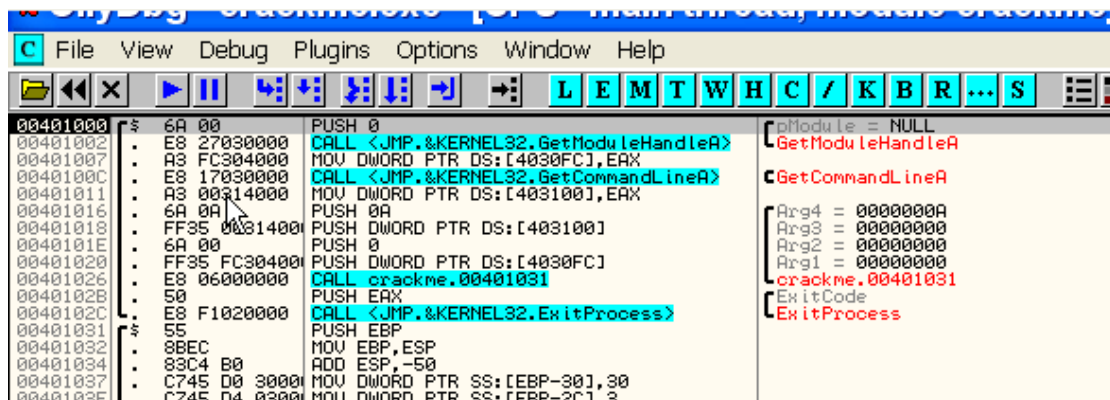
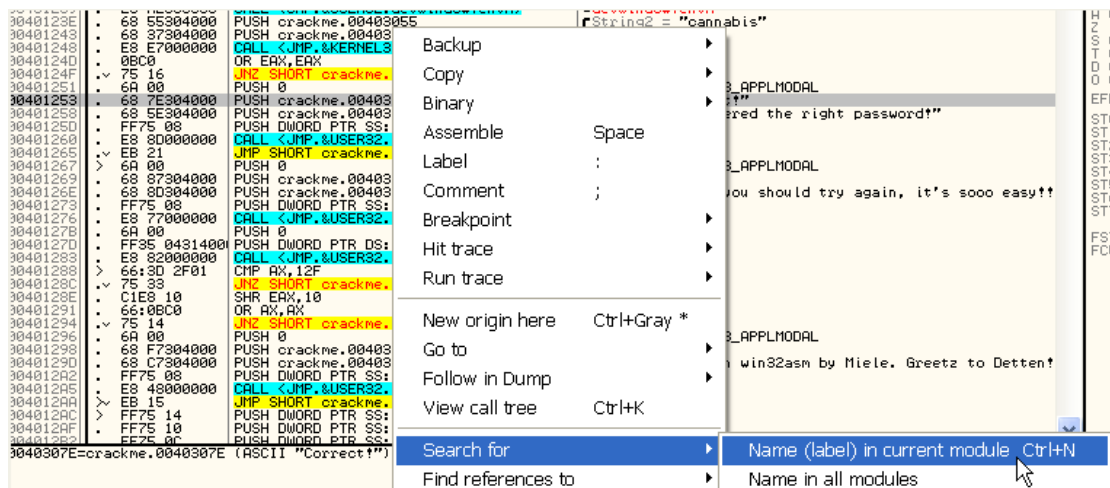


第十四章-硬编码序列号追踪-Part2

首先我来解答一下上一章留下的那个 CrackMe。



用 OD 加载名为 mielecraackme1 的 CrackMe。断在了入口点处,我们单击鼠标右键选择-Search for Name(label) in current module 看看该 CrackMe 使用了哪些 API 函数。



