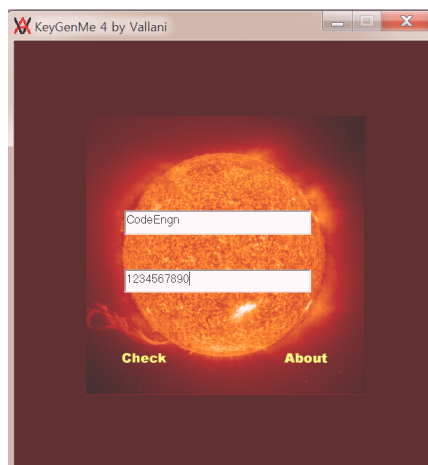


## 코드 엔진 Challenges: Advance 03

Author: Vallani

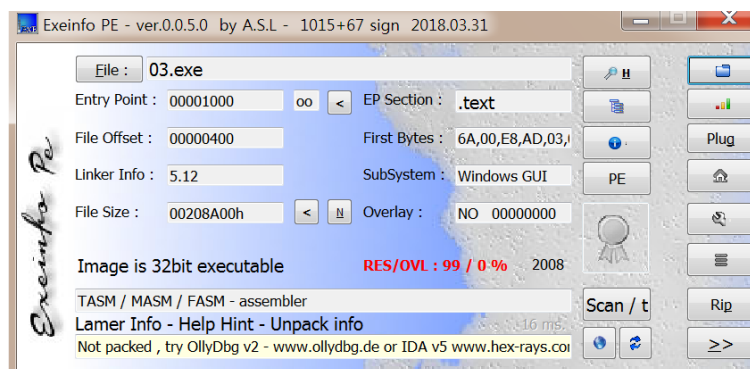
Korean: Name이 CodeEngn 일 때 Serial 은 무엇인가

문제의 유형은 지난 Basic에서도 많이 보았던 문제이다 .



파일을 실행하고 CodeEngn을 입력 후 Serial을 입력해보았다.

PEID를 통해서 패킹 여부와 특이점을 살펴보고 프로그램을 분석해보도록 하자.



패킹도 따로되어 있지 않고 별다른 특이점이 보이지 않는다. 올리디버거를 통해 파일을 분석해보자.

Address	Disassembly	Destination
00401000	PUSH 0	(Initial CPU selection)
00401002	CALL <JMP.&kernel32.GetModuleHandleA>	kernel32.GetModuleHandleA
00401020	CALL <JMP.&user32.DialogBoxParamA>	user32.DialogBoxParamA
00401027	CALL <JMP.&kernel32.ExitProcess>	kernel32.ExitProcess
00401066	CALL <JMP.&winmm.PlaySoundA>	winmm.PlaySoundA
00401076	CALL <JMP.&user32.LoadIconA>	user32.LoadIconA
00401086	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
00401099	CALL <JMP.&kernel32.VirtualProtect>	kernel32.VirtualProtect
004010A6	CALL <JMP.&user32.GetDlgItem>	user32.GetDlgItem
004010AC	CALL <JMP.&user32.SetFocus>	user32.SetFocus
004010B6	CALL <JMP.&user32.GetWindowLongA>	user32.GetWindowLongA
004010C6	CALL <JMP.&user32.SetWindowLongA>	user32.SetWindowLongA
004010D7	CALL <JMP.&user32.SetLayeredWindowAttributes>	user32.SetLayeredWindowAttributes
004010EC	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401120	CALL <JMP.&user32.GetDlgItemTextA>	user32.GetDlgItemTextA
00401138	CALL <JMP.&user32.MessageBoxA>	user32.MessageBoxA
00401162	CALL <JMP.&user32.GetDlgItemTextA>	user32.GetDlgItemTextA
00401177	CALL <JMP.&user32.wsprintfA>	user32.wsprintfA
00401196	CALL <JMP.&kernel32.lstrcpA>	kernel32.lstrcpA
004011AC	CALL <JMP.&user32.MessageBoxA>	user32.MessageBoxA
004011C8	CALL <JMP.&user32.MessageBoxA>	user32.MessageBoxA
004013A6	CALL <JMP.&user32.MessageBoxA>	user32.MessageBoxA

올리디버거를 통해 열어보고 함수들이 대강 무엇이 쓰였는지 확인해보았다 우리가 몇 번 보았던 GetDlgItemTextA와 MessageBox 가 눈에 띈다 .

