxes have three tabs: ASCII, UNICODE, and HEX +00. The HEX +00 tab displays memory contents as a grid of hex values. In the left box, the first four columns of the first row (containing '00 04 00 40') are highlighted with a pink rectangle. In the right box, the entire first row ('00 04 00 00 00 0A 00 00 40 00 00 00 00 00 00 00') is highlighted with a pink rectangle. A red arrow points from the left box to the right box, indicating a change or comparison between the two states.

(변경 전)

(변경 후)

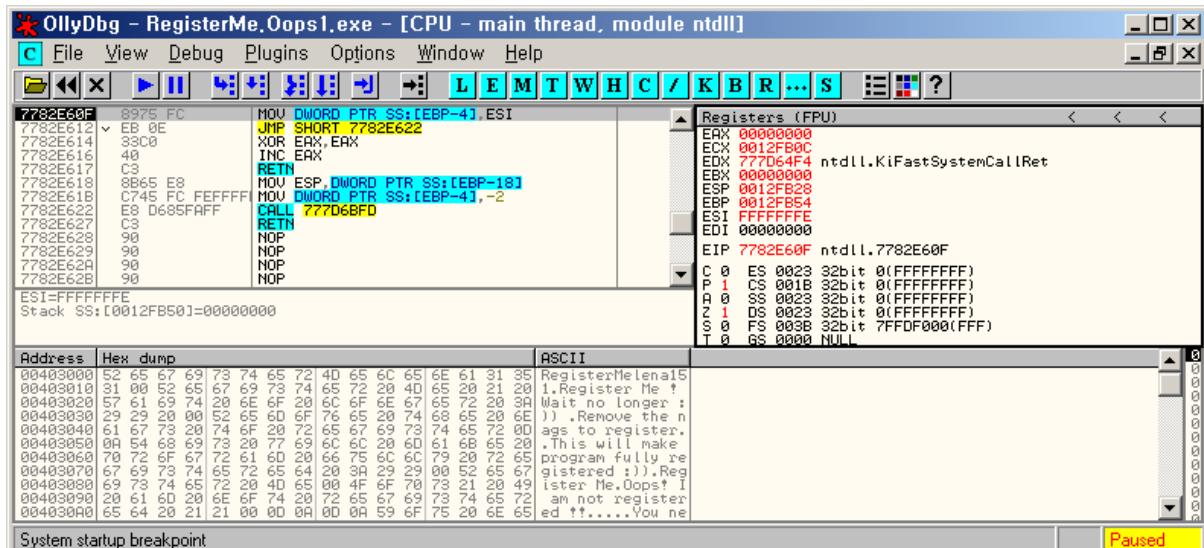
(그림 3-12)

변경된 내용을 (그림 3-13)과 같이 확인할 수 있다.

Address	Hex dump	ASCII
0040000DC	00 04 00 00 00 0A 00 00 00 00 00 00 00 00 00 00	♦.....@..
0040000EC	00 10 00 00 00 20 00 00 00 00 40 00 00 10 00 00	♦.....@..
0040000FC	00 02 00 00 04 00 00 00 00 00 00 00 04 00 00 00	♦.....P..
00400010C	00 00 00 00 00 50 00 00 00 04 00 00 99 B4 00 00	.....P..
00400011C	02 00 00 00 00 00 10 00 00 10 00 00 00 00 10 00	.....P..
00400012C	00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00	.....P..
00400013C	00 00 00 00 50 20 00 00 3C 00 00 00 00 40 00 00	.....P..<...@..
00400014C	9C 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00	?.....
00400015C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

(그림 3-13)

이것을 바이너리로 저장한 후 OllyDbg로 다시 읽어보면, (그림 3-4)와 같은 “executable file format” 에러가 표시되지 않고 정상적으로 이미지가 로딩되는 것을 볼 수 있다.



(그림 3-14)

(그림 3-15)의 메모리 맵을 봐도, (그림 3-10)과는 달리 원본(RegisterMe.exe)의 그림(2-6)처럼 PE header에 연속적으로 관련 정보(code, imports, data...)가 붙어 있는 것을 볼 수 있다.

Memory map												
Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as				
00400000 00003000	Register		PE header		Priu	R/W	R/W					
00401000 00001000	Register		.text		Imag	R	RWE					
00402000 00001000	Register		.rdata		code	Imag	R	RWE				
00403000 00001000	Register		.data		imports	Imag	R	RWE				
00404000 00001000	Register		.rsrc		data	Imag	R	RWE				
75A80000 00001000	KERNELBA		resources		resources	Imag	R	RWE				
75A81000 00043000	KERNELBA		.text		PE header	Imag	R	RWE				
					code, import	Imag	R	RWE				

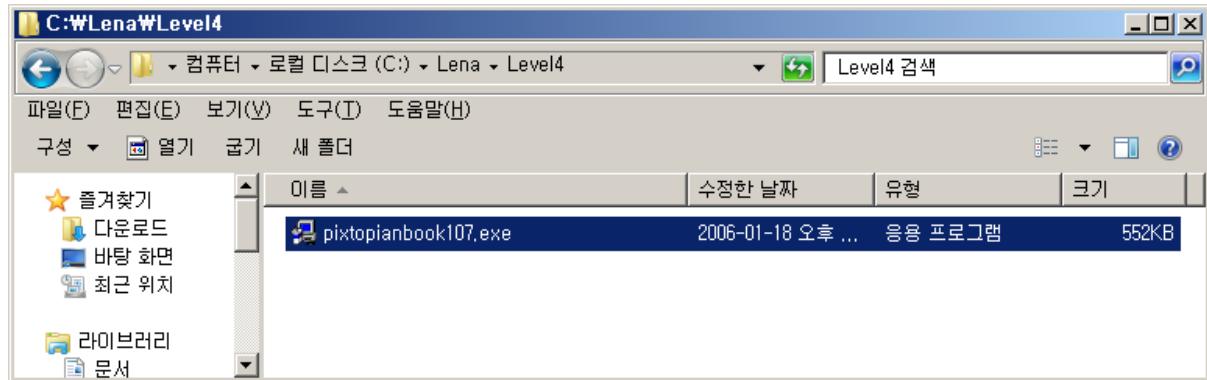
(그림 3-15)

이렇게 PE 구조를 맞출 수 있다면, 프로그램을 완벽하게 분석할 수 있다. 하지만, PE 구조 전체를 세세하게 알기에는 많은 것을 공부해야 하기 때문에 쉽지 않다. 즉, 많은 공부를 해야 한다.

