

codeengn Basic RCE 12.

## Basic RCE L12

Key를 구한 후 입력하게 되면 성공메시지를 볼 수 있다

이때 성공메시지 대신 Key 값이 MessageBox에 출력 되도록 하려면 파일을 HexEdit로 오픈 한 다음 0x???? ~ 0x???? 영역에 Key 값을 overwrite 하면 된다.

문제 : Key값과 + 주소영역을 찾으시오

Ex) 77777777???????

00401000	6A 00	PUSH 0	pModule = NULL
00401002	E8 AB010000	CALL <JMP.&KERNEL32.GetModuleHandleA>	GetModuleHandleA
00401007	A3 64364000	MOV DWORD PTR DS:[403664],EAX	
0040100C	6A 00	PUSH 0	lParam = NULL
0040100E	68 29104000	PUSH 12.00401029	DlgProc = 12.00401029
00401013	6A 00	PUSH 0	hOwner = NULL
00401015	6A 65	PUSH 65	pTemplate = 65
00401017	FF35 64364000	PUSH DWORD PTR DS:[403664]	hInst = 00400000
0040101D	E8 60010000	CALL <JMP.&USER32.ShowDialogBoxParamA>	DialogBoxParamA
00401022	6A 00	PUSH 0	ExitCode = 0
00401024	E8 89010000	CALL <JMP.&KERNEL32.ExitProcess>	ExitProcess
00401029	55	PUSH _EBP	

ep가 401000이고 루틴이 상당히 단순해 보인다. DialogBoxparamA라는 함수에서 휴지통 아이콘이 나오는 프로그램을 실행하고, 키를 입력할 수 있게 된다. 바로 밑에 그 코드가 있다.

0040107B	EB EB	JMP SHORT 12.00401068	
0040107D	3D BF96287A	CMP EAX,7A2896BF	
00401082	75 14	JNZ SHORT 12.00401098	
00401084	6A 40	PUSH 40	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
00401086	68 30354000	PUSH 12.00403530	Title = "In the Bin"
0040108B	68 3B354000	PUSH 12.0040353B	Text = "Congratulation, you found the right key"
00401090	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hOwner
00401093	E8 02010000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
00401098	EB 6C	JMP SHORT 12.00401105	
0040109A	EB 61	JMP SHORT 12.004010FD	
0040109C	66:83F8 02	CMP AX,2	
004010A0	75 10	JNZ SHORT 12.004010B2	
004010A2	6A 00	PUSH 0	lParam = 0
004010A4	6A 00	PUSH 0	wParam = 0
004010A6	6A 10	PUSH 10	Message = WM_CLOSE
004010A8	FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
004010AB	E8 F0000000	CALL <JMP.&USER32.SendMessageA>	SendMessageA
004010B0	EB 4B	JMP SHORT 12.004010FD	

JNZ를 기점으로 위는 성공, 아래는 실패로 보인다. JNZ 앞에 CMP에서는 EAX를 무언가와 비교하고 있다. EAX는 언제 만들어지는지 살펴보자.

00401063	BE 00304000	MOV ESI,12.00403000	ASCII "0q1qb4EhM/4jISMjlzQf6kp6QwLrG+GEIY"
00401068	833E 00	CMP DWORD PTR DS:[ESI],0	
0040106B	75 04	JNZ SHORT 12.00401071	
0040106D	EB 0E	JMP SHORT 12.0040107D	
0040106F	EB 0C	JMP SHORT 12.0040107D	
00401071	8B1E	MOV EBX,DWORD PTR DS:[ESI]	
00401073	E8 97000000	CALL 12.0040110F	
00401078	83C6 04	ADD ESI,4	
0040107B	EB EB	JMP SHORT 12.00401068	
0040107D	3D BF96287A	CMP EAX,7A2896BF	
00401082	75 14	JNZ SHORT 12.00401098	

반복문이 존재한다. 우선 ESI에 매우 긴 문자를 넣고 이 ESI를 4씩 증가시켜가며 문자열이 끝날 때까지(0이 될 때 까지) 비교한다. 0이 되면 40107D로 점프하게 되고 아까 보았던 CMP를 수행한다. 이 과정에서 40110F에서는 하나의 함수가 계속 반복된다. 함수 안을 살

펴보자.

0040110F	51	PUSH ECX
00401110	52	PUSH EDX
00401111	8BD3	MOV EDX,EBX
00401113	8BC8	MOV ECX,EAX
00401115	40	INC EAX
00401116	F7D0	NOT EAX
00401118	43	INC EBX
00401119	F7D3	NOT EBX
0040111B	40	INC EAX
0040111C	43	INC EBX
0040111D	23C2	AND EAX,EDX
0040111F	23D9	AND EBX,ECX
00401121	03C3	ADD EAX,EBX
00401123	5A	POP EDX
00401124	59	POP ECX
00401125	C3	RET

아무 쓰잘데기 없어보이는 괴상한 함수이다. 무언가의 계산을 통해 암호화하는 것처럼 보이기도 한다. 루틴에 들어가기 전에 이것저것 대입해서 분석할 결과, eax는 내가 입력한 정수 (16진수로 바꿔서)이고, ebx에는 괴상하게 긴 문자열의 처음 4바이트가 저장된다.("0qiq")

실제 함수를 작동시켜본 결과, 내가 입력한 처음의 정수 값은 암호화 과정에서 하나의 솔트로 작용하는 것 같고, 함수의 알고리즘은 마치 하나의 암호를 만드는 듯하다. 어떡하징.. 1부터 하나하나 대입하면서 차분법을 사용해보자.

