

코드 엔진 Challenges: Basic 14

Author: BENGALY

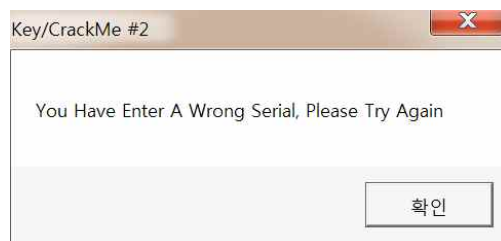
Korean: Name이 CodeEngn일 때 Serial을 구하시오

(이 문제는 정답이 여러개 나올 수 있는 문제이며 5개의 숫자로 되어있는 정답을 찾아야 함, BruteForce 필요)

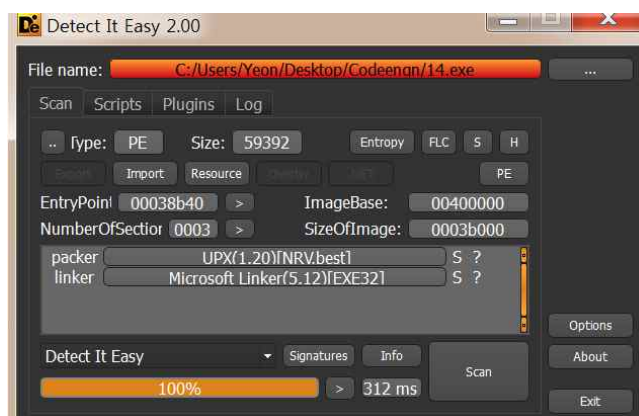
Ex) 11111

시리얼값을 생성하는 알고리즘을 찾아서 분석하는게 문제의 키이다. 문제를 통해서 얻을 수 있는 약간의 정보는 이름에 따라서 시리얼이 바뀐다는 것이다.

문제를 확인했으니 파일을 다운로드 받아서 실행해보자.



위와 같은 화면이 나와서 아무값이나 넣어보니 "You~"이러한 문자가 나왔다. 아직은 어떠한 힌트도 얻지못하였다. DE를 통해서 패킹여부를 확인해보자.



DE로 파일을 열어보니 이 파일이 UPX로 패킹되어있다는 것을 알 수 있다. 그리고 올리디버거로 열어보니 따로 StolenByte가 된건 아닌거 같다.

UPX를 통해 언패킹한후 올리디버거로 파일을 열어분석해보자.

(언패킹은 많이 해봤으니 생략)

00401000	6A 00	PUSH 0	pModule = NULL
00401002	E8 23040000	CALL <JMP.&KERNEL32.GetModuleHandleA>	GetModuleHandleA
00401007	A3 F0344000	MOV DWORD PTR DS:[4034F0],EAX	
0040100C	50	PUSH EAX	
0040100D	E8 13000000	CALL Basic_14.00401025	Arg1 Basic_14.00401025
00401012	6A 00	PUSH 0	ExitCode = 0
00401014	E8 0B040000	CALL <JMP.&KERNEL32.ExitProcess>	ExitProcess
00401019	6A 6A	DB 6A	CHAR 'j'
0040101A	00	DB 00	
0040101B	E8	DB E8	
0040101C	0A	DB 0A	
0040101D	04	DB 04	
0040101E	00	DB 00	
0040101F	00	DB 00	
00401020	A3	DB A3	
00401021	F0344000	DD Basic_14.004034F0	
00401025	55	PUSH EBP	
00401026	8BEC	MOV EBP,ESP	
00401028	83C4 B0	ADD ESP,-50	
0040102B	6A 64	PUSH 64	RsrcName = 100.
0040102D	FF35 F0344000	PUSH DWORD PTR DS:[4034F0]	hInst = NULL
00401033	E8 B0030000	CALL <JMP.&USER32.LoadIconA>	LoadIconA
00401038	8945 E8	MOV DWORD PTR SS:[EBP-18],EAX	

이 프로그램은 이름값을 입력받고 시리얼값을 생성하기 때문에 어느 한 주소에 시리얼값이 저장되어있진 않을 것이다. 사용자 입력값을 가져와서 사용하는 방식이라는 것을 예상할 수 있을 것이다. 시리얼 키를 찾기위해 문자열들을 확인해보자.

