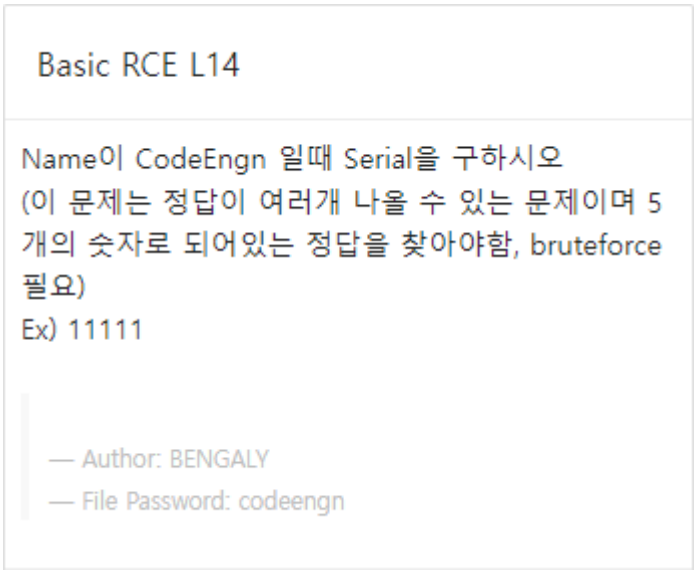
codeengn-basic-L14 풀이

리버싱 문제풀이 / Wonlf / 2022. 4. 8. 15:17



Basic RCE L14

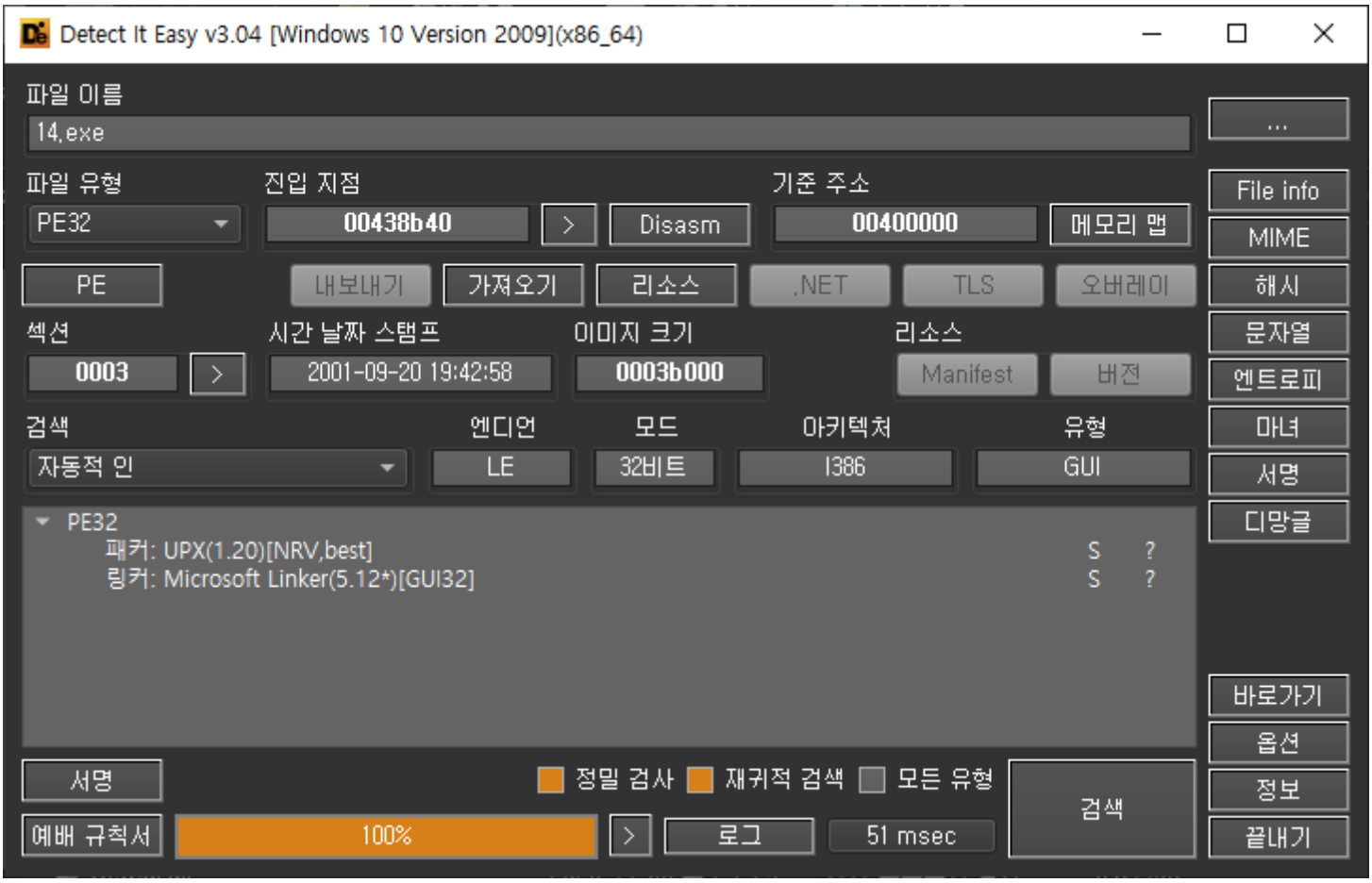
Name이 CodeEngn 일때 Serial을 구하시오
(이 문제는 정답이 여러개 나올 수 있는 문제이며 5개의 숫자로 되어있는 정답을 찾아야함, bruteforce 필요)
Ex) 11111

