

codeengn-basic-L11 풀이

리버싱 문제풀이 / Wonlf / 2022. 4. 2. 22:32

Basic RCE L11

OEP를 찾으시오. Ex) 00401000 / Stolenbyte 를 찾으시오.
Ex) FF35CA204000E84D000000 정답인증은 OEP+ Stolenbyte
Ex) 00401000FF35CA204000E84D000000

— Author: abex

— File Password: codeengn



이번 문제는 9번 문제와 완벽히 동일하다. 먼저 9번 링크를 참고하겠다.

2022.03.27 - [리버싱 문제 풀이/CodeEngn.com] - codeengn-basic-L09 풀이

9번과 동일하지만 포인트로 짚고 넘어가야 할 것이 한 개 추가 되었다. 내려가면서 확인해보도록 하자.

문제는 OEP와 Stolenbyte를 원하고 있다.

9번 풀이대로 해보면...

The screenshot shows a debugger window with the following details:

- Assembly View:** Address 004071F0 to 00407217. Instructions include `pushad`, `mov esi, 11.407000`, `lea edi, dword ptr ds:[esi-6000]`, `push edi`, `or ebp, FFFFFFFF`, `jmp 11.407212`, `mov al, byte ptr ds:[esi]`, `inc esi`, `mov byte ptr ds:[edi], al`, `inc edi`, `add ebx, ebx`, `jne 11.407219`, `mov ebx, dword ptr ds:[esi]`, `sub esi, FFFFFFFC`, and `adc ebx, ebx`.
- Registers View:** EAX: 0019FFCC, ECX: 00228000, EDI: 004071F0, EBP: 0019FF80, ESP: 0019FF54, ESI: 004071F0, EDI: 004071F0. EIP: 004071F1.
- ESI:EntryPoint:** Address 004071F1. The label is highlighted in the assembly view.
- Right Pane:** Shows the 'ESI:EntryPoint' label and a list of memory addresses and values.

pushad이후 하드웨어 브레이크 포인트를 걸어주고,

popad에서 멈추고 보게되면...

0040736D	61	popad	
0040736E	6A 00	push 0	
00407370	68 00204000	push 11.402000	
00407375	68 12204000	push 11.402012	402000:"abex' 3rd crackme"
0040737A	8D4424 80	lea eax,dword ptr ss:[esp-80]	402012:"Click OK to check for the keyfile."
0040737E	6A 00	push 0	
00407380	39C4	cmp esp,eax	
00407382	75 FA	jne 11.40737E	
00407384	83EC 80	sub esp,FFFFFF80	
00407387	E9 809CFFFF	jmp 11.40100C	

popad이후 stolenbyte를 복구하는 구문과 OEP로 추정되는 것이 보인다.

그래서 Auth에 "40100C6A0068002040006812204000"

이대로 입력해보니 incorrect가 뜨길래 맞는데 뭐가 문제지 했는데

지금 저 40100C로 보이는 부분이 OEP로 보일 수도 있겠지만, stolenbyte바이트를 복구하지 않은 상태의 주소이다.

그러니까 40100C위에 stolenbyte가 복구 되었을 때의 EP가 OEP가 되는 것이다.

현재 확인한 stolenbyte는 "6A0068002040006812204000" 로 총 12바이트이다.

00401000	. 90	nop	sub_401000
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	

nop로 되어 있는 부분도 총 12바이트. 그러니 stolenbyte를 복구 해줬을 때를 보면

00401000	. 6A 00	push 0	sub_401000
00401002	. 68 00204000	push 11.402000	402000:"abex' 3rd crackme"
00401007	. 68 12204000	push 11.402012	402012:"Click OK to check for the keyfile."
0040100C	. 6A 00	push 0	
0040100E	. E8 8C000000	call <JMP.&MessageBoxA>	

딱 맞아 떨어지고 00401000이 EP가 되었다.

답은 "004010006A0068002040006812204000"

더 쉽게 보면 0040100C에 stolenbyte(12byte)를 뺐을 때 실제 OEP가 나오게 된다.

```
>>> hex(0x0040100C - 12)
'0x401000'
```

이번 문제는 stolenbyte가 있을 때 stolenbyte를 생각해서 OEP를 찾아야 되는 것을 알려주는 문제였다.