

CodeEngn

Basic RCE



김성현(gh0st@Ma9icAna1y\$is)

<http://gh0st.xe.to>

Reverse L01 Start

먼저 문제를 확인하겠다.

Reverse L01 Start

Author : abex

Korea :

HDD를 CD-Rom으로 인식시키기 위해서는 GetDriveTypeA의 리턴값이 무엇이 되어야 하는가

English :

What value must GetDriveTypeA return in order to make the computer recognize the HDD as a CD-Rom

Down

총 171 분이 이 문제를 푸셨습니다. / 171 people solved this problem.

1번 문제는 MSDN 사이트에서 GetDriveTypeA 의 리턴값만 조사하면 쉽게 풀 수 있는 문제이다.

여기서 잠깐!

GetDriveType 함수에 대해서 알아보고 지나가자.

GetDriveType [Quick Info](#)

The **GetDriveType** function determines whether a disk drive is a removable, fixed, CD-ROM, RAM disk, or network drive.

UINT GetDriveType

LPCWSTR lpRootPathName // address of root path

);

Parameters

lpRootPathName

Points to a null-terminated string that specifies the root directory of the disk to return information about. If lpRootPathName is NULL, the function uses the root of the current directory.

Return Values

The return value specifies the type of drive. It can be one of the following values:

Value	Meaning
0	The drive type cannot be determined.
1	The root directory does not exist.
DRIVE_REMOVABLE	The drive can be removed from the drive.
DRIVE_FIXED	The disk cannot be removed from the drive.
DRIVE_REMOTE	The drive is a remote (network) drive.
DRIVE_CDROM	The drive is a CD-ROM drive.
DRIVE_RAMDISK	The drive is a RAM disk.

See Also

[GetDiskFreeSpace](#)

그림을 보게 되면 어떠한 파라미터 값을 받게 되면 그에 따른 리턴값을 내어주는 함수이다.

그런데 리턴값에서 0,1 다음에는 문자열로 지정되어있다. 정확하게 리턴값을 알고자 하면 MSDN 에서 검색하여 본다. 검색 결과는 다음과 같다.

Return code/value

```
drive_unknown 0
drive_no_root_dir 1
drive_removable 2
drive_fixed 3
drive_remote 4
drive_cdrom 5
drive_ramdisk 6
```

위 그림에서 볼 때 리턴값이 5일때 cdrom 을 인식한다고 나와있다.

Reverse L02 Start

먼저 문제를 확인하자.

Reverse L02 Start

Author : ArturDents

Korea :
패스워드로 인증하는 실행파일이 손상되어 실행이 안되는 문제가 생겼다. 패스워드가 무엇인지 분석하시오

English :
The program that verifies the password got messed up and ceases to execute. Find out what the password is.

[Down](#)

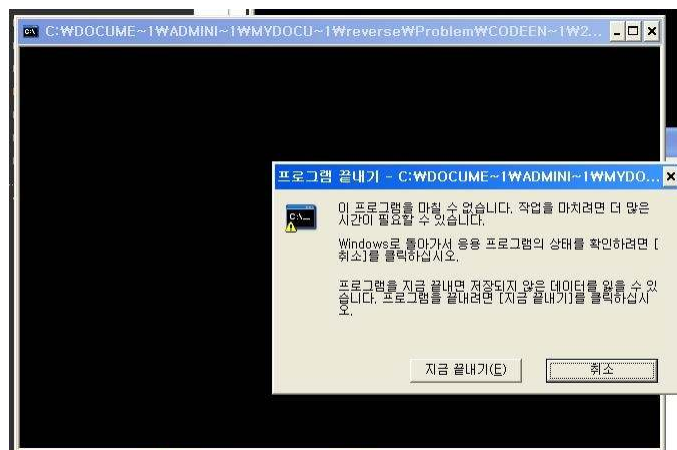
총 114 분이 이 문제를 푸셨습니다. / 114 people solved this problem.

Down을 클릭하면 파일 하나를 받을 수 있는데 그 파일을 분석하여 패스워드를 구하는 문제인것 같다. 다운 받으면 아래와 같은 파일을 볼 수 있다.

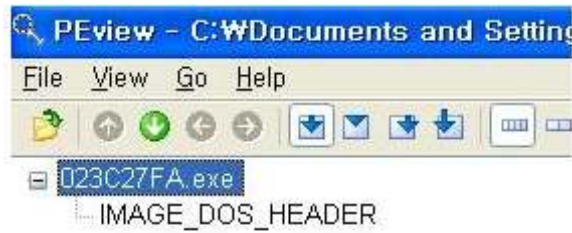


일단 실행시켜 본다.

실행을 하게 되면 아무것도 뜨지 않은 채 커서만 반짝 거리는걸 볼 수 있다. 게다가 종료할 때에도 정상적인 종료가 되지 않는것을 볼 수 있다.



딱 보니 PE File format을 제대로 지켜서 만든 파일이 아닌것 같다. PEview를 이용하여 확인해 본다.



위 그림과 같이 PE file format을 제대로 갖추지 않고 달랑 IMAGE_DOS_HEADER 구조체 하나만 있는것을 볼 수 있다. PEView를 이용하여 IMAGE_DOS_HEADER 의 Raw_data 값을 보도록 한다.

```

00000740  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000750  41 44 44 69 61 6C 6F 67 00 41 72 74 75 72 44 65  ADDialog.ArturDe
00000760  6E 74 73 20 43 72 61 63 6B 4D 65 23 31 00 00 00  nts CrackMe#1...
00000770  00 00 00 00 00 4E 6F 70 65 2C 20 74 72 79 20 61  ....Nope, try a
00000780  67 61 69 6E 21 00 59 65 61 68 2C 20 79 6F 75 20  gain!.Yeah, you
00000790  64 69 64 20 69 74 21 00 43 72 61 63 6B 6D 65 20  did it!.Crackme
000007A0  23 31 00 4A 4B 33 46 4A 5A 68 00 00 00 00 00 00  #1.JK3FJZh.....
000007B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....

```

위 그림에서 ASCII 코드값을 보면 끝 부분에 Crakme#1 JK3FJZh 라는 문구를 볼 수 있다.

Reverse L03 Start

3번 문제를 보도록 하자.

Reverse L03 Start Author : Blaster99 [DCD]

Korea :
비주얼베이직에서 스트링 비교함수 이름은?

English :
What is the name of the Visual Basic function that compares two strings?

[Down](#)

총 102 분이 이 문제를 푸셨습니다. / 102 people solved this problem.

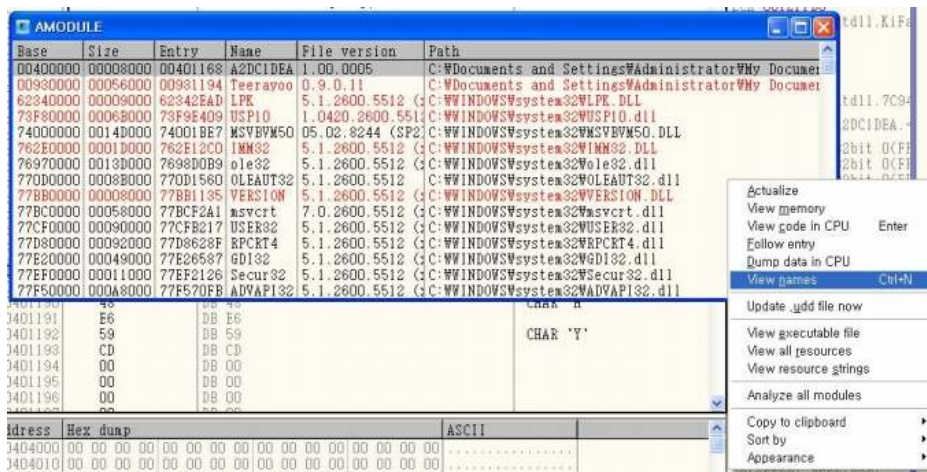
3번 문제는 스트링비교함수를 찾는 문제이다. 일단 다운 받아 실행시켜 본다.



실행시키면 위와 같은 그림을 볼 수 있다. 이제 이 실행파일을 올리디버거로 열어보자. 우리가 필요한 것은 스트링비교함수를 찾는 것이다. 그러므로 이 실행파일에 사용된 함수 목록을 찾아 그중에 스트링비교함수를 찾으려 하는 것이다. 올리디버거 메뉴에서 E(show modules) 버튼을 누른다.

```
FixDBG - {A2DC1DEA.exe} - [UPC= main thread, module A2DC1DEA]
파일(F) 보기(V) 디버그(D) 플러그인 설정(C) 실행(R) 도움말(H) Tools Custom Languages
E T W H C 7 K B R S
00401168 $ 68 B8184000 PUSH A2DC1DEA.004018B8
0040116D . E8 F0FFFFFF CALL <JMP.&MSVBVM50.#100>
00401172 . 0000 ADD BYTE PTR DS:[EAX],AL
00401174 . 0000 ADD BYTE PTR DS:[EAX],AL
00401176 . 0000 ADD BYTE PTR DS:[EAX],AL
00401178 . 3000 XOR BYTE PTR DS:[EAX],AL
0040117A . 0000 ADD BYTE PTR DS:[EAX],AL
0040117C . 40 INC EAX
0040117D . 0000 ADD BYTE PTR DS:[EAX],AL
0040117F . 0000 ADD BYTE PTR DS:[EAX],AL
00401181 . 0000 ADD BYTE PTR DS:[EAX],AL
00401183 . 003F ADD BYTE PTR DS:[EDI],BH
00401185 . 186A 8B SBB BYTE PTR DS:[EDX-75],CH
00401188 . 9E SAHF
00401189 . C3 RETN
0040118A . D2 DB D2
0040118B . 11 DB 11
0040118C . 80 DB 80
0040118D . E1 DB E1
0040118E . 00 DB 00
```

버튼을 누른 후 활성화 된 module 창 위에서 오른쪽 버튼을 눌러 View names을 클릭한다.