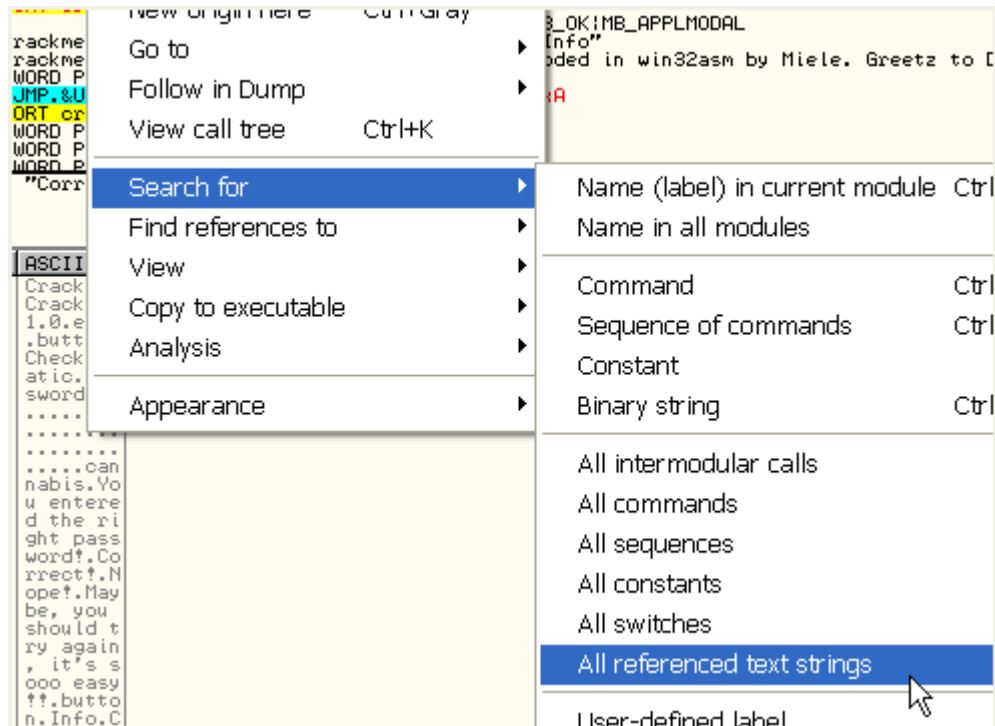
下面是找到的 API 函数列表:

Address	Section	Type	Name	Comment
00402048	.rdata	Import	USER32.CreateWindowExA	
00402028	.rdata	Import	USER32.DefWindowProcA	
00402040	.rdata	Import	USER32.DispatchMessageA	
00402008	.rdata	Import	KERNEL32.ExitProcess	
0040203C	.rdata	Import	USER32.GetMessageA	
00402004	.rdata	Import	KERNEL32.GetModuleHandleA	
00402024	.rdata	Import	USER32.GetWindowTextA	
00402020	.rdata	Import	USER32.LoadCursorA	
00402014	.rdata	Import	USER32.LoadIconA	
00402000	.rdata	Import	KERNEL32.lstrcmpA	
00402018	.rdata	Import	USER32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
0040201C	.rdata	Import	USER32.PostQuitMessage	
00402044	.rdata	Import	USER32.RegisterClassExA	
0040204C	.rdata	Import	USER32.SetFocus	
0040202C	.rdata	Import	USER32.SetWindowTextA	
00402030	.rdata	Import	USER32.ShowWindow	
00402034	.rdata	Import	USER32.TranslateMessage	
00402038	.rdata	Import	USER32.UpdateWindow	