## 4. Reversing for Newbies Part 4

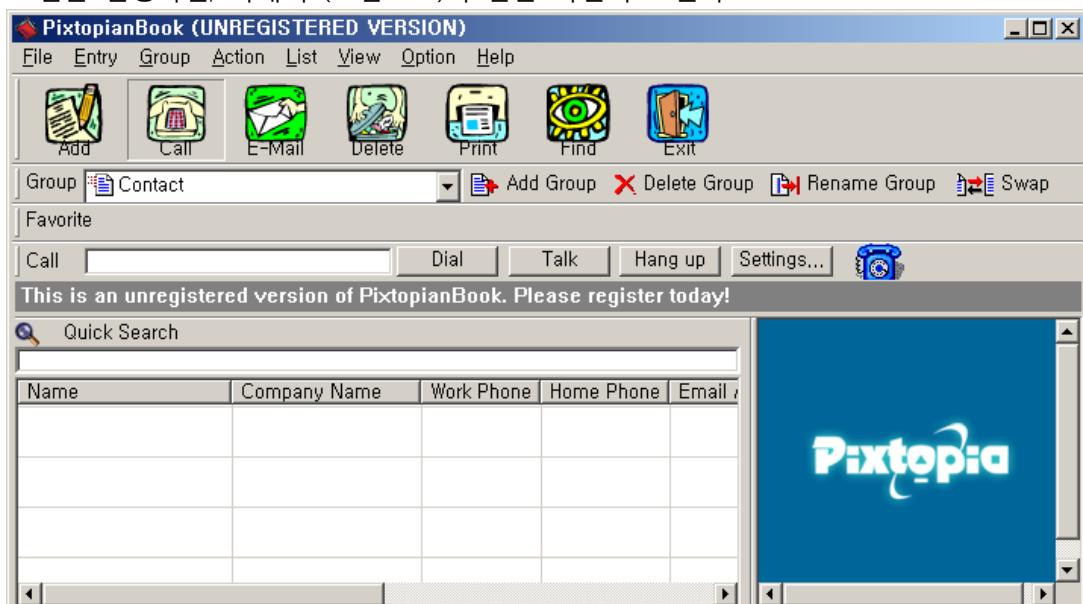
- site : <http://www.tuts4you.com/download.php?view.125>

- contents :



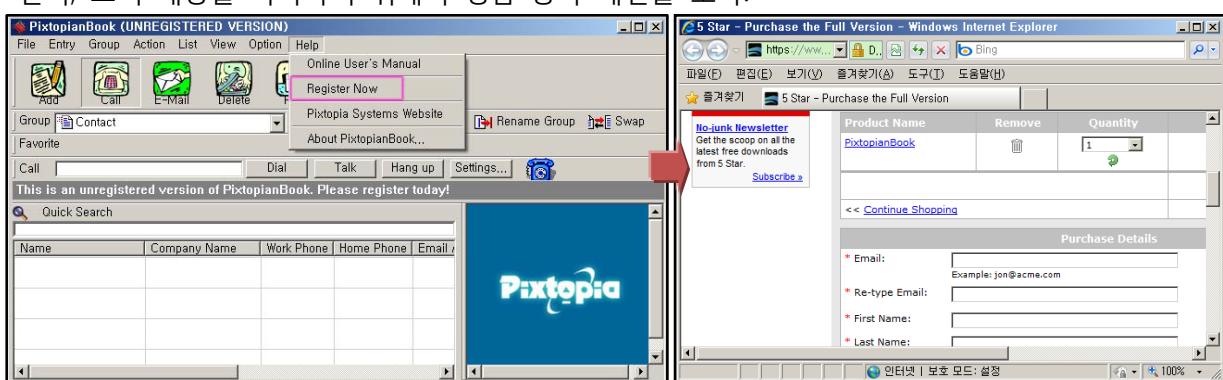
이번 프로그램은 설치버전(구버전의 상용 프로그램)이다. 먼저, 해당 파일을 PC에 설치한다.

프로그램을 실행하면, 아래의 (그림 4-1)과 같은 화면이 보인다.



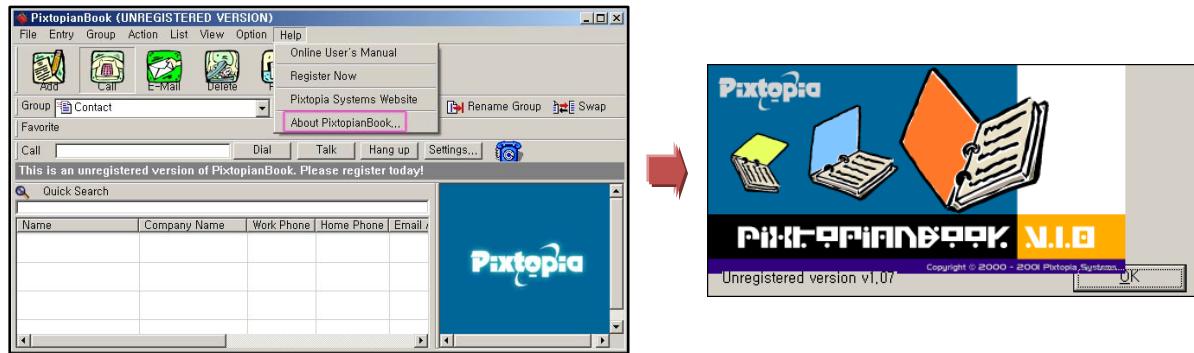
(그림 4-1)

먼저, 크랙 대상을 파악하기 위해서 정품 등록 패턴을 보자.



(그림 4-2)

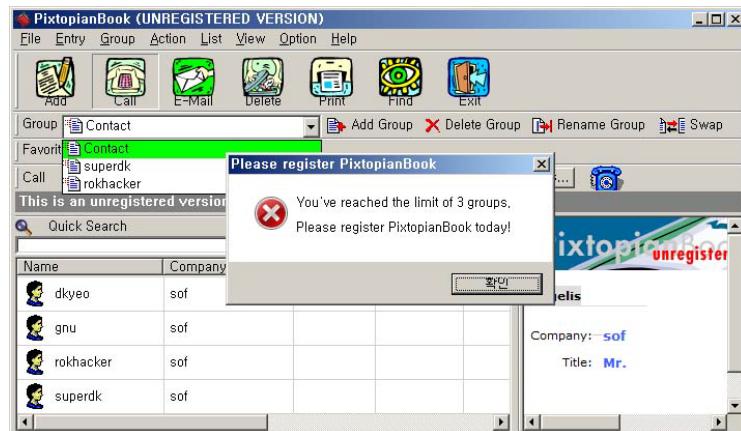
(그림 4-2)와 같이 “Register Now”를 누르면, 시리얼키 입력을 받는 게 아니라, 구매를 유도하는 웹사이트가 뜬다. 즉, 이 부분에서는 우리가 바이너리를 수정하더라도 프로그램을 정품으로 수정할 수 없으므로, 수정 대상이 아니다.



(그림 4-3)

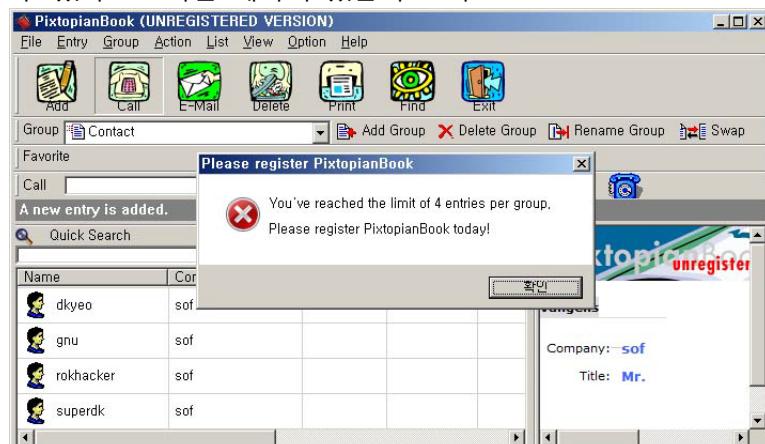
(그림 4-3) 역시 시리얼 같은 내용은 없고, 단순하게 Unregistered version 이라는 정보만 있다. 역시 관심 가질 항목은 아니다. Part 1~2 처럼 시리얼을 등록하는 프로세스인 경우 시리얼 값을 찾아서 정품으로 강제 인식시키면 되지만, 그러한 방법으로 패치할 수 있는 것이 아니므로, 프로그램의 동작 상태를 좀 더 구체적으로 파악해 보자.