View names 를 클릭하게 되면 실행파일에 사용된 함수의 목록을 볼 수 있다.

00405120	.idata	Import	MSVBVM50.EVENT_SINK_AddRef
0040513C	.idata	Import	MSVBVM50.EVENT_SINK_QueryInterface
00405134	.idata	Import	MSVBVM50.EVENT_SINK_Release
00401168	.text	Export	<ModuleEntryPoint>
0040511C	.idata	Import	MSVBVM50.__vbaChkstk
004050F0	.idata	Import	MSVBVM50.__vbaEnd
00405140	.idata	Import	MSVBVM50.__vbaExceptionHandler
0040514C	.idata	Import	MSVBVM50.__vbaFPException
00405188	.idata	Import	MSVBVM50.__vbaFreeObj
00405184	.idata	Import	MSVBVM50.__vbaFreeStr
004050E8	.idata	Import	MSVBVM50.__vbaFreeVar
004050EC	.idata	Import	MSVBVM50.__vbaFreeVarList
004050FC	.idata	Import	MSVBVM50.__vbaHresultCheckObj
00405168	.idata	Import	MSVBVM50.__vbaI4Var
00405104	.idata	Import	MSVBVM50.__vbaLateMemSt
00405108	.idata	Import	MSVBVM50.__vbaObjSet
0040512C	.idata	Import	MSVBVM50.__vbaObjVar
00405124	.idata	Import	MSVBVM50.__vbaStrCmp
00405170	.idata	Import	MSVBVM50.__vbaVarCopy
0040516C	.idata	Import	MSVBVM50.__vbaVarDup
004050E4	.idata	Import	MSVBVM50.__vbaVarMove
00405128	.idata	Import	MSVBVM50.__vbaVarTstEq

함수의 목록중에 vbaStrCmp 라는 함수를 볼 수 있다. C언어를 공부하면서 strcmp를 많이 보았을 것이다.

Reverse L04 Start

4번째 문제는 디버거를 탐지하는 함수의 이름을 찾는 문제이다.

Korea :

이 프로그램은 디버거 프로그램을 탐지하는 기능을 갖고 있다. 디버거를 탐지하는 함수의 이름은 무엇인가

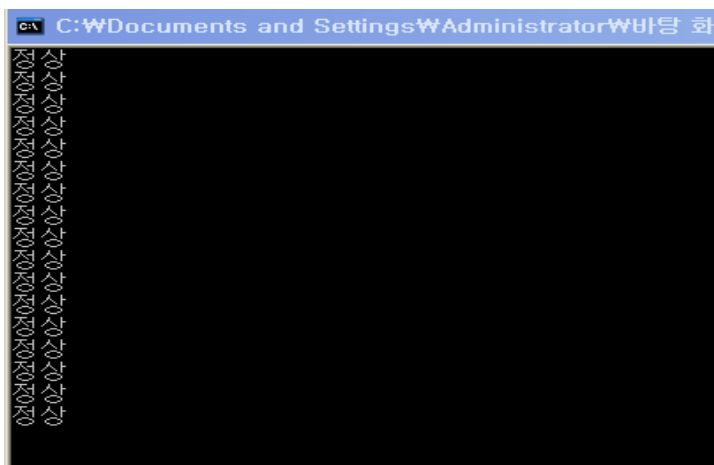
English :

This program can detect debuggers. Find out the name of the debugger detecting function the program uses.

[Down](#)

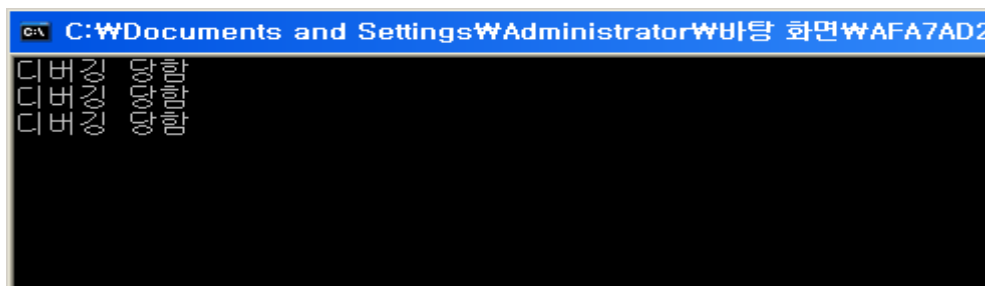
총 227 분이 이 문제를 푸셨습니다. / 227 people solved this problem.

먼저 파일을 다운받아 실행을 해보면 아래 그림과 같이 출력된다.



이번엔 올리디버거로 다운받은 파일을 실행시켜 본다.

아래그림과 같이 '디버깅 당함'이라고 출력되는것을 볼 수 있다.



디버거를 탐지하는 방법에는 여러 가지가 있는데 이 프로그램에서는 IsDebuggerPresent() 함수를 사용함을 알 수 있다. 올리디버거로 열어 intermodular calls에서 확인해본다.

IsDebuggerPresent() API는 PEB.BeingDebugged(PEB 구조체 +0x002)의 값을 참조하여 디버깅 여부를 판별한다.

0040104F	. FF15 68B14300	CALL DWORD PTR DS:[&KERNEL32.Sleep]	LSleep
00401055	. 3BF4	CMR ESI,ESP	
00401057	. E8 B4710000	CALL AFA7AD21.00408210	
0040105C	. 8BF4	MOV ESI,ESP	
0040105E	. FF15 64B14300	CALL DWORD PTR DS:[&KERNEL32.IsDebuggerPresent]	IsDebuggerPresent

위 코드를 보면 IsDebuggerPresent() 있음을 확인할 수 있다.F7(Step Into)를 통해 내부를 살펴보겠다.

7C7E3133	Is	\$ 64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
7C7E3139		. 8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]
7C7E313C		0FB640 02	MOVZX EAX,BYTE PTR DS:[EAX+2]

위 3줄의 코드는 TEB 구조체를 이용해서 PEB 구조체를 찾아 PEB구조체의 BeingDebugged 값을 확인한다.

FS[0x18] -> TEB

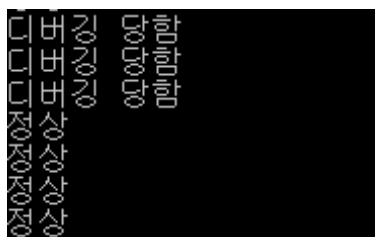
DS[EAX+0x30] -> PEB

PEB에서 +0x2 한 값이 +0x002 BeingDebugged : Uchar 이 값이다.

이 값이 디버거로 실행될시 1로 셋팅되고, 그냥 실행될시 0으로 셋팅된다.

DS:[7FFD6002]=01									
EAX=7FFD6000									
Address	Hex dump								
7FFD6002	01 00 FF FF	FF FF 00 00	40 00 A0 1E	24 00 00 00					
7FFD6012	02 00 00 00	00 00 00 00	14 00 20 06	9B 7C 00 10					
7FFD6022	93 7C E0 10	93 7C 01 00	00 00 00 00	00 00 00 00					
7FFD6032	00 00 00 00	00 00 00 00	00 00 00 00	00 00 E0 05					
7FFD6042	9B 7C 01 00	00 00 00 00	00 00 00 00	6F 7F 00 00					
7FFD6052	6F 7F 88 06	6F 7F 00 00	FA 7F 00 00	FA 7F 00 10					
7FFD6062	FD 7F 02 00	00 00 70 00	00 00 00 00	00 00 00 80					
7FFD6072	9B A7 6D F8	FF FF 00 00	1A 00 00 2A	00 00 00 00					

01 값을 00 으로 패치해주면 정상이 뜨는 것을 확인할 수 있다.



Reverse L05 Start

먼저 문제를 보도록 한다. 프로그램의 등록키를 찾는 문제이다.

Korea :
이 프로그램의 등록키는 무엇인가

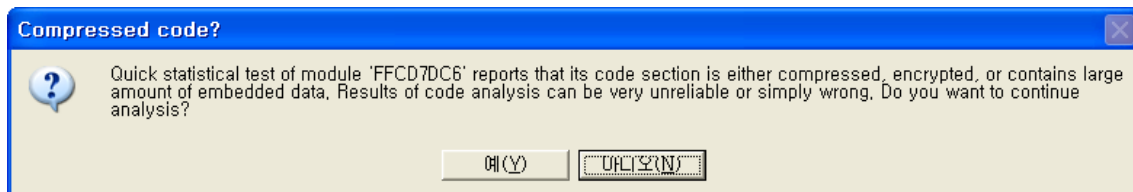
English :
The registration key of this program is?

[Down](#)

실행시켜보면 user name과 key 값을 입력하는 박스가 보인다. 잘못된 값을 입력 시 아래와 같이 에러가 뜬다.



올리디버거로 열어 Key 값을 찾아보도록 한다. 올리디버거로 열게 되면 아래와 같은 에러를 확인할 수 있다. 코드가 압축되어있는데 계속 하겠냐고 묻는 것이다. 다른 tool로 확인해보면 UPX로 압축되어 있음을 알 수 있다.



그럼 먼저 올리디버거로 UPX 파일을 Unpack 하는 방법부터 알아본다. 예를 누른후 코드를 보게되면 가장 처음 명령 부분에 이런 명령이 나온다.