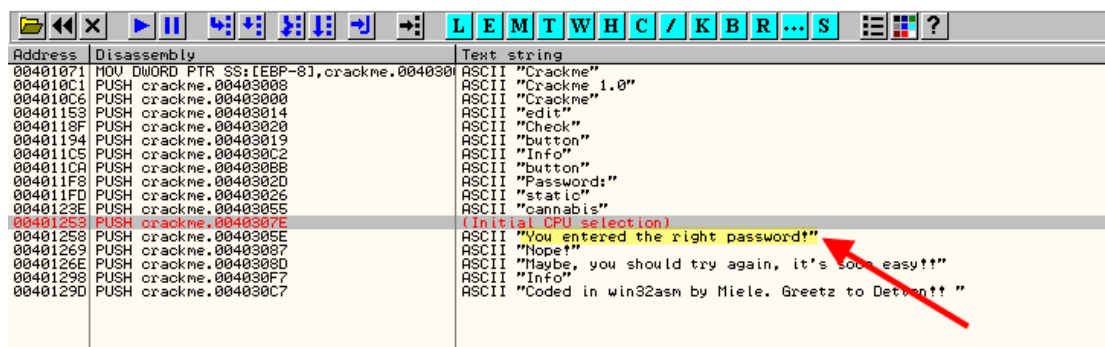
其中有几个 API 函数比较重要,GetWindowTextA:获取用户输入的序列号。lstrcmpA,上一章最后提到的,用于字符串的比较。MessageBoxA:用于显示一条消息,提示是正确或者错误的序列号。

我们可

以给这几个 API 设置断点,当我们输入错误的序列号的时候,就会断下来。但这里我们使用更加简单,快速的做法,我们来看看程序中使用的字符串。



我们通过在反汇编窗口中单击鼠标右键选择-Search for-All referenced text strings 打开字符串列表窗口。



我们可以看到上面列表中有提示成功以及失败的字符串。如果我们在该字符串上面双击鼠标左键,就可以来到 MessageBoxA 调用代码附近。现在我们在"You entered the right password!"字符串上面双击鼠标左键。

0040121E	bb30 2001	UNP HX,120		
00401222	75 64	JNZ SHORT crackme.00401288		
00401224	C1E8 10	SHR EAX,10		
00401227	66 0BC0	OR AX,AX		
0040122A	75 5C	JNZ SHORT crackme.00401288		
0040122C	6A 1E	PUSH 1E		
0040122E	68 37304000	PUSH crackme.00403037		
00401233	FF35 04314001	PUSH DWORD PTR DS:[4031041]		
00401239	E8 A2000000	CALL <JMP.&USER32.GetWindowTextA>		
0040123E	68 55304000	PUSH crackme.00403055		
00401243	68 37304000	PUSH crackme.00403037		
00401248	E8 E7000000	CALL <JMP.&KERNEL32.lstrcmpA>		
0040124D	0BC0	OR EAX,EAX		
0040124F	75 16	JNZ SHORT crackme.00401267		
00401251	6A 00	PUSH 0		
00401253	68 7E304000	PUSH crackme.0040307E		
00401258	68 5E304000	PUSH crackme.0040305E		
0040125D	FF75 08	PUSH DWORD PTR SS:[EBP+8]		
00401260	E8 80000000	CALL <JMP.&USER32.MessageBoxA>		
00401265	EB 21	JMP SHORT crackme.00401288		
00401267	6A 00	PUSH 0		
00401269	68 87304000	PUSH crackme.00403087		
0040126E	68 80304000	PUSH crackme.00403080		
00401273	FF75 08	PUSH DWORD PTR SS:[EBP+8]		
00401276	E8 77000000	CALL <JMP.&USER32.MessageBoxA>		
0040127B	6A 00	PUSH 0		
0040127D	FF35 04314001	PUSH DWORD PTR DS:[4031041]		
00401283	E8 82000000	CALL <JMP.&USER32.SetWindowTextA>		
00401288	66 30 2F01	CMP AX,12F		
0040128C	75 33	JNZ SHORT crackme.004012C1		
00401291	C1E8 10	SHR EAX,10		
00401293	66 0BC0	OR AX,AX		

