

POC2008 Hacker'S Dream reversing 2 solution

Author: [LEONY] email: codexb@gmail.com

2008. 10

Contents

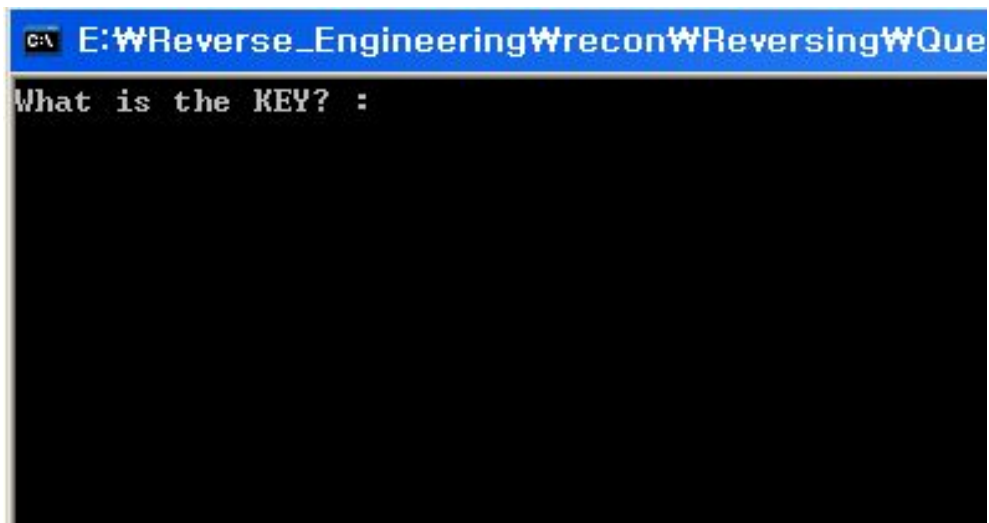
Chapter 1. 문제 소개

Chapter 2. 문제 풀이

Chapter 1. 문제 소개

POC2008 대회에서의 리버싱 2번 문제 입니다.

다음과 같이 바이너리를 실행하였을 때 KEY 값을 물어보는 것으로 보아 이 문제의 미션은 KEY 값을 찾고 있습니다.



[그림 1] 바이너리 실행 화면

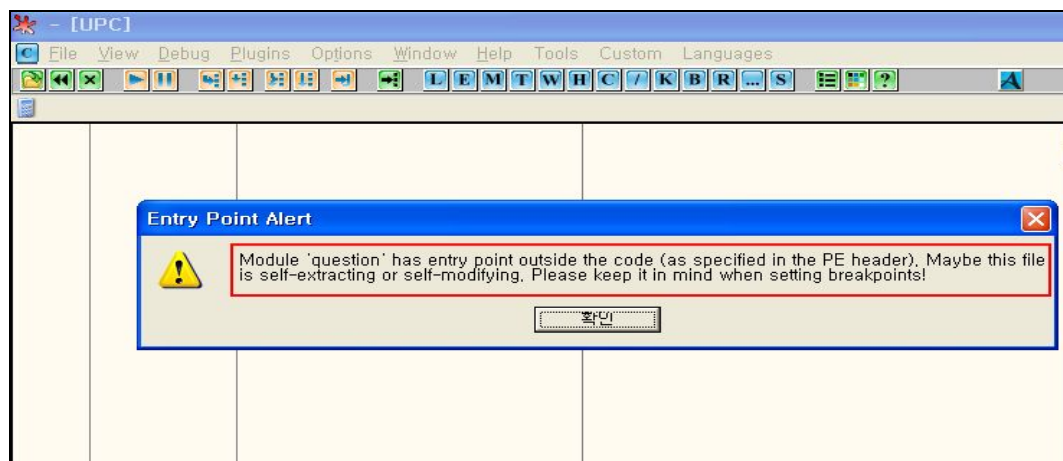
[파일 정보]

- FILENAME : **question2.exe**
- PACKING? : IT IS PACKED with **unknown packer** (-□-;;)
- Compiler : **Visual C++ 5.0 ~ 6.0**

Chapter 2. 문제 풀이

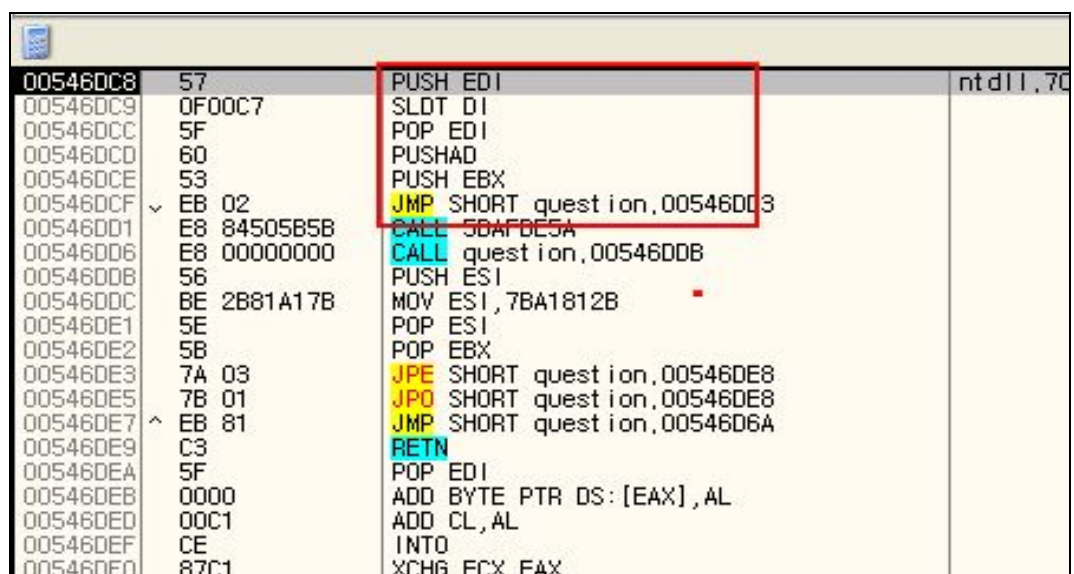
Question2.exe 파일을 OLLYDBG를 이용하여 오픈하면 다음과 같이 **Entry Point Alert** 라는 경고 창을 띄우고 있다.

이것으로 파일이 PACKER 에 의해 보호되고 있는 걸 알 수 있다.



[그림 2] 올리디버거 Entry Point 변경 알림

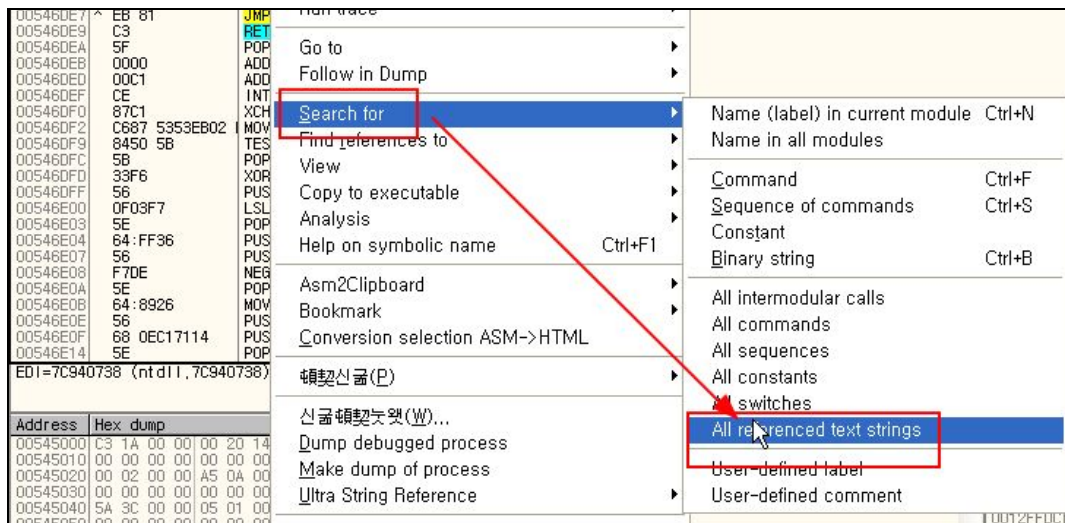
확인 버튼을 클릭하면 실행파일의 처음 명령어 시작 위치인 EP(Entry Point) 부분이 나오는데 정상적인 컴파일러의 EP 시그내처가 아니고 **패커의 EP** 부분이 나온다.



[그림 3] Packer의 Entry Point

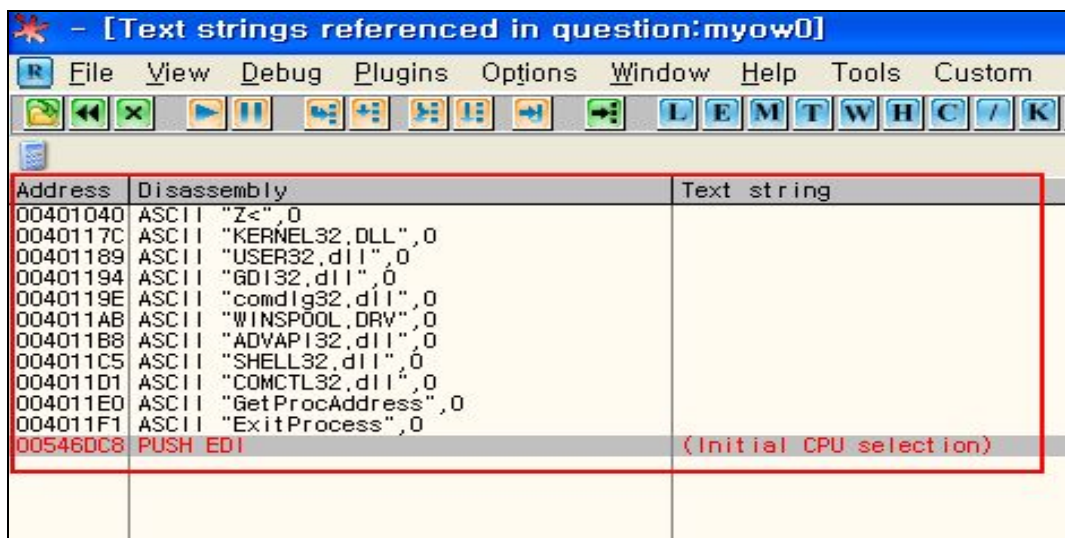
일반적으로 패킹되어 있지 않은 바이너리는 바이너리 안의 텍스트 정보를 확인할 수 있어 문제를 해결하는데 도움이 될 수 있으나, 패킹에 의해 보호된 실행파일일 경우 언팩하기 전에는 바이너리의 텍스트 정보를 확인하기가 어렵다. 패킹된 파일을 구별할 때 전형적인 형태라고 생각할 수 있다.

다음은 OLLYDBG의 All refenced text strings 의 텍스트 뷰 기능을 이용했을 때 결과 화면이다.