00401026	MOV EBP,ESP	(Initial CPU selection)
00401072	MOV DWORD PTR SS:[EBP-8],Basic_14.004034F0	ASCII "Bengaly"
00401080	PUSH Basic_14.00403021	ASCII "MainWindow"
00401179	PUSH Basic_14.004034DC	ASCII "Key/CrackMe - #2"
00401278	PUSH Basic_14.00403462	ASCII "Key/CrackMe #2"
0040127C	PUSH Basic_14.0040323C	ASCII "Key/CrackMe #2"
004012E1	PUSH Basic_14.00403462	ASCII "Key/CrackMe #2"
004012E6	PUSH Basic_14.00403000	ASCII "Please Fill in 1 more Char!!"
00401348	PUSH Basic_14.00403462	ASCII "Key/CrackMe #2"
00401345	PUSH Basic_14.004034B8	ASCII "Good Job, I Wish You the Very Best"
00401355	PUSH Basic_14.00403462	ASCII "Key/CrackMe #2"
00401358	PUSH Basic_14.00403486	ASCII "You Have Enter A Wrong Serial, Please Try Again "

시리얼키와 가장 근접한 문자열일 거 같은 "Good~"문자열의 주소로 이동하자.

0040132B	49	DEC ECX	
0040132C	75 DB	JNZ SHORT Basic_14.00401309	
0040132E	56	PUSH ESI	
0040132F	68 38314000	PUSH Basic_14.00403138	
00401334	E8 4A000000	CALL Basic_14.00401383	
00401339	5E	POP ESI	
0040133A	3BC6	CMP EAX,ESI	
0040133C	75 15	JNZ SHORT Basic_14.00401353	
0040133E	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
00401340	68 62344000	PUSH Basic_14.00403462	Title = "Key/CrackMe #2 "
00401345	68 B8344000	PUSH Basic_14.004034B8	Text = " Good Job, I Wish You the
0040134A	6A 00	PUSH 0	hOwner = NULL
0040134C	E8 9D000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
00401351	EB 13	JMP SHORT Basic_14.00401366	
00401353	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
00401355	68 62344000	PUSH Basic_14.00403462	Title = "Key/CrackMe #2 "
0040135A	68 86344000	PUSH Basic_14.00403486	Text = " You Have Enter A Wrong Se
0040135F	6A 00	PUSH 0	hOwner = NULL
00401361	E8 88000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
00401366	EB 15	JMP SHORT Basic_14.0040137D	
00401368	FF75 14	PUSH DWORD PTR SS:[EBP+14]	lParam
0040136B	FF75 10	PUSH DWORD PTR SS:[EBP+10]	wParam
0040136E	FF75 0C	PUSH DWORD PTR SS:[EBP+0C]	Message

위 코드를 잘 보면 JNZ라는 분기문이 보인다. 00401353으로 가보니 시리얼값이 틀렸을때 나오는 문자열을 띄우는 함수가 보인다. 분기문의 전을 살펴보면 EAX와 ESI의 값을 비교할 수 있다. 비교하는 EAX와 ESI값은 어떤 값이 들어간지 알기위해 분기문 근처의 함수에 BP를 걸고 진행해보자.

분기문의 근처함수는 strlenA 함수이고 이 함수는 문자열의 길이를 반환해준다. 함수에 대한 반환값은 EAX에 저장되는 것을 보면 시리얼의 길이를 비교하는 것을 예상 할 수 있다. 확인해보기위해 실행을 해보자.

#lstrlenA 함수

//지정된 문자열의 길이를 결정합니다 (종료 널 문자 제외).

int lstrlenA(

LPCSTR lpString

);

매개 변수

-lpString

=유형 : LPCTSTR

=검사 할 Null 종료 문자열입니다.

반환 값

=유형 : int

=이 함수는 문자열의 길이를 문자로 반환합니다. lpStringNULL일 경우 , 함수는 0을 반환합니다.



Address	Disassembly	Comment
004012ED	E8 FC000000	CALL <JMP.8USER32.MessageBox>
004012F2	C9	LEAVE
004012F3	C2 1000	RETN 10
004012F6	> 68 38304000	PUSH Basic_14.00403038
004012F8	E8 30010000	CALL <JMP.8KERNEL32.lstrlenA>
00401300	33F6	XOR ESI,ESI
00401302	8BC8	MOV ECX,EAX
00401304	B8 01000000	MOV EAX,1
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

Register	Value
EAX	00000000
ECX	75DC730C KERNELBA.75DC730C
EDX	00403038 ASCII "odeEngn"
EBX	00000000
ESP	0012FBEC
EBP	0012FBEC
ESI	00000111
EDI	0012FC68
EIP	00401300 Basic_14.00401300

예측했던대로 lstrlenA의 값은 EAX에 저장되는 것을 알 수 있다.