쉐어웨어는 분명히 뭔가 제약이 있으므로, 예러가 보일 때까지 그룹을 만들어 보자.



(그림 4-4)

(그림 4-4)와 같이 그룹은 3개까지만 등록이 가능하다. 이 프로그램은 저장 가능한 수를 제한하는 쉐어웨어로 볼 수 있다. 또 다른 제약이 있는지 보자.

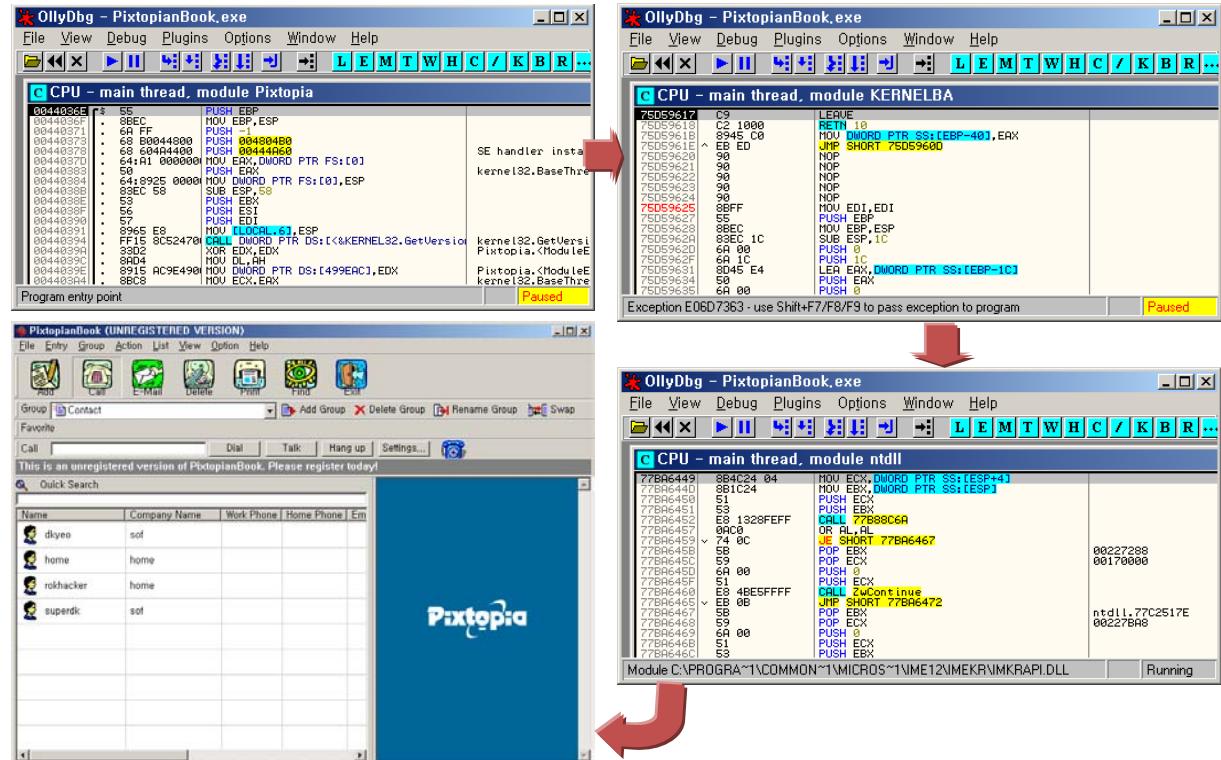


(그림 4-5)

이 PhoneBook 프로그램은 쉐어웨어로 3개의 그룹과 그룹당 4명(최대 12명)까지 등록이 가능하다. 즉, 체험으로 12명까지 등록해서 써보고 마음에 들면 구매하라는 것이다.

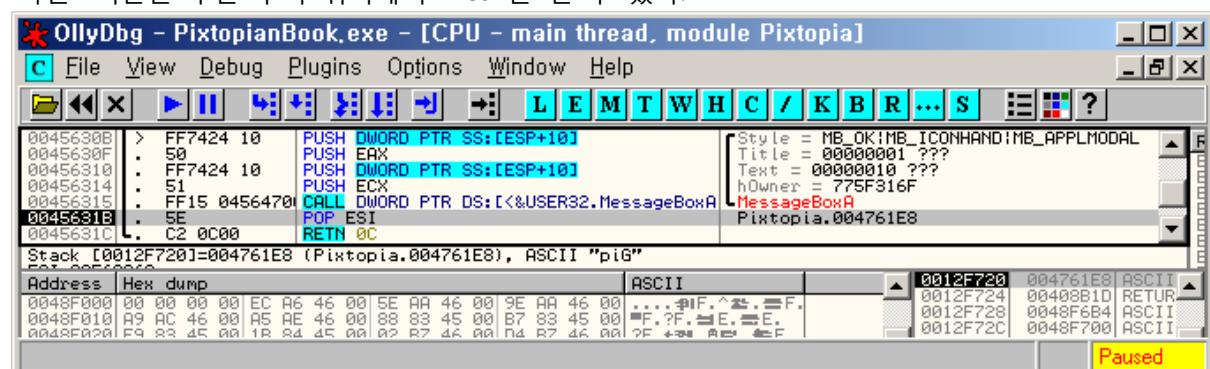
그러면, 이 프로그램은 Group Add 버튼을 누를 때와, User Add 버튼을 누를 때, 조건문으로 3개의 그룹과 그룹당 4명을 초과할 수 없도록 프로그램 되어 있을 것이며, 우리는 그 부분을 조작하면 될 것이다. 바이너리를 분석해 보자.

바이너리를 로딩한 후 (그림 4-6)과 같이 "F9", "Shift+F7/F8/F9"를 교대로 반복하여 실행하다 보면, 해당 프로그램이 실행되는 디버깅 모드로 진입할 수 있다. 이것은 로딩 중 "Exception Error"가 발생했기 때문으로, 에러로 인해 Break 된 상태에서 "Shift+F7/F8/F9"를 이용해 계속할 수 있다.



(그림 4-6)

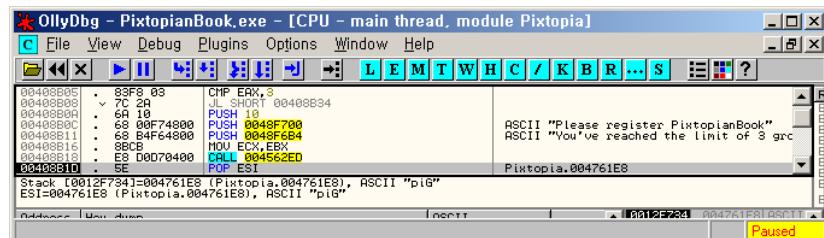
(그림 4-4)와 같이 "그룹을 더 추가할 수 없다."는 에러 창을 띄운 후, Ollydbg에서 "Pause" 버튼이나 "F12"를 눌러서 프로그램을 일시중지시키고, 다시 "Alt+F9"를 이용해서 사용자의 입력 코드가 있을 때까지만 실행하도록 하면, "3개의 그룹을 초과해서 그룹을 생성할 수 없다"는 팝업에서 "확인" 버튼을 누른 후의 위치에서 Break를 걸 수 있다.