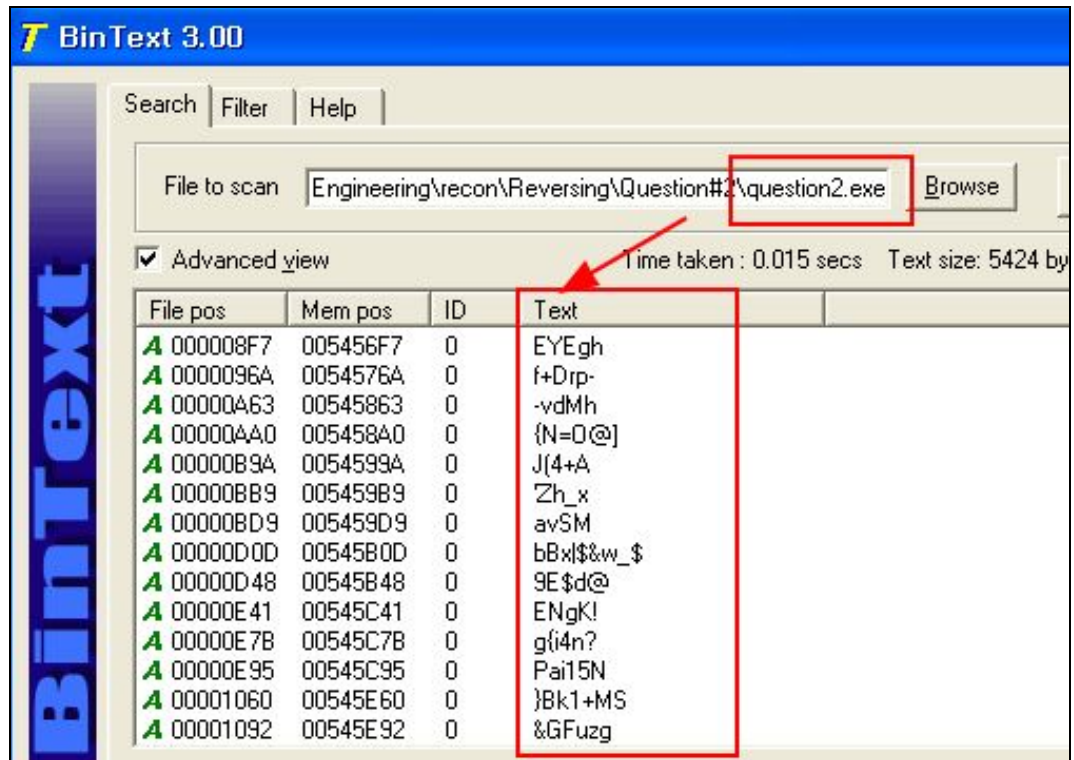
[그림 4] 바이너리의 텍스트 정보 확인

다음과 같이 DLL 과 API 외에는 다른 정보가 보이지 않는다.



[그림 5] 바이너리의 텍스트 정보

마찬가지로 Bintext 툴을 이용하여도 원래의 텍스트 정보를 알기가 어렵다.



[그림 6] bintext 툴을 이용한 바이너리 텍스트

Entry Point Alert 및 보이지 않거나 인코딩된 텍스트는 패킹된 파일의 전형적인 모습이라 할 수 있다.

바이너리를 언팩하여 보자.

Unpacking 하는 방법은 여러가지가 있을 수 있는데 본 문서에서는 다음의 순서로 Unpacking을 수행하였다.

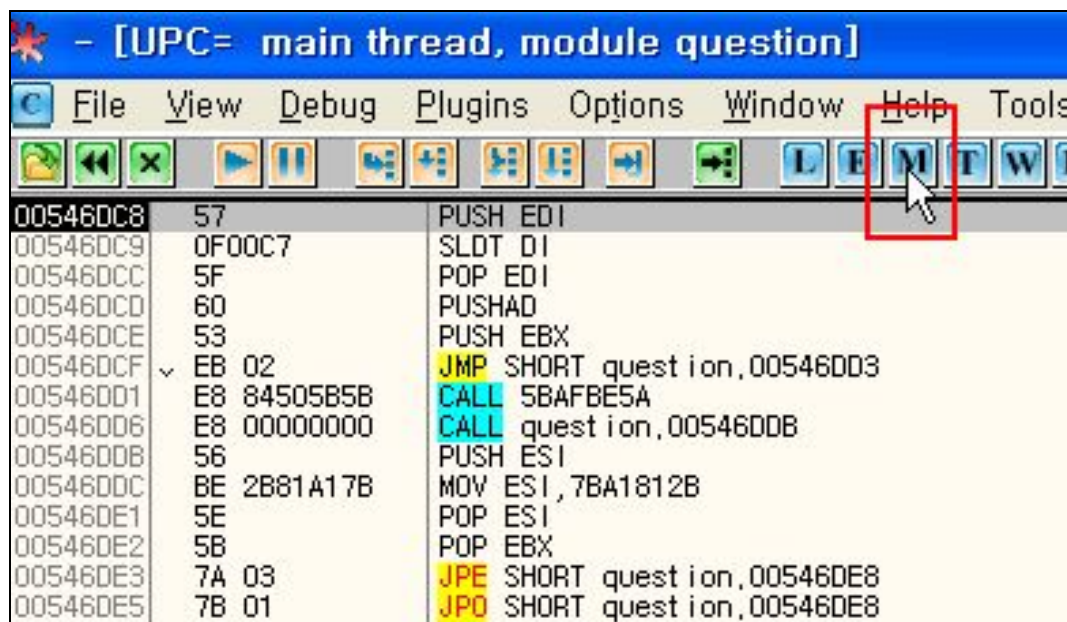
1. 실행파일이 풀릴 CODE 영역에 BreakPoint 설정
2. 스택을 이용한 리턴주소 유추하여 OEP 근접하기
3. OEP 점프 문 찾기

이외에도 무한 삽질로 OEP 근접하기, LoadLibraryA 함수 에서 브레이크 포인트 설정 후 OEP 근접하기, 실행 파일 실행 후 어태치로 붙인 후 컴파일러 시그내처로 찾기 등등이 있을 수 있겠다.

위의 첫번째 방법을 이용하여 언팩하여 보자.

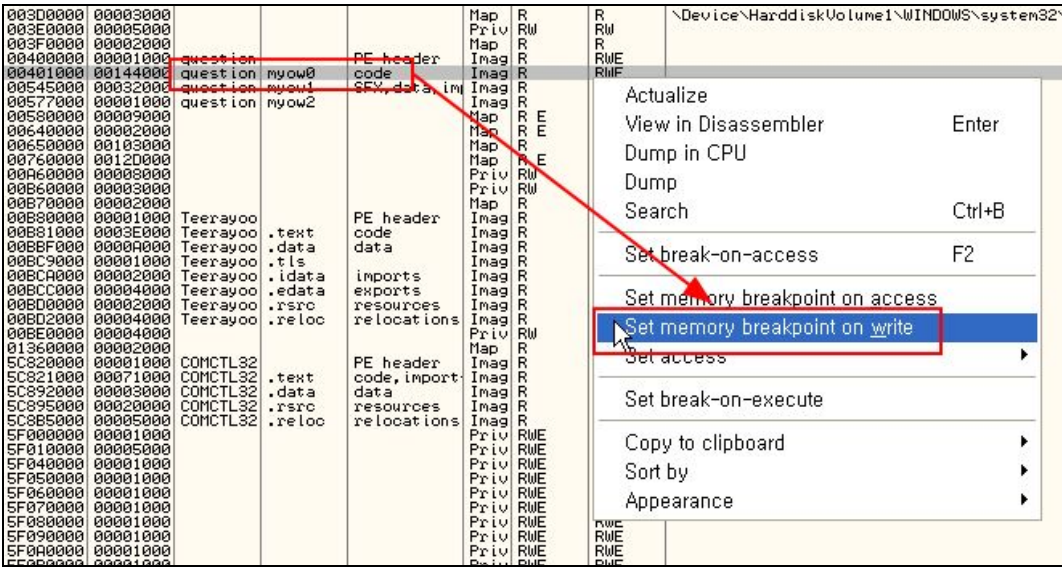
1. 실행파일이 풀릴 CODE 영역에 BreakPoint 설정

일단 바이너리의 CODE영역을 보기 위해 OLLYDBG의 단축 링크인 M을 클릭한다.



[그림 7] MEMORY 확인

다음과 같은 화면을 볼 수 있으며, 본 문제의 패커는 오리지널 바이너리를 메모리에 풀 때 code영역에 풀기 때문에 Set memory breakpoint on write 기능을 이용하여 code 영역에 어떤 데이터 값을 쓸 때 브레이크 포인트가 걸릴 수 있도록 설정한다.



[그림 8] CODE 영역 브레이크 포인트 설정

브레이크포인트 설정 후 F9 를 눌러 프로그램을 달리면 아래 위치에서 실행이 중단된다.

00CE0273 880429 MOV BYTE PTR DS:[ECX+EBP],AL

이 부분의 위치가 패커가 코드영역에 원래의 바이너리를 푸는 루틴이다.

