

codeengn-basic-L17 풀이

리버싱 문제풀이 / Wonlf / 2022. 4. 21. 15:59

Basic RCE L17

Key 값이 BEDA-2F56-BC4F4368-8A71-870B 일때 Name은 무엇인가

힌트 : Name은 한자리인데.. 알파벳일수도 있고 숫자일수도 있고..

정답인증은 Name의 MD5 해쉬값(대문자)

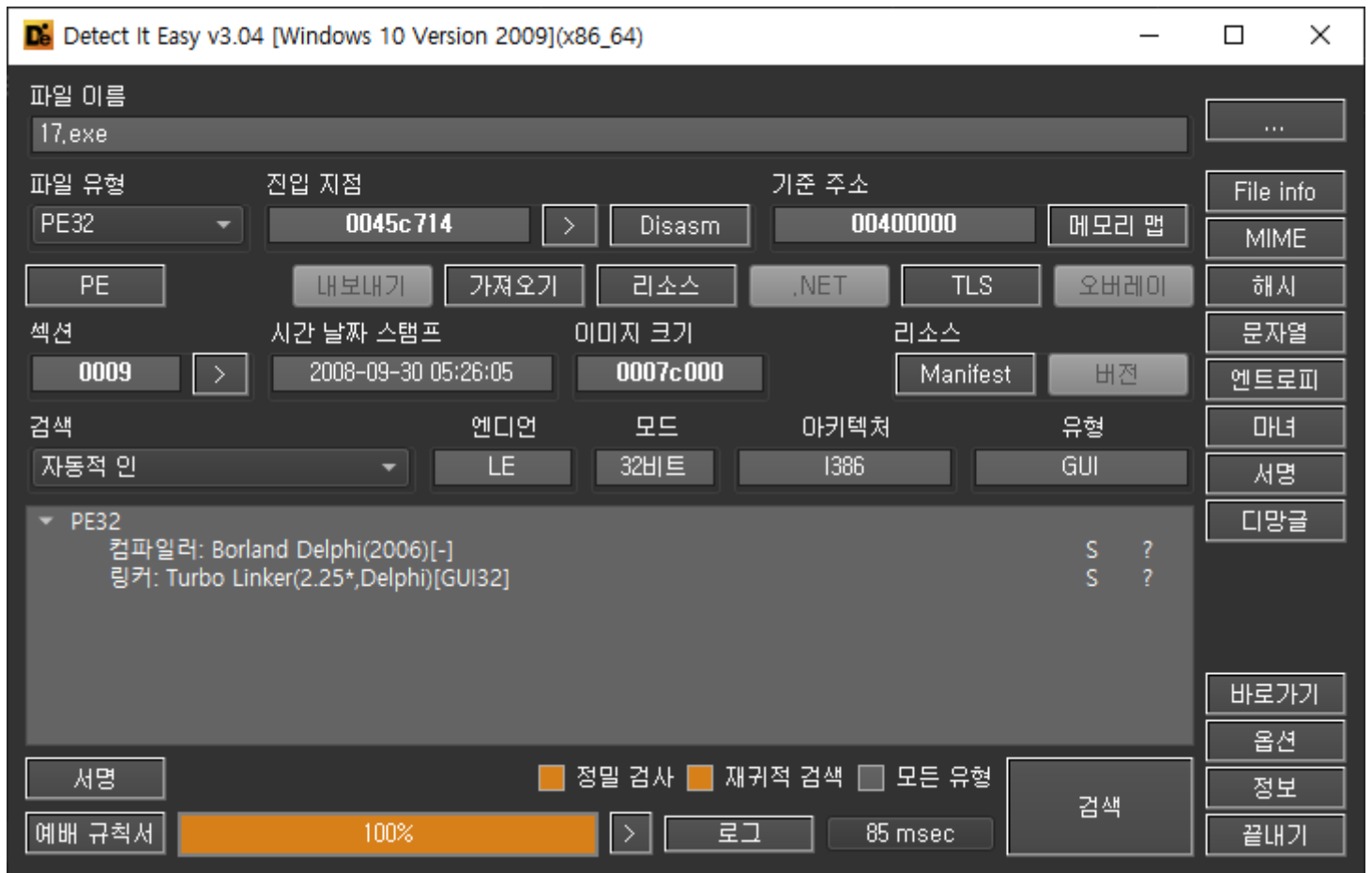
— Author: WarRock

— File Password: codeengn



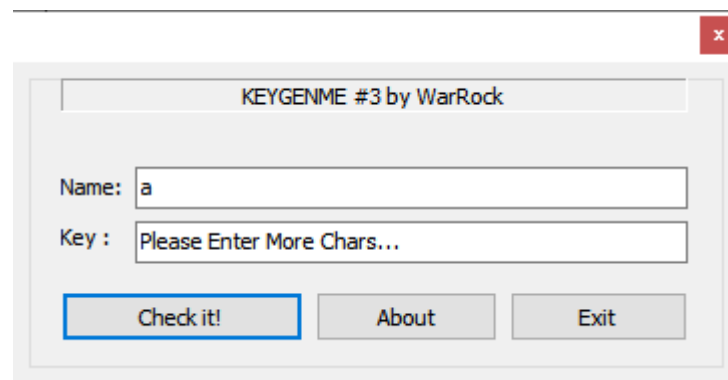
문제는 Key가 BEDA-2F56-BC4F4368-8A71-870B 일때 한글자인 name의 MD5 해쉬값을 원하고 있다.

Die로 열어보면,



특이사항은 보이지 않는다.

파일을 실행해본다.



Name은 한글자라는데 한글자를 입력하면 Please Enter More Chars 더 많은 문자를 입력해달라고 한다. 이렇게 되면 Name을 구할 수 없으니 디버거로 열어서 문제를 파악해야겠다.

문자열 찾기를 사용하여 "Please Enter More Chars..." 문자열을 사용하는 곳을 찾았다.

0045BB1F	83E8 04	sub eax,4	
0045BB22	8B00	mov eax,dword ptr ds:[eax]	
0045BB24	83F8 03	cmp eax,3	
0045BB27	7D 15	jge 17.45BB3E	
0045BB29	BA 18BC4500	mov edx,17.45BC18	45BC18: "Please Enter More Chars..."
0045BB2E	8B83 74030000	mov eax,dword ptr ds:[ebx+374]	ebx+374: "....."
0045BB34	E8 6BE5FDFF	call 17.43A0A4	
