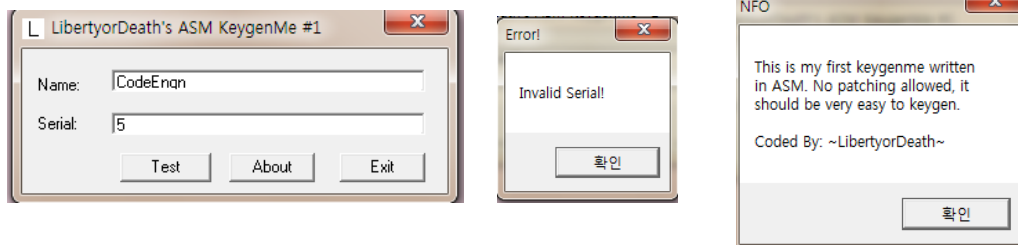


코드 엔진 Challenges: Advance 04

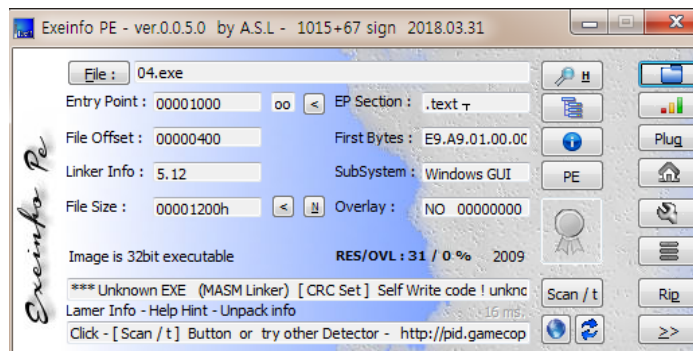
Author: LibertyorDeath

Korean: Name이 CodeEngn 일 때 Serial 은 무엇인가

문제는 지금까지 많이 풀어왔던 유형과 비슷하게 Name에 따라 달라지는 Serial을 찾아내는 것이다. 파일을 실행해서 CodeEngn 을 입력하고 Serial에 아무 값이나 넣었을 때 Invalid serial 이라는 에러 메시지가 나온다.



또 특이하게 About 버튼을 누르면 어셈블리어로 제작되었다는 메시지를 보여준다. PEID로 열어여부를 확인하고 바로 올리디버거로 분석해보자.



어떠한 정보도 나와있지않다. 바로 올리디버거로 분석해보자.

00401000	\$.E9 A9010000	JMP 04.0040110E	
00401005	90	NOP	
00401006	> 4F	DEC EDI	
00401007	25 CDE42425	AND EAX, 2524E4CD	
0040100C	25 86991565	AND EAX, 65159986	
00401011	25 4F254D13	AND EAX, 134D254F	
00401016	35 65254F25	XOR EAX, 254F2565	
0040101B	4D	DEC EBP	
0040101C	CC	INT3	
0040101D	26:25 25DA1095	AND EAX, 9910DA25	
00401023	15 6525CD90	ADC EAX, 90CD2565	
00401028	24 25	AND AL, 25	
0040102A	25 86011465	AND EAX, 65140186	
0040102F	25 75CDB724	AND EAX, 24B7CD75	
00401034	25 2570AEC9	AND EAX, C9AE7025	
00401039	A4	MOVS BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]	
0040103A	58	POP EAX	
0040103B	29	DB 29	CHAR '):'
0040103C	35	DB 35	CHAR '5'
0040103D	24	DB 24	CHAR '\$'
0040103E	25	DB 25	CHAR 'y'

코드가 이상하고 사용된 함수 목록과 문자열을 봤을 때 어떠한 정보도 알 수 없다. 우리가 알지 못하는 패킹방법으로 패킹되어 있는 것 같다. 일단 위의 코드를 그대로 분석해보도록 하자.

-00401000 004011AE로 점프

-004011AE 00401006의 값을 ESI에 복사한다. ESI에는 00401006 함수가 저장된다.