Address	Hex dump	Disassembly	Comment
01015380	60	PUSHAD	

PUSHAD 는 현재 레지스터에 저장된 모든값을 전부 스택에 저장 하라는 명령이다. 그리고 이런 명령이 있다는 것은 패킹을 했다는 증거로 삼을 수도 있다.

이유는 ?

: 패킹된 프로그램과 패킹되지 않은 프로그램은 동일하게 동작을 한다. 이렇게 패킹된 프로그램도 정상적으로 동작을 하기 위해서는 언패킹 과정을 거치고 OEP로 가서 실제 코드를 실행해야 하기 때문에 프로그램 시작당시 레지스터 값들을 스택에 저장하고 언패킹과정을 거친후 다시 POPAD를 사용해 스택에 저장되어있던 레지스터 값들을 가져와 실제 코드를 실행 하기 때문이다

이제 F8(Step over) 번을 눌러 명령을 수행해 본다. 수행한 결과 아래와 같은 화면을 볼 수 있다.

<F8 누르기전 EBP 기준의 Stack 모습>

EBP-2C	7C817067	RETURN to kernel32.7C817067
EBP-28	7C940208	ntdll.7C940208
EBP-24	FFFFFFFF	
EBP-20	7FFDE000	
EBP-1C	805532FA	
EBP-18	0006FFC8	
EBP-14	863EB020	
EBP-10	FFFFFFFF	End of SEH chain
EBP-C	7C839AC0	SE handler
EBP-8	7C817070	kernel32.7C817070
EBP-4	00000000	
EBP ==>	00000000	
EBP+4	00000000	
EBP+8	01015330	notepad_.<ModuleEntryPoint>
EBP+C	00000000	

<F8 누른후 EBP 기준의 Stack 모습>

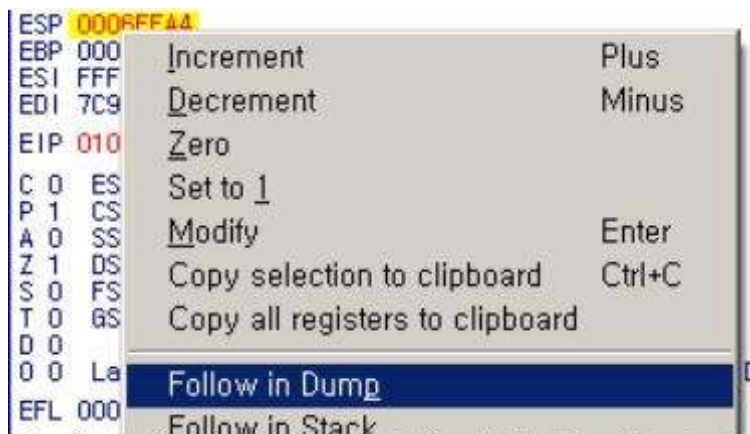
EBP-4C	7C940208	ntdll.7C940208
EBP-48	FFFFFFFF	
EBP-44	0006FFFO	
EBP-40	0006FFC4	
EBP-3C	7FFDE000	
EBP-38	7C93E4F4	ntdll.KiFastSystemCallRet
EBP-34	0006FFB0	
EBP-30	00000000	
EBP-2C	7C817067	RETURN to kernel32.7C817067
EBP-28	7C940208	ntdll.7C940208
EBP-24	FFFFFFFF	
EBP-20	7FFDE000	
EBP-1C	805532FA	
EBP-18	0006FFC8	
EBP-14	863EB020	
EBP-10	FFFFFFFF	End of SEH chain
EBP-C	7C839AC0	SE handler
EBP-8	7C817070	kernel32.7C817070
EBP-4	00000000	

두개의 그림을 비교해보면 EBP-2C를 기준으로 PUSHAD 명령 후 8개의 값이 추가 된것을 볼 수있다. EAX 부터 차례대로 8개의 값이 입력 되었다.

이제 어딘가에서 Stack에 들어간 레지스터값을 POPAD 할 것이므로 그 부분을 찾아보도록 하겠다. 분명히 POPAD 를 하면 Stack주소를 건들 것 이므로 그 부분에 Hardware break point를 걸어둔다. Stack 창에선 break point 가 걸어지지 않으므로



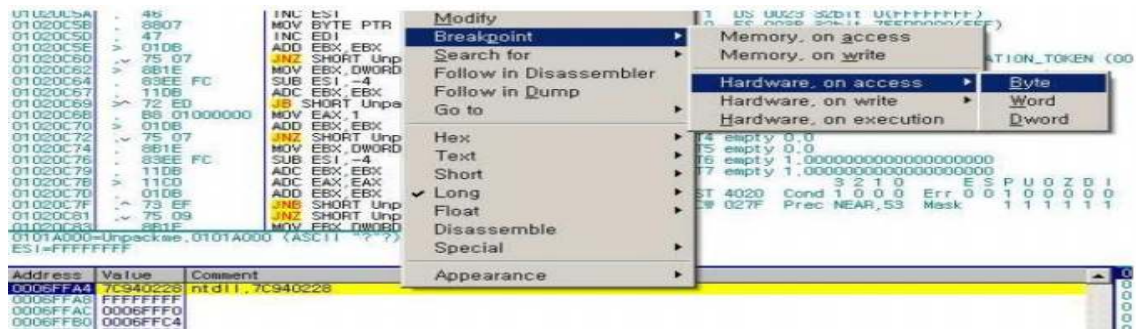
그림과 같이 스택 옆 창에서 이렇게 바꿔준 다음에



레지스터 창에서 ESP에 우클릭 후 Follow in Dump 를 해주면



이렇게 바뀐걸 볼 수 있다.



해당 위치에서 우클릭 후 저렇게 걸어 주면 된다. 그러면 해당 부분에서 접근하면 멈출 것이다. 이제 F9(Run) 를 눌러 실행을 해준다. 실행을 해주니 POPAD가 끝난 위치에서 멈춰 있는걸 볼 수 있다.

Address	Hex dump	Disassembly	Comment
010154AC	58	POP EAX	
010154AD	61	POPAD	
010154AE	804424 80	LEA EAX, DWORD PTR SS:[ESP-80]	

보이는가 010154AD 부분을 보게되면 POPAD 명령을 수행한 것을 볼 수 있다. 그 위치에서 F8(Step over)를 수행하게 되면 아래와 같은 그림의 위치로 이동한다.

00441270	> 55	PUSH EBP	
00441271	. 8BEC	MOV EBP,ESP	
00441273	? 83C4 F4	ADD ESP,-0C	

이제 이 해당 부분에서 Dump debugged process를 클릭해서 덤프를 뜨게되면 unpack이 완료된다. Rebulid import 부분을 체크하여 IAT를 복구해준다. 정상적으로 복구가 안되는 경우는 ImportREConstrutor를 사용하도록 한다.

이제 unpack 된 파일을 가지고 분석해 보겠다. 올리디버거로 열어 Text strings을 확인해 보면 아래와 같은 그림을 볼 수 있다.

```

00440EA7 ASCII "Unit1"
00440EDC MOV ECX,FFCD7DC6.00440FC8 ASCII "No Name entered"
00440EE1 MOV EDX,FFCD7DC6.00440FD8 ASCII "Enter a Name!"
00440F08 MOV ECX,FFCD7DC6.00440FE8 ASCII "No Serial entered"
00440F0D MOV EDX,FFCD7DC6.00440FFC ASCII "Enter a Serial!"
00440F2F MOV EDX,FFCD7DC6.00441014 ASCII "Registered User"
00440F4C MOV EDX,FFCD7DC6.0044102C ASCII "GFX-754-IER-954"
00440F5A MOV ECX,FFCD7DC6.0044103C ASCII "CrackMe cracked successfully"
00440F5F MOV EDX,FFCD7DC6.0044105C (Initial CPU selection)
00440F74 MOV ECX,FFCD7DC6.00441080 ASCII "Beggart off!"
00440F79 MOV EDX,FFCD7DC6.0044108C ASCII "Wrong Serial,try again!"
00440F8E MOV ECX,FFCD7DC6.00441080 ASCII "Beggart off!"
00440F93 MOV EDX,FFCD7DC6.0044108C ASCII "Wrong Serial,try again!"
00440FC8 ASCII "No Name entered",0

```

Reverse L06 Start

문제를 보면 OEP와 serial 값을 찾는 문제이다. OEP는 Original Entry Point를 뜻한다.

Korea :

Unpack을 한 후 Serial을 찾으시오. 정답인증은 OEP + Serial

Ex) 00400000PASSWORD

English :

Unpack, and find the serial. The solution should be in this format : OEP + Serial

Ex) 00400000PASSWORD

[Down](#)

파일을 다운 받아 바로 올디버거로 열어본다. 패킹이 되어 있음을 알 수 있다. 5번 문제
에서와 같이 Unpack을 수행한다. Unpack을 수행하면 OEP 주소가 00401360임을 알 수
있다.

00401360	55	PUSH EBP
00401361	8BEC	MOV EBP,ESP
00401363	6A FF	PUSH -1

다음은 serial 키를 찾아보도록 한다. serial 키가 만들어지는 방법에는 여러 가지가 있지만
이 RCE에선 이미 입력된 serial 값과 사용자가 입력한 입력값을 비교한다. 그러므로 스트링
을 비교하는 함수나 serial키를 입력받는 함수를 찾아보도록 한다.

0040105C	CALL DWORD PTR DS:[&USER32.GetDlgItemTextA]	USER32.GetDlgItemTextA
00401081	PUSH 40	(Initial CPU selection)
00401094	CALL DWORD PTR DS:[&USER32.MessageBoxA]	USER32.MessageBoxA

문자열을 입력받는 함수를 찾을 수 있다. 해당 함수가 위치한 곳으로 가면 serial값을 찾을
수 있다.