ESP
EBP
ESI
EDI
EIP

C 0
P 1
A 0
S 1
T 0
D 0
O 0

EFL

ST0
ST1
ST2
ST3
ST4
ST5
ST6
ST7

FST
FCW

来到了比较关键的地方。

首先是 GetWindowTextA 获取用户输入的序列号,然后 lstrcmpA 将输入序列号与正确的序列号进行比较,相同的话就 MessageBoxA 显示“You entered the right password”的提示框,如果不相同就 MessageBoxA 提示“Maybe, you should try again,it's sooo easy!!”的提示框。

所以我们给 lstrcmpA 函数设置一个断点,看看是如何进行比较的。

00401224	C1E8 10	SHR EAX,10		
00401227	66 0BC0	OR AX,AX		
0040122A	75 5C	JNZ SHORT crackme.00401288		
0040122C	6A 1E	PUSH 1E		
0040122E	68 37304000	PUSH crackme.00403037		
00401233	FF35 04314001	PUSH DWORD PTR DS:[4031041]		
00401239	E8 A2000000	CALL <JMP.&USER32.GetWindowTextA>		
0040123E	68 55304000	PUSH crackme.00403055		
00401243	68 37304000	PUSH crackme.00403037		
00401248	E8 E7000000	CALL <JMP.&KERNEL32.lstrcmpA>		
0040124D	0BC0	OR EAX,EAX		
0040124F	75 16	JNZ SHORT crackme.00401267		
00401251	6A 00	PUSH 0		

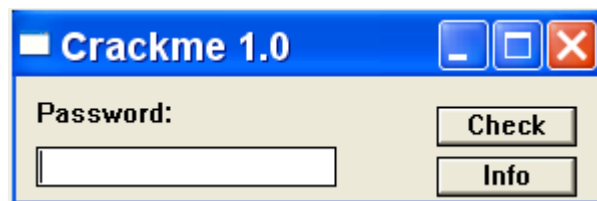
Count = 1E (30.)
Buffer = crackme.00403037
hWnd = NULL
GetWindowTextA
String2 = "cannabis"
String1 = ""
lstrcmpA

Style = MB_OK|MB_APPLMODAL
Title = "Correct!"
Text = "You entered the right password!"
hOwner
MessageBoxA

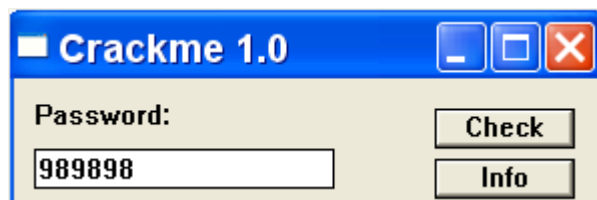
Style = MB_OK|MB_APPLMODAL
Title = "Nope!"
Text = "Maybe, you should try again, it's sooo easy!!"
hOwner
MessageBoxA

Text = NULL
hWnd = NULL
SetWindowTextA

按 F9 键运行起来。



我们在弹出的窗口中随意输入一个序列号,例如:这里我们输入 989898。



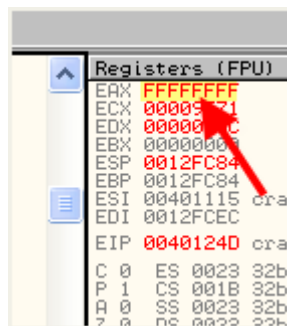
单击 Check 按钮,就会断在我们刚刚设置的断点处。

00401239	E8 A2000000	CALL <JMP.&USER32.GetWindowTextA>		
0040123E	68 55304000	PUSH crackme.00403055		
00401243	68 37304000	PUSH crackme.00403037		
00401248	E8 E7000000	CALL <JMP.&KERNEL32.lstrcmpA>		
0040124D	0BC0	OR EAX,EAX		

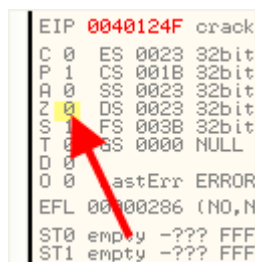
GetWindowTextA
String2 = "cannabis"
String1 = "989898"
lstrcmpA