0045BB39	E9 91000000	jmp 17.45BBCE	
0045BB3E	8D55 F4	lea edx,dword ptr ss:[ebp-C]	
0045BB41	8B83 68030000	mov eax,dword ptr ds:[ebx+368]	ebx+368: "....."
0045BB47	E8 28E5FDFF	call 17.43A074	
0045BB4C	8B45 F4	mov eax,dword ptr ss:[ebp-C]	
0045BB4F	8945 F8	mov dword ptr ss:[ebp-8],eax	
0045BB52	8B45 F8	mov eax,dword ptr ss:[ebp-8]	
0045BB55	85C0	test eax,eax	
0045BB57	74 05	jle 17.45BB5E	
0045BB59	83E8 04	sub eax,4	
0045BB5C	8B00	mov eax,dword ptr ds:[eax]	
0045BB5E	83F8 1E	cmp eax,1E	
0045BB61	7E 12	jle 17.45BB75	
0045BB63	BA 3CBC4500	mov edx,17.45BC3C	45BC3C: "Please Enter Not More Than 30 Chars..."
0045BB68	8B83 74030000	mov eax,dword ptr ds:[ebx+374]	ebx+374: "....."
0045BB6E	E8 31E5FDFF	call 17.43A0A4	
0045BB73	EB 5A	jmp 17.45BBCE	
0045BB75	8D55 F0	lea edx,dword ptr ss:[ebp-10]	
0045BB78	8B83 74030000	mov eax,dword ptr ds:[ebx+374]	ebx+374: "....."
0045BB7E	E8 F1E4FDFF	call 17.43A074	

더 아래를 보니 문자열이 30길이를 넘으면 안되는 구문도 보인다.