0040105C	FF15 B0524200	CALL DWORD PTR DS:[&USER32.GetDlgItemTextA]	USER32.GetDlgItemTextA
00401062	3BF4	CMP ESI,ESP	
00401064	E8 B7020000	CALL 386D13B0.00401320	
00401069	68 D4354200	PUSH 386D13B0.004235D4	
0040106E	68 302A4200	PUSH 386D13B0.00422A30	ASCII "AD46DFS547"

Reverse L07 Start

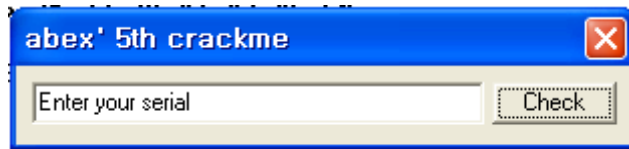
문제를 보니 해당 프로그램이 자신의 C드라이브 이름을 가져와 시리얼값을 생성하는 것 같다.

Korea :
컴퓨터 C 드라이브의 이름이 CodeEngn 일경우 시리얼이 생성될때 CodeEngn은 "어떤것"으로 변경되는가

English :
Assuming the drive name of C is CodeEngn, what does CodeEngn transform into in the process of the serial construction

[Down](#)

다운을 받은 후 실행해보면 사용자로부터 시리얼 값을 입력받는다.



그렇다면 사용자로부터 값을 입력 받는 함수를 찾아 로직을 살펴보도록 하자. GetDlgItemtext를 찾아 브레이크포인트를 설정하고 실행한다.

00401024	CALL <JMP.&KERNEL32.ExitProcess>	kernel32.ExitProcess
00401078	CALL <JMP.&USER32.GetDlgItemTextA>	USER32.GetDlgItemTextA
00401099	CALL <JMP.&KERNEL32.GetVolumeInformationA>	kernel32.GetVolumeInformationA

실행하면 입력창이 활성화 되고 값을 입력하고 check를 눌러 올리디버거로 시리얼값이 어떻게 생성되는지 확인해본다.

00401078	E8 F4000000 CALL <JMP.&USER32.GetDlgItemTextA>	GetDlgItemTextA
0040107D	6A 00 PUSH 0	pFileSystemNameSize = NULL
0040107F	6A 00 PUSH 0	pFileSystemNameBuffer = NULL
00401081	드라이브이름에 추	pFileSystemFlags = BABB04F7.004020C8
00401086	가 입력 스트링값	pMaxFilenameLength = BABB04F7.00402190
00401088		pVolumeSerialNumber = BABB04F7.00402190
00401090		MaxVolumeNameSize = 32 (50.)
00401092	68 5C224000 PUSH BABB04F7.0040225C	VolumeNameBuffer = BABB04F7.0040225C
00401097	6A 00 PUSH 0	ConcatPathName = NULL
00401099	E8 85000000 CALL <JMP.&KERNEL32.GetVolumeInformationA>	GetVolumeInformationA
0040109E	68 F3234000 PUSH BABB04F7.004023F3	StringToAdd = "4562-ABEX"
004010A3	68 5C224000 PUSH BABB04F7.0040225C	ConcatString = ""
004010A8	E8 94000000 CALL <JMP.&KERNEL32.lstrcatA>	lstrcatA
004010AD	B2 02 MOV DL, 2	
004010AF	8305 5C224000 ADD DWORD PTR DS:[40225C], 1	
004010B6	8305 5D224000 ADD DWORD PTR DS:[40225D], 1	
004010BD	8305 5E224000 ADD DWORD PTR DS:[40225E], 1	
004010C4	8305 5F224000 ADD DWORD PTR DS:[40225F], 1	
004010CB	FECA DEC DL	
004010CD	75 E0 JNZ SHORT BABB04F7.004010AF	
004010CF	68 FD234000 PUSH BABB04F7.004023FD	StringToAdd = "L2C-5781"
004010D4	68 00204000 PUSH BABB04F7.00402000	ConcatString = ""
004010D9	E8 63000000 CALL <JMP.&KERNEL32.lstrcatA>	lstrcatA
004010DE	68 5C224000 PUSH BABB04F7.0040225C	StringToAdd = ""
004010E3	68 00204000 PUSH BABB04F7.00402000	ConcatString = ""
004010E8	E8 54000000 CALL <JMP.&KERNEL32.lstrcatA>	lstrcatA
004010ED	68 24234000 PUSH BABB04F7.00402324	String2 = ""
004010F2	68 00204000 PUSH BABB04F7.00402000	String1 = ""
004010F7	E8 51000000 CALL <JMP.&KERNEL32.lstrcpiaA>	lstrcpiaA

문자 순서는 로직

브레이크포인트 지점으로부터 한단계씩 실행해보면 GetVolumeInfomation API로부터 드라이버정보를 가져오고

1. 드라이버이름과 StringToAdd 값을 합친다.
2. 드라이버이름의 순서를 변경
3. 1번과2번 과정을 거친 값과 StringToAdd 값을 합친다.

그 후에 사용자가 입력한 값이랑 비교한다.

이 문제는 드라이버이름이 CodeEngn일 경우 CodeEngn 문자열이 어떻게 변경되는 것인가를 묻는 문제이다. 그러므로 원래의 드라이버이름을 CodeEngn으로 변경해주고 알고리즘을 실행하면 원하는 값이 나온다. CodeEngn의 hex값은 아스키코드표를 참조한다.

The screenshot shows a debugger window with assembly code on the left and a hex dump on the right. A red arrow points from the assembly code to the hex dump. A red text annotation says "이 부분을 Follow in dump 하여 hex 값 수정".

Address	Hex dump	ASCII
0040225C	43 6F 64 65 45 6E 67 6E 00 00 00 00 00 00 00 00	CodeEngn-----
0040226C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	-----

알고리즘이 완료되면 아래그림과 같이 최종값이 생성된다.

The screenshot shows a debugger window with assembly code on the left and a hex dump on the right. A red box highlights the final String1 value "L2C-5781EqfgEngn4562-ABEX".

Address	Hex dump	ASCII
0040109E	68 F3234000	PUSH BABB04F7.004023F3
004010A3	68 5C224000	PUSH BABB04F7.0040225C
004010A8	E8 94000000	CALL <JMP.&KERNEL32.lstrcata>
004010AD	B2 02	MOV DL,2
004010AF	> 8305 5C224000	ADD DWORD PTR DS:[40225C],1
004010B6	8305 5D224000	ADD DWORD PTR DS:[40225D],1
004010BD	8305 5E224000	ADD DWORD PTR DS:[40225E],1
004010C4	8305 5F224000	ADD DWORD PTR DS:[40225F],1
004010CB	FECA	DEC DL
004010CD	^ 75 E0	JNZ SHORT BABB04F7.004010AF
004010CF	68 FD234000	PUSH BABB04F7.004023FD
004010D4	68 00204000	PUSH BABB04F7.00402000
004010D9	E8 63000000	CALL <JMP.&KERNEL32.lstrcata>
004010DE	68 5C224000	PUSH BABB04F7.0040225C
004010E3	68 00204000	PUSH BABB04F7.00402000
004010E8	E8 54000000	CALL <JMP.&KERNEL32.lstrcata>
004010ED	68 24234000	PUSH BABB04F7.00402324
004010F2	68 00204000	PUSH BABB04F7.00402000

Reverse L08 Start

이번 문제는 5번문제와 6번문제에서 다뤘던 Unpack 문제이다.

Korea :

OEP를 구하십시오

Ex) 00400000

English :

Find the OEP

Ex) 00400000

[Down](#)

5번과 같이 Unpack을 수행하면 바로 OEP를 확인할 수 있다.

01012475	6A 70	PUSH 70	
01012477	68 E0150001	PUSH 1BF0364F.010015E0	
0101247C	E8 47030000	CALL 1BF0364F.010127C8	
01012481	33DB	XOR EBX,EBX	
01012483	53	PUSH EBX	
01012484	8B3D 20100001	MOV EDI,DWORD PTR DS:[1001020]	kerne132.GetModuleHandleA
0101248A	FFD7	CALL EDI	
0101248C	66:8138 4D5A	CMP WORD PTR DS:[EAX],5A4D	

Reverse L09 Start

Korea :

StolenByte를 구하시오
Ex) 75156A0068352040

English :

Find the StolenByte
Ex) 75156A0068352040

[Down](#)

문제에서 stolenbyte를 찾으라고 한다. stolenbyte를 직역하자면 훔친 바이트이다.

Stolenbyte란?

: Stolenbyte는 패커가 위치를 이동시킨 코드로써 보호된 프로그램의 코드의 윗부분(보통은 엔트리 포인트의 몇 개의 명령어)이다. 이 부분의 명령어는 이동된 곳이나 할당 받은 메모리 공간에서 실행된다. 보호된 프로세스의 메모리가 덤프 되었을 때 Stolenbyte를 복구하지 못한다면 덤프된 실행파일은 작동하지 않을 수 있다.

먼저 파일을 받아 확인해보면 패킹되어있음을 알 수 있다. 올리디버거로 Unpack을 해주는데 이 파일은 Stolenbyte 때문에 코드를 약간 수정해주어야 한다.

POPAD부분까지와서 확인해보면 JMP를 하기전에 아래 그림처럼 프로그램의 첫 부분에 있을 법한 코드가 여기에 위치하게 된다. 이 부분은 매개변수로 쓰이는데, OEP로 진입하여 덤프를 하게되면 손실되어 프로그램을 실행할 수 없게된다. 그러므로 Unpack 할때 이코드들을 앞으로 이동시켜줘야 한다.