그 밑의 코드를 따라가면

004012F6 PUSH 403038 name 값 (codeengn)

004012F8 CALL 00401430 lstrlen 함수 호출 길이 값 EAX에 저장

00401300 XOR ESI,ESI ESI 0 으로 초기화

00401302 MOV ECX,EAX ECX 에 EAX 값 저장

00401304 MOV EAX,1 EAX 값에 1 저장

00401309 MOV EDX,DWORD PTR DS:[403038]

EDX 에 첫번째 부터 4번째 문자열이 저장

0040130F MOV DL,BYTE PTR DS:[EAX+403037]

DL 에 첫번째 문자를 저장

00401315 AND EDX,0FF EDX 에 0FF 를 AND 시켜서 EDX 에 DL 많이 남도록 함

0040131B MOV EBX,EDX EDX 값을 EBX에 저장

0040131D IMUL EBX,EDX EBX 값과 EDX 값을 곱한다. 즉 제공 값을 EBX 에 저장

00401320 ADD ESI,EBX ESI 값에 EBX (제공 값) 를 더한다.

00401322 MOV EBX,EDX EDX 값을 EBX에 저장한다. 첫글자를 EBX에 저장한다.

00401324 SAR EBX,1 쉬프트 연산을 진행한다. (S : 쉬프트 A : 산술 R : 우측)

00401326 ADD ESI,EBX ESI 값에 EBX 값을 더한다. (제공값 + 첫글자 쉬프트 연산값)

00401328 SUB ESI,EDX ESI 값에서 다시 EDX 값 (첫글자) 을 빼준다.
 0040132A INC EAX EAX 값에 1을 증가시킨다. 반복문 카운팅 용도
 0040132B DEC ECX ECX 값에 1을 감소시킨다. 반복문 카운팅 용도
 0040132C JNZ SHORT 00401309
 ECX 값이 0이면 (문자의 길이 만큼 반복하면) 점프
 0040132E PUSH ESI ESI 값을 PUSH
 0040132F PUSH 403138 입력한 Serial 값을 PUSH
 00401334 CALL 00401383 401383 (16진수로 변환하는 코드) 호출
 00401339 POP ESI ESI 값을 POP
 0040133A CMP EAX,ESI EAX 와 ESI 비교
 0040133C JNZ SHORT 00401353 같으면 분기

0040132C	75 DB	JNZ SHORT Basic_14.00401309	
0040132E	56	PUSH ESI	
0040132F	68 38314000	PUSH Basic_14.00403138	ASCII "asdf"
00401334	E8 4A000000	CALL Basic_14.00401383	
00401339	5E	POP ESI	
0040133A	3BC6	CMP EAX,ESI	
0040133C	75 15	JNZ SHORT Basic_14.00401353	
00403138=Basic_14.00403138 (ASCII "asdf")			

*입력한 시리얼값이 입력되는 곳은 00403138이다.

이런식인데 우리가 찾고자하는 것은 성공 메시지가 나왔을때의 시리얼값이다.
 결국 레지스터값을 확인해보게 되면 EAX는 사용자 입력값이며 그와 비교되는 값인 ESI의 10진수 변환한 값이 우리가 찾고자 한 값이다.

004012FB	E8 30010000	CALL <JMP.&KERNEL32.1strlen@>	strlen@
00401300	33F6	XOR ESI,ESI	
00401302	8BC8	MOV ECX,EAX	
00401304	B8 01000000	MOV EAX,1	
00401309	> 8B15 38304000	MOV EDI,DWORD PTR DS:[4030381]	
0040130F	8A90 37304000	MOV DL,BYTE PTR DS:[EAX+4030371]	
00401315	81F2 FF000000	AND EDX,0FF	
0040131B	8BD0	MOV EBX,EDX	
0040131D	0FAFDA	IMUL EBX,EDX	
00401320	03F3	ADD ESI,EBX	
00401322	8BD0	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 Basic_14.00401309	
0040132E	56	PUSH ESI	
0040132F	68 38314000	PUSH Basic_14.00403138	ASCII "asdf"
00401334	E8 4A000000	CALL Basic_14.00401383	
00401339	5E	POP ESI	
0040133A	3BC6	CMP EAX,ESI	
0040133C	75 15	JNZ SHORT Basic_14.00401353	

Registers (FPU)	
EAX	00000009
ECX	00000000
EDX	0000000E
EBX	00000037
ESP	0012F814
EBP	0012FBEC
ESI	000129A1
EDI	0012FC68
EIP	0040132F Basic_14.0040132F
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFDF000(FFF)
T 0	GS 0000 NULL
D 0	
O 0	LastErrr ERROR_SUCCESS (00000000)
EFL	00000246 (NO_MB,E,BE,MS,PE,GE,IE)
ST0	empty 0.0
ST1	empty 0.0
ST2	empty 0.0

이때 ESI값은 00129A1이다. 십진수로 변환해보면 76193이다