(그림 4-7)

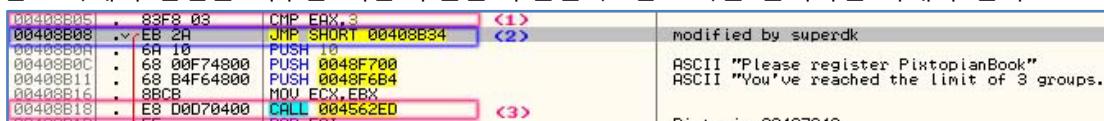
(그림 4-7)과 같이 "확인"을 누르는 시점에서 Break 가 잡힌다. F8을 눌러 "RETN 0C"를 지나면,

(그림 4-8)과 같이 경고창을 띄우는 프로세스를 호출했던 call 문의 다음으로 돌아갈 수 있다.



(그림 4-8)

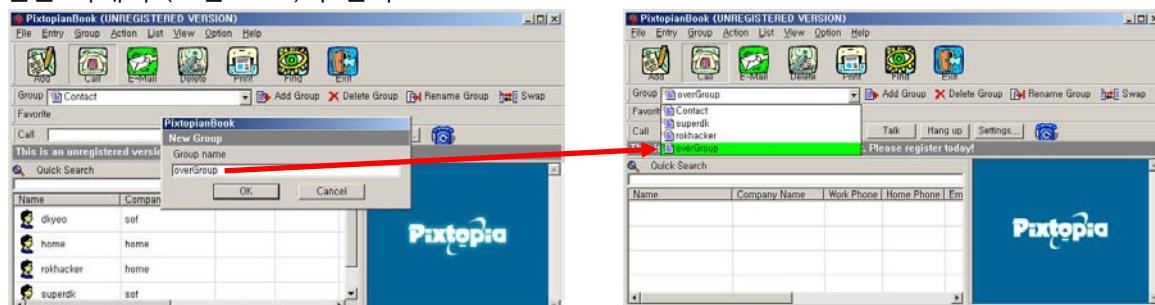
(그림 4-8)에서 팝업을 띄우는 핵심 부분을 추출한 (그림 4-9)를 분석하면 아래와 같다.



(그림 4-9)

(1)번 라인이 그룹을 3개까지만 추가할 수 있는 조건을 나타내고, (2)번 라인에서 3개 이하일 때만 그룹을 추가할 수 있는 부분으로 이동하고, (3)번 라인에서 경고창을 띄우는 함수를 호출하여 추가 그룹의 생성을 막고 있다.

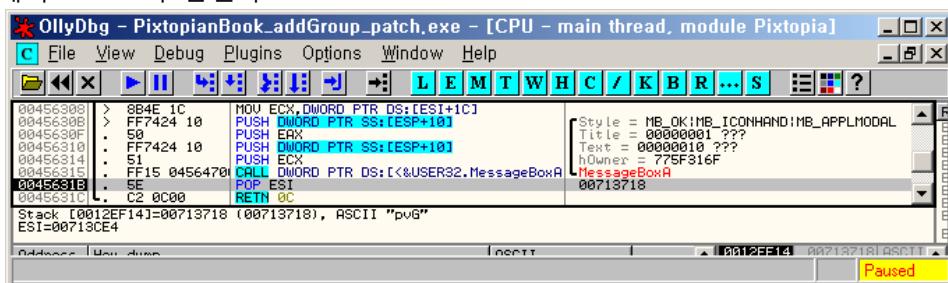
이 부분을 수정하는 간단한 2가지 방법을 말하면, 하나는 (1)번 라인에서 "숫자 3"을 10000 정도로 충분히 크게 수정해서 10000개의 그룹을 등록할 수 있도록 하는 것과, (2)번 라인에서 "JL SHORT 00408B34" 명령어를 "JMP SHORT 00408B34"로 바꾸어서 제한 없이 무한대로 그룹을 추가할 수 있도록 하는 것이다. 이렇게 해당 코드를 수정한 결과 그룹을 더 추가할 수 있게 된 화면은 아래의 (그림 4-10)과 같다.



(그림 4-10)

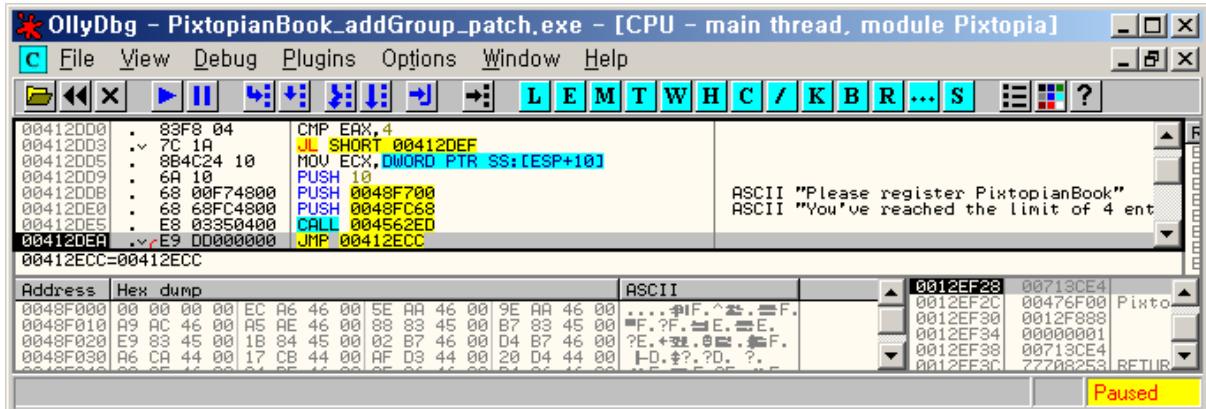
(그림 4-10)을 보면 "overGroup"이 4번째 그룹으로 추가되었다

이제 그룹마다 추가할 사람의 수를 수정해 보자. 그룹을 수정할 때와 마찬가지로 (그림 4-5)의 "주소를 더 추가할 수 없다"는 에러 메시지가 뜬 상태에서 "Pause"를 눌러서 프로그램을 일시정지 시킨 후, (그림 4-7)에서와 같이 "Alt+F9"로 사용자 입력시 Break를 걸고, OK 버튼을 누르면 해당하는 부분에서 Break 가 잡힌다.



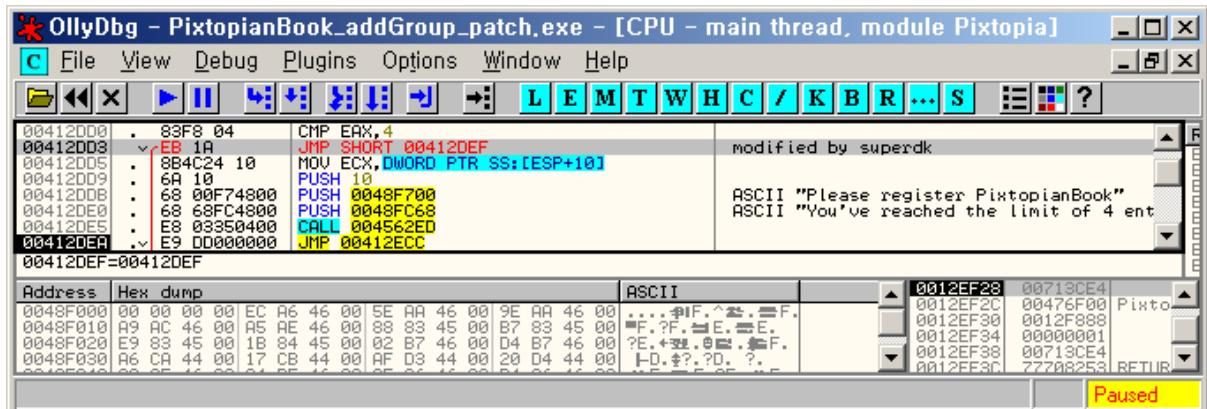
(그림 4-11)

(그림 4-7)과 마찬가지로 "F8"로 "RETN 0C"를 지나가면, (그림 4-12)처럼 해당하는 조건이 있는 바이너리 부분을 찾을 수 있다.