cmp와 점프분기가 있으니 높은 확률로 eax에는 길이가 들어갈 것 같고 실제로도 확인해보면,

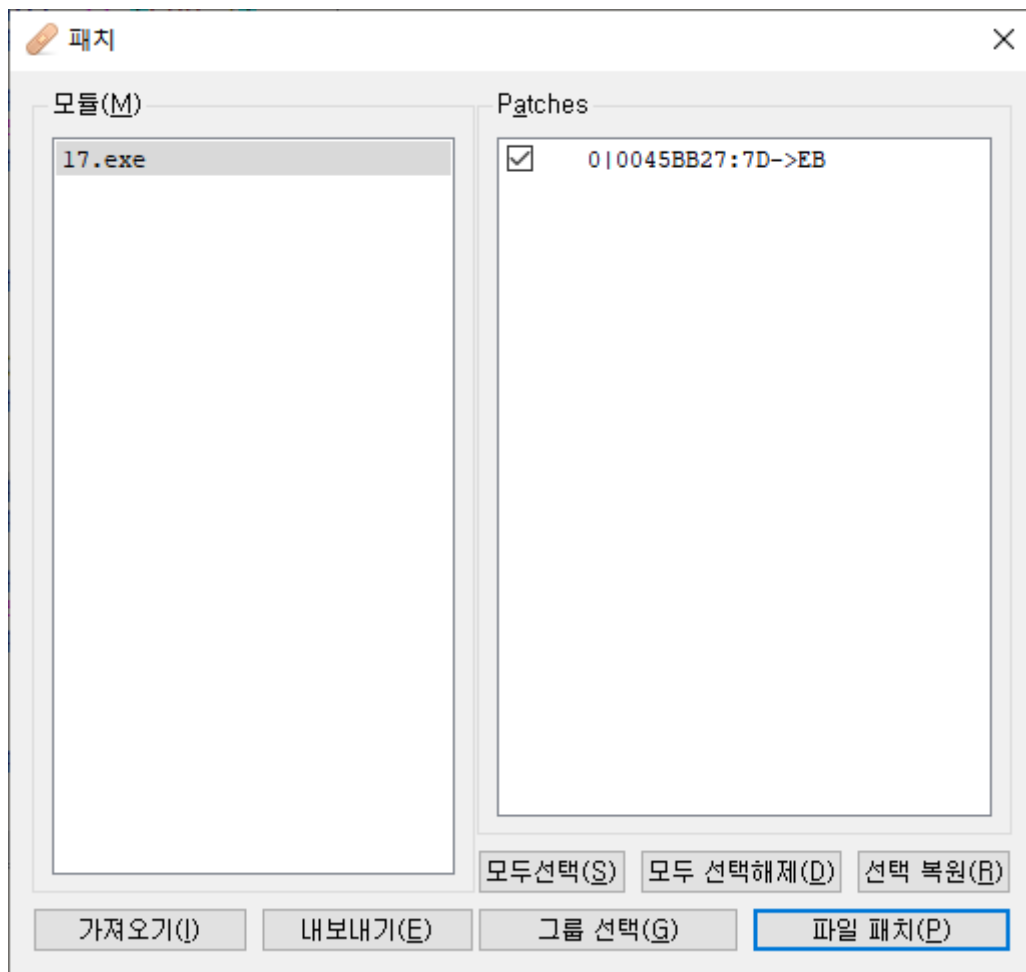
```
eax=1
```

길이

길이가 맞다.

우리는 한글자만 입력하면 되니 jge쪽을 보게되면 jge는 $eax \geq 0x3$ 일때 점프를 뛴다.

점프를 뛰어야 그 다음 검증인 30길이를 넘는지 안넘는지 구문까지 갈 수 있기에 jge를 jmp로 바꾸어 주었다.

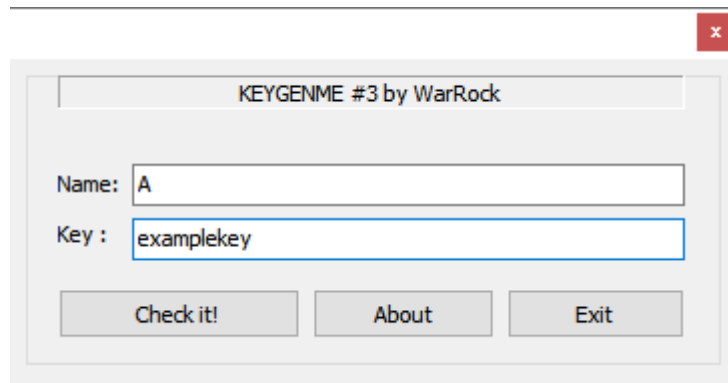


jmp로 바꾼 파일을 패치해서 저장 하고 저장한 파일을 디버깅하기 시작했다.