— Author: BENGALY
— File Password: codeengn

이번 문제는 Name이 CodeEngn일 때, Serial을 원한다.

문제에는 bruteforce가 필요하다고 하지만, 코드 짜는것 없이 풀 수 있었다.

첫번째로 Die로 확인해보니 UPX로 패킹이 되어 있다.

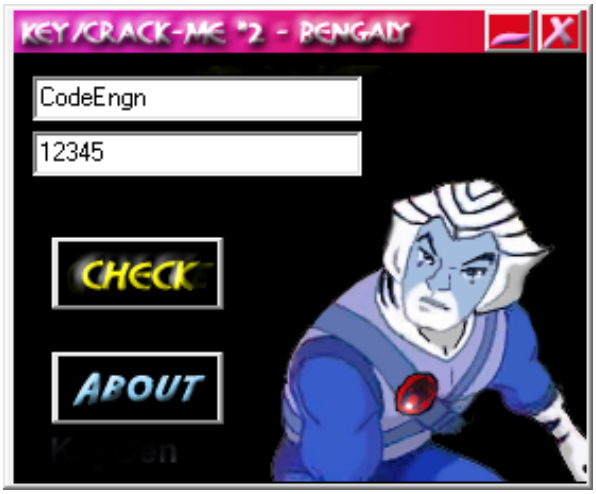


언패커로 언패킹 해주고 디버거로 열어보면,

```
0040133A  .  3BC6          cmp eax,esi
0040133C  .v 75 15         jne 14.401353
0040133E  .  6A 00         push 0
00401340  .  68 62344000   push 14.403462                  403462:"Key/CrackMe #2    "
00401345  .  68 B8344000   push 14.4034B8                  4034B8:" Good Job, I Wish You the Very Best"
0040134A  .  6A 00         push 0
0040134C  .  E8 9D000000   call <JMP.&MessageBoxA>
00401351  .v EB 13         jmp 14.401366
00401353  >  6A 00         push 0
00401355  .  68 62344000   push 14.403462                  403462:"Key/CrackMe #2    "
0040135A  .  68 86344000   push 14.403486                  403486:" You Have Enter A Wrong Serial, Please Try Again "
0040135F  .  6A 00         push 0
00401361  .  E8 88000000   call <JMP.&MessageBoxA>
00401366  >v EB 15         jmp 14.40137D
```

비교하는 구문을 찾았습니다. 저 두개가 같아야 Good Job이 출력 될 것 같다.

eax와 esi의 데이터를 알아보기 위해 브레이크 포인트를 걸고 확인했다.




```
EAX   00003039
EBX   00000037   '7'
ECX   00000000
EDX   00403139   "2345"
EBP   0019FA74
ESP   0019FA74
ESI   000129A1
EDI   00000111   L'ď'
```

serial에 12345를 입력 했을때 레지스터를 보면..