00CE0273	880429	MOV BYTE PTR DS:[ECX+EBP],AL	
00CE0276	45	INC EBP	
00CE0277	3B6C24 60	CMP EBP,DWORD PTR SS:[ESP+60]	
00CE027B	0F02 20FFFFFF	JB 00CE01A1	
00CE0281	5F	POP EDI	
00CE0282	5E	POP ESI	
00CE0283	5D	POP EBP	
00CE0284	33C0	XOR EAX,EAX	
00CE0286	5B	POP EBX	
00CE0287	83C4 2C	ADD ESP,2C	
00CE028A	C3	RETN	
00CE028B	8D5424 28	LEA EDX,DWORD PTR SS:[ESP+28]	
00CE028F	8D845F 8001000	LEA EAX,DWORD PTR DS:[EDI+EBX*2+180]	
00CE0296	52	PUSH EDX	
00CE0297	50	PUSH EAX	
00CE0298	C74424 20 0100	MOV DWORD PTR SS:[ESP+20],1	
00CE02A0	E8 6F020000	CALL 00CE0514	
00CE02A5	83C4 08	ADD ESP,8	
00CE02A8	83F8 01	CMP EAX,1	
00CE02AB	0F85 E6000000	JNZ 00CE0397	
00CE02B1	8D4C24 28	LEA ECX,DWORD PTR SS:[ESP+28]	
00CE02B5	8D945F 9801000	LEA EDX,DWORD PTR DS:[EDI+EBX*2+198]	
00CE02BC	51	PUSH ECX	
00CE02BD	52	PUSH EDX	
00CE02BE	E8 51020000	CALL 00CE0514	
00CE02C3	83C4 08	ADD ESP,8	
00CE02C6	85C0	TEST EAX,EAX	
00CE02C8	8D4424 28	LEA EAX,DWORD PTR SS:[ESP+28]	
00CE02CC	50	PUSH EAX	
00CE02CD	75 46	JNZ SHORT 00CE0315	
00CE02CF	8D5424 54	MOV EDX,DWORD PTR SS:[ESP+54]	
00CE02D3	8D4C24 28	LEA ECX,DWORD PTR SS:[ESP+28]	
AL=CC			
DS:[00401000]=C3			
Address	Hex dump	ASCII	
00545000	C3 1A 00 00 00 20 14 00 00 00 40 00 00 00 00 00	?... 1A...@.....	
00545010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 CE 00?.....	
00545020	00 02 00 00 A5 0A 00 00 00 00 00 00 00 00 00 00	...?.....	
00545030	00 00 00 00 00 00 00 00 00 00 00 00 00 10 40 00+@.	
00545040	5A 3C 00 00 05 01 00 00 00 00 00 00 00 00 00 00	Z<... r.....+@.?	
00545050	00 00 00 00 00 00 00 00 00 00 10 40 00 AF F7 02 00+@.?	
00545060	00 00 00 00 77 10 80 7C C0 AD 80 7C D0 1A 80 7C	...w□ 윗□ ?□	
00545070	71 9A 80 7C 04 9B 80 7C 52 5D 83 7C 24 1A 80 7C	q? J? R ?}\$+□	
00545080	A7 0D 81 7C 67 9B 80 7C 82 CA 81 7C 00 00 00 00	??g? 첼?...	
00545090	10 9B 01 77 00 00 00 00 00 A0 E2 77 00 00 00 00	+첼w.....첼w....	
005450A0	CE EE 30 76 00 00 00 00 67 37 F6 72 00 00 00 00	첼0v.....g7?....	
005450B0	1B 76 F5 77 00 00 00 00 C9 14 61 7D 00 00 00 00	+v?....?a}....	
005450C0	CF 65 82 5C 00 00 00 00 00 00 00 00 00 00 00 00	??.....	
005450D0	00 00 00 00 7C 51 14 00 64 50 14 00 00 00 00 00 0에.dP에....	
005450E0	00 00 00 00 00 00 00 00 89 51 14 00 80 50 14 00에에...에	
Command: bo LoadLibraryA			
Memory breakpoint when writing to [00401000]			

[그림 9] 중단 점이 걸린 위치

F8 로 몇번 STEP OVER 하면서 00401000 위치에 어떤 변화가 일어나는지 확인해 보면 hexs 코드를 해당 영역에 덮어씌우는 걸 알 수 있다.

[덮어써지기 전의 00401000 주소]

Address	Disassembly
00401000	G3
00401001	1A00
00401003	0000
00401005	201400
00401008	B2 53
0040100A	14 00
0040100C	0000
0040100E	0000
00401010	0000
00401012	0000
00401014	0000
00401016	0000
00401018	0000
0040101A	0000
0040101C	0000
0040101E	0000
00401020	0002
00401022	0000
00401024	A5
00401025	0A00
00401027	0000
00401029	0000
0040102B	0000
0040102D	0000

[그림 10] before writing

[덮어써진 이후의 00401000]

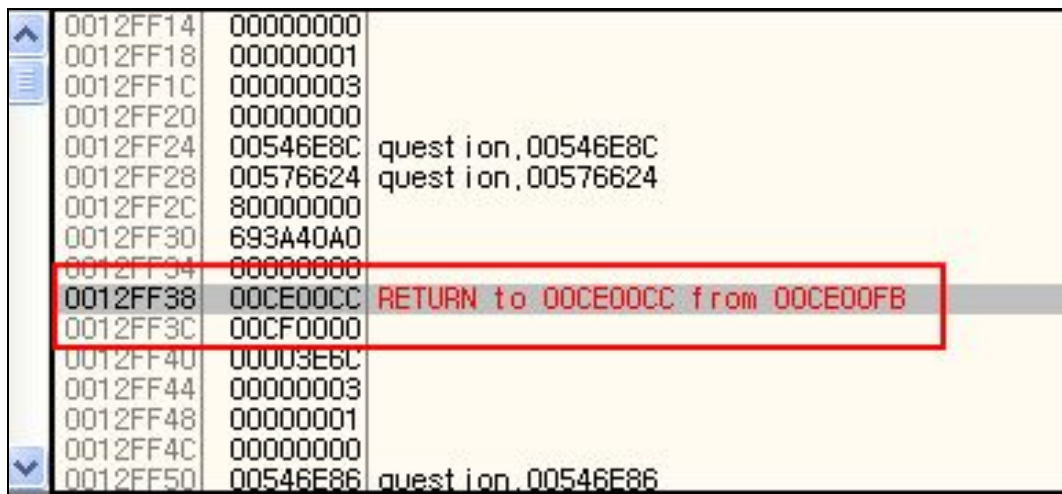
Address	Disassembly
00401000	CC
00401001	CC
00401002	CC
00401003	CC
00401004	CC
00401005	- E9 0500033C
0040100A	- E9 050000AC
0040100F	- E9 0500004C
00401014	- E9 05002D2C
00401019	- E9 050005AC
0040101E	- E9 050004EC
00401023	CC
00401024	CC
00401025	CC
00401026	CC
00401027	CC
00401028	CC
00401029	CC
0040102A	CC
0040102B	CC
0040102C	0000
0040102E	0000

[그림 11] after writing

언팩의 한 과정인 이 과정을 끝내고 밖으로 나오는 위치인 리턴주소를 찾아 그쪽으로 점프하면 OEP에 근접할 수 있다는 걸 추측하여...

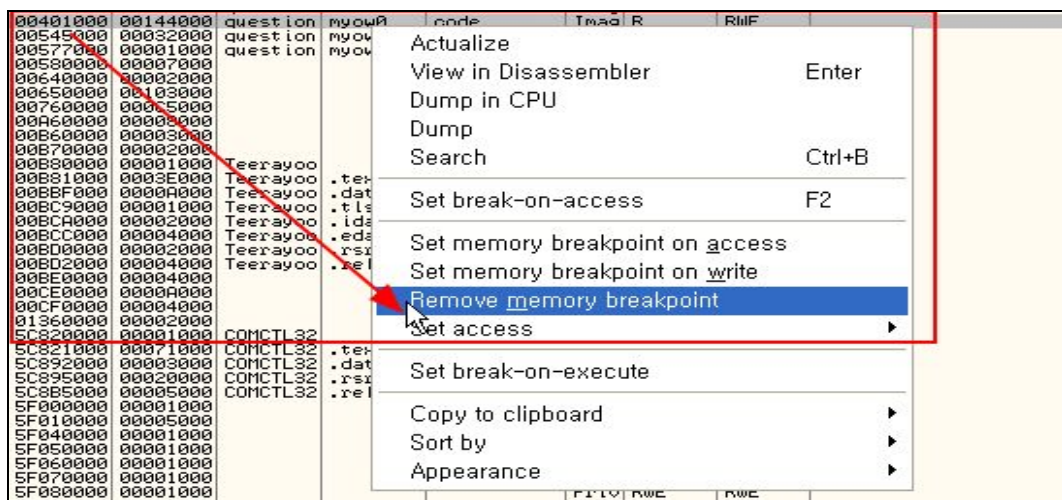
함수 호출 시 삽입한 리턴주소를 스택에서 찾아보자.

다음의 위치가 삽입된 리턴주소 이다. 올리에서 친절하게 RETURN 이라고 표시하여 준다. ^_^



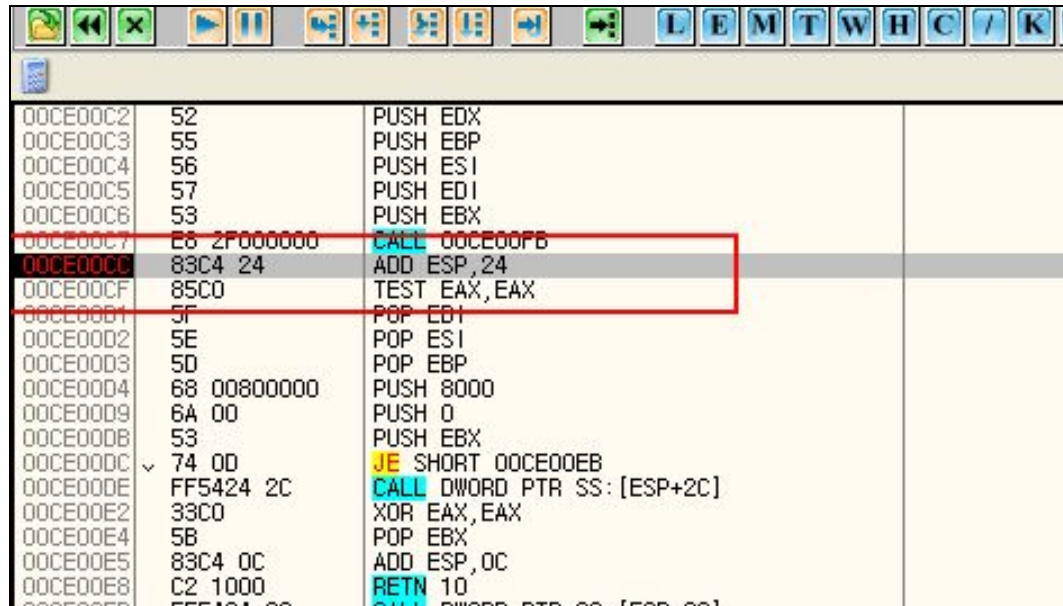
[그림 12] 삽입된 리턴 주소 00CE00CC

코드 영역에 브레이크 포인트를 제거하고 위의 리턴 주소에 브레이크 포인트 설정 후 F9로 달려보자.



[그림 13] CODE영역 브레이크 포인트 제거

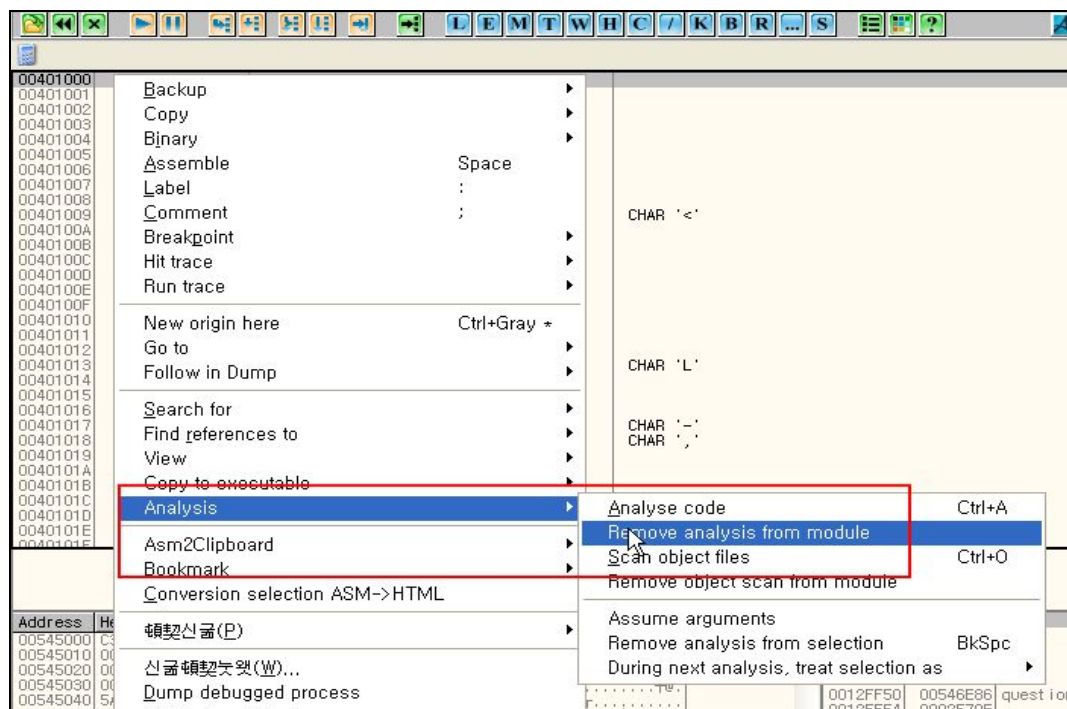
00CE00CC 위치에서 실행이 중단된다.