1 입력했을 때 eax 1 (목표는 7a2896bf)

2 입력했을 때 eax 2

10 입력했을 때 eax A

64 입력했을 때 eax 40

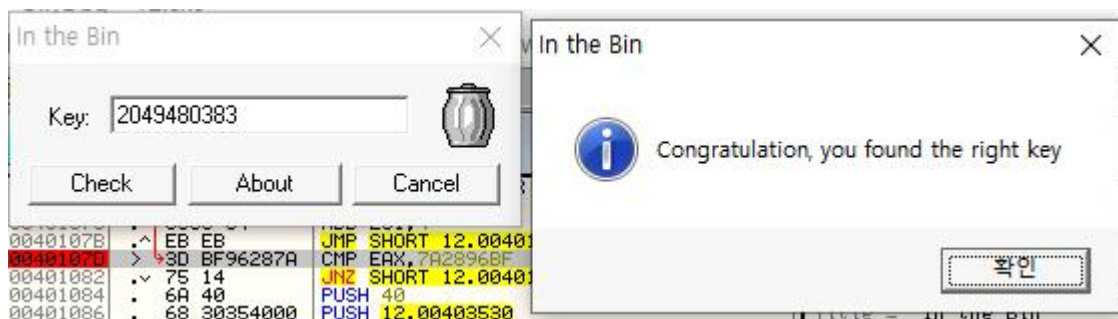
??? 엄청 복잡해보였는데, 막상 그냥 십진수를 16진수로 바꿔줬을 뿐이다.

그렇다면 key는 2049480383

7A28 96BF	
HEX	7A28 96BF
DEC	2,049,480,383

정확히 일치함

Registers (FPU)	
EAX	7A2896BF
ECX	011E758D
EDX	0019F808
EBX	00000000



Key는 구했고, 이제 성공 메시지 대신에 key값이 메시지박스에 출력하도록 파일을 수정해 보자.

```
0040353B 43 6F 6E 67 72 61 74 75 Congratu
00403543 6C 61 74 69 6F 6E 2C 20 lation,
0040354B 79 6F 75 20 66 6F 75 6E you foun
00403553 64 20 74 68 65 20 72 69 d the ri
0040355B 67 68 74 20 68 65 79 00 ght key.
```

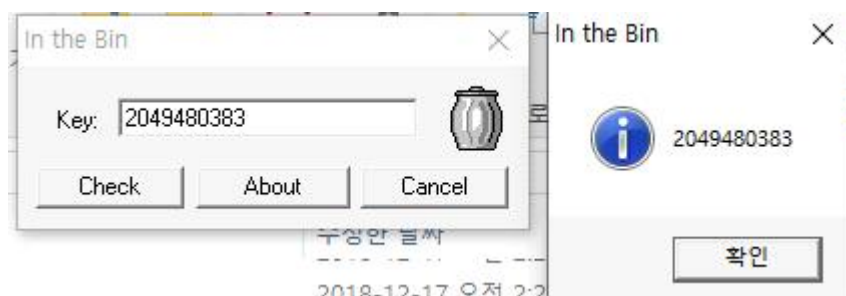
0x40353B 부분부터 Congratu... 로 시작하는 문자열이 있는 것을 확인할 수 있다.

```
00000D30 49 6E 20 74 68 65 20 42 69 6E 00 43 6F 6E 67 72 In the Bin. Congr
00000D40 61 74 75 6C 61 74 69 6F 6E 2C 20 79 6F 75 20 66 atulation, you f
00000D50 6F 75 6E 64 20 74 68 65 20 72 69 67 68 74 20 6B ound the right k
00000D60 65 79 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ey.....
00000D70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000D80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

hxd로 열었을 때도, 0x0D3B부터 문자열이 존재하는 것을 확인할 수 있다.

```
00000D30 49 6E 20 74 68 65 20 42 69 6E 00 32 30 34 39 34 In the Bin.20494
00000D40 38 30 33 38 33 00 69 6F 6E 2C 20 79 6F 75 20 66 80383. on, you f
00000D50 6F 75 6E 64 20 74 68 65 20 72 69 67 68 74 20 6B ound the right k
00000D60 65 79 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Key 값의 길이가 더 짧기 때문에 마지막에 Null값을 추가해서 파일을 수정하고 실행해보자.



성공! 주소값은 0x0D3B ~ 0x0D45