

CodeEngn Basic RCE L19

: 이 프로그램은 몇 밀리세컨드 후에 종료 되는가

아마도 timeout과 관련된 문제인 것 같다.

004AF382	. 57	PUSH EDI	
004AF383	. FFD5	CALL EBP	
004AF385	. 58	POP EAX	
004AF386	. 61	POPAD	
004AF387	. 8D4424 80	LEA EAX,DWORD PTR SS:[ESP-80]	
004AF38B	> 6A 00	PUSH 0	
004AF38D	. 39C4	CMP ESP,EAX	
004AF38F	. ^75 FA	JNZ SHORT 19.004AF38B	
004AF391	. 83EC 80	SUB ESP,-80	
004AF394	. -E9 D783F6FF	JMP 19.00417770	
004AF399	00	DB 00	
004AF39A	00	DB 00	
004AF39B	00	DB 00	
004AF39C	00	DB 00	
004AF39D	00	DB 00	
004AF39E	00	DB 00	
004AF39F	00	DB 00	

올리디버거로 열어주었더니 오랜만에 PUSHAD가 보여서 command 찾기로 POPAD를 찾아주었고, 점프 관련 명령어가 2개가 있어서 살펴보니 JNZ는 004AF38B로 가면서 루프를 돈다. 즉 OEP로 가는 점프명령어는 JMP 00417770이다. 해당 주소에 BP를 걸어주고 F9 -> F8을 누르면 실제 코드로 이동한다.

실제코드에서 딱히 단서가 될만한 것이 없어서 search for -> all intermodular calls 눌러서 우클릭 -> sort by -> destination 으로 함수 별 탐색을 하였다.

0040E95B	57	PUSH EDI	
0040E95C	E8 1DFFFFFF	CALL 19.0040C880	
0040E961	FF15 20D34700	CALL DWORD PTR DS:[47D320]	KERNEL32.IsDebuggerPresent
0040E967	90	NOP	
0040E968	90	NOP	
0040E969	~0F85 6F4F0200	JNZ 19.004338DE	
0040E96F	8B4424 0F	MOV BYTE PTR SS:[ESP+F],AL	
0040E973	BE 30044A00	MOV ESI,19.004A0430	

방향성을 못잡고 있어서 검색을 해 본 결과 IsDebuggerPresent 함수가 있으면 해당 부분을 NOP으로 채워야 한다고 한다. 위 함수가 존재를 해서 NOP으로 채워주었다.

함수를 내리다 보면 timeGetTime을 찾을 수 있는데, 이 함수는 현재 시간을 불러오는 함수이다. 시간 관련 함수이므로 문제의 답을 구하는 것과 관련이 있을 것이라 생각했다.

00444C39	C3	RETN	
00444C3A	53	PUSH EBX	
00444C3B	55	PUSH EBP	
00444C3C	56	PUSH ESI	
00444C3D	57	PUSH EDI	
00444C3E	8B3D 58D74700	MOV EDI,DWORD PTR DS:[47D758]	WINMM.timeGetTime
00444C44	FFD7	CALL EDI	
00444C46	803D D3E84800 0	CMP BYTE PTR DS:[48E8D3],0	
00444C4D	8BF0	MOV ESI,EAX	
00444C4F	~0F84 FF000000	JE 19.00444D54	
00444C55	8B5C24 14	MOV EBX,DWORD PTR SS:[ESP+14]	
00444C59	8B2D 58D14700	MOV EBP,DWORD PTR DS:[47D158]	KERNEL32.Sleep
00444C5F	FFD7	CALL EDI	
00444C61	3BC6	CMP EAX,ESI	
00444C63	~0F83 CF000000	JNB 19.00444D38	
00444C69	2BC6	SUB EAX,ESI	
00444C6B	43	DEC EAX	
00444C6C	~E9 C9000000	JMP 19.00444D3A	
00444C71	8B03	MOV EAX,DWORD PTR DS:[EBX]	
00444C73	6A 00	PUSH 0	
00444C75	68 FC064300	PUSH 19.004306FC	

timeGetTime 부터 Sleep함수까지 살펴보면,

```

CALL EDI // EDI 호출 == timeGetTime함수 호출
MOV ESI, EAX // ESI = EAX : ESI에 EAX값 저장
JE 00444D54 // 0이면 00444D54로 JMP
.
.
CALL EDI // 다시 timeGetTime 함수 호출
CMP EAX, ESI // EAX - ESI
JNB 00444D38 // **Jump Not Below : 작지 않을 경우 jmp
=> EAX가 ESI보다 더 크면 JMP
SUB EAX, ESI // EAX = EAX - ESI
DEC EAX // EAX = EAX - 1
JMP 00444D3A // 해당 주소로 JMP

```

위에서 timeGetTime 함수를 2번 호출하고 있다. 즉 코드를 살펴보면 처음으로 호출한 부분에서 그때의 시각을 ESI에 저장하고, 2번째 호출을 한 시각을 EAX에 저장을 해서 두 개를 비교를 하는데, 당연히 2번째 때 호출한 EAX가 ESI보다 클 수 밖에 없다. 그래서 JNB로 무조건 이동한다.