我们可以看到 OD 中显示的参数,进行比较的两个字符串,分别是“989898”和“cannabis”。

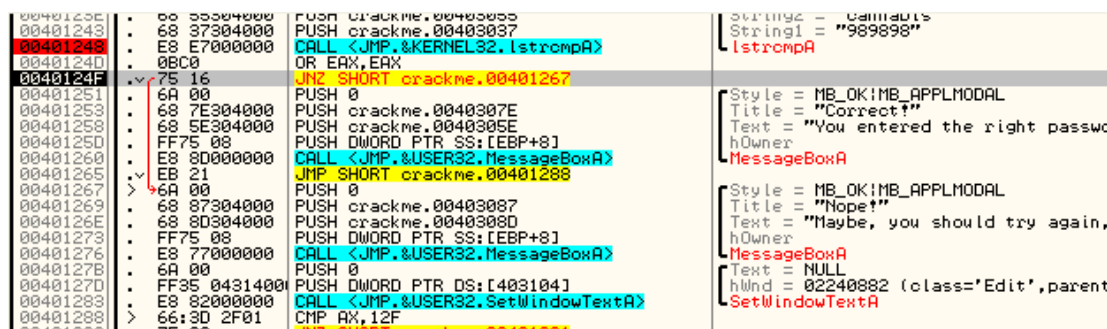
我们按 F8 键单步步过这个 API 调用。



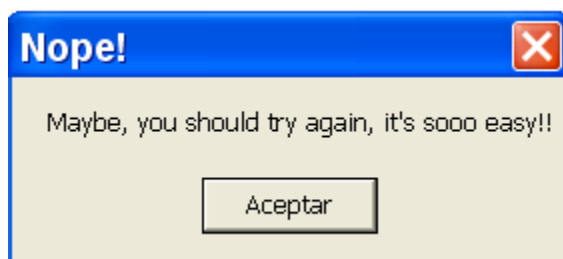
EAX 中存放的返回值为 FFFFFFFF,意味着比较的两个字符串不相同。



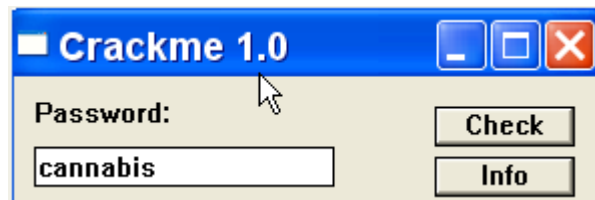
因为比较的结果不为零,所以零标志位 Z 不置位。JNZ 跳转就会实现。(记住:JNZ 当零标志位 Z 置 0 跳转,置 1 不跳转)



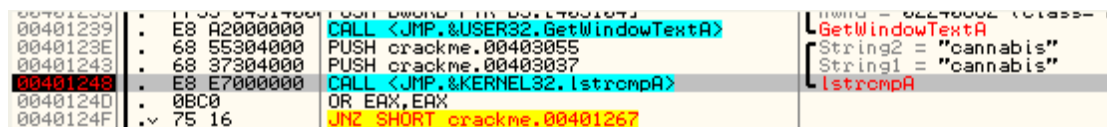
跳转实现以后就弹出一个错误消息框。所以,与我们输入序列号进行比较的"canabis"就是正确的序列号。我们继续运行程序。



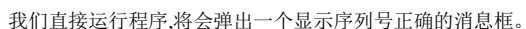
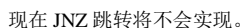
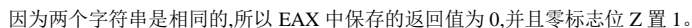
单击 OK,我们回到主窗口,输入正确的序列号"canabis"。