EAX는 3039 즉 10진수로 12345가 나온다. 이렇게 EAX는 serial에 입력한 값이 들어간다는 것을 알 수 있었다.

ESI를 129A1이라는 값이 들어 있는데 이것을 10진수로 확인해보면, 76193이 나온다.

Name = CodeEngn, Serial = 76193 이렇게 key가 나왔다.

ESI에 저런 값이 왜 들어갔는지 보면,

```
004012F6  >  68 38304000      push 14.403038
004012FB  .  E8 30010000      call <JMP.&lstrlen>
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,FF
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            jne 14.401309

403038:"CodeEngn"
name의 길이를 잼. CodeEngn이니까 eax = 8
esi를 0으로 만듬
ecx = 8
eax = 1
00403038:"CodeEngn"
eax+403037:"eEngn"
입력한 값의 앞 4자리와 & 0xFF
위 구문을 진행하면 앞 1바이트만 남음
문자 맨 앞을 제곱함
제곱한 값을 esi와 더함

맨 앞자리를 1 sar함
맨 앞자리 제곱한 값과 맨 앞자리 1 sar한 값을 더함
위 구문 진행한 것에서 맨 앞자리를 뺌
```

이런 암호화 방식을 통해 CodeEngn 문자열 을 암호화 하여 ESI에 넣었다.

EAX에는 입력한 값이 그대로 들어간 이유는 모르겠지만 숫자를 입력했을 때, 숫자가 그대로 eax에 저장되고,

문자열을 입력했을때 문자열과 관련없는 수가 eax에 저장되는 것을 봐서

아마 문자열과 숫자를 걸러주기 위한 함수가 아닐까 싶다.

```
00401383  r$  55          push ebp                    sub_401383
00401384  .   8BEC        mov ebp,esp
00401386  .   FF75 08     push dword ptr ss:[ebp+8]   [ebp+8]:L"-core-win32k-full-flo
00401389  .   E8 A2000000 call <JMP.&lstrlen>
0040138E  .   53          push ebx
0040138F  .   33DB        xor ebx,ebx
00401391  .   8BC8        mov ecx,eax                 문자열 길이 = ecx
00401393  .   8B75 08     mov esi,dword ptr ss:[ebp+8] [ebp+8]:L"-core-win32k-full-flo
00401396  >   51          push ecx                    push 문자열 길이
00401397  .   33C0        xor eax,eax                 eax 0으로 만듦
00401399  .   AC          lodsb                       시리얼 문자열의 한자리를 가져옴
0040139A  .   83E8 30     sub eax,30                  serial[i] - 0x30
0040139D  .   49          dec ecx                     문자열 길이 - 1
0040139E  .v  74 05       je 14.4013A5                for문
004013A0  >   6BC0 0A     imul eax,eax,A              (serial[i] - 0x30) * 0xA
004013A3  .^  E2 FB       loop 14.4013A0              문자열 길이만큼 위에 구문을 반복
004013A5  >   03D8        add ebx,eax                 전에 했던 거랑 다음에 하는거랑 더함
004013A7  .   59          pop ecx                     문자열 길이만큼 하니까 pop해줘야함
004013A8  .^  E2 EC       loop 14.401396
004013AA  .   8BC3        mov eax,ebx
004013AC  .   5B          pop ebx
004013AD  .   C9          leave
004013AE  L.  C2 0400     ret 4
```

bruteforce없이 풀 수 있는 문제였다.