여기서 주목해야 하는 것은 JNB이다. $EAX > ESI$ 면 해당 주소로 점프하라는건데 보통 타임아웃은 지정 시간이 현재시간보다 초과될 경우 발생한다. 그럼 EAX가 초과한 경우 해당 주소로 이동하라는 것 같아서 EAX가 경과된 시간이라고 추측했다. 그럼 00444D38주소로 가면 EAX가 어떻게 작동하는지 더 볼 수 있을 것이다.

00444D38	2BC6	SUB EAX,ESI	
00444D3A	3B43 04	CMP EAX,DWORD PTR DS:[EBX+4]	
00444D3D	^0F83 2EFFFFFF	JNB 19.00444C71	
00444D43	6A 0A	PUSH 0A	
00444D45	FFD5	CALL EBP	
00444D47	803D D3E84800 0	CMP BYTE PTR DS:[48E8D3],0	
00444D4E	^0F85 0BFFFFFF	JNZ 19.00444C5F	
00444D54	5F	POP EDI	
00444D55	5E	POP ESI	
00444D56	5D	POP EBP	
00444D57	33C0	XOR EAX,EAX	
00444D59	5B	POP EBX	
00444D5A	C2 0400	RETN 4	
00444D5D	83EC 08	SUB ESP,8	
00444D60	56	PUSH ESI	
00444D61	57	PUSH EDI	
00444D62	8B7C24 24	MOV EDI,DWORD PTR SS:[ESP+24]	
00444D6C	33F6	XOR ESI,ESI	

```

SUB EAX, ESI // EAX = EAX - ESI
CMP EAX, DWORD PTR DS : [EBX+4] // EAX와 EBX+4주소의 데이터영역의 4바이트 값을 비교

```

정답으로 가는 KEY이다. 왜냐면 $EAX - ESI$ 는 2번째 호출 시각 - 1번째 호출 시각이기 때문이다. 아마 경과 시간이라고 생각된다. 근데 이 경과 시간을 비교하고 있으니 프로그램에서 지정해둔 타임아웃을 시간을 EBX+4주소의 4바이트가 가지고 있을 것이라고 추측된다.

00444D38	2BC6	SUB EAX,ESI	
00444D39	3B43 04	CMP EAX,DWORD PTR DS:[EBX+4]	
00444D3D	^0F83 2EFFFFFF	JNB 19.00444C71	
00444D43	6A 0A	PUSH 0A	
00444D45	FFD5	CALL EBP	
00444D47	803D D3E84800 0	CMP BYTE PTR DS:[48E8D3],0	
00444D4E	^0F85 0BFFFFFF	JNZ 19.00444C5F	
00444D54	5F	POP EDI	
00444D55	5E	POP ESI	
00444D56	5D	POP EBP	
00444D57	33C0	XOR EAX,EAX	
00444D59	5B	POP EBX	
00444D5A	C2 0400	RETN 4	
00444D5D	83EC 08	SUB ESP,8	
00444D60	56	PUSH ESI	
00444D61	57	PUSH EDI	
00444D62	8B7C24 24	MOV EDI,DWORD PTR SS:[ESP+24]	

Stack DS:[008BF87C]=00002B70
EAX=00000000

Address	Hex dump	ASCII
008BF87C	70 2B 00 00 64 E0 45 00 00 00 00 00 C8 E6 6F 03	p+..d?.....o
008BF88C	08 E5 6F 03 00 00 01 00 14 44 00 00 60 F6 49 00	o...0.1D..'?
008BF89C	58 F9 8B 00 54 F0 8B 00 60 F6 49 00 07 22 41 00	x?-T?..'?

BP를 걸고 실행시켜주면 해당 주소 DS 부분에 2B70이 쓰여져 있는 것을 볼 수 있다. (리틀 엔디안이라서 HEX DUMP에서 오른쪽부터 읽어주어야 한다.) 2B70을 10진수로 바꾸면 11120 이고 이 프로그램은 11120 밀리세컨드 후에 종료된다는 것을 알 수 있다.

ms = 10의 -3승 s이므로 11.12초 이후에 종료된다.