[그림 14] 00CE00CC로 리턴

[참고]

또한 00401000 주소로 이동하여 코드를 분석하여 보면 풀린 코드를 볼 수 있다.



[그림 15] Analyse code

Address	Hex	Assembly
0040104E	CC	INT3
0040104F	CC	INT3
00401050	55	PUSH EBP
00401051	8BEC	MOV EBP,ESP
00401053	83EC 40	SUB ESP,40
00401056	53	PUSH EBX
00401057	56	PUSH ESI
00401058	57	PUSH EDI
00401059	8D7D C0	LEA EDI,DWORD PTR SS:[EBP-40]
0040105C	B9 10000000	MOV ECX,10
00401061	B8 CCCCCCCC	MOV EAX,CCCCCCCC
00401066	F3:AB	REP STOS DWORD PTR ES:[EDI]
00401068	64:A1 30000000	MOV EAX,DWORD PTR FS:[30]
0040106E	0FB640 02	MOVZX EAX,BYTE PTR DS:[EAX+2]
00401072	0AC0	OR AL,AL
00401074	74 09	JE SHORT question,0040107F
00401076	EB 00	JMP SHORT question,00401078
00401078	B8 01000000	MOV EAX,1
0040107D	EB 02	JMP SHORT question,00401081
0040107F	33C0	XOR EAX,EAX
00401081	5F	POP EDI
00401082	5E	POP ESI
00401083	5B	POP EBX
00401084	83C4 40	ADD ESP,40
00401087	3BEC	CMP EBP,ESP
00401089	E8 05003A2C	CALL 2C7A1093
0040108E	8BE5	MOV ESP,EBP
00401090	5D	POP EBP
00401091	C3	RETN
00401092	CC	INT3

[그림 16] 변경된 코드 영역

00CE00CC 위치에서 다시 리턴 코드까지 실행하여 빠져 나오면 00545661 주소에 위치한다.

Address	Hex	Assembly
0054565A	57	PUSH EDI
0054565B	FF05 6AFCFFFF	CALL DWORD PTR SS:[EBP-396]
00545661	5B	POP EBX
00545662	5A	POP EDX
00545663	59	POP ECX
00545664	5F	POP EDI
00545665	83F9 00	CMP ECX,0
00545668	74 05	JE SHORT question,0054566F
0054566A	83C3 08	ADD EBX,8
0054566D	EB CE	JMP SHORT question,0054563D
0054566F	68 00800000	PUSH 8000
00545674	6A 00	PUSH 0
00545676	FFB5 6AFCFFFF	PUSH DWORD PTR SS:[EBP-396]
0054567C	FF95 C2FCFFFF	CALL DWORD PTR SS:[EBP-33E]
00545682	8DB5 8AFCFFFF	LEA ESI,DWORD PTR SS:[EBP-376]
00545688	8B4E 04	MOV ECX,DWORD PTR DS:[ESI+4]
00545688	8D56 08	LEA EDX,DWORD PTR DS:[ESI+8]
0054568E	8B36	MOV ESI,DWORD PTR DS:[ESI]
00545690	8BFE	MOV EDI,ESI
00545692	83F9 00	CMP ECX,0

[그림 17] 리턴된 위치

밑으로 내려가다 보면 패커가 저장시켜 놓 레지스터 정보와 플래그 정보를 복원하는 코드가 나오고 JMP 00405370 으로 즉 OEP로 점프하는 구문이 나오는 걸 볼 수 있다.

00545797	50	PUSH EAX
00545798	FF95 BAFCEFFF	CALL DWORD PTR SS:[EBP-346]
0054579E	5A	POP EDX
0054579F	5B	POP EBX
005457A0	59	POP ECX
005457A1	5E	POP ESI
005457A2	83C3 0C	ADD EBX,0C
005457A5	^ E2 E1	LOOPD SHORT question,00545788
005457A7	E8 7E020000	CALL question,00545A2A
005457AC	61	POPAD
005457AD	9D	POPFD
005457AE	- E9 BDFBE8FF	JMP question,00405370
005457B3	8B85 52FCFFFF	MOV ESI,DWORD PTR SS:[EBP-3AE]
005457B9	0BF6	OR ESI,ESI
005457BB	^ 0F84 A4000000	JE question,00545865
005457C1	8B95 56FCFFFF	MOV EDX,DWORD PTR SS:[EBP-3AA]
005457C7	03F2	ADD ESI,EDX
005457C9	833E 00	CMP DWORD PTR DS:[ESI],0
005457CC	^ 75 11	JNZ SHORT question,005457DF
005457CE	837E 04 00	CMP DWORD PTR DS:[ESI+4],0

[그림 18] OEP 점프 구문

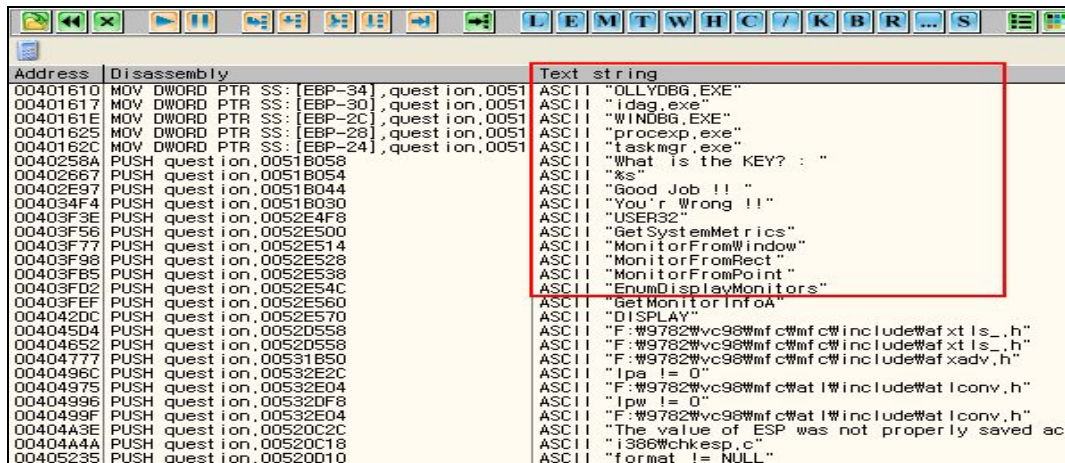
[COMPLETE]

해당 주소로 점프하면 바이너리가 언팩된 OEP 영역이 나온다.

00405370	55	PUSH EBP
00405371	8BEC	MOV EBP,ESP
00405373	6A FF	PUSH -1
00405375	68 300D5200	PUSH question,00520D30
0040537A	68 40514000	PUSH question,00405140
0040537F	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
00405385	50	PUSH EAX
00405386	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
0040538D	83C4 F0	ADD ESP,-10
00405390	53	PUSH EBX
00405391	56	PUSH ESI
00405392	57	PUSH EDI
00405393	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
00405396	FF15 A0E35300	CALL DWORD PTR DS:[53E3A0]
0040539C	A3 9CAB5300	MOV DWORD PTR DS:[53AB9C],EAX
004053A1	A1 9CAB5300	MOV EAX,DWORD PTR DS:[53AB9C]
004053A6	C1E8 08	SHR EAX,8
004053A9	25 FF000000	AND EAX,0FF
004053AE	A3 A8AB5300	MOV DWORD PTR DS:[53ABA8],EAX
004053B3	8B0D 9CAB5300	MOV ECX,DWORD PTR DS:[53AB9C]
004053B9	81E1 FF000000	AND ECX,0FF
004053BF	890D A4AB5300	MOV DWORD PTR DS:[53ABA4],ECX
004053C5	8B15 A4AB5300	MOV EDX,DWORD PTR DS:[53ABA4]
004053CB	C1E2 08	SHL EDX,8
004053CE	0315 A8AB5300	ADD EDX,DWORD PTR DS:[53ABA8]
004053D4	8915 A0AB5300	MOV DWORD PTR DS:[53ABA0],EDX
004053DA	A1 9CAB5300	MOV EAX,DWORD PTR DS:[53AB9C]
004053DF	C1E8 10	SHR EAX,10
004053E2	25 55550000	AND EAX,05555

[그림 19] UNPACKING COMPLETE

언팩 전에 확인할 수 없었던 바이너리가 저장하고 있는 텍스트 정보를 확인할 수 있다. Good Job!!, You're Wrong!! 같은 정보가 보인다.