单击 Check 按钮,依然断在了我们设置的断点处。



可以看到待比较的两个字符串是相同的,按 F8 键单步步过这个 API 函数。



我们再来看一个更加复杂的硬编码的 CrackMe(比之前的两个复杂)。

这个 CrackMe 并不是序列号直接进行比较。用 OD 加载这个名为“crakmeeasy”的 CrackMe。

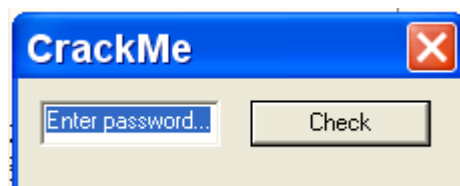


Address	Section	Type	Name	Comment
00403110	.idata	Import	msvrt.atexit	
004030F4	.idata	Import	msvrt._cexit	
00403130	.idata	Import	USER32.DialogBoxParamA	
00403134	.idata	Import	USER32.EndDialog	
004030D4	.idata	Import	KERNEL32.ExitProcess	
004030F8	.idata	Import	msvrt._fileno	
004030FC	.idata	Import	msvrt._fmode	
00403100	.idata	Import	msvrt._fpreset	
004030D8	.idata	Import	KERNEL32.GetCommandLineA	
00403138	.idata	Import	USER32.GetDlgItem	
0040313C	.idata	Import	USER32.GetDlgItemTextA	
0040310C	.idata	Import	msvrt._getmainargs	
004030DC	.idata	Import	KERNEL32.GetModuleHandleA	
004030E0	.idata	Import	KERNEL32.GetStartupInfoA	
00403140	.idata	Import	USER32.GetWindowTextLengthA	
004030E4	.idata	Import	KERNEL32.GlobalAlloc	
00403104	.idata	Import	msvrt._iob	
00403118	.idata	Import	msvrt.memset	
00403144	.idata	Import	USER32.MessageBoxA	
004011F0	.text	Export	<ModuleEntryPoint>	
00403114	.idata	Import	msvrt._p__environ	
00403148	.idata	Import	USER32.SetDlgItemTextA	
00403108	.idata	Import	msvrt._setmode	
004030E8	.idata	Import	KERNEL32.SetUnhandledExceptionFilter	
00403124	.idata	Import	msvrt._set_app_type	
0040311C	.idata	Import	msvrt.signal	
00403120	.idata	Import	msvrt.strlen	

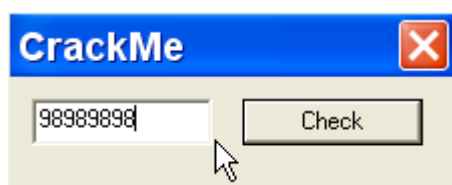
我们在命令栏窗口中输入 bp GetDlgItemTextA。

004020C0 00 00 00 00 00 00 00 00
 004020C8 00 00 00 00 00 00 00 00
 Command Bp GetDlgItemTextA BP a
 Program entry point

我们现在按 F9 键运行程序并输入序列号。



随便输入一个错误的序列号



单击 Check 按钮,断在我们刚刚设置的断点处。

Address	Disassembly	Comment
77D6AC1E	8BFF	MOV EDI,EDI
77D6AC20	55	PUSH EBP
77D6AC21	8BEC	MOV EBP,ESP
77D6AC23	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
77D6AC26	FF75 08	PUSH DWORD PTR SS:[EBP+8]
77D6AC29	E8 E89BF8FF	CALL USER32.GetDlgItem
77D6AC2E	85C0	TEST EAX,EAX
77D6AC30	74 0E	JE SHORT USER32.77D6AC40
77D6AC32	FF75 14	PUSH DWORD PTR SS:[EBP+14]
77D6AC35	FF75 10	PUSH DWORD PTR SS:[EBP+10]
77D6AC38	50	PUSH EAX
77D6AC39	E8 FE74FCFF	CALL USER32.GetWindowTextA
77D6AC3E	EB 0E	JMP SHORT USER32.77D6AC4E
77D6AC40	837D 14 00	CMP DWORD PTR SS:[EBP+14],0
77D6AC44	74 06	JE SHORT USER32.77D6AC4C
77D6AC46	8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]
77D6AC49	C600 00	MOV BYTE PTR DS:[EAX],0
77D6AC4C	33C0	XOR EAX,EAX
77D6AC4E	5D	POP EBP
77D6AC4F	C2 1000	RETN 10
77D6AC52	90	NOP