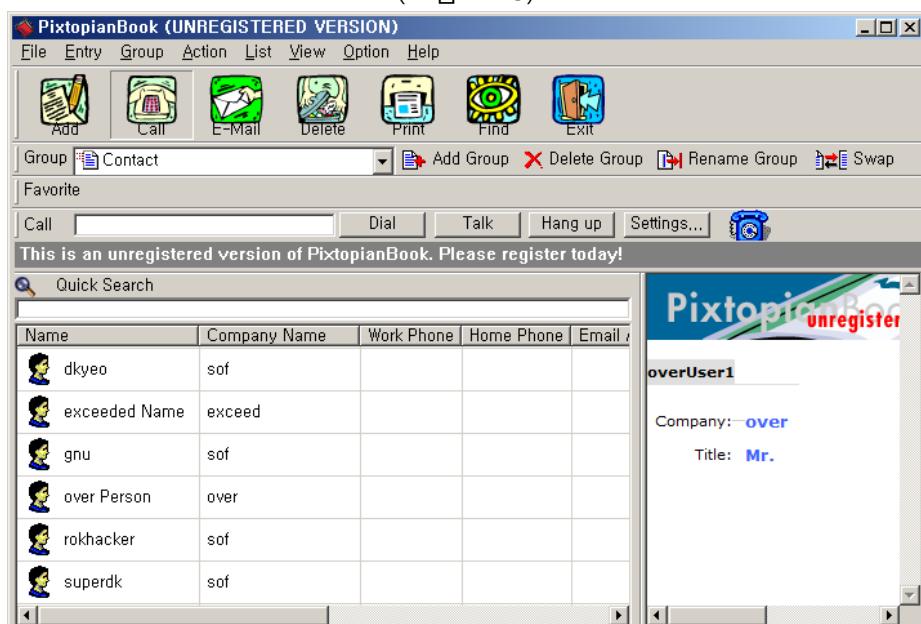


(그림 4-12)

이번 역시 (그림4-9)에서처럼, "CMP EAX,4" 라인의 4를 10000과 같이 충분히 큰 수로 바꾸거나, "JL SHORT 00412DEF"을 "JMP SHORT 00412DEF"로 바꾸어서 (그림 4-13)과 같이 조건식을 무력화시키게 되면, 아래의 (그림 4-14)처럼 정품과 동일하게 많은 수의 사람을 등록할 수 있게 된다.



(그림 4-13)



(그림 4-14)

## 5. Reversing for Newbies Part 5

- site : <http://www.tuts4you.com/download.php?view.126>

- contents :



19개의 DLL 파일은 "VisualSite Designer.exe"가 실행될 때 참고하는 파일이므로, 반드시 있어야 하는 파일들이다.

프로그램을 실행시켜 보면, Part3 과 동일한 "전/후 Nag 패턴"을 띤다. 하지만, 이 프로그램은 10번만 사용할 수 있는 쉐어웨어이다.

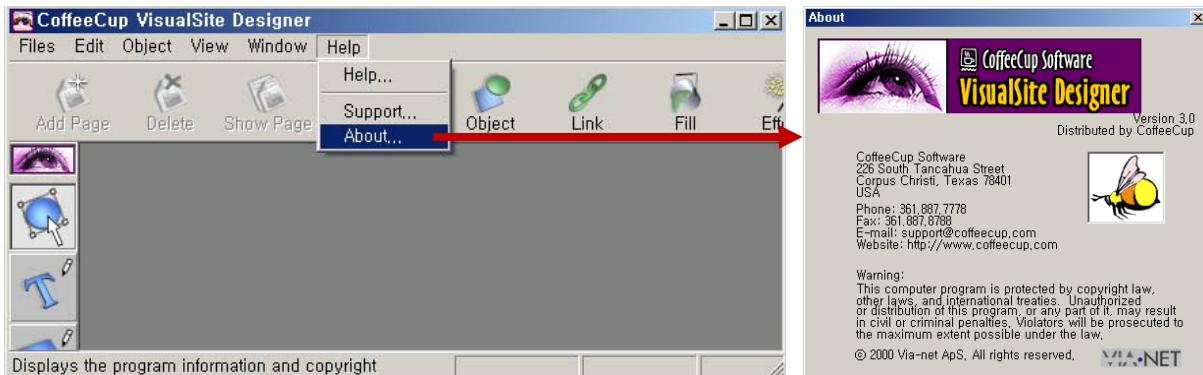


(그림 5-1)

(그림 5-1)의 첫 Nag Screen의 (1)번 영역에서 사용 가능한 횟수를 표시하고, (2)번은 "Buy Now" 버튼으로 인터넷으로 구매할 수 있는 사이트로 연결되며, (3)번은 "Start Program" 버튼으로 사용 가능한 횟수를 1만큼 빼면서 본 프로그램이 시작되고, 본 프로그램이 종료되면 마지막 Nag Screen 을 띠운다.

여기에서 하나의 취약점이 있다. 눈치 빠른 독자라면 이미 눈치를 챘겠지만, 첫 Nag 에서 "Start Program" 버튼을 클릭하지 않고, "Enter"를 치면, 사용 가능한 카운트의 차감 없이 본 프로그램으로 곧바로 진입한다. Nag 가 불편하지 않은 독자라면 굳이 패치할 필요가 없겠으나, 우리의 목적은 이런 패턴의 프로그램에 대해서 공부하는데 있으므로 패치를 해 보자.

이번 역시 패치해야 하는 포인트를 찾아야 하므로, 본 프로그램에서 정품 등록이 있는지 확인해 보자.



(그림 5-2)

"Help" 메뉴에서 "Support"는 개발사 홈페이지로 이동하게 되어 있고, "About"은 단순히 프로그램 정보만을 보여주고 있다. 그러므로, 시리얼 키를 찾는 것 보다는 Nag 를 없애고, 사용 가능한 카운트가 줄어드는 것을 없애는 방향으로 패치해야 한다.

먼저, 저자는 사용 가능한 카운트가 줄어드는 것을 막는 부분을 찾기 위해서 10번의 카운트를 모두 소진했고, (그림 5-3)과 같이 "더 이상 사용할 수 없다."는 에러 메시지를 만나게 되었다.



(그림 5-3)

10번의 수치를 판단하는 조건식을 찾아서, Ollydbg 로 분석을 시작해 보자.

004BD444	> p46	INC ESI MOV [LOCAL.29],ESI MOV AL,BYTE PTR DS:[ESI] CMP AL,BL JE SHORT 004BD452 CMP AL,22 JNZ SHORT 004BD444	path info by superdk
004BD445	.	8975 8C	
004BD448	.	8A06	
004BD449	.	3AC3	
004BD44C	▼	74 04	
004BD44E	.	3C 22	
004BD450	^	75 F2	
ESI=00211F1A, (ASCII ":\\Lena\\Levels\\VisualSite Designer.exe")			

(그림 5-4-1) : 파일의 전체 경로를 가져온다.