0040736E	. 6A 00	PUSH 0	
00407370	. 68 00204000	PUSH 7EA5E6FB.00402000	ASCII "abex' 3rd crackme"
00407375	. 68 12204000	PUSH 7EA5E6FB.00402012	ASCII "Click OK to check for the keyfile."

OEP 부분으로 오게되면 주소가 0040100C 인걸 확인할 수 있다. 원래는 여기서 덤프를 하면 되겠지만 stolenbyte가 있기 때문에 NOP으로 채워진곳에 Stolenbyte로 채워준다. 이 문제파일은 예제이기 때문에 딱 맞게 빈공간이 존재하지만, 모든 경우가 이렇지는 않다고 한다.

00401000	90	NOP
00401001	90	NOP
00401002	90	NOP
00401003	90	NOP
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	90	NOP
00401008	90	NOP
00401009	90	NOP
0040100A	90	NOP
0040100B	90	NOP
0040100C	6A 00	PUSH 0

00401000에서 Ctrl+E 눌러 Stolenbyte를 채워준다.

00401000	6A 00	PUSH 0	
00401002	68 00204000	PUSH 7EA5E6FB.00402000	ASCII "abex' 3rd crackme"
00401007	68 12204000	PUSH 7EA5E6FB.00402012	ASCII "Click OK to check for the keyfile."

00401000에서 덤프를 해주고 IAT를 복구해주게 되면 Unpack이 완료된다.

Reverse L10 Start

Korea :

OEP를 구한 후 "등록성공"으로 가는 분기점의 OPCODE를 구하시오. 정답인증은 OEP + OPCODE

EX) 00400000EB03

English :

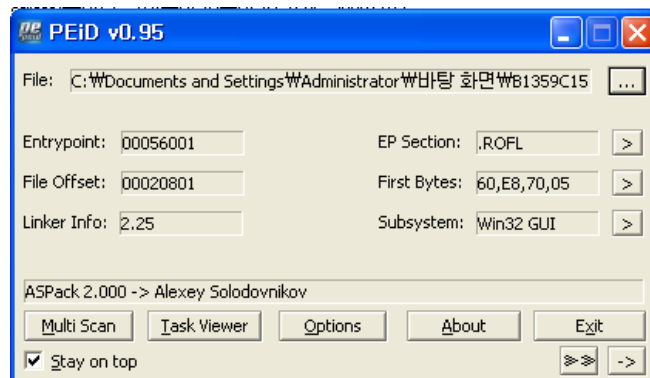
After finding the OEP, find the OPCODE of the branch instruction going to the "goodboy routine"

The solution should be in this format : OEP + Serial

EX) 00400000EB03

[Down](#)

10번문제는 OEP와 분기점의 OPCODE를 구하는 문제이다. 우선 PEID로 해당 파일을 확인해본다. 이 파일은 아래 그림과 같이 Aspack으로 패킹되어 있다.



Aspack을 Unpack 하는 방법은 UPX와 거의 비슷하다. UPX패킹을 Unpack 하는 방법으로 아래그림부분 까지 온다.

00445834	55	DB 55	CHAR 'U'
00445835	8B	DB 8B	
00445836	EC	DB EC	
00445837	83	DB 83	
00445838	C4	DB C4	
00445839	F4	DB F4	
0044583A	B8	DB B8	
0044583B	F4	DB F4	
0044583C	56	DB 56	CHAR 'V'
0044583D	44	DB 44	CHAR 'D'
0044583E	00	DB 00	
0044583F	E8	DB E8	
00445840	04	DB 04	
00445841	08	DB 08	
00445842	FC	DB FC	

여기서 Ctrl+a 를 누르면 코드가 제대로 보이게 되고 덤프를 해주면 된다. 그리고 ImportRCE Tool로 IAT를 복구 시켜준다.

Unpack을 완료한 후 등록성공 분기점의 OPCODE를 찾아본다. Text String을 확인해보면 아래와 같이 'Registered ... well done!' 을 볼 수 있다. 분기점의 OPCODE는 아마 저 string 위에 있을 것이다.

```

00445393 ASCII "15935785264587569231133566485712546985721"
004453D5 MOV EDX,123_..004455A0 ASCII "cm5.dat"
004453ED MOV EDX,123_..0044561C ASCII "Name must be at least 5 characters long!"
00445472 MOV EAX,123_..0044562C ASCII "Registered ... well done!"
0044550C MOV EDX,123_..00445660
004455A0 ASCII "15935785264587569231133566485712546985721"

```

00445D4에서 분기점 OPCODE를 발견할 수 있다.

```

004454D4 . 75 55 JNZ SHORT 123_..0044552B
004454D6 . 8D85 F4FDFFF1 LEA EAX,DWORD PTR SS:[EBP-20C]
004454DC . 8D95 17FEFFF1 LEA EDX,DWORD PTR SS:[EBP-1E9]
004454E2 . E8 1DE6FBFF CALL 123_..00403B04
004454E7 . 8B95 F4FDFFF1 MOV EDX,DWORD PTR SS:[EBP-20C]
004454ED . 8B87 D4020000 MOV EAX,DWORD PTR DS:[EDI+2D4]
004454F3 . E8 B4F5FDFF CALL 123_..00424AAC
004454F8 . 8B87 D8020000 MOV EAX,DWORD PTR DS:[EDI+2D8]
004454FE . 8B55 FC MOV EDX,DWORD PTR SS:[EBP-4]
00445501 . E8 A6F5FDFF CALL 123_..00424AAC
00445506 . 8B87 E8020000 MOV EAX,DWORD PTR DS:[EDI+2E8]
0044550C . BA 60564400 MOV EDX,123_..00445660 ASCII "Registered ... well done!"
00445511 . E8 96F5FDFF CALL 123_..00424AAC

```

Reverse L11 Start

Korea :

OEP를 찾으시오. Ex) 00401000 / Stolenbyte 를 찾으시오. Ex) FF35CA204000E84D000000

정답인증은 OEP+ Stolenbyte

Ex) 00401000FF35CA204000E84D000000

English :

Find the OEP. Ex) 00401000 / Find the Stolenbyte. Ex) FF35CA204000E84D000000

The solution should be in this format : OEP + Serial

Ex) 00401000FF35CA204000E84D000000

[Down](#)

11번 문제는 9번에서 사용된 파일과 동일한 파일이므로 다르게 없다.

Reverse L12 Start

Korea :

Key를 구한 후 입력하게 되면 성공메시지를 볼 수 있다

이때 성공메시지 대신 Key 값이 MessageBox에 출력 되도록 하려면 파일을 HexEdit로 오픈 한 다음

0x???? ~ 0x???? 영역에 Key 값을 overwrite 하면 된다.

문제 : Key값과 + 주소영역을 찾으시오

Ex) 77777777????????

English :

You will see a success message after finding the key.

If you would want the Key itself to replace the success message in the MessageBox,

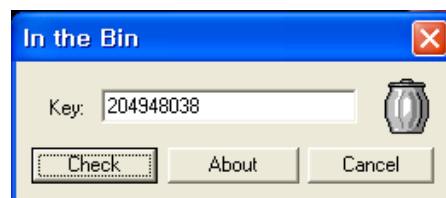
open up a Hex Editor and overwrite the key value in the offset range 0x???? ~ 0x????.

Q : find the key value and the offset range and write the solution in this format : key????????

(first ??? for the start and the next 4 ?s for the end).

[Down](#)

이 문제는 key 값과 주소영역을 찾아야 하는 문제이다. 먼저 파일을 실행해보면 key값을 입력받는 부분이 있다. 값을 입력하고 체크를 누르면 아무런 반응이 없다. 틀린 값을 넣으면 반응이 없는 것 같다.



올리디버거로 분석해본다. 일단은 값을 입력받는 함수(GetDlgItemInt)부분에 브레이크포인트를 걸어 실행해보자. 사용자로부터 입력받은 값은 EAX에 저장되고 아래그림과 같은 알고리즘을 반복하게 된다.

0040105F	E8 31010000	CALL <JMP.&USER32.GetDlgItemInt>	GetDlgItemInt
00401063	BE 00304000	MOV ESI,B643D2BD.00403000	ASCII "0q1qb4EhM/4jISMj1zQf6kpGQwLrG+GEIY4bPcOJL/jWB
00401068	> 833E 00	CMP DWORD PTR DS:[ESI],0	
0040106B	> 75 04	JNZ SHORT B643D2BD.00401071	
0040106D	> EB 0E	JMP SHORT B643D2BD.0040107D	
0040106F	> EB 0C	JMP SHORT B643D2BD.0040107D	
00401071	> 8B1E	MOV EBX,DWORD PTR DS:[ESI]	
00401073	E8 97000000	CALL B643D2BD.0040110F	
00401078	83C6 04	ADD ESI,4	
0040107B	EB EB	JMP SHORT B643D2BD.00401068	

간단하게 분석해보면

1. ESI에 ASCII 값이 저장된 주소값을 mov 한다
2. ASCII 값을 4바이트 단위씩 0과 비교한다.
3. zero flag가 0이 아닐 경우 00401071로 점프
4. 4바이트의 ASCII 값을 EBX에 넣는다.
5. 알고리즘을 호출한다.

아래는 5번에서 호출하는 알고리즘이다.