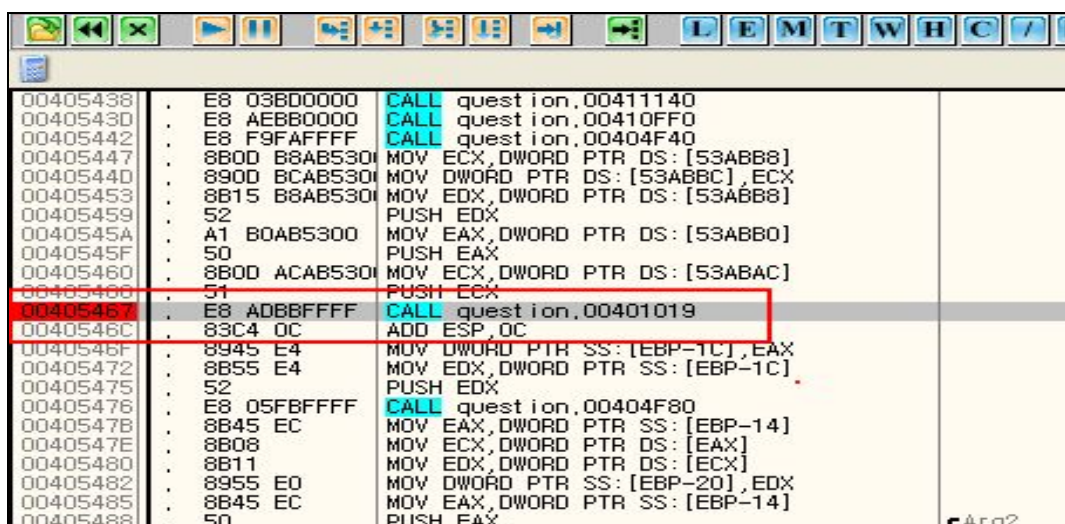
Address	Disassembly	Text string
00401610	MOV DWORD PTR SS:[EBP-34],question,0051	ASCII "OLLYDBG.EXE"
00401617	MOV DWORD PTR SS:[EBP-30],question,0051	ASCII "idag.exe"
0040161E	MOV DWORD PTR SS:[EBP-2C],question,0051	ASCII "WINDBG.EXE"
00401625	MOV DWORD PTR SS:[EBP-28],question,0051	ASCII "proexp.exe"
0040162C	MOV DWORD PTR SS:[EBP-24],question,0051	ASCII "taskmgr.exe"
0040258A	PUSH question,0051B058	ASCII "What is the KEY? : "
00402667	PUSH question,0051B054	ASCII "%s"
00402E97	PUSH question,0051B044	ASCII "Good Job !! "
004034F4	PUSH question,0051B030	ASCII "You'r Wrong !!"
00403F3E	PUSH question,0052E4F8	ASCII "USER32"
00403F56	PUSH question,0052E500	ASCII "GetSystemMetrics"
00403F77	PUSH question,0052E514	ASCII "MonitorFromWindow"
00403F98	PUSH question,0052E528	ASCII "MonitorFromRect"
00403FB5	PUSH question,0052E538	ASCII "MonitorFromPoint"
00403FD2	PUSH question,0052E54C	ASCII "EnumDisplayMonitors"
00403FEF	PUSH question,0052E560	ASCII "GetMonitorInfoA"
004042DC	PUSH question,0052E570	ASCII "DISPLAY"
004045D4	PUSH question,0052D558	ASCII "F:\9782\vc98\mf\include\afx\afx.h"
00404652	PUSH question,0052D558	ASCII "F:\9782\vc98\mf\include\afx\afx.h"
00404777	PUSH question,00531B50	ASCII "F:\9782\vc98\mf\include\afx\afx.h"
0040496C	PUSH question,00532E2C	ASCII "lpw != 0"
00404975	PUSH question,00532E04	ASCII "F:\9782\vc98\mf\include\afx\afx.h"
00404996	PUSH question,00532DF8	ASCII "lpw != 0"
0040499F	PUSH question,00532E04	ASCII "F:\9782\vc98\mf\include\afx\afx.h"
00404A3E	PUSH question,00520C2C	ASCII "The value of ESP was not properly saved ac"
00404A4A	PUSH question,00520C18	ASCII "i386\chkesp.c"
00405235	PUSH question,00520D10	ASCII "format != NULL"

[그림 20] 바이너리의 텍스트 정보

OEP로 진입 후 코딩을 살펴 본 결과 6개 이상의 안티 디버깅 코드가 삽입되어 있어(-;~;) 위의 패스워드 비교 구문 근처에 브레이크 포인터를 걸고 실행시키면 디버거가 뺏는 현상이 발생하였다.

안티 디버깅 코드를 차례대로 제거하여 보자.

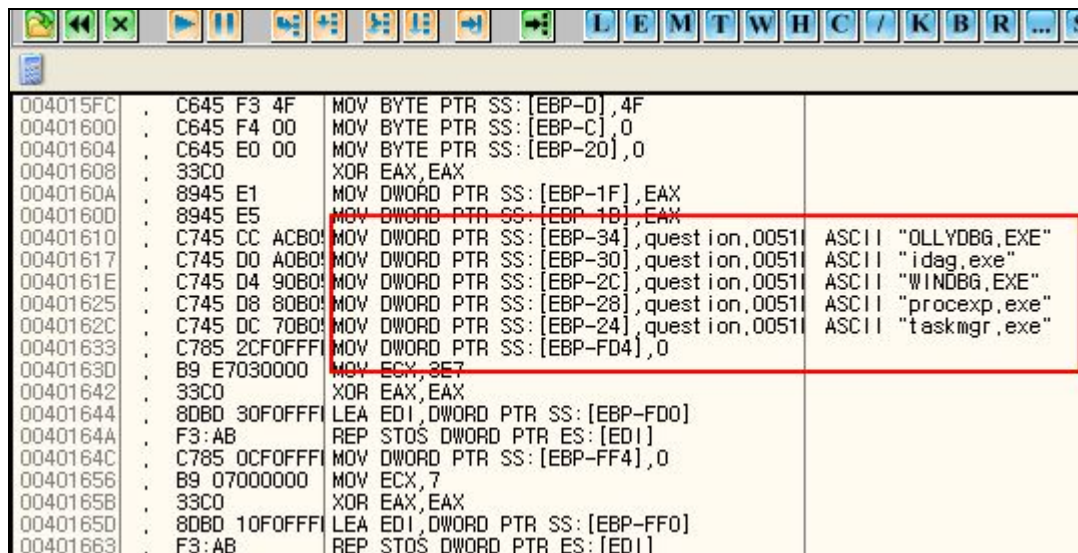
00401019에 브레이크 포인트(F2)를 설정하고 콜 안으로 진입하자.



00405438	. E8 03BD0000	CALL question,00411140
0040543D	. E8 AEBB0000	CALL question,00410FF0
00405442	. E8 F9FAFFFF	CALL question,00404F40
00405447	. 8B0D B8AB5301	MOV ECX,DWORD PTR DS:[53ABB8]
0040544D	. 890D BCAB5301	MOV DWORD PTR DS:[53ABBC],ECX
00405453	. 8B15 B8AB5301	MOV EDX,DWORD PTR DS:[53ABB8]
00405459	. 52	PUSH EDX
0040545A	. A1 B0AB5300	MOV EAX,DWORD PTR DS:[53ABB0]
0040545F	. 50	PUSH EAX
00405460	. 8B0D ACAB5301	MOV ECX,DWORD PTR DS:[53ABAC]
00405468	. 51	PUSH ECX
00405467	. E8 ADBBFFFF	CALL question,00401019
0040546C	. 83C4 0C	ADD ESP,0C
0040546F	. 8945 E4	MOV DWORD PTR SS:[EBP-1C],EAX
00405472	. 8B55 E4	MOV EDX,DWORD PTR SS:[EBP-1C]
00405475	. 52	PUSH EDX
00405476	. E8 05FBFFFF	CALL question,00404F80
0040547B	. 8B45 EC	MOV EAX,DWORD PTR SS:[EBP-14]
0040547E	. 8B08	MOV ECX,DWORD PTR DS:[EAX]
00405480	. 8B11	MOV EDX,DWORD PTR DS:[ECX]
00405482	. 8955 E0	MOV DWORD PTR SS:[EBP-20],EDX
00405485	. 8B45 EC	MOV EAX,DWORD PTR SS:[EBP-14]
00405488	. 50	PUSH EAX

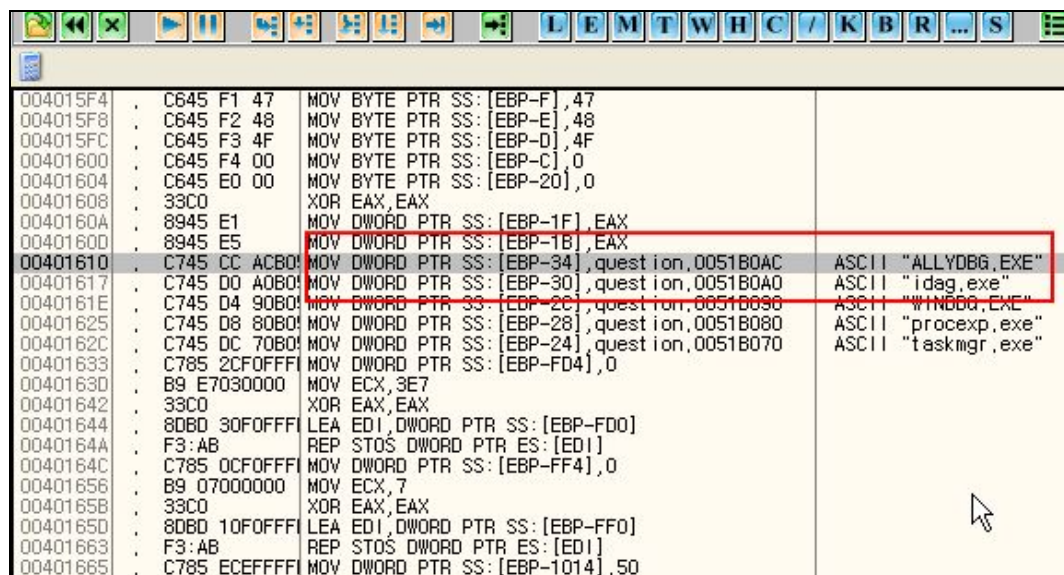
[그림 21] CALL 00401019 브레이크 포인트 설정

함수 안으로 들어가면, 다음과 같이 5개의 실행파일 이름을 스택의 특정 주소로 삽입시키는데... 실행되고 있는 특정 프로세스를 조사하는 안티 디버깅 기법으로 추측되어 OLLYDBG.EXE 이름을 다른 이름으로 변경하였다.



004015FC	C645 F3 4F	MOV BYTE PTR SS:[EBP-D],4F	
00401600	C645 F4 00	MOV BYTE PTR SS:[EBP-C],0	
00401604	C645 E0 00	MOV BYTE PTR SS:[EBP-20],0	
00401608	33C0	XOR EAX,EAX	
0040160A	8945 E1	MOV DWORD PTR SS:[EBP-1F],EAX	
0040160D	8945 E5	MOV DWORD PTR SS:[EBP-1B],EAX	
00401610	C745 CC ACB0	MOV DWORD PTR SS:[EBP-34],question,0051	ASCII "OLLYDBG.exe"
00401617	C745 D0 A0B0	MOV DWORD PTR SS:[EBP-30],question,0051	ASCII "idag.exe"
0040161E	C745 D4 90B0	MOV DWORD PTR SS:[EBP-2C],question,0051	ASCII "WINDBG.exe"
00401625	C745 D8 80B0	MOV DWORD PTR SS:[EBP-28],question,0051	ASCII "procexp.exe"
0040162C	C745 DC 70B0	MOV DWORD PTR SS:[EBP-24],question,0051	ASCII "taskmgr.exe"
00401633	C785 2CF0FFF	MOV DWORD PTR SS:[EBP-FD4],0	
0040163D	B9 E7030000	MOV ECX,3E7	
00401642	33C0	XOR EAX,EAX	
00401644	8DBD 30F0FFF	LEA EDI,DWORD PTR SS:[EBP-FD0]	
0040164A	F3:AB	REP STOS DWORD PTR ES:[EDI]	
0040164C	C785 0CF0FFF	MOV DWORD PTR SS:[EBP-FF4],0	
00401656	B9 07000000	MOV ECX,7	
0040165B	33C0	XOR EAX,EAX	
0040165D	8DBD 10F0FFF	LEA EDI,DWORD PTR SS:[EBP-FF0]	
00401663	F3:AB	REP STOS DWORD PTR ES:[EDI]	