그리고 더 내려가면 30길이를 넘는지 안넘는지 구문은 한글자만 입력 했으니 당연히 통과 할것이고 더 아래로 가게 되면,

0045BB59	83E8 04	sub eax,4	
0045BB5C	8B00	mov eax,dword ptr ds:[eax]	
0045BB5E	83F8 1E	cmp eax,1E	
0045BB61	7E 12	jle 17_patched.45BB75	
0045BB63	BA 3CBC4500	mov edx,17_patched.45BC3C	edx:"U영갓齣兆E", 45BC3C:"Please Enter Not More Than 30 Chars..."
0045BB68	8B83 74030000	mov eax,dword ptr ds:[ebx+374]	
0045BB6E	E8 31E5FDFF	call 17_patched.43A0A4	
0045BB73	EB 5A	jmp 17_patched.45BB8F	
0045BB75	8D55 F0	lea edx,dword ptr ss:[ebp-10]	
0045BB78	8B83 74030000	mov eax,dword ptr ds:[ebx+374]	
0045BB7E	E8 F1E4FDFF	call 17_patched.43A074	
0045BB83	8B45 F0	mov eax,dword ptr ss:[ebp-10]	
0045BB86	50	push eax	
0045BB87	8D55 E8	lea edx,dword ptr ss:[ebp-18]	
0045BB8A	8B83 68030000	mov eax,dword ptr ds:[ebx+368]	
0045BB90	E8 DFE4FDFF	call 17_patched.43A074	
0045BB95	8B45 E8	mov eax,dword ptr ss:[ebp-18]	
0045BB98	8D55 EC	lea edx,dword ptr ss:[ebp-14]	
0045BB9B	E8 B0FCFFFF	call 17_patched.45BB80	
0045BBA0	8B55 EC	mov edx,dword ptr ss:[ebp-14]	
0045BBA3	58	pop eax	
0045BBA4	E8 9390FAFF	call 17_patched.404C3C	
0045BBA9	75 1A	jne 17_patched.45BBC5	
0045BBAB	6A 40	push 40	
0045BBAD	B9 64BC4500	mov ecx,17_patched.45BC64	ecx:"U영갓齣兆E", 45BC64:"Good Boy!!!"
0045BBB2	BA 70BC4500	mov edx,17_patched.45BC70	edx:"U영갓齣兆E", 45BC70:"Well done!"
0045BBB7	A1 C0E94500	mov eax,dword ptr ds:[45E9C0]	

0045BB75부터 시리얼이 맞는지 틀린지를 판단하는 구문의 시작인 것 같아 브레이크 포인트를 걸고



특정 값을 넣은뒤 한줄씩 실행시켜보았다.

→ 0045BB75	8D55 F0	lea edx,dword ptr ss:[ebp-10]	[ebp-10]: "examplekey"
• 0045BB78	8B83 74030000	mov eax,dword ptr ds:[ebx+374]	[ebx+374]: "뽕B"
• 0045BB7E	E8 F1E4FDFF	call 17_patched.43A074	
• 0045BB83	8B45 F0	mov eax,dword ptr ss:[ebp-10]	[ebp-10]: "examplekey"
• 0045BB86	50	push eax	
• 0045BB87	8D55 E8	lea edx,dword ptr ss:[ebp-18]	
• 0045BB8A	8B83 68030000	mov eax,dword ptr ds:[ebx+368]	[ebx+368]: "뽕B"
• 0045BB90	E8 DFE4FDFF	call 17_patched.43A074	
• 0045BB95	8B45 E8	mov eax,dword ptr ss:[ebp-18]	
• 0045BB98	8D55 EC	lea edx,dword ptr ss:[ebp-14]	[ebp-14]: "FFE3-2C73-0502A34C-8A48-E1CB"
• 0045BB9B	E8 B0FCFFFF	call 17_patched.45B850	
→ 0045BBA0	8B55 EC	mov edx,dword ptr ss:[ebp-14]	[ebp-14]: "FFE3-2C73-0502A34C-8A48-E1CB"
• 0045BBA3	58	pop eax	

ebp-10에는 Key로 입력한 "examplekey" 가 들어 있었고 더 실행하다보면, 45B850 함수를 호출 했을 때, Name에 대한 Key가 스택에 저장 되었다.

A를 넣어서 저런 Key가 나왔으니 B도 넣어서 확인해본다.