0040110F	\$ 51	PUSH ECX
00401110	. 52	PUSH EDX
00401111	. 8BD3	MOV EDX,EBX
00401113	. 8BC8	MOV ECX,EAX
00401115	. 40	INC EAX
00401116	. F7D0	NOT EAX
00401118	. 43	INC EBX
00401119	. F7D3	NOT EBX
0040111B	. 40	INC EAX
0040111C	. 43	INC EBX
0040111D	. 23C2	AND EAX,EDX
0040111F	. 23D9	AND EBX,ECX
00401121	. 03C3	ADD EAX,EBX
00401123	. 5A	POP EDX
00401124	. 59	POP ECX
00401125	. C3	RETN

EAX, EBX(ASCII값), ECX, EDX 을 가지고, 즉 EAX(입력한 값)과 EBX(ASCII값)값을 가지고 연산하고, 위의 다섯단계 로직에서 봤듯이 ASCII 값이 0이 나올때 까지 반복된다.

모든 반복이 끝나면

CMP EAX, 7A2896BF

를 수행하고 두 개의 값이 같을 경우 key값이 인증되게 된다.

그렇다면 우린 여기서 우리가 입력한 EAX값과 ASCII값이 어떤 식으로 계산되는지 알아내려고 할 것이다. 하지만 이것은 속임수이다. 해당값을 알아내려고 하면 더욱 패닉에 빠질 것이다. 여기서 ASCII 값을 유심히 보면 대칭됨을 알 수 있다. 이 의미는 EAX값을 ASCII 값에 의해 값이 변경되는것 처럼 보이지만 결국에는 본래의 EAX값으로 되돌아 옴을 의미한다. 이는 CMP EAX, 7A2896BF 비교구문에 브레이크포인트를 걸어 EAX값을 확인해봄으로써 알 수 있다.

Reverse L13 Start

Korea :

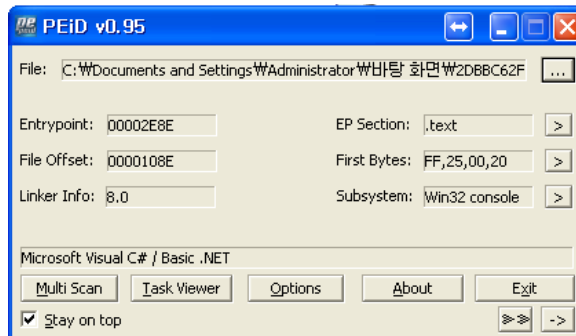
정답은 무엇인가

English :

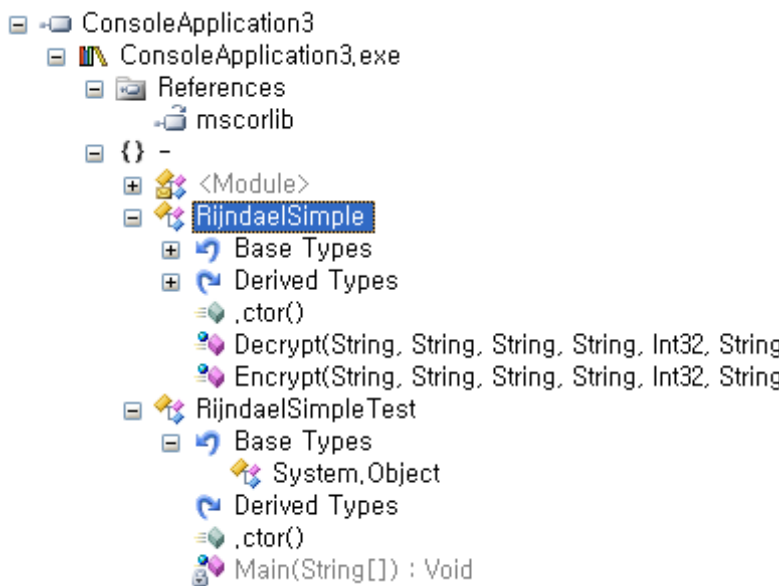
Find the answer

[Down](#)

문제가 아주 간단하다. 파일을 받아 실행해본다. 패스워드를 입력하라고 한다. 패스워드를 찾기 위해 올리디버거로 열어보지만 C#으로 작성된 파일이라 올리디버거로 분석이 되지 않는다.



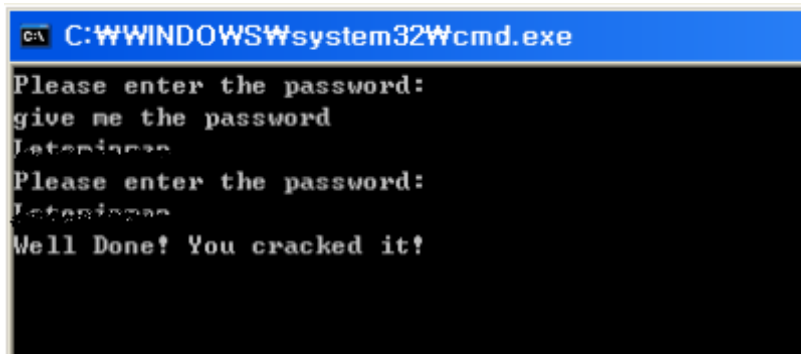
C#으로 작성되었으니 .NET Reflector 로 소스코드를 확인해본다.



보아하니 Rijndael 이란 사람의 암호알고리즘인 것 같다. c#을 다룰줄 모른다면 소스코드를 분석하여 다른 언어로 작성해도 되겠지만 아는사람이라면 소스코드를 그대로 가져가서 사용하면 된다. 단 `using System.Security.Cryptography;` `using System.IO;` 두 개만 추가해 준다. 모든 코드를 옮긴 후 `plaintext`를 출력되도록 코드를 수정하면 된다.

```
[STAThread]
private static void Main(string[] args)
{
    string plainText = "";
    string cipherText = "BnCxBiN4aJDE+qUe2yIm8Q==";
    string passPhrase = "~F79e!k56$#x00a3";
    string saltValue = "DHj47&+)$h";
    string hashAlgorithm = "MD5";
    int passwordIterations = 0x400;
    string initVector = "&!#x00a3$X^&+( )CvHgE!";
    int keySize = 0x100;
    RijndaelSimple.Encrypt(plainText, passPhrase, saltValue, hashAlgorithm, passwordIterations, initVector, keySize);
    plainText = RijndaelSimple.Decrypt(cipherText, passPhrase, saltValue, hashAlgorithm, passwordIterations, initVector,
Label_0056:
    Console.WriteLine("Please enter the password: ");
    if (Console.ReadLine() == plainText)
    {
        Console.WriteLine("Well Done! You cracked it!");
        Console.ReadLine();
    }
    else
    {
        Console.WriteLine(plaintext);
        goto Label_0056;
    }
}
```

틀린 암호를 입력하면 `else`문이 실행될 것이니 `else`문에 `plaintext`가 출력되도록 수정하면 된다.



Reverse L14 Start

Korea :

Name이 CodeEngn 일때 Serial을 구하시오

(이 문제는 정답이 여러개 나올 수 있는 문제이며 5개의 숫자로 되어있는 정답을 찾아야함, bruteforce 필요)

Ex) 11111

English :

Find the Serial when the Name of CodeEngn

(This problem has several answers, and the answer should be a 5 digit number. Brute forcing is required.)

Ex) 11111

[Down](#)

이 프로그램은 name에 따른 serial을 생성한다. 그러므로 올디버거로 해당 파일을 열어 어떤 알고리즘으로 serial을 생성하는지 확인해보자. 우선 Upack을 수행하고 분석한다. 사용자로부터 입력받은 값을 가지고 시리얼을 생성하는 알고리즘이 시작하는 부분을 찾아 브레이크포인트를 건다. 이 프로그램에선 사용자가 입력한 name 문자열 길이 만큼 알고리즘을 반복하는데, 문자열길이 값을 리턴하는 strlen에서 브레이크 포인트를 걸어주면 적합할 것 같다.

004012F6	68 38304000	PUSH 1,00403038	ASCII "CodeEngn"	Registers (FPU)
004012FB	E8 30010000	CALL <JMP.&kernel32.lstrlen>		EAX 00000008
00401300	33F6	XOR ESI,ESI		ECX 00000008

브레이크포인트를 걸고 CodeEngn을 입력하니 8(문자열길이) 이라는 값이 ECX에 들어갔다.

00401309 ~ 0040132C 까지 ECX값만큼 알고리즘을 반복한다.

00401309	8B15 38304000	MOV EDX,DWORD PTR DS:[403038]
0040130F	8A90 37304000	MOV DL,BYTE PTR DS:[EAX+403037]
00401315	81E2 FF000000	AND EDX,0FF
0040131B	8BDA	MOV EBX,EDX
0040131D	0FAFDA	IMUL EBX,EDX
00401320	03F3	ADD ESI,EBX
00401322	8BDA	MOV EBX,EDX
00401324	D1FB	SAR EBX,1
00401326	03F3	ADD ESI,EBX
00401328	2BF2	SUB ESI,EDX
0040132A	40	INC EAX
0040132B	49	DEC ECX
0040132C	75 DB	JNZ SHORT 1.00401309

해당 알고리즘을 분석하지 못하더라도 CodeEngn을 입력했을때 생성되는 serial값은 찾을 수 있다.