[그림 22] 안티 디버깅 - I



004015F4	C645 F1 47	MOV BYTE PTR SS:[EBP-F],47	
004015F8	C645 F2 48	MOV BYTE PTR SS:[EBP-E],48	
004015FC	C645 F3 4F	MOV BYTE PTR SS:[EBP-D],4F	
00401600	C645 F4 00	MOV BYTE PTR SS:[EBP-C],0	
00401604	C645 E0 00	MOV BYTE PTR SS:[EBP-20],0	
00401608	33C0	XOR EAX,EAX	
0040160A	8945 E1	MOV DWORD PTR SS:[EBP-1F],EAX	
0040160D	8945 E5	MOV DWORD PTR SS:[EBP-1B],EAX	
00401610	C745 CC ACB0	MOV DWORD PTR SS:[EBP-34],question,0051B0AC	ASCII "ALLYDBG.exe"
00401617	C745 D0 A0B0	MOV DWORD PTR SS:[EBP-30],question,0051B0A0	ASCII "idag.exe"
0040161E	C745 D4 90B0	MOV DWORD PTR SS:[EBP-2C],question,0051B098	ASCII "WINDBG.exe"
00401625	C745 D8 80B0	MOV DWORD PTR SS:[EBP-28],question,0051B080	ASCII "procexp.exe"
0040162C	C745 DC 70B0	MOV DWORD PTR SS:[EBP-24],question,0051B070	ASCII "taskmgr.exe"
00401633	C785 2CF0FFF	MOV DWORD PTR SS:[EBP-FD4],0	
0040163D	B9 E7030000	MOV ECX,3E7	
00401642	33C0	XOR EAX,EAX	
00401644	8DBD 30F0FFF	LEA EDI,DWORD PTR SS:[EBP-FD0]	
0040164A	F3:AB	REP STOS DWORD PTR ES:[EDI]	
0040164C	C785 0CF0FFF	MOV DWORD PTR SS:[EBP-FF4],0	
00401656	B9 07000000	MOV ECX,7	
0040165B	33C0	XOR EAX,EAX	
0040165D	8DBD 10F0FFF	LEA EDI,DWORD PTR SS:[EBP-FF0]	
00401663	F3:AB	REP STOS DWORD PTR ES:[EDI]	
00401665	C785 ECEFFFF	MOV DWORD PTR SS:[EBP-1014],50	

[그림 23] 안티 디버깅 제거 - I

004016B5에 브레이크 포인트 설정 후 함수안으로 진입한다.

0040166F	C785 F0FFFFFF	MOV DWORD PTR SS:[EBP-1010],4E	
00401679	C785 F4FFFFFF	MOV DWORD PTR SS:[EBP-100C],41	
00401683	C785 F8FFFFFF	MOV DWORD PTR SS:[EBP-1008],42	
0040168D	C785 FCEFFFFFF	MOV DWORD PTR SS:[EBP-1004],47	
00401697	C785 00FFFFFF	MOV DWORD PTR SS:[EBP-1000],4D	
004016A1	C785 04FFFFFF	MOV DWORD PTR SS:[EBP-FFC],4F	
004016AD	C785 06FFFFFF	MOV DWORD PTR SS:[EBP-FF8],57	
004016B5	E8 64F9FFFF	CALL question,0040101E	
004016BA	8BF4	MOV ESI,ESP	
004016BC	5A 00	PUSH 0	
004016BE	FF15 18E95301	CALL DWORD PTR DS:[53E918]	[Show = FALSE ShowCursor
004016C4	3BF4	CMP ESI,ESP	
004016C6	E8 65330000	CALL question,00404A30	
004016CB	66:53	PUSH BX	
004016CD	66:83C3 13	ADD BX,13	
004016D1	66:83C3 5F	ADD BX,5F	
004016D5	66:83EB 49	SUB BX,49	
004016D9	66:5B	POP BX	
004016DB	66:53	PUSH BX	
004016DD	66:83C3 40	ADD BX,40	
004016E1	66:83EB 5D	SUB BX,5D	
004016E5	66:BB 6200	MOV BX,62	
004016E9	66:5B	POP BX	

[그림 24] 004016B5 브레이크 포인트

함수안으로 진입 후 0040152F 주소에 브레이크 포인트 설정하고 Shift+F9를 누른다. 이유는 INT 2D 명령어를 F7 이나 F8로 달렸을 경우 올리가 프로세스를 중지시켰기 때문에 Shift+F9로 해당 명령어를 실행시켰다.

00401519	B9 10000000	MOV ECX,10	
0040151E	B8 CCCCCCCC	MOV EAX,CCCCCCCC	
00401523	F3:AB	REP STOS DWORD PTR ES:[EDI]	
00401525	C745 FC 0000	MOV DWORD PTR SS:[EBP-4],0	
0040152C	CD 2D	INT 2D	
0040152E	E9	DB E9	
0040152F	C745 FC FFFF	MOV DWORD PTR SS:[EBP-4],-1	
00401536	EB 12	JMP SHORT question,0040154A	
00401538	B8 01000000	MOV EAX,1	
0040153D	C3	RETN	
0040153E	8B65 E8	MOV ESP,DWORD PTR SS:[EBP-18]	
00401541	C745 FC FFFF	MOV DWORD PTR SS:[EBP-4],-1	
00401548	EB 1B	JMP SHORT question,00401565	
0040154A	> 8BF4	MOV ESI,ESP	
0040154C	68 E8030000	PUSH 3E8	
00401551	FF15 1CE35301	CALL DWORD PTR DS:[53E31C]	[Timeout Sleep
00401557	3BF4	CMP ESI,ESP	
00401559	E8 D2340000	CALL question,00404A30	
0040155E	5A 00	PUSH 0	
00401560	E8 1B3A0000	CALL question,00404F80	
00401565	> 8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
00401568	64:890D 0000	MOV DWORD PTR FS:[0],ECX	
0040156E	5F	POP EDI	

[그림 25] INT 2D 우회

00401560 주소에 브레이크 포인트 설정 후 함수 안으로 진입한다.

00401541	. C745 FC FFFF	MOV DWORD PTR SS:[EBP-4],-1
00401548	EB 1B	JMP SHORT question,00401565
0040154A	> 8BF4	MOV ESI,ESP
0040154C	. 68 E8030000	PUSH 3E8
00401551	. FF15 1CE35301	CALL DWORD PTR DS:[53E31C]
00401557	. 3BF4	CMP ESI,ESP
00401559	. E8 D2340000	CALL question,00404A30
0040155E	. 6A 00	PUSH 0
00401560	E8 1B3A0000	CALL question,00404F80
00401565	> 8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]
00401568	. 64:890D 0000	MOV DWORD PTR FS:[0],ECX
0040156F	. 5F	POP EDI
00401570	. 5E	POP ESI
00401571	. 5B	POP EBX
00401572	. 83C4 58	ADD ESP,58
00401575	. 3BEC	CMP EBP,ESP
00401577	. E8 B4340000	CALL question,00404A30
0040157C	. 8BE5	MOV ESP,EBP
0040157E	. 5D	POP EBP
0040157F	. C3	RETN
00401580	. CC	INT3
00401581	. CC	INT3
00401582	. CC	INT3

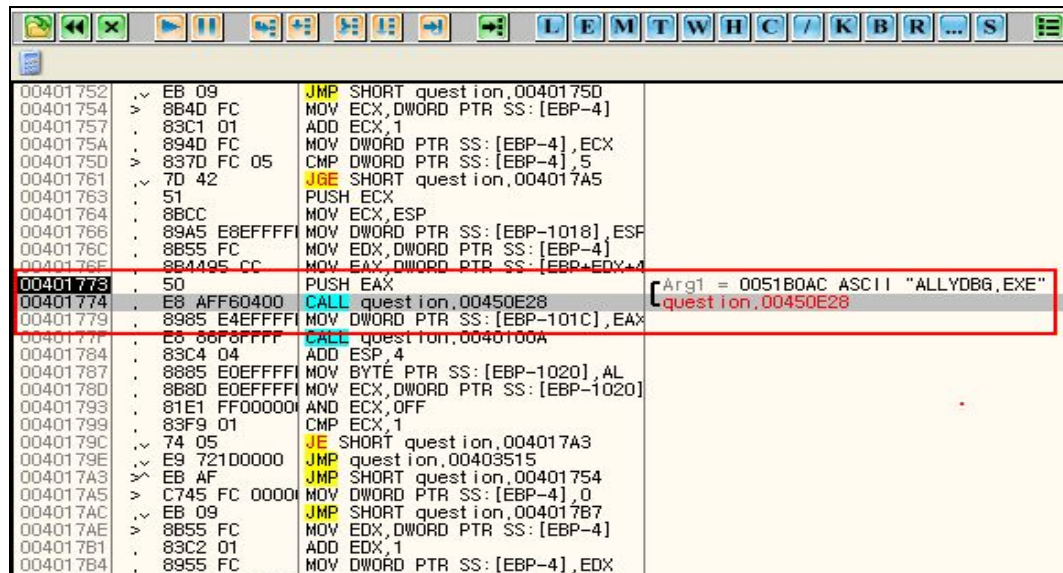
[그림 26] 00401560 브레이크 포인트 설정

다음과 같이 무언가를 조사한 다음 제로 플래그가 설정되면 ExitProcess 함수를 호출하는 곳으로 점프되어버려 JE 코드를 JNE 로 패치하여 우회할 수 있었다.