※GetDlgItemText 박스가 GetDlgItemText 멤버 함수 포인터가 가리키는 위치에 텍스트를 복사 lpStr 및 복사 하는 바이트의 개수를 반환 한다는 것을 기억하고 가자.

Address	Disassembly	Text string
0040102C	PUSH EBP	(Initial CPU selection)
0040112C	PUSH 03.00403116	ASCII "You failed..."
00401131	PUSH 03.004030F1	ASCII "No, that is not the right answer :)"
00401160	PUSH 03.004030B4	ASCII "No"
00401100	PUSH 03.00403116	ASCII "You failed..."
00401105	PUSH 03.004030F1	ASCII "No, that is not the right answer :)"
0040110C	PUSH 03.00403221	ASCII "About"
004011C1	PUSH 03.00403124	ASCII "KeyGenMe 4 by Vallani))Solutions for this challenge are only keyGens, no patches, no selfgens, no loaders ))Greeting
0040139A	PUSH 03.004030E0	ASCII "You succeeded..."
0040139F	PUSH 03.00403087	ASCII "Yes, you see the Good Boy Message :) , now write a KeyGen to master my little challange!"

위는 이 파일에서 사용된 문자열이다. “You failed” 나 “You Success”라는 문자열이 보이는 것으로 봐서 이 문자열들이 사용된 주소 가까이에 분기문이 있을 것이라고 예측된다. 일단 “You Success”가 실행되는 주소로 이동해보자.

0040130B	MOV DWORD PTR SS:[EBP+4],EAX	
0040138E	LEAVE	
0040138F	RET 4	
00401392	MOV EBP,DWORD PTR DS:[40325C]	
00401398	PUSH 30	
0040139A	PUSH 03.004030E0	Structured exception handler
0040139F	PUSH 03.00403087	Style = MB_OK MB_ICONEXCLAMATION MB_APPL
004013A4	PUSH 0	Title = "You succeeded..."
004013A6	CALL <JMP.&user32.MessageBoxA>	Text = "Yes, you see the Good Boy Messag
004013AB	RET 4	hOwner = NULL
		MessageBoxA

이 근처의 코드에 You Failed도 있을 것 같아 일단 이곳에 BP를 설정 후 위로 계속 이동해 보았다.

00401153	. 6A 20	PUSH 20	Count = 20 (32.)
00401155	. 68 64324000	PUSH 03.00403264	Buffer = 03.00403264
0040115A	. 68 E0300000	PUSH 3ED	ControlID = 3ED (1005.)
0040115F	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
00401162	. E8 77020000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401167	. FF35 00304000	PUSH DWORD PTR DS:[403000]	<u> = 0
0040116D	. 68 84304000	PUSH 03.00403084	Format = "%u"
00401172	. 68 84324000	PUSH 03.00403284	s = 03.00403284
00401177	. E8 40020000	CALL <JMP.&user32.wsprintfA>	wsprintfA
0040117C	. 83C4 0C	ADD ESP,0C	
0040117F	. 33C0	XOR EAX,EAX	
00401181	. A3 00304000	MOV DWORD PTR DS:[403000],EAX	
00401186	. 892D 5C324000	MOV DWORD PTR DS:[40325C],EBP	
0040118C	. 68 64324000	PUSH 03.00403264	String2 = ""
00401191	. 68 84324000	PUSH 03.00403284	String1 = ""
00401196	. E8 25020000	CALL <JMP.&kernel32.lstrcmpA>	lstrcmpA
0040119B	. 99	CDQ	
0040119C	. F7F8	IDIV EAX	
0040119E	. 6A 10	PUSH 10	Style = MB_OK MB_ICONHAND MB_APPLMODAL
004011A0	. 68 16314000	PUSH 03.00403116	Title = "You failed..."
004011A5	. 68 F1304000	PUSH 03.004030F1	Text = "No, that is not the right answer"
004011AA	. 6A 00	PUSH 0	hOwner = NULL
004011AC	. E8 3F020000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
004011B1	. > EB 1E	JMP SHORT 03.004011D1	
004011B3	. > 3D E0300000	CMP EAX,3EB	
004011B8	. > 75 17	JNZ SHORT 03.004011D1	
004011BA	. 6A 40	PUSH 40	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
004011BC	. 68 21324000	PUSH 03.00403221	Title = "About"
004011C1	. 68 24314000	PUSH 03.00403124	Text = "KeyGenMe 4 by Vallani's Solutions"
004011C6	. 6A 00	PUSH 0	hOwner = NULL
004011C8	. E8 23020000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA

문자열이 같은지 비교해 주는 함수인 lstrcmpA 함수가 보인다. 이곳이 시리얼과 입력받은 값을 비교하는 부분이라고 추측할 수 있다. 이곳에서 어떤 값을 받고 어떤 값이 어디에 저장되는지 직접 실행하여 확인해보자. 프로그램에 CodeEngn 그리고 5를 입력 한 후

00401125	. 83F8 03	CMP EAX,3	
00401128	. > 73 10	JNB SHORT 03.00401142	
00401129	. 6A 10	PUSH 10	Style = MB_OK MB_ICONHAND MB_APPLMODAL
0040112C	. 68 16314000	PUSH 03.00403116	Title = "You failed..."
00401131	. 68 F1304000	PUSH 03.004030F1	Text = "No, that is not the right answer :)"
00401136	. 6A 00	PUSH 0	hOwner = NULL
00401138	. E8 B3020000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401139	. > EB 1E	JMP SHORT 03.004011D1	
0040113A	. > 3D E0300000	CMP EAX,3EB	ASCII "CodeEngn"
0040113B	. > 75 17	JNZ SHORT 03.004011D1	
00401147	. A3 58324000	MOV DWORD PTR DS:[403258],EAX	
0040114C	. 6A 00	PUSH 0	
0040114E	. 68 24010000	PUSH 03.00401277	
00401153	. 6A 20	PUSH 20	Count = 20 (32.)
00401155	. 68 64324000	PUSH 03.00403264	Buffer = 03.00403264
00401158	. 68 E0300000	PUSH 3ED	ControlID = 3ED (1005.)
0040115F	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
00401162	. E8 77020000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401167	. FF35 00304000	PUSH DWORD PTR DS:[403000]	<u> = 0
0040116D	. 68 84304000	PUSH 03.00403084	Format = "%u"
00401172	. 68 84324000	PUSH 03.00403284	s = 03.00403284
00401177	. E8 40020000	CALL <JMP.&user32.wsprintfA>	wsprintfA
0040117C	. 83C4 0C	ADD ESP,0C	
0040117F	. 33C0	XOR EAX,EAX	
00401181	. A3 00304000	MOV DWORD PTR DS:[403000],EAX	
00401186	. 892D 5C324000	MOV DWORD PTR DS:[40325C],EBP	
0040118C	. 68 64324000	PUSH 03.00403264	String2 = ""
00401191	. 68 84324000	PUSH 03.00403284	String1 = ""
00401196	. E8 25020000	CALL <JMP.&kernel32.lstrcmpA>	lstrcmpA
0040119B	. 99	CDQ	
0040119C	. F7F8	IDIV EAX	
0040119E	. 6A 10	PUSH 10	Style = MB_OK MB_ICONHAND MB_APPLMODAL
004011A0	. 68 16314000	PUSH 03.00403116	Title = "You failed..."
004011A5	. 68 F1304000	PUSH 03.004030F1	Text = "No, that is not the right answer :)"
004011AA	. 6A 00	PUSH 0	hOwner = NULL
004011AC	. E8 3F020000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
004011B1	. > EB 1E	JMP SHORT 03.004011D1	
004011B3	. > 3D E0300000	CMP EAX,3EB	
004011B8	. > 75 17	JNZ SHORT 03.004011D1	
004011BA	. 6A 40	PUSH 40	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
004011BC	. 68 21324000	PUSH 03.00403221	Title = "About"
004011C1	. 68 24314000	PUSH 03.00403124	Text = "KeyGenMe 4 by Vallani's Solutions for this challenge are only keyGens, no patches,
004011C6	. 6A 00	PUSH 0	hOwner = NULL
004011C8	. E8 23020000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA

CodeEngn 문자열이 00401142에 들어간 것이 보인다

00401142 입력한 CodeEngn이 EAX에 저장됨

00401147 CodeEngn이 저장된 EAX값이 00403258에 4바이트만큼 옮겨짐

0040114C push 0

0040114E 03.00401277을 호출

00401153 버퍼의 크기 =>20

00401155 push 03.00403264 컨트롤의 텍스트를 받을 버퍼를 가리킴

0040115A 컨트롤의 ID

0040115F 대화 상자의 핸들

00401162 Call Jmp GetDlgItemTextA를 호출

\*wsprintfA 함수는 원하는 문자열을 담고 싶을 때 사용한다. 리턴되는 값은 담은 문자열의 개수다. 반복문으로 일일이 옮기는 과정과 차이점은 서식문자도 지원하는 것이다.

예를들어 `wsprintf(buffer, "$s", "1")`이런 식으로 사용할 수 있다.

함수의 원형은

```
int __cdecl wsprintf(  
    _Out_ LPSTR lpOut, //서식화된 출력을 내보낼 문자열 버퍼  
    _In_ LPSTR lpFmt, //서식문자열  
    _In_ ... //서식 문자열 내의 서식과 대응될 변수를 서식의 개수만큼 변수제공  
);이다
```

00401167 403000의 값(C2A776FA=>3265754874)을 push 한다.

0040116D 03.403084 포맷지정 %u

00401172 03.00403284 push

00401177 call jmp wsprintf 호출

0040117C 0C를 ESP에 더한다.

0040117F EAX를 XOR 한다. 0이된다.

00401181 EAX의 값을 00403000 4바이트 만큼 복사한다. 08383D3B000000이된다.

00401186 EBP의 값을 0040325C에 4바이트만큼 복사한다.

0040118C 00403264의 값을 푸쉬함 우리가 입력한 5가 비교를 위해 push된다.

00401191 00403284의 값이 푸쉬되는데 "3265754874"값이다.

00401196 lstrcmpA 함수에의해 비교된다.

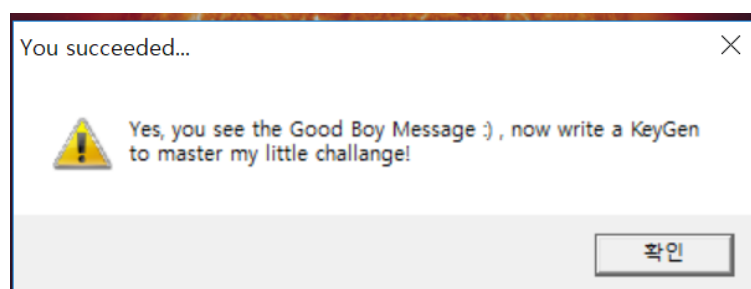
0040119B CDQ

0040119C IDIV EAX

004010119E~ 실패 메시지가뜸

lstrcmpA에서 우리가 입력한 값과 00403284에 저장된 값이 다르자 실패메시지가 호출된다.

00403284에 저장된 3265754874를 입력하고 인증을 해보자.



성공메시지가뜸다.

※CodeEngn을 입력 후 Serial을 만드는 코드 부분

00401286	> 58	POP EAX
00401287	. 33C9	XOR ECX, ECX
00401288	> 8B1419	MOV DL, BYTE PTR DS:[ECX+EBX]
00401289	. 3BF2 38	XOR DL, 38
0040128F	. 8B10	MOV BYTE PTR DS:[EBX], DL
004012C1	. 40	INC EBX
004012C3	. 41	INC ECX
004012C9	. 83F9 1C	CMPL ECX, 1C
004012CB	. ^72 F1	JNB SHORT 004012B9