00401339	5E	POP ESI	
0040133A	3BC6	CMP EAX,ESI	
0040133C	75 15	JNZ SHORT 1.00401353	
0040133E	6A 00	PUSH 0	
00401340	68 62344000	PUSH 1.00403462	ASCII "Key/CrackMe #2 "
00401345	68 B8344000	PUSH 1.004034B8	ASCII " Good Job, I Wish You the Ver
0040134A	6A 00	PUSH 0	
0040134C	E8 9D000000	CALL <JMP.&USER32.MessageBoxA>	
00401351	EB 13	JMP SHORT 1.00401366	
00401353	6A 00	PUSH 0	
00401355	68 62344000	PUSH 1.00403462	ASCII "Key/CrackMe #2 "
0040135A	68 86344000	PUSH 1.00403486	ASCII " You Have Enter A Wrong Serie

이유는 성공메시지를 띄우기 위해 비교하는 값은 EAX와 ESI임을 알 수 있다. 즉 우리는 알고리즘을 모르더라도 name값에 따라 EAX와 ESI값이 어떻게 변하는지만 분석하면 된다. 확인해보면 EAX에 사용자가 입력한 시리얼값이 들어가고 ESI에 name값에 따른 시리얼값이 생성된다. 그러므로 CMP 윗 부분에 브레이크 포인트를 걸고 CodeEngn을 입력했을때의 시리얼값 즉 ESI 값을 10진수로 변경한 값을 입력하면 된다.

Reverse L15 Start

Korea :

Name이 CodeEngn일때 Serial을 구하시오

English :

Find the Serial when the Name is CodeEngn

[Down](#)

이 문제 역시 CodeEngn일때 serial 값을 찾는 문제이다. 14번 문제와 푸는방식이 같다. 성공메시지 주위의 CMP를 찾아 사용자가 입력하는 값에 따라 생성되는 시리얼값이 어디에 저장되는지 확인하여 값을 구할 수 있다.

00458831	. 3B05 44B84500	CMP EAX, DWORD PTR DS:[45B844]	
00458837	.. 75 1B	JNZ SHORT 311F4179.00458854	
00458839	. B8 88884500	MOV EAX, 311F4179.00458888	ASCII "You cracked the UBC CrackMe#1
0045883E	. E8 29C1FEFF	CALL 311F4179.0044496C	
00458843	. BA E8884500	MOV EDX, 311F4179.004588E8	ASCII "CRACKED"
00458848	. A1 3CB84500	MOV EAX, DWORD PTR DS:[45B83C]	
0045884D	. E8 9ECDFCFF	CALL 311F4179.004255F0	
00458852	.. EB 0A	JMP SHORT 311F4179.0045885E	
00458854	> B8 F8884500	MOV EAX, 311F4179.004588F8	ASCII "Try Again !"
00458859	. E8 0EC1FEFF	CALL 311F4179.0044496C	
0045885F	.. 80 00	CBP EAX, EAX	

이 문제에선 사용자가 입력한 시리얼값이 EAX에 저장되고 name값에 따른 시리얼값은 DS:[45B844] 부분에 저장됨을 알 수 있다. 그러므로 CodeEngn 값을 입력하고 DS:[45B844]에 있는값을 10진수 값으로 변경해주면 되겠다.

Reverse L16 Start

Korea :

Name이 CodeEngn일때 Serial을 구하시오

English :

Find the Serial when the Name is CodeEngn

[Down](#)

이번 문제도 14,15번 문제와 같이 CodeEngn일때 시리얼값을 구하는 문제이다.
성공메시지 주위의 CMP 구문을 찾아 분석해본다.

0040159F	. 3B45 C4	CMP EAX, DWORD PTR SS:[EBP-3C]	
004015A2	. 0F85 9400000	JNZ 1F651B57.0040163C	
004015A8	. C70424 F5FFF	MOV DWORD PTR SS:[ESP], -0B	
004015AF	. E8 8CF60000	CALL <JMP.&KERNEL32.GetStdHandle>	GetStdHandle
004015B4	. 83EC 04	SUB ESP, 4	
004015B7	. C74424 04 0A	MOV DWORD PTR SS:[ESP+4], 0A	
004015BF	. 890424	MOV DWORD PTR SS:[ESP], EAX	
004015C2	. E8 89F60000	CALL <JMP.&KERNEL32.SetConsoleTextAttri	SetConsoleTextAttribute
004015C7	. 83EC 08	SUB ESP, 8	
004015CA	. C74424 04 A8	MOV DWORD PTR SS:[ESP+4], 1F651B57.0048B	
004015D2	. C70424 C0334	MOV DWORD PTR SS:[ESP], 1F651B57.004433C	
004015D9	. E8 528D0200	CALL 1F651B57.0042A330	
004015DE	. C74424 04 D9	MOV DWORD PTR SS:[ESP+4], 1F651B57.00440	ASCII " Good Job!☺"

EAX값과 SS:[EBP-3C]에 있는 값을 비교한다. EAX에는 사용자가 입력한 password 값이 들어가게 되고 SS:[EBP-3C]에는 사용자가 입력한 name에 따른 password가 생성된다.

Reverse L17 Start

Korea :

Key 값이 BEDA-2F56-BC4F4368-8A71-870B 일때 Name은 무엇인가

힌트 : Name은 한자리인데.. 알파벳일수도 있고 숫자일수도 있고..

정답인증은 Name의 MD5 해쉬값(대문자)

English :

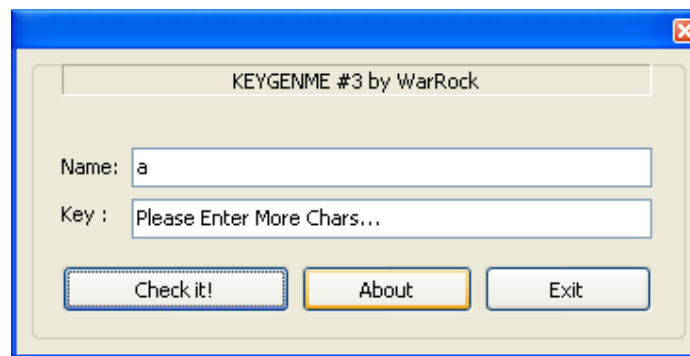
What is Name when the Key is BEDA-2F56-BC4F4368-8A71-870B

Hint : The name is 1 letter and it could be either alphabetic or numeric.

Verify your solution with the MD5 value of the Name.

[Down](#)

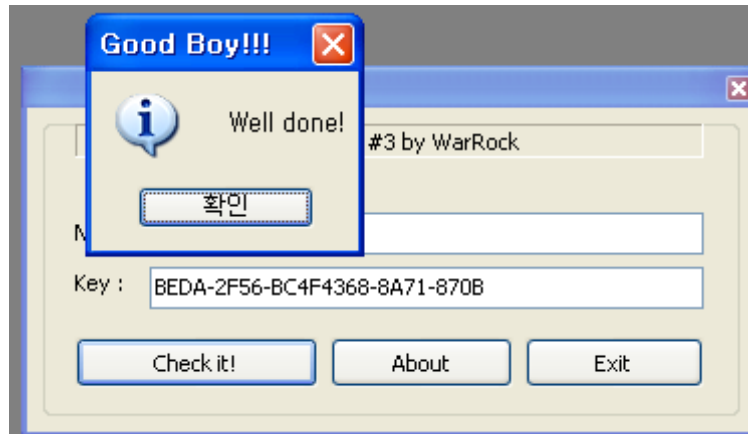
이번 문제는 이전 문제와 달리 key값에 따른 name값을 구하는 문제이다. 자신이 분석능력이 뛰어나다면 알고리즘을 분석하여 프로그래밍해서 name값을 구하는것이 가장 좋은 방법 이긴 하나 그렇지 못하다면 문제의 힌트를 가지고 bruteforce 해보는 방법이 있다. 힌트에서 문자열이 길이가 한자리고 했으니 많이 걸리지 않을것 같다.



key값에 주어진 값을 넣고 name에 한글자를 넣었더니 문자를 더 입력하라고 한다. 이 부분은 올리디버거로 열어 조건문을 수정해주면 된다.

```
0045BB24 |> 83F8 01 | CMP EAX, 1 |
0045BB27 |> 7D 15 | JGE SHORT E683EC0B.0045BB3E |
0045BB29 |> BA 18BC4500 | MOV EDX, E683EC0B.0045BC18 | ASCII "Please Enter More Chars..."
```

string으로 찾아 조건문의 3을 1로 변경해주고 패치하면 문자 하나를 입력해도 에러가 나지 않는다.



Reverse L18 Start

Korea :

Name이 CodeEngn일때 Serial은 무엇인가

English :

Find the Serial when the Name is CodeEngn.

[Down](#)

CodeEngn일때 serial 값을 구하는 문제이다. 이 문제는 strcmp 비교함수를 이용해 사용자가 입력한 값과 name값으로 생성된 시리얼값을 비교한다.

```
004011E5 . 68 F0804000 PUSH 5AF8B382.004080F0 String2 = "0"
004011EA . 68 F07E4000 PUSH 5AF8B382.00407E00 String1 = "1"
004011EF . E8 DA000000 CALL <JMP.&kernel32.lstrcmpiA> lstrcmpiA
```

Reverse L19 Start

KO : 이 프로그램은 몇 밀리세컨드 후에 종료 되는가

EN : How many milliseconds does it take for this program to terminate

[Down](#)

19번 문제는 프로그램이 실행된 후 종료되는 시간을 확인하는 문제이다. 프로그램을 실행하면 아래그림같이 메시지박스를 띄우고 시간 지나면 사라진다.