0040508D	. E8 7E000000	CALL question,00405110
00405092	. 83C4 08	ADD ESP,8
00405095	. 833D DCAB5301	CMP DWORD PTR DS:[53ABDC],0
0040509C	> 75 20	JNZ SHORT question,004050BE
0040509E	. 6A FF	PUSH -1
004050A0	. E8 3B320000	CALL question,004082E0
004050A5	. 83C4 04	ADD ESP,4
004050A8	. 83E0 20	AND EAX,20
004050AB	. 85C0	TEST EAX,EAX
004050AD	> 74 0F	JE SHORT question,004050BE
004050AF	. C705 DCAB5301	MOV DWORD PTR DS:[53ABDC],1
004050B9	. E8 D23C0000	CALL question,00408D90
004050BE	> 837D 10 00	CMP DWORD PTR SS:[EBP+10],0
004050C2	> 74 07	JE SHORT question,004050CB
004050C4	E8 37000000	CALL question,00405100
004050C9	> EB 14	JMP SHORT question,004050DF
004050CB	> C705 D8AB5301	MOV DWORD PTR DS:[53ABD8],1
004050D5	. 8B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]
004050D8	. 51	PUSH ECX
004050D9	. FF15 7CE45301	CALL DWORD PTR DS:[53E47C]
004050DF	> 8BE5	MOV ESP,EBP
004050E1	. 5D	POP EBP
004050E2	. C3	RETN
004050E3	. CC	INT3
004050E4	. CC	INT3

[그림 27] JE 구문 패치

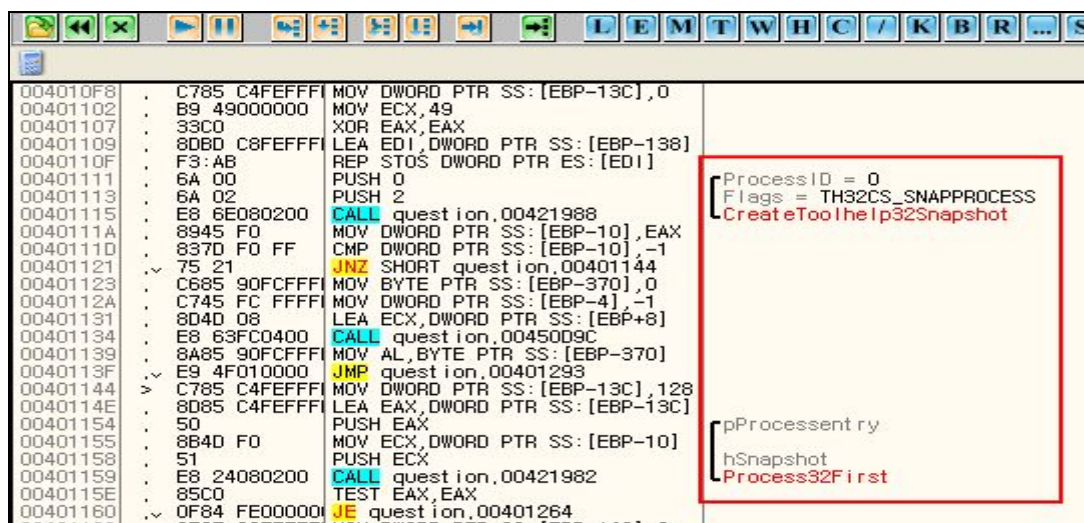
다음의 코드 분석 중 00401773 주소에 처음에 삽입된 5개 문자열 중 하나를 가져와서 원가를 하는 코드가 있다. 해당 문자열은 이전에 변경된 ALLYDBG.EXE 로 변경되어 삽입되는 걸 볼 수 있다.



[그림 28] 문자열 패치

CreateToolhelp32Snapshot , Process32First 와 같은 API가 있는 것으로 보아 5개의 실행파일 문자열과 프로세스 리스트를 비교하여 디버거가 실행 중인지 체크하는 코드가 삽입되어 있다는 걸 추측할 수 있다.

변경된 문자열이 인자 값으로 삽입되어 우회된다.



[그림 29] 안티 디버깅 - II

5개의 실행파일 문자열에 대한 조사가 끝나면, 다음과 같이 분석을 어렵게 하는 난독화 코드가 나온다. 우리가 풀려고 하는 것과 상관없고 단지 분석을 어렵게 하는 용도로 쓰였기 때문에 넘어가자.

Address	Disassembly	Comment
004017CE	JMP SHORT question_004017AE	
004017D0	PUSH BX	
004017D2	ADD BX, 13	
004017D6	ADD BX, 5F	
004017DA	SUB BX, 49	
004017DE	POP BX	
004017E0	PUSH BX	
004017E2	ADD BX, 40	
004017E6	SUB BX, 5D	
004017EA	MOV BX, 6200	
004017EE	POP BX	
004017F0	PUSH AX	
004017F2	ADD AX, 19	
004017F6	ADD AX, 30	
004017FA	ADD AX, 4B	
004017FE	ADD AX, 8	
00401802	POP AX	
00401804	PUSH AX	
00401806	SUB AX, 17	
0040180A	SUB AX, 21	
0040180E	ADD AX, 57	
00401812	ADD AX, 5C	
00401816	POP AX	
00401818	PUSH BX	
0040181A	ADD BX, 13	
0040181E	ADD BX, 5F	
00401822	SUB BX, 49	
00401826	POP BX	
00401828	PUSH BX	
0040182A	ADD BX, 40	
0040182E	SUB BX, 5D	
00401832	MOV BX, 6200	

[그림 30] 난잡화 코드

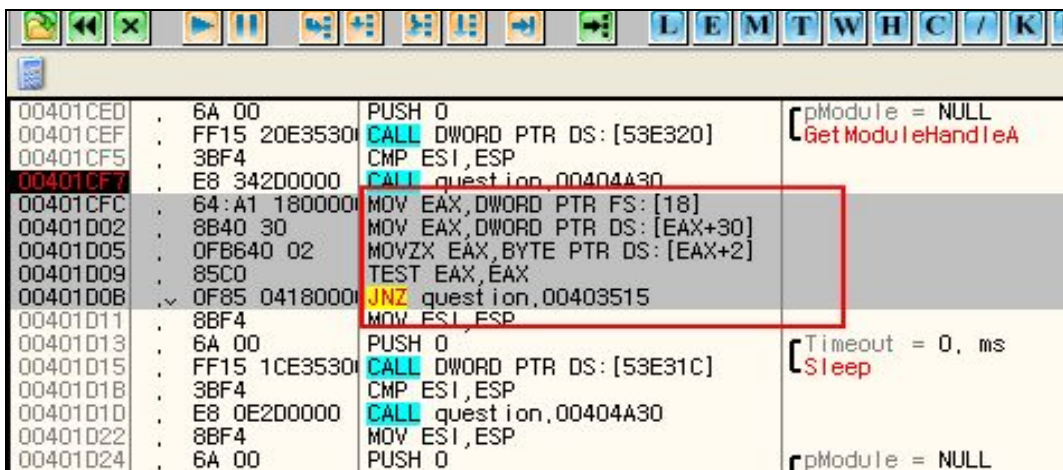
이곳에 쓰인 난독화 코드는 분석과 관계가 없으므로 00401CF7에 브레이크 포인트 설정 후 달린다.

다음과 같이 00401CFC 에 안티 디버깅 기법이 삽입되어 있다.

프로세스 환경 블록인 PEB에서 디버깅 정보를 가져와서 프로세스가 디버깅 되어 있다면 종료되는 코드로 넘어간다.

JNZ 를 JE 로 패치하여 우회할 수 있다.

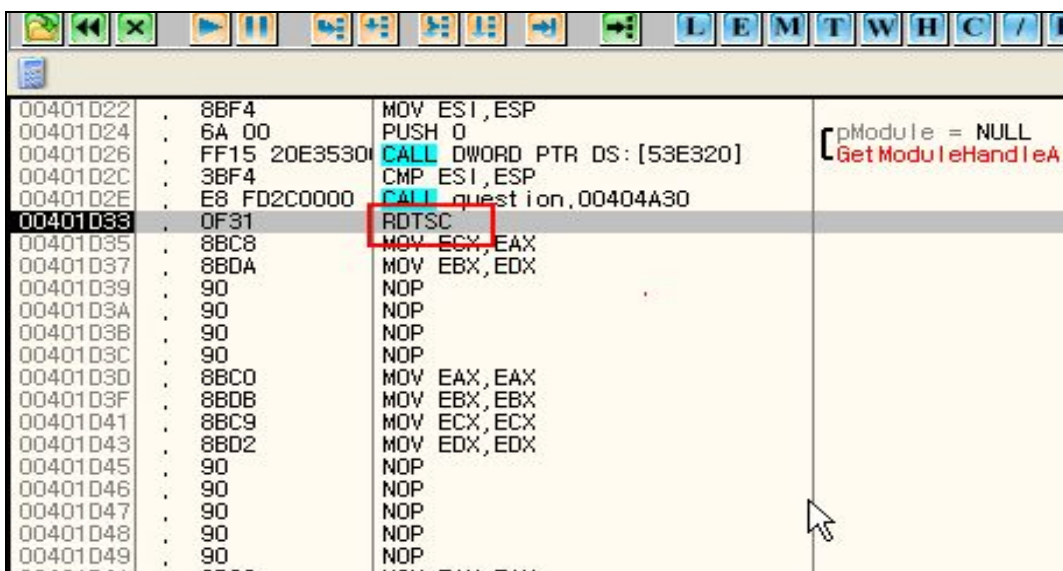
또는 올리 디버거의 안티 디버깅 우회 플러그인을 사용할 수 있다.



00401CED	6A 00	PUSH 0	pModule = NULL
00401CEF	FF15 20E3530	CALL DWORD PTR DS:[53E320]	GetModuleHandleA
00401CF5	3BF4	CMP ESI,ESP	
00401CF7	E8 342D0000	CALL question,00404A30	
00401CFC	64:A1 180000	MOV EAX,DWORD PTR FS:[18]	
00401D02	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]	
00401D05	0FB640 02	MOVZX EAX,BYTE PTR DS:[EAX+2]	
00401D09	85C0	TEST EAX,EAX	
00401D0B	0F85 0418000	JNZ question,00403515	
00401D11	8BF4	MOV ESI,ESP	
00401D13	6A 00	PUSH 0	Timeout = 0, ms
00401D15	FF15 1CE3530	CALL DWORD PTR DS:[53E31C]	Sleep
00401D1B	3BF4	CMP ESI,ESP	
00401D1D	E8 0E2D0000	CALL question,00404A30	
00401D22	8BF4	MOV ESI,ESP	
00401D24	6A 00	PUSH 0	pModule = NULL

[그림 30] 안티 디버깅 코드 - III

다음의 안티 디버깅 코드는 00401D33에 위치하고 있다. RDTSC는 Read-time Stamp Counter 로써 CPU 실행 타임을 체크한다.



00401D22	8BF4	MOV ESI,ESP	
00401D24	6A 00	PUSH 0	pModule = NULL
00401D26	FF15 20E3530	CALL DWORD PTR DS:[53E320]	GetModuleHandleA
00401D2C	3BF4	CMP ESI,ESP	
00401D2E	E8 FD2C0000	CALL question,00404A30	
00401D33	0F31	RDTSC	
00401D35	8BC8	MOV ECX,EAX	
00401D37	8BDA	MOV EBX,EDX	
00401D39	90	NOP	
00401D3A	90	NOP	
00401D3B	90	NOP	
00401D3C	90	NOP	
00401D3D	8BC0	MOV EAX,EAX	
00401D3F	8BDB	MOV EBX,EBX	
00401D41	8BC9	MOV ECX,ECX	
00401D43	8BD2	MOV EDX,EDX	
00401D45	90	NOP	
00401D46	90	NOP	
00401D47	90	NOP	
00401D48	90	NOP	
00401D49	90	NOP	

[그림 31] 안티 디버깅 코드 - IV

Address	Disassembly	Comment
00401D82	MOV ECX,ECX	
00401D84	MOV EDX,EDX	
00401D86	NOP	
00401D87	NOP	
00401D88	NOP	
00401D89	NOP	
00401D8A	NOP	
00401D8B	PUSH EAX	
00401D8C	POP EAX	
00401D8D	NOP	
00401D8E	RDTSC	
00401D90	CMP EDX,EBX	
00401D92	JNBE question,00403515	
00401D98	SUB EAX,ECX	
00401DA9	CMP EAX,2000	
00401DAF	JNBE question,00403515	
00401DA5	MOV ESI,ESP	
00401DA7	PUSH 0	
00401DA9	CALL DWORD PTR DS:[53E31C]	
00401DAF	CMP ESI,ESP	
00401DB1	CALL question,00404A30	
00401DB6	MOV ESI,ESP	
00401DB8	PUSH 0	
00401DBA	CALL DWORD PTR DS:[53E320]	
00401DC0	CMP ESI,ESP	

Timeout = 0, ms
Sleep
pModule = NULL
GetModuleHandle

```
00401E22 . 0F31          RDTSC
00401E24 . 3BD3          CMP EDX,EBX
00401E26 . 0F87 E9160000 JA question.00403515
00401E2C . 2BC1          SUB EAX,ECX
00401E2E . 3D 00200000   CMP EAX,2000
00401E33 . 0F87 DC160000 JA question.00403515
```


다음의 안티 디버깅 기법은 PEB.NtGlobalFlag를 이용하여 프로세스가 디버깅 중인 지 체크한다. 디버깅 중이 아닐 때, NtGlobalFlag은 0으로 설정되나, 프로세스가 디버깅 중이라면, 필드 값은 0x70 이 된다. JNZ 코드를 JE로 패치하거나 플래그 레지스터를 수정하여 우회할 수 있다.