-004011B3 ESI에 들어있는 데이터의 1바이트 값을 AL 에 복사

```
DS:[00401006]=4F ('O')
AL=33 ('3')
Jump from 004011C0
```

-004011B5 AL과 25를 xor 한다.

```
AL=4F ('O')
```

-004011B7 AL의 값을 ESI의 1바이트 만큼 복사 해서 ESI에 저장한다.

```
AL=6A ('j')
DS:[00401006]=4F ('O')
```

-004011B9 ESI의 값을 1만큼 증가 시킨다 .00401006 증가 =>00401007

-004011BA ESI의 값과 004011A7 비교해서 같지 않으면 점프한다.

```
004011A7=04.004011A7
ESI=00401007 (04.00401007)
```

-004011C0 위의 값이 같지않으므로 004011B3으로 점프한다.

만약 같았다면 004011C2에서 00401006으로 분기하게된다. !

004011C2에 BP를 걸고 실행하여 f8을 눌러 실행하니

00401006	> 6A 00	PUSH 0	
00401008	? E8 C1010000	CALL <JMP.&kernel32.GetModuleHandleA>	
0040100D	? A3 BC304000	MOV DWORD PTR DS:[4030BC],EAX	
00401012	? 6A 00	PUSH 0	
00401014	? 68 36104000	PUSH 04.00401036	
00401019	? 6A 00	PUSH 0	
0040101B	? 68 E9030000	PUSH 3E9	
00401020	? FF35 BC304000	PUSH DWORD PTR DS:[4030BC]	
00401026	? E8 B5010000	CALL <JMP.&user32.DialogBoxParamA>	
0040102B	? A3 24314000	MOV DWORD PTR DS:[403124],EAX	
00401030	? 50	PUSH EAX	
00401031	? E8 92010000	CALL <JMP.&kernel32.ExitProcess>	
00401036	? 55	PUSH EBP	
00401037	? 8BEC	MOV EBP,ESP	
00401039	? 817D 0C 10010000	CMP DWORD PTR SS:[EBP+C],110	
00401040	? 75	DB 75	CHAR 'u'
00401041	? 8B	DB 8B	CHAR 'a'

다음과 같은 코드가 나온다. 이 부분이 OEP인 것 같다. 일단 이 부분을 OEP로 해서 덤프를 떠보자.

00401006	6A 00	PUSH 0	pModule = NULL
00401008	E8 C1010000	CALL <JMP.&kernel32.GetModuleHandleA>	GetModuleHandleA
0040100D	A3 BC304000	MOV DWORD PTR DS:[4030BC],EAX	
00401012	6A 00	PUSH 0	lParam = NULL
00401014	68 36104000	PUSH 4_dump.00401036	DlgProc = 4_dump.00401036
00401019	6A 00	PUSH 0	hOwner = NULL
0040101B	68 E9030000	PUSH 3E9	pTemplate = 3E9
00401020	FF35 BC304000	PUSH DWORD PTR DS:[4030BC]	hInst = NULL
00401026	E8 B5010000	CALL <JMP.&user32.DialogBoxParamA>	DialogBoxParamA
0040102B	A3 24314000	MOV DWORD PTR DS:[403124],EAX	
00401030	50	PUSH EAX	
00401031	E8 92010000	CALL <JMP.&kernel32.ExitProcess>	ExitCode
00401036	55	PUSH EBP	ExitProcess
00401037	8BEC	MOV EBP,ESP	
00401039	817D 0C 100100	CMP DWORD PTR SS:[EBP+C1],110	
00401040	75 38	JNZ SHORT 4_dump.00401070	

덤프 뜯 파일을 열자 제대로 된 형태의 코드를 볼 수 있다 문자열 찾기를 통해 이제까지 해본 대로 해보도록 하자.