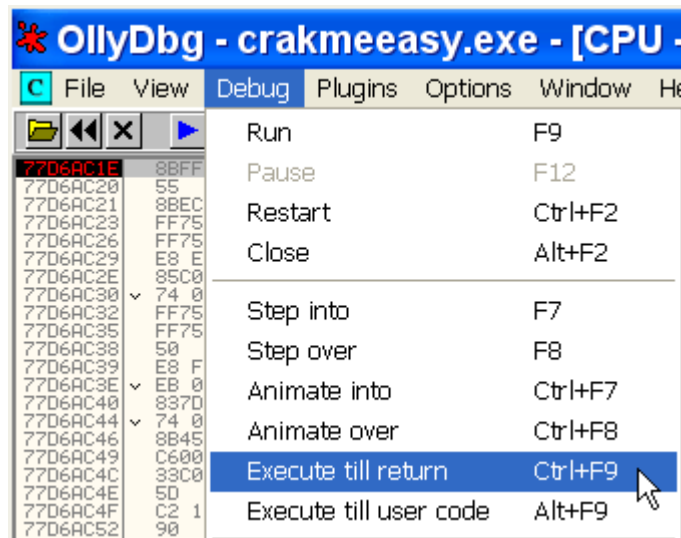
我们来看看堆栈中参数。

0240F998	00401303	CALL to GetDlgItemTextA from crakmea.004012FE
0240F99C	00770A36	hWnd = 00770A36 ('CrackMe',class='#32770')
0240F9A0	00000191	ControlID = 191 (401.)
0240F9A4	00044BB8	Buffer = 00044BB8
0240F9A8	00000009	Count = 9
0240F9AC	05011BD1	
0240F9B0	0240FA3C	

这里我们可以注意到 Buffer 参数:用于保存用户输入的序列号,我们在这个参数上单击鼠标右键选择-Follow in Dump。

Address	Hex dump	ASCII
00044BB8	00 00 00 00 00 00 00 00
00044BC0	00 AB AB AB AB AB AB AB
00044BC8	AB FE EE FE EE FE EE FE
00044BD0	00 00 00 00 00 00 00 00
00044BD8	56 00 05 00 EE 04 EE 00	U..-.-.
00044BE0	28 04 04 00 28 04 04 00	(..)(..)
00044BE8	EE FE EE FE EE FE EE FE
00044BF0	EE FE EE FE EE FE EE FE

这里缓冲区是空的,因为该函数还没有执行,我们选择主菜单-Debug-Execute till return。



执行到函数返回。

77D6AC46	8B45 10	MOV EAX, DWORD PTR SS:[EBP+10]
77D6AC49	C600 00	MOV BYTE PTR DS:[EAX], 0
77D6AC4C	33C0	XOR EAX, EAX
77D6AC4E	5D	POP EBP
77D6AC4F	C2 1000	RETN 10
77D6AC52	90	NOP
77D6AC53	90	NOP
77D6AC54	90	NOP
77D6AC55	90	NOP

我们按下 F7 键,返回到主程序代码中。

Address	Hex dump	ASCII
00044BB8	39 38 39 38 39 38 39 38	98989898
00044BC0	00 AB AB AB AB AB AB AB
00044BC8	AB FE EE FE EE FE EE FE
00044BD0	00 00 00 00 00 00 00 00
00044BD8	56 00 05 00 EE 04 EE 00	U..-.-.
00044BE0	28 04 04 00 28 04 04 00	(..)(..)
00044BE8	EE FE EE FE EE FE EE FE
00044BF0	EE FE EE FE EE FE EE FE