004BD48F	.	53	PUSH EBX	pModule = NULL
004BD490	.	FF15 C4C14C00	CALL DWORD PTR DS:[&KERNEL32.GetModuleHandleA]	GetModuleHandleA
004BD496	.	50	PUSH EAX	VisualSi.00400000
<b>004BD497</b>	.	E8 74000000	CALL 004BD510	call Nag & program by superdk
004BD49C	.	8945 98	MOU [LOCAL.261],EAX	VisualSi.00400000
004BD49F	.	50	PUSH EAX	status = 400000 (4194304.)
004BD4A0	.	FF15 2CCC4C00	CALL DWORD PTR DS:[&MSVCRT.exit]	exit

(그림 5-4-2) : Nag를 포함한 프로그램 호출이 시작된다.

여기에서는 "F7"을 눌러서 call 안으로 들어가야만 분석을 계속할 수 있다.

004BD510	-\$	FF7424 10	PUSH DWORD PTR SS:[ESP+10]	VisualSi.004BD49C
004BD514	.	FF7424 10	PUSH DWORD PTR SS:[ESP+10]	VisualSi.004BD49C
004BD518	.	FF7424 10	PUSH DWORD PTR SS:[ESP+10]	VisualSi.004BD49C
004BD51C	.	FF7424 10	PUSH DWORD PTR SS:[ESP+10]	VisualSi.004BD49C
<b>004BD520</b>	.	E8 43000000	CALL <JMP.&MFC42.#1576>	

(그림 5-4-3) : 역시 call 안으로 들어간다.(MFC 로 들어간다.)

6B8B348A	8B16	MOV EDX,DWORD PTR DS:[ESI]	VisualSi.004D81A0
6B8B348C	8B42 58	MOV EAX,DWORD PTR DS:[EDX+58]	VisualSi.00489310
6B8B348F	8BCE	MOV ECX,ESI	VisualSi.0058B230
<b>6B8B3491</b>	FFD0	CALL EAX	VisualSi.00489310
6B8B3493	85C0	TEST EAX,EAX	VisualSi.00489310
6B8B3495	^ 0F84 23A60000	JF 6B8BDABE	VisualSi.00489310

(그림 5-4-4) : 역시 call 안으로 들어간다.

0048951B	> 8DB7 E8000000 LEA ESI, DWORD PTR DS:[EDI+E8]		
00489521	. 68 94715100 PUSH 00517194	ASCII "ftp_lock_name"	VisualSi.0058B318
00489526	. 8BCF MOV ECX,ESI		
00489528	. C605 6CB35800 MOV BYTE PTR DS:[58B36C1],0		
0048952F	. E8 303A0300 CALL <JMP.&MFC42.#913>		
00489534	. 50 PUSH EAX		
00489535	. B9 78B35800 MOV ECX,0058B378		
0048953A	. E8 A9310300 CALL <JMP.&MFC42.#858>		
0048953F	. 68 80715100 PUSH 00517180	ASCII "ftp_default_value"	VisualSi.0058B318
00489544	. 8BCF MOV ECX,ESI		
:			
00489604	. 68 70715100 PUSH 00517170	fileName = "Resources.dll"	
00489609	. FF15 A4C14C00 CALL DWORD PTR DS:[<&KERNEL32.LoadLibraryA]	LoadLibraryA	
0048960F	. 8BF0 MOV ESI,EAX		
00489611	. 85F6 TEST ESI,ESI		
00489613	. 89B7 C4000000 MOV DWORD PTR DS:[EDI+C4],ESI		
00489619	.^ 74 B2 JE SHORT 004896CD		
0048961B	. E8 C4310300 CALL <JMP.&MFC42.#1168>		
00489620	. 8970 0C MOV DWORD PTR DS:[EAX+C1],ESI		
00489623	. E8 1E390300 CALL <JMP.&MFC42.#6438>		
00489628	. 8D5424 58 LEA EDX,DWORD PTR SS:[ESP+58]		
0048962C	. 52 PUSH EDX		
0048962D	. 68 64715100 PUSH 00517164	fileHandle = ntdll.KiFastSystemCallRet	
00489632	. E8 482D0300 CALL <JMP.&VERSION.GetFileVersionInfoSizeA>	GetFileVersionInfoSizeA	
00489637	. 8BF0 MOV ESI,EAX		
00489639	. 85F6 TEST ESI,ESI		
0048963B	.^ 76 65 JBE SHORT 004896A2		
0048963D	. 56 PUSH ESI		
0048963E	. E8 51300300 CALL <JMP.&MFC42.#823>		
00489643	. 83C4 04 ADD ESP,4		
00489646	. 8BE8 MOV EBP,EAX		
00489648	. 55 PUSH EBP	Buffer = FFFFFFFF	
00489649	. 56 PUSH ESI	BufSize = 7A4 (1956.)	
0048964A	. 6A 00 PUSH 0	Reserved = 0	
0048964C	. 68 64715100 PUSH 00517164	fileName = "Resources"	
00489651	. E8 262D0300 CALL <JMP.&VERSION.GetFileVersionInfoA>	GetFileVersionInfoA	

(그림 5-4-5) : DLL 등 필요한 환경을 읽어 들인다.

004898C5	. E8 58360300 CALL <JMP.&MFC42.#5503>		VisualSi.0058B230
004898CA	. 8BCF MOV ECX,EDI		
004898CC	. E8 AFF0FFFF CALL 00488980		
004898D1	. 84C0 TEST AL,AL		
004898D3	.^ 0F84 FF000000 JE 004899D8		
004898D9	. 8A87 E0000000 MOV AL,BYTE PTR DS:[EDI+E0]		
004898DF	. 84C0 TEST AL,AL		
004898E1	.^ 0F85 42010000 JNZ 00489A29		
004898E7	. 8B87 E4000000 MOV EAX,DWORD PTR DS:[EDI+E4]		
004898ED	. 6A 00 PUSH 0		
004898EF	«1» 85C0 TEST EAX,EAX	count by superdk	
004898F1	«2» 0F8E A100000 JLE 00489998		
004898F7	. 8D8C24 100201 LEA ECX,DWORD PTR SS:[ESP+210]		
004898FE	. E8 6D240200 CALL 004ABD70		
00489903	. 8D8C24 0C0201 LEA ECX,DWORD PTR SS:[ESP+20C]		
0048990A	. C68424 788701 MOV BYTE PTR SS:[ESP+8778],0B		
00489912	«3» E8 132B0300 CALL <JMP.&MFC42.#2514>		
00489917	. 83F8 01 CMP EAX,1		
0048991A	.^ 74 40 JE SHORT 00489950		
0048991C	. 8BCF MOV ECX,EDI		VisualSi.0058B230
0048991E	. E8 0DF6FFFF CALL 00488F30		

(그림 5-4-6)

(그림 5-4-6)의 (1)번 라인에서 "TEST EAX,EAX"를 실행할 때, EAX에는 첫 번째 Nag의 좌측 하단에 있는 Count 가 들어있다. 즉, 이 조건식의 결과값을 가지고, (2)번 라인에서 첫 번째 Nag를 (그림 5-1)로 보여줄 지, (그림 5-3)으로 보여줄지를 결정한다. 그리고, (3)번 라인에서 Nag를 띄운다.