00401149	0FAFFE	IMUL EDI,ESI	
0040114C	3B0D 00314000	CMP ECK,DWORD PTR DS:[40310D]	
00401152	75 E3	JNZ SHORT 4_dump.00401137	
00401154	57	PUSH EDI	<%IX>
00401155	56	PUSH ESI	<%Iu>
00401156	68 0E304000	PUSH 4_dump.0040300E	Format = "LOD-Xlu-XIX"
0040115B	68 04314000	PUSH 4_dump.00403104	s = 4_dump.00403104
00401160	E8 75000000	CALL <JMP.&user32.wsprintfA>	~wsprintfA
00401165	83C4 10	ADD ESP,10	
00401168	68 E0304000	PUSH 4_dump.004030E0	String2 = ""
0040116D	68 04314000	PUSH 4_dump.00403104	String1 = ""
00401172	E8 5D000000	CALL <JMP.&kernel32.lstrcmpA>	~lstrcmpA
00401177	83F8 00	CMP EAX,0	
0040117A	5F	POP EDI	
0040117B	75 14	JNZ SHORT 4_dump.00401191	
0040117D	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
0040117F	68 92304000	PUSH 4_dump.00403092	Title = "Vay!"
00401184	68 84304000	PUSH 4_dump.00403084	Text = "Valid Serial!"
00401189	6A 00	PUSH 0	hOwner = NULL
0040118B	E8 6E000000	CALL <JMP.&user32.MessageBoxA>	~MessageBoxA
00401190	C3	RET	
00401191	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
00401193	68 A7304000	PUSH 4_dump.004030A7	Title = "Error!"
00401198	68 97304000	PUSH 4_dump.00403097	Text = "Invalid Serial!"
0040119D	6A 00	PUSH 0	hOwner = NULL
0040119F	E8 5D000000	CALL <JMP.&user32.MessageBoxA>	~MessageBoxA
004011A4	68 24314000	PUSH 4_dump.00403124	ExitCode = 403124
004011A9	E8 1A000000	CALL <JMP.&kernel32.ExitProcess>	~ExitProcess
004011AE	> BE 06104000	MOV ESI,4_dump.<ModuleEntryPoint>	
004011B3	> 8006	MOV AL,BYTE PTR DS:[ESI]	
004011B5	> 34 25	XOR AL,25	

지금까지와 비슷하게 성공했을때의 문자열 ,실패했을때의 문자열을 호출하는 메시지 박스와 함께 그 근처에 lstrcmpA와 함께 분기문이 있는 것이 보인다. 비교문에 bp를 걸고 실행을해 보자.

00401147	8BF0	MOV EDI,EDX	
00401149	0FAFFE	IMUL EDI,ESI	
0040114C	3B0D 00314000	CMP ECK,DWORD PTR DS:[40310D]	
00401152	75 E3	JNZ SHORT 4_dump.00401137	
00401154	57	PUSH EDI	<%IX>
00401155	56	PUSH ESI	<%Iu>
00401156	68 0E304000	PUSH 4_dump.0040300E	Format = "LOD-Xlu-XIX"
0040115B	68 04314000	PUSH 4_dump.00403104	s = 4_dump.00403104
00401160	E8 75000000	CALL <JMP.&user32.wsprintfA>	~wsprintfA
00401165	83C4 10	ADD ESP,10	
00401168	68 E0304000	PUSH 4_dump.004030E0	String2 = "5"
0040116D	68 04314000	PUSH 4_dump.00403104	String1 = "LOD-59919-A0024900"
00401172	E8 5D000000	CALL <JMP.&kernel32.lstrcmpA>	~lstrcmpA
00401177	83F8 00	CMP EAX,0	
0040117A	5F	POP EDI	
0040117B	75 14	JNZ SHORT 4_dump.00401191	
0040117D	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
0040117F	68 92304000	PUSH 4_dump.00403092	Title = "Vay!"
00401184	68 84304000	PUSH 4_dump.00403084	Text = "Valid Serial!"
00401189	6A 00	PUSH 0	hOwner = NULL
0040118B	E8 6E000000	CALL <JMP.&user32.MessageBoxA>	~MessageBoxA

lstrcmpA에 우리가 입력한 값과 CodeEngn에 대한 시리얼 값이 나오는 것을 알 수 있다. Serial을 얻는 방법은 앞의 3번 문제와 완전히 유사하고 이 문제의 의도는 풀어보지 못한 새로운 유형의 패킹문제를 경험하게 하려는 의도로 보인다.