004021CD	5306	XOR EBX,EBX	
004021CF	64:8B1D 3000	MOV EBX,DWORD PTR FS:[30]	
004021D6	807B 68 00	CMP BYTE PTR DS:[EBX+68],0	
004021DA	0F85 35130000	JNZ question,00403515	
004021E0	8B5B 18	MOV EBX,DWORD PTR DS:[EBX+18]	
004021E3	807B 0C 02	CMP BYTE PTR DS:[EBX+C],2	
004021E7	0F85 28130000	JNZ question,00403515	
004021ED	807B 10 00	CMP BYTE PTR DS:[EBX+10],0	
004021F1	0F85 1E130000	JNZ question,00403515	
004021F7	8BF4	MOV ESI,ESP	
004021F9	6A 00	PUSH 0	
004021FB	FF15 1CE35300	CALL DWORD PTR DS:[53E31C]	
00402201	3BE4	CMP ESI,ESP	

Address	Hex	dump	ASCII
7FFD4068	70 00 00 00	00 00 00 00 00 80 9B 07 6D E8 FF FF	p...
7FFD4078	88 00 10 00	00 20 00 00 00 00 01 00 00 10 00 00	...+...
7FFD4088	0A 00 00 00	10 00 00 00 80 DE 9A 7C 00 00 65 00	...-...
7FFD4098	00 00 00 00	14 00 00 00 D8 C0 9A 7C 05 00 00 00	...9...
7FFD40A8	01 00 00 00	28 0A 00 02 02 00 00 00 03 00 00 00	...J...
7FFD40B8	04 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
7FFD40C8	00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00

[그림 33] 안티 디버깅 코드 - V

다음의 안티 디버깅 코드는 PEB.ProcessHeap.Flags 를 이용하여 프로세스가 디버깅 인지 체크한다. JNZ 코드를 JE로 패치하거나 플래그 레지스터를 수정하여 우회할 수 있다.

004021CB	66:58	POP AX	
004021CD	33DB	XOR EBX,EBX	
004021CF	64:8B1D 3000	MOV EBX,DWORD PTR FS:[30]	
004021D6	807B 68 00	CMP BYTE PTR DS:[EBX+68],0	
004021DA	0F85 3513000	JNZ question,00403515	
004021E0	8B5B 18	MOV EBX,DWORD PTR DS:[EBX+18]	
004021E3	807B 0C 02	CMP BYTE PTR DS:[EBX+C],2	
004021E7	0F85 2813000	JNZ question,00403515	
004021E0	807B 10 00	CMP BYTE PTR DS:[EBX+10],0	
004021F1	0F85 1E13000	JNZ question,00403515	
004021F7	8BF4	MOV ESI,ESP	
004021F9	6A 00	PUSH 0	
004021FB	FF15 1CE3530	CALL DWORD PTR DS:[53E31C]	[Timeout = 0, ms] Sleep

[그림 34] 안티 디버깅 코드 - VI

다음의 안티 디버깅 코드는 PEB.ProcessHeap.ForceFlags 를 이용하여 프로세스가 디버깅 인지 체크한다. JNZ 코드를 JE로 패치하거나 플래그 레지스터를 수정하여 우회할 수 있다.

004021C3	66:05 5700	ADD AX,57	
004021C7	66:05 5C00	ADD AX,5C	
004021CB	66:58	POP AX	
004021CD	33DB	XOR EBX,EBX	
004021CF	64:8B1D 3000	MOV EBX,DWORD PTR FS:[30]	
004021D6	807B 68 00	CMP BYTE PTR DS:[EBX+68],0	
004021DA	0F85 3513000	JNZ question,00403515	
004021E0	8B5B 18	MOV EBX,DWORD PTR DS:[EBX+18]	
004021E3	807B 0C 02	CMP BYTE PTR DS:[EBX+C],2	
004021E7	0F85 2813000	JNZ question,00403515	
004021E0	807B 10 00	CMP BYTE PTR DS:[EBX+10],0	
004021F1	0F85 1E13000	JNZ question,00403515	
004021F7	8BF4	MOV ESI,ESP	
004021F9	6A 00	PUSH 0	
004021FB	FF15 1CE3530	CALL DWORD PTR DS:[53E31C]	[Timeout = 0, ms] Sleep

[그림 35] 안티 디버깅 코드 - VII

안티 디버깅을 모두 제거하면 키 값을 받는 위치가 나온다.

00402656	FF15 20E35301	CALL DWORD PTR DS:[53E320]	[GetModuleHandleA
0040265C	3BF4	CMP ESI,ESP	
0040265E	E8 CD230000	CALL question.00404A30	
00402663	8D4D E0	LEA ECX,DWORD PTR SS:[EBP-20]	
00402666	51	PUSH ECX	
00402667	68 54B05100	PUSH question.0051B054	[Arg2
0040266C	E8 AF2B0000	CALL question.00405220	Arg1 = 0051B054 ASCII "%s"
00402671	83C4 08	ADD ESP,8	question.00405220
00402674	8BF4	MOV ESI,ESP	
00402676	6A 00	PUSH 0	
00402678	FF15 1CE35301	CALL DWORD PTR DS:[53E31C]	[Timeout = 0, ms
0040267E	3BF4	CMP ESI,ESP	Sleep
00402680	E8 AB230000	CALL question.00404A30	
00402685	8BF4	MOV ESI,ESP	
00402687	6A 00	PUSH 0	
00402689	FF15 20E35301	CALL DWORD PTR DS:[53E320]	[pModule = NULL
0040268F	3BF4	CMP ESI,ESP	GetModuleHandleA
00402691	E8 9A230000	CALL question.00404A30	

[그림 36] 키 값 입력

테스트 키 값으로 abcdefghi 를 입력하였다.



[그림 37] 테스트 키 값

여러 연산을 거치고 00402C4A 위치에 오면 테스트로 입력한 패스워드와 오리지널 패스워드를 비교하여 bad 메시지를 보여줄 지 good 메시지를 보여줄 지 정하는 부분이 나온다.

다음은 패스워드 문자 비교와 연산 횟수를 나타내고 있다. 연산횟수는 7번이고 연산되어 나오는 패스워드 위치는 SS:[EBP+EDX*4-FF4] -> 요기 인걸 알수 있다.

00402C58 CMP ECX,DWORD PTR SS:[EBP+EDX*4-FF4] ; 문자 비교

00402C65 CMP DWORD PTR SS:[EBP-4],7 ; 패스워드 문자 동일 시 연산 횟수

00402C45	. E8 E6FC0000	CALL question.00412930
00402C4A	. 83C4 08	ADD ESP,8
00402C4D	. 8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
00402C50	. 0FBF4C05 E0	MOVSB ECX,BYTE PTR SS:[EBP+EAX-20]
00402C55	. 8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]
00402C58	. 3B8C95 0CF0F1	CMP ECX,DWORD PTR SS:[EBP+EDX*4-FF4]
00402C5F	. 0F85 55020001	JNZ question.00402EBA
00402C65	. 837D FC 07	CMP DWORD PTR SS:[EBP-4],7
00402C69	. 0F85 4B020001	JNZ question.00402EBA
00402C6F	. 66:53	PUSH BX
00402C71	. 66:83C3 13	ADD BX,13
00402C75	. 66:83C3 5F	ADD BX,5F
00402C79	. 66:83EB 49	SUB BX,49
00402C7D	. 66:5B	POP BX
00402C7F	. 66:53	PUSH BX
00402C81	. 66:83C3 40	ADD BX,40
00402C85	. 66:83EB 5D	SUB BX,5D

[그림 38] 패스워드 문자 비교 구문

패스워드 문자 비교에서 jnz 구문의 제로 플래그를 변경하여 7번의 오리지널 패스워드 연산을 유도하여준다.

00402C41	. 8B55 D0	MOV EDX,DWORD PTR SS:[EBP-30]
00402C44	. 52	PUSH EDX
00402C45	. E8 E6FC0000	CALL question.00412930
00402C4A	. 83C4 08	ADD ESP,8
00402C4D	. 8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
00402C50	. 0FBF4C05 E0	MOVSB ECX,BYTE PTR SS:[EBP+EAX-20]
00402C55	. 8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]
00402C58	. 3B8C95 0CF0F1	CMP ECX,DWORD PTR SS:[EBP+EDX*4-FF4]
00402C5F	. 0F85 55020001	JNZ question.00402EBA
00402C65	. 837D FC 07	CMP DWORD PTR SS:[EBP-4],7
00402C69	. 0F85 4B020001	JNZ question.00402EBA
00402C6F	. 66:53	PUSH BX
00402C71	. 66:83C3 13	ADD BX,13
00402C75	. 66:83C3 5F	ADD BX,5F
00402C79	. 66:83EB 49	SUB BX,49
00402C7D	. 66:5B	POP BX
00402C7F	. 66:53	PUSH BX
00402C81	. 66:83C3 40	ADD BX,40
00402C85	. 66:83EB 5D	SUB BX,5D

[그림 39] 패스워드 연산 유도

연산되는 위치는 DWORD PTR SS:[EBP+EDX*4-FF4] 이곳이고 7번의 연산을 유도하면 다음과 같이 패스워드가 스택의 주소에 표시된다.

패스워드 : **POCACHIP**

Address	Hex dump	ASCII
0012EF8C	50 00 00 00 4F 00 00 00 43 00 00 00 41 00 00 00	P...O...C...A...
0012EF9C	43 00 00 00 48 00 00 00 49 00 00 00 50 00 00 00	C...H...I...P...
0012EFA0	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012EFB0	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012EFC0	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012EFD0	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012EFE0	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012EFF0	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F000	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F010	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F020	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F030	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F040	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F050	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F060	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F070	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F080	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F090	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
0012F0A0	01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00

[그림 40] 패스워드 연산 결과

Mission Complete