→ 0045BB75	8D55 F0	lea edx,dword ptr ss:[ebp-10]	[ebp-10]: "examplekey"
• 0045BB78	8B83 74030000	mov eax,dword ptr ds:[ebx+374]	[ebx+374]: "뽕B"
• 0045BB7E	E8 F1E4FDFF	call 17_patched.43A074	
• 0045BB83	8B45 F0	mov eax,dword ptr ss:[ebp-10]	[ebp-10]: "examplekey"
• 0045BB86	50	push eax	
• 0045BB87	8D55 E8	lea edx,dword ptr ss:[ebp-18]	[ebp-18]: &"liyo"
• 0045BB8A	8B83 68030000	mov eax,dword ptr ds:[ebx+368]	[ebx+368]: "뽕B"
• 0045BB90	E8 DFE4FDFF	call 17_patched.43A074	
• 0045BB95	8B45 E8	mov eax,dword ptr ss:[ebp-18]	[ebp-18]: &"liyo"
• 0045BB98	8D55 EC	lea edx,dword ptr ss:[ebp-14]	[ebp-14]: "A27E-2D07-BA91EB06-8A50-A5D4"
• 0045BB9B	E8 B0FCFFFF	call 17_patched.45B850	
→ 0045BBA0	8B55 EC	mov edx,dword ptr ss:[ebp-14]	[ebp-14]: "A27E-2D07-BA91EB06-8A50-A5D4"
• 0045BBA3	58	pop eax	

이런식으로 Name에 따라서 Key가 바뀌는 것을 알 수 있는데, 여기서 사실 답을 알아버렸다.

어떠한 Name이 입력 됐을 때, key가 문제에 있던 BEDA-2F56-BC4F4368-8A71-870B 이것과 같으면 되기에 알파벳을 입력해보다 "F" 를 입력했을때 성공구문으로 가는 것을 알았다

하지만 그렇게 풀면 내 자신이 한심하기 때문에 Key를 만드는 45B850 함수를 뒤져보기로 한다.

함수를 뒤져보다 보니 특정 연산을 하는 코드를 발견했다.

0045B87F	> 33D2	xor edx,edx	edx 0으로 초기화
0045B881	33F6	xor esi,esi	esi 0으로 초기화
0045B883	33C0	xor eax,eax	eax 0으로 초기화
0045B885	8945 F0	mov dword ptr ss:[ebp-10],eax	
0045B888	8B45 FC	mov eax,dword ptr ss:[ebp-4]	[ebp-4]:&"liyo"
0045B88B	85C0	test eax,eax	
0045B88D	74 05	je 17_patched.45B894	
0045B88F	83E8 04	sub eax,4	
0045B892	8B00	mov eax,dword ptr ds:[eax]	
0045B894	> 85C0	test eax,eax	
0045B896	7E 2C	jle 17_patched.45B8C4	
0045B898	B9 01000000	mov ecx,1	
0045B89D	> 8B5D FC	mov ebx,dword ptr ss:[ebp-4]	[ebp-4]:&"liyo"
0045B8A0	0FB6740B FF	movzx esi,byte ptr ds:[ebx+ecx-1]	ebx+ecx*1-1:&"liyo"
0045B8A5	03F2	add esi,edx	
0045B8A7	69F6 72070000	imul esi,esi,772	
0045B8AD	8BD6	mov edx,esi	
0045B8AF	0FAFD6	imul edx,esi	
0045B8B2	03F2	add esi,edx	
0045B8B4	0BF6	or esi,esi	
0045B8B6	69F6 74040000	imul esi,esi,474	
0045B8BC	03F6	add esi,esi	
0045B8BE	8BD6	mov edx,esi	
0045B8C0	41	inc ecx	
0045B8C1	48	dec eax	
0045B8C2	75 D9	jne 17_patched.45B89D	
0045B8C4	> 8B45 FC	mov eax,dword ptr ss:[ebp-4]	[ebp-4]:&"liyo"
0045B8C7	85C0	test eax,eax	
0045B8C9	74 05	je 17_patched.45B8D0	
0045B8CB	83E8 04	sub eax,4	
0045B8CE	8B00	mov eax,dword ptr ds:[eax]	
0045B8D0	> 83F8 01	cmp eax,1	
0045B8D3	7C 24	j1 17_patched.45B8F9	
0045B8D5	> 8B55 FC	mov edx,dword ptr ss:[ebp-4]	[ebp-4]:&"liyo"

edx=A27E1920
 esi=A27E1920

특정 연산을 하고 edx에 저장되는 값은 Name에 B를 입력했을 때의 Key의 앞 4자리인 A27E와 유사한 A27E1920가 들어있다.