먼저 UPX로 패키징되어있기 때문에 Unpack을 해주고 안티디버깅이 되어 있으므로 IsDebuggerPresent()함수를 우회하고 파일을 저장하여 올리디버거로 분석한다. 이 문제는 몇 밀리세컨드 뒤에 종료되는지 확인하는 문제니 시간에 관련된 함수를 있나 찾아본다.

```
0040B350 CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
0040E6CA CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
004301F3 CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
004305BC CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
0043197F CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
00431D70 CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
00431EED CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
00444C44 CALL EDI (Initial CPU select
00451882 CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
0045189E CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
00456F68 CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
0046FBC1 CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
0046FBCD CALL DWORD PTR DS:[<&WINMM.timeGetTime>] WINMM.timeGetTime
```

timeGetTime이라는 함수를 찾을 수 있다. 이 함수는 윈도우가 시작되어서 지금까지 흐른 시간을 1/1000초(milliseconds) 단위로 DWORD 형을 리턴하는 함수이다. 모든 timeGetTime 함수에 브레이크포인트를 건 후 프로그램을 실행해본다. 이 알고리즘에 대해 간단히 보자면

1. timeGetTime 함수를 호출하여 현재시간을 EAX에 리턴한다.
2. EAX값을 ESI로 복사한다.
3. timeGetTime 함수를 또 호출하여 현재시간을 EAX에 리턴한다. 이 때의 리턴값은 1번의 리턴값 보다 큰 값일 것이다.
4. EAX값에서 ESI값을 뺀다.
5. EAX값과 DS:[EBX+ 4] 를 비교하여 EAX값이 DS:[EBX+ 4]값보다 크다면 프로그램을 실행한다. 그렇지 않으면 다시 timeGetTime 함수를 호출하여 EAX에 새로운 리턴값을 저장하고 4번과 5번을 조건이 만족할 때 까지 다시 수행한다.

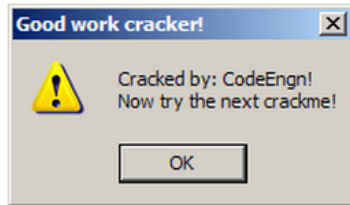
00444C3E	. 8B3D 58D74700	MOV EDI,DWORD PTR DS:[&WINMM.timeGetTime]	WINMM.timeGetTime
00444C44	. FFD7	CALL EDI	WINMM.timeGetTime; <&WINMM.timeGetTime>
00444C46	. 803D D3E84800	CMP BYTE PTR DS:[48E8D3],0	1번째 호출
00444C4D	. 8BF0	MOV ESI,EAX	
00444C4F	. 0F84 FF000000	JE B5352594.00444D54	
00444C55	. 8B5C24 14	MOV EBX,DWORD PTR SS:[ESP+14]	
00444C59	. 8B2D 58D14700	MOV EBP,DWORD PTR DS:[&KERNEL32.Sleep]	kerne132.Sleep
00444C5F	> FFD7	CALL EDI	
00444C61	. 3BC6	CMP EAX,ESI	2번째 호출
00444C63	. 0F83 CF000000	JNB B5352594.00444C66	
00444C69	. 2BC6	SUB EAX,ESI	
00444C6B	. 48	DEC EAX	
00444C6C	. E9 C9000000	JMP B5352594.00444D3A	

EDI에 timeGetTime 함수의 주소를 넣고 call EDI로 호출하는 것을 볼 수 있다.

00444D38	> 2BC6	SUB EAX,ESI
00444D3A	> 3B43 04	CMP EAX,DWORD PTR DS:[EBX+4]
00444D3D	. ^ 0F83 2EFFFFFF	JNB B5352594.00444C71

계산을 하고 DS:[EBX+4]에 있는 값보다 EAX값이 작을 경우 다시 반복하고 작지 않을 경우 프로그램을 실행한다. 즉 원하는값은 DS:[EBX+4] 여기에 저장되어 있다.

Reverse L20 Start



Korea :

이 프로그램은 Key파일을 필요로 하는 프로그램이다.

위 문구가 출력되도록 하려면 crackme3.key 파일안의 데이터는 무엇이 되어야 하는가

Ex) 41424344454647

(정답이 여러개 있는 문제로 인증시 맞지 않다고 나올 경우 게시판에 비공개로 올려주시면 확인해드리겠습니다)

English :

This program needs a key file.

What does the data in the file crackme3.key have to be to make it print the above message.

Ex) 41424344454647

(This problem has multiple answers, so post your answer on the messageboard in a private thread and we will verify it for you.)

[Down](#)

crackme3.key 파일안의 데이터를 찾는 문제이다. 올리디버거로 열어 살펴보도록 한다. 먼저 CreateFileA 함수를 호출하는데 이 함수는 파일을 열고 핸들을 반환한다. 이는 dwCreationDistribution 자리를 확인하면 알 수 있다.

```
00401016 | . 6A 00          PUSH 0
00401018 | . 68 80000000    PUSH 80
0040101D | . 6A 03          PUSH 3
0040101F | . 6A 00          PUSH 0
00401021 | . 6A 03          PUSH 3
00401023 | . 68 000000C0   PUSH C0000000
00401028 | . 68 D7204000   PUSH 42564675.004020D7
0040102D | . E8 76040000    CALL <JMP. &KERNEL32.CreateFileA>
hTemplateFile = NULL
Attributes = NORMAL
Mode = OPEN_EXISTING
pSecurity = NULL
ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE
Access = GENERIC_READ|GENERIC_WRITE
FileName = "CRACKME3.KEY"
CreateFileA
```

다음으로 ReadFile 함수를 호출하여 생성된 파일로부터 0x12바이트(18바이트)를 읽어온다.

```
00401052 | . 6A 00          PUSH 0
00401054 | . 68 A0214000    PUSH 42564675.004021A0
00401059 | . 50            PUSH EAX
0040105A | . 53            PUSH EBX
0040105B | . FF35 F5204000 PUSH DWORD PTR DS:[4020F5]
00401061 | . E8 30040000    CALL <JMP. &KERNEL32.ReadFile>
pOverlapped = NULL
nBytesRead = 42564675.004021A0
BytesToRead => 12 (18.)
Buffer => 42564675.00402008
hFile = 00000084
ReadFile
```


다음은 읽어온 문자열을 바꿔주는 알고리즘이다. 여기서 파일에 입력한값이 CodeEngn값으로 바뀌어야 한다.

```

00401311 |$ 33C9      XOR ECX, ECX
00401313 |. 33C0      XOR EAX, EAX
00401315 |. 8B7424 04  MOV ESI, DWORD PTR SS:[ESP+4]
00401319 |. B3 41     MOV BL, 41
0040131B |> 8A06     MOV AL, BYTE PTR DS:[ESI]
0040131D |. 32C3     XOR AL, BL
0040131F |. 8806     MOV BYTE PTR DS:[ESI], AL
00401321 |. 46       INC ESI
00401322 |. FEC3     INC BL
00401324 |. 0105 F9204000 ADD DWORD PTR DS:[4020F9], EAX
0040132A |. 3C 00     CMP AL, 0
0040132C |. 74 07     JE SHORT 42564675.00401335
0040132E |. FEC1     INC CL
00401330 |. 80FB 4F  CMP BL, 4F
00401333 |.^ 75 E6     JNZ SHORT 42564675.0040131B
00401335 |> 890D 49214000 MOV DWORD PTR DS:[402149], ECX
0040133B |. C3       RETN

```

1. BL 레지스터에 0x41 값을 넣는다.
2. AL 레지스터에 CRACKME3.KEY안의 문자열중 1바이트를 가져온다.
3. AL 레지스터와 BL 레지스터를 XOR 연산하고 PTR DS:[ESI]에 원래문자대신 넣는다.
4. ESI 증가, BI 증가
5. XOR한 값을 PTR DS:[4020F9]에 더한다.
6. AL이 0일 경우 점프, 즉 AL과 BL이 같으면 점프
7. 0부터 CL을 1씩 증가시키고 BL과 4F랑 비교하여 같지 않으면 1번부터 다시반복, 같으면 빠져나간다.

위의 CRACKME3.KEY의 데이터와 위의 알고리즘을 가지고 CodeEngn을 찍으려면 직접 계산하는 방법도 있겠지만 프로그래밍을 하면 더욱 수월 하겠다. 프로그래밍을 할때 주의할점은 5번째 문자가 해당 알고리즘을 수행할 때 BL에 0x45값이 들어가는데 이때 CodeEngn의 E(0x45)가 들어가므로 AL레지스터와 BL레지스터 값이 같게 되므로 알고리즘이 종료되게 된다. 이를 방지하기 위해 0x1D 값을 사용한다. AL^BL=0x1D가 될 경우 아무런 값도 표시되지 않는다. 18바이트중 14바이트만 문자변환을 수행하는데 14바이트중 CodeEngn은 8바이트이므로 나머지 6바이트는 0x1D로 만들어준다.

```

#include <stdio.h>
#include <string.h>
void main(){
    char name[]={'C','o','d','e','X','E','n','g','n'};
    char buf;
    int i,k,j,h;
    char xor=0x41;
    char xor1=0x4A;

    for(k=0;k<9;k++){
        {
            for(i=0x0 ; i<=0x7F ; i++){
                buf=i^xor;
                if(buf==name[k])
                    printf("%c",i);
            }
            xor=xor+0x01;
        }
        printf("\n");
        printf("-----0x1D-----\n");

        for(j=0x4A ; j<=0x4F ; j++){
            {
                for(h=0x01 ; h<=0x7F ; h++){
                    if((j^h)==0x1D)
                        printf("%x",h);
                }
            }
        }
    }
}

```

메시지박스를 띄우기 위해 해당 점프문을 NOP으로 바꿔 메시지박스를 출력한다.

00401188	. 3C 01	CMP AL, 1		
0040118A	90	NOP		
0040118B	90	NOP		
0040118C	. 68 86214000	PUSH 42564675.00402186		ASCII "Now try the next crackme!"
00401191	. 68 6A214000	PUSH 42564675.0040216A		ASCII "Cracked by: Now t
00401196	. 68 08204000	PUSH 42564675.00402008		
0040119B	. E8 C2010000	CALL 42564675.00401362		

메시지박스를 출력한 화면이다.