我们可以看到,缓冲区里面保存了错误的序列号。

004012FA	. 8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
004012FD	. 50	PUSH EAX	hWnd
004012FE	. E8 4D030000	CALL <JMP.&USER32.GetDlgItemTextA>	GetDlgItemTextA
00401303	. C745 F0 0000	MOV DWORD PTR SS:[EBP-10],0	
00401308	. B8 22124000	MOV EAX,crakmeea.00401222	ASCII "10445678951"
0040130F	. 8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX	
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]	
00401317	. 8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX	
0040131A	. 8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]	
0040131D	. 8945 D8	MOV DWORD PTR SS:[EBP-28],EAX	
00401320	. 8D45 DC	LEA EAX,DWORD PTR SS:[EBP-24]	
00401323	. 83C4 FC	ADD ESP,-4	
00401326	. 6A 08	PUSH 8	n = 8
00401328	. 6A 00	PUSH 0	c = 00
0040132A	. 50	PUSH EAX	s
0040132B	. E8 F0020000	CALL <JMP.&msvcrt.memset>	memset
00401330	. 83C4 10	ADD ESP,10	
00401333	. C745 CC 0000	MOV DWORD PTR SS:[EBP-34],0	
0040133A	. 8DB6 00000000	LEA ESI,DWORD PTR DS:[ESI]	
00401340	. 83C4 F4	ADD ESP,-0C	
00401343	. 8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]	
00401346	. 50	PUSH EAX	s
00401347	. E8 DC020000	CALL <JMP.&msvcrt.strlen>	strlen
0040134C	. 83C4 10	ADD ESP,10	
0040134F	. 89C0	MOV EAX,EAX	
00401351	. 8D50 FF	LEA EDX,DWORD PTR DS:[EAX-1]	
00401354	. 3955 F0	CMP DWORD PTR SS:[EBP-10],EDX	
00401357	. 72 07	JB SHORT crakmeea.00401360	
00401359	. EB 35	JMP SHORT crakmeea.00401390	
0040135B	. 90	NOP	
0040135C	. 8D7426 00	LEA ESI,DWORD PTR DS:[ESI]	
00401360	. 8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
00401363	. 8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]	
00401366	. 01D0	ADD EAX,EDX	
00401368	. 0FBF10	MOVSX EDX,BYTE PTR DS:[EAX]	
0040136B	. 8D42 EC	LEA EAX,DWORD PTR DS:[EDX-14]	
0040136E	. 8D55 D0	LEA EDX,DWORD PTR SS:[EBP-30]	
00401371	. 8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
00401374	. 0FBF1411	MOVSX EDX,BYTE PTR DS:[ECX+EDX]	
00401378	. 39D0	CMP EAX,EDX	
0040137A	. 75 0D	JNZ SHORT crakmeea.00401389	
0040137C	. 8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]	
0040137E	. 8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]	
Stack SS:[0240FA04]=0240000F			

这里我们看到了一长串数字字符串。有些人可能会问这是正确的序列号吗?呵呵,我也不知道。

004012FE	. E8 4D030000	CALL <JMP.&USER32.GetDlgItemTextA>	GetDlgItemTextA
00401303	. C745 F0 0000	MOV DWORD PTR SS:[EBP-10],0	
00401308	. B8 22124000	MOV EAX,crakmeea.00401222	ASCII "10445678951"
0040130F	. 8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX	
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]	

这里 EAX 保存了 401222 这个常量地址,该地址指向一个固定的字符串。

Registers (FPU)			
EAX	00401222	ASCII "10445678951"	
ECX	77D3219D	USER32.77D3219D	
EDX	00040608		
EBX	00000000		
ESP	0240F9AC		
EBP	0240FA14		
ESI	00401240	crakmeea.00401240	
EDI	0240FA7C		
EIP	0040130F	crakmeea.0040