더 아래로 진행해보면,

0045B8CE	8B00	mov eax,dword ptr ds:[eax]	
0045B8D0	> 83F8 01	cmp eax,1	
0045B8D3	7C 24	j1 17_patched.45B8F9	
0045B8D5	> 8B55 FC	mov edx,dword ptr ss:[ebp-4]	[ebp-4]:&"liyo"
0045B8D8	0FB65402 FF	movzx edx,byte ptr ds:[edx+eax-1]	
0045B8DD	83C2 11	add edx,11	
0045B8E0	83EA 05	sub edx,5	
0045B8E3	69D2 92000000	imul edx,edx,92	
0045B8E9	03D2	add edx,edx	
0045B8EB	69D2 19080000	imul edx,edx,819	
0045B8F1	0155 F0	add dword ptr ss:[ebp-10],edx	
0045B8F4	48	dec eax	
0045B8F5	85C0	test eax,eax	
0045B8F7	75 DC	jne 17_patched.45B8D5	
0045B8F9	> 8D45 E8	lea eax,dword ptr ss:[ebp-18]	
0045B8FC	B9 C0BA4500	mov ecx,<17_patched.sub_45BAC0>	
0045B901	8B55 FC	mov edx,dword ptr ss:[ebp-4]	[ebp-4]:&"liyo"
0045B904	E8 2392FAFF	call <17_patched.sub_404B2C>	
0045B909	8B45 E8	mov eax,dword ptr ss:[ebp-18]	

dword ptr ss:[ebp-10]=[0019F3B4]=2D07038
 edx=2D07038

또 다른 특정 연산 구문이 있고 이 연산 구문을 지나고 나면 Key의 두번째인 2D07과 유사한 2D07038이 EDX에 들어있다.

이로써 추측한 바로는, 아마 더 아래로 내려 갔을때, Key의 다른 부분을 정하는 연산 코드가 있을 것이고, 나중에 그것을 조합하고 입력한 Key와 비교 할것 같다.

그리고 계속 Name에 "B"를 입력 했을때 첫번째 4자리와 두번째 4자리의 Key가 변경되지 않았으니, 또 다른 경우의 수를 찾을 필요는 없을 것 같다.

그렇기 때문에 처음 보았던 연산 코드로 Key의 첫번째 4자리만 브루트포스 해준다면 key를 구할 수 있을 것이다.

0045B898	B9 01000000	mov ecx,1	[ebp-4]:&L"liyo"
0045B89D	8B5D FC	mov ebx,dword ptr ss:[ebp-4]	ebx+ecx*1-1:&L"liyo"
0045B8A0	0FB6740B FF	movzx esi,byte ptr ds:[ebx+ecx-1]	edx는 00이니까 입력받은 esi = Name
0045B8A5	03F2	add esi,edx	esi = Name * 0x772
0045B8A7	69F6 72070000	imul esi,esi,772	edx = Name * 0x772
0045B8AD	8BD6	mov edx,esi	edx = (Name * 0x772) * (Name * 0x772)
0045B8AF	0FAFD6	imul edx,esi	esi = ((Name * 0x772) * (Name * 0x772)) + (Name * 0x772)
0045B8B2	03F2	add esi,edx	같은 값을 or하면 똑같은 값이 나오니 이걸 무시해도 됨
0045B8B4	0BF6	or esi,esi	esi = (((Name * 0x772) * (Name * 0x772)) + (Name * 0x772)) * 0x474
0045B8B6	69F6 74040000	imul esi,esi,474	esi = esi + esi
0045B8BC	03F6	add esi,esi	edx = esi
0045B8BE	8BD6	mov edx,esi	
0045B8C0	41	inc ecx	
0045B8C1	48	dec eax	

정리한 연산 코드이다. 특정 Name을 입력 했을때, EDX에는 문제의 Key였던 BEDA???? 이런 수가 들어 있을 것이다.

이것을 바탕으로 코드를 짜보면,

```
#include <stdio.h>

int main(void) {
    for (int i = 33; i < 134; ++i) {
        if (((((((i * 0x772) * (i * 0x772)) + (i * 0x772)) * 0x474) + (((i * 0x772) * (i * 0x772)) + (i * 0x772)) * 0x474)) & 0xffff0000) >> 16) == 0xBEDA){
            printf( format: "%c", i);
        }
    }
}
```

main

test

C:\Users\ao2\Desktop\project\test\cmake-build-debug\test.exe

F

대문자 "F" 가 나오는 것을 알 수 있다.

문제의 정답은 Name의 MD5였기 때문에 "F"의 MD5를 구해서 입력해주면,

여기 MD5 해시하고자하는 텍스트를 붙여 넣습니다

F

MD5 해시를 생성!

당신의 MD5 메시지 여기에서 소화 복사합니다.

800618943025315F869E4E1F09471012

통과하게 된다.

이번 문제는 좀 헛갈렸다.