004898C5	. E8 58360300 CALL <JMP.&MFC42.#5503>		VisualSi.0058B230
004898CA	. 8BCF MOV ECX,EDI		
004898CC	. E8 AFF0FFFF CALL 00488980		
004898D1	. 84C0 TEST AL,AL		
004898D3	.^ 0F84 FF000000 JE 004899D8		
004898D9	. 8A87 E0000000 MOV AL,BYTE PTR DS:[EDI+E0]		
004898DF	. 84C0 TEST AL,AL		
004898E1	.^ 0F85 42010000 JNZ 00489A29		
004898E7	. 8B87 E4000000 MOV EAX,DWORD PTR DS:[EDI+E4]		
004898ED	. 6A 00 PUSH 0		
004898EF	«1» 85C0 TEST EAX,EAX	count by superdk	
004898F1	«2» 90 NOP		
004898F2	90 NOP		
004898F3	90 NOP		
004898F4	90 NOP		
004898F5	90 NOP		
004898F6	90 NOP		
004898F7	. 8D8C24 100201 LEA ECX,DWORD PTR SS:[ESP+210]		
004898FE	. E8 6D240200 CALL 004ABD70		
00489903	. 8D8C24 0C0201 LEA ECX,DWORD PTR SS:[ESP+20C]		
0048990A	. C68424 788701 MOV BYTE PTR SS:[ESP+8778],0B		
00489912	«3» E8 132B0300 CALL <JMP.&MFC42.#2514>		
00489917	. 83F8 01 CMP EAX,1		
0048991A	.^ 74 40 JE SHORT 00489950		
0048991C	. 8BCF MOV ECX,EDI		VisualSi.0058B230

(그림 5-4-7)

(그림 5-4-7)과 같이 “더 이상 실행할 수 없다”는 Nag를 띄우는 “JLE 00489998”을 없애면, 카운트 “0”이 되어서도 실행된다.



(그림 5-4-8)

물론, “Homepage Creator”가 띄우는 팝업도 제거할 수 있겠지만 (그림 5-4-8)은 Reversing 에는 여러 관점이 있음을 보여주는, 분석에 대한 공부를 위한 것이었을 뿐이며, 올바른 방법은 앞/뒤로 뜨는 Nag 를 제거하는 것이다. 이제 올바른 방법으로 처리를 해 보자.

004898C5	.	E8 58360300	CALL <JMP.&MFC42.#5503>
004898CA	.	8BCF	MOV ECX,EDI
004898CC	.	E8 AFF0FFFF	CALL 00488980
004898D1	.	84C0	TEST AL,AL
004898D3	✓	0F84 FF00000	JE 004899D8
004898D9	(1)	BA87 E000000	MOV AL,BYTE PTR DS:[EDI+E0]
004898DF	(2)	84C0	TEST AL,AL
004898E1	(3)	0F85 4201000	JNZ 00489A29
004898E7	.	8B87 E400000	MOV EAX,DWORD PTR DS:[EDI+E4]
004898ED	.	6A 00	PUSH 0
004898EF	.	85C0	TEST EAX,EAX
004898F1	✓	0F8E A100000	JLE 00489998
004898F7	.	8D8C24 10020	LEA ECX,DWORD PTR SS:[ESP+210]
004898FE	.	E8 6D240200	CALL 004ABD70
00489903	.	8D8C24 0C020	LEA ECX,DWORD PTR SS:[ESP+20C]
0048990A	.	C68424 78870	MOV BYTE PTR SS:[ESP+8778],0B
00489912	.	E8 132B0300	CALL <JMP.&MFC42.#2514>
00489917	.	83F8 01	CMP EAX,1
0048991A	✓	74 40	JE SHORT 00489950
0048991C	.	8BCF	MOV ECX,EDI
0048991E	.	E8 0DF6FFFF	CALL 00488F30

(그림 5-4-9)

(그림 5-4-9)를 보면 본 프로그램 전에 뜨는 Nag 와 관계된 조건절이 있다. (1)번 라인에서 “EDI+E0”에 있는 값을 AL 에 넣고, (2)번 라인에서 “TEST AL,AL”을 수행하는 것을 볼 수 있다. 하지만, 아래의 (그림 5-4-10)과 같이 “EDI+E0”에 해당하는 “00583B310=00583B230+E0”에 들어있는 값이 “0”이라는 것을 알 수 있다. 즉, “TEST 0,0” 은 항상 “0”이므로 (3번)라인의 “JNZ 00489A29”에서 점프하지 못하여 항상 Nag가 뜨는 것이다.

Address	Hex dump	ASCII
0058B310	00 00 00 00 00 00 00 00 6C D0 DF 5D 68 71 97 00	.....L...?h?
0058B320	11 00 00 00 02 00 00 00 2C 86 97 00 08 86 97 00	.....e...,.u..u.
0058B330	0A 00 00 00 B0 A7 ED 5D A8 34 BD 00 18 33 BD 00	.....=??+3?
0058B340	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

(그림 5-4-10)

그렇다면, (그림 5-4-11)과 같이 코드를 역할에 따라 구분할 수 있다. (A)로 표시된 영역이 시작 Nag 를 띄우는 코드이고, 지금까지 10번 이상을 사용했다면, “더 이상 사용할 수 없다”는 내용의 메시지 창을, 10번 미만으로 사용했다면 잔여 카운트 수와 함께 우리가 없애야 할 Nag 창을 (4)번 라인에서 호출한다.

그러므로, 우리는 (3)번 라인인 “JNZ 00489A29”를 “JMP 00489A29”로 바꾸면 시작 Nag 창을 제거할 수 있다.

004898C5	. E8 58360300	CALL <JMP.&MFC42.#5503>	VisualSi.0058B230
004898CA	. 8BCF	MOV ECX,EDI	
004898CC	. E8 AFF0FFFF	CALL 00488980	
004898D1	. 84C0	TEST AL,AL	
004898D3	.> 0F84 FF000000	JE 004899D8	
004898D9	(1) 8A87 E0000000	MOV AL,BYTE PTR DS:[EDI+E0]	
004898E0	(2) 84C0	TEST AL,AL	
004898E1	(3) 0F85 42010000	JNZ 00489A29	
004898E7	. 8B87 E4000000	MOV EAX,DWORD PTR DS:[EDI+E4]	
004898ED	. 6A 00	PUSH 0	count by superdk
004898EF	. 85C0	TEST EAX,EAX	
004898F1	.> 0F8E A1000000	JLE 00489998	
004898F7	. 8D8C24 100200	LEA ECX,DWORD PTR SS:[ESP+210]	
004898FE	. E8 6D240200	CALL 004AB070	<a>
00489903	. 8D8C24 0C0200	LEA ECX,DWORD PTR SS:[ESP+20C]	
0048990A	. C68424 788700	MOV BYTE PTR SS:[ESP+87781],0B	
00489912	(4) E8 132B0300	CALL <JMP.&MFC42.#2514>	
00489917	. 83F8 01	CMP EAX,1	
0048991A	.> 74 40	JE SHORT 0048995C	
0048991C	. 8BCF	MOV ECX,EDI	
0048991E	. E8 00F6FFFF	CALL 00488F30	VisualSi.0058B230

(그림 5-4-11)

(그림 5-4-12) 와 같이 실제로 바꿔서 실행해 보자.

004898D9	. 8A87 E0000000	MOV AL,BYTE PTR DS:[EDI+E0]	
004898E0	. 84C0	TEST AL,AL	
004898E1	(1) 8A87 E9 43010000	JMP 00489A29	modified by superdk
004898E6	90	NOP	
004898E7	8B87 E4000000	MOV EAX,DWORD PTR DS:[EDI+E4]	

(그림 5-4-12)

오타 없이 정확하게 수정하였다면, 본 프로그램 앞에 뜨던 Nag 가 없어질 것이다.

이제는 본 프로그램이 끝날 때 나오는 웹페이지 모양의 Nag 를 없애야 하는데, 이 Nag 를 제거하는 것이 독자에게는 쉽지 않을 것이다. 왜냐하면, 지금까지 했던 것처럼 "Step by Step" 형태로 따라가서는 쉽게 없앨 수 없기 때문이다. 한 라인씩 침착하게 따라가는데도 쉽게 없앨 수 없는 이유는 무엇일까? 첫째는 너무 많은 라인이 있고, 둘째는 첫 번째 Nag를 없앴던 것과는 달리 앞에서 이미 만들어 뒀기 때문이다. 마지막으로, Windows 는 이벤트가 callback 으로 동작하기 때문에 순차적으로 따라가서는 바이너리상의 어디에서 해당 이벤트가 호출될 지 감으로 잡기가 쉽지 않기 때문이다.

그래서, 본 프로그램을 종료하고 웹페이지 형태의 Nag 가 떴을 때, Ollydbg를 Pause 시킨 후, Call stack(단축 아이콘 : K, 단축키 : Alt+K)을 호출해서 어디에서 해당 Nag를 띄우는지를 확인하는 것이 빠르다.

Call stack of main thread					
Address	Stack	Procedure / arguments	Called from	Frame	
0012F094	76868F8F	Includes ntdll.KiFastSystemCallRet	USER32.76868F8D	0012F0B8	
0012F098	768628AF	USER32.768628AA	USER32.768628AA	0012F0B8	
0012F0B0	50095398	USER32.GetMessageA	MFC42.50095395	0012F0B8	
0012F0C0	0058B264	pMsg = VisualSi.0058B264			
0012F0C4	00000000	hWnd = NULL			
0012F0C8	00000000	MsgFilterMin = 0			
0012F0CC	00000000	MsgFilterMax = 0			
0012F0D8	500AEB1A	Includes MFC42.5009539B	MFC42.500AEB18	0012F0F4	
0012F0F8	500C1640	MFC42.#5718	MFC42.500C163B	0012F0F4	
0012F13C	004880C29	? <JMP.&MFC42.#2514>	VisualSi.00480C24 (1)	0012F138	
0012F1AC	5008A5A1	Includes VisualSi.00480C29	MFC42.5008A59F	0012F228	
0012F22C	50083687	Includes MFC42.5008A5A1	MFC42.50083685	0012F228	
0012F254	5008A361	Includes MFC42.50083687	MFC42.5008A35F	0012F250	
0012F2BC	5008A2B9	MFC42.#1109	MFC42.5008A2B4	0012F2B8	
0012F2E0	5008A571	MFC42.#1578	MFC42.5008A56C	0012F2DC	
0012F2E4	004D0FCE	Arg1 = 004D0FCE			
0012F2E8	00000002	Arg2 = 00000002			
0012F2EC	00000000	Arg3 = 00000000			
0012F2F0	00000000	Arg4 = 00000000			
0012F314	768686EF	Includes MFC42.5008A571	USER32.768686EC	0012F310	
0012F340	76868876	? USER32.768686CC	USER32.76868871	0012F33C	
0012F3B8	768670F4	? USER32.768687C3	USER32.768670EF	0012F3B4	

(그림 5-4-13)

(그림 5-4-13)의 (1)번 라인만이 유일하게 VisualSite에서 호출한 함수이다. 그러므로, 마지막 Nag를 띄운 가장 유력한 호출 지점으로 보인다. (1)번 라인을 더블클릭해서 해당하는 코드로 이동해 보자.

```

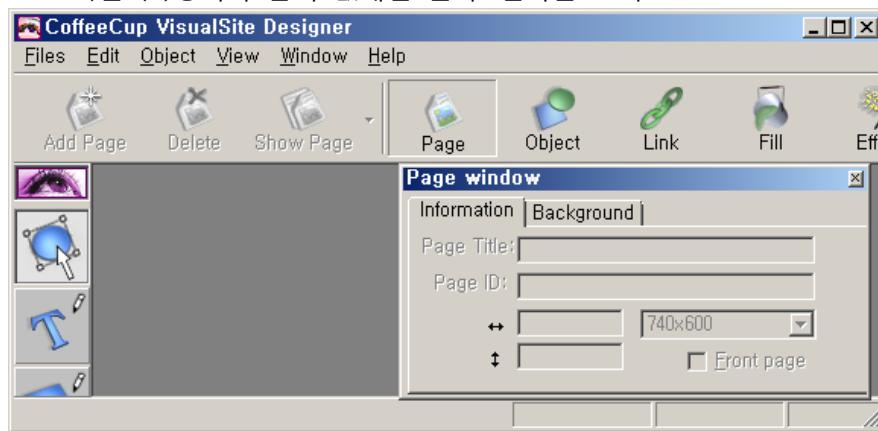
00480BFE . 64:8925 000000 MOU DWORD PTR FS:[0],ESP
00480C05 . 83EC 60 SUB ESP,60
00480C08 . E8 37C20300 CALL <JMP.&MFC42.#4501>
00480C0D . 6A 00 PUSH 0
00480C0F . 804C24 04 LEA ECX,DWORD PTR SS:[ESP+4]
00480C13 . E8 2833EFF CALL 00463F40
00480C18 . 804C24 00 LEA ECX,DWORD PTR SS:[ESP]
00480C1C . C74424 68 0000 MOU DWORD PTR SS:[ESP+68],0
00480C24 . E8 01B80300 CALL <JMP.&MFC42.#2514> <1>
00480C29 . 804C24 00 LEA ECX,DWORD PTR SS:[ESP]
00480C2D . C74424 68 FFFF MOU DWORD PTR SS:[ESP+68],-1
00480C35 . E8 DEBA0300 CALL <JMP.&MFC42.#641>
00480C3A . 804C24 60 MOU ECX,DWORD PTR FS:[0],ECX
00480C3E . 64:8900 000000 MOU DWORD PTR FS:[0],ESP
00480C45 . 83C4 6C ADD ESP,6C
00480C48 . C3 RETN

00480BFE . 64:8925 000000 MOU DWORD PTR FS:[0],ESP
00480C05 . 83EC 60 SUB ESP,60
00480C08 . E8 37C20300 CALL <JMP.&MFC42.#4501>
00480C0D . 6A 00 PUSH 0
00480C0F . 804C24 04 LEA ECX,DWORD PTR SS:[ESP+4]
00480C13 . E8 2833EFF CALL 00463F40
00480C18 . 804C24 00 LEA ECX,DWORD PTR SS:[ESP]
00480C1C . C74424 68 0000 MOU DWORD PTR SS:[ESP+68],0
00480C24 . 90 NOP
00480C25 . 90 NOP <A>
00480C26 . 90 NOP
00480C27 . 90 NOP
00480C28 . 90 NOP
00480C29 . 804C24 00 LEA ECX,DWORD PTR SS:[ESP]
00480C2D . C74424 68 FFFF MOU DWORD PTR SS:[ESP+68],-1

```

(그림 5-4-14)

(그림 5-4-14)의 (1)번 라인에서 마지막의 웹페이지 형태의 Nag를 호출한다. 그러므로, (1)번 라인("CALL 004BC42A")을 (A)영역과 같이 없애면 된다. 결과를 보자.



(그림 5-4-15)

(그림 5-4-15)와 같이 본 프로그램의 앞/뒤로 뜨던 Nag Screen이 사라진 것을 볼 수 있다.

#### \* 도움이 된 참고 사이트

항 목	참 고 주 소
OllyDbg 사용법	<a href="http://www.tuts4you.com/download.php?list.29">http://www.tuts4you.com/download.php?list.29</a>
PE Header	<a href="http://win32assembly.online.fr/pe-tut1.html">http://win32assembly.online.fr/pe-tut1.html</a> ~ ~ <a href="http://win32assembly.online.fr/pe-tut7.html">pe-tut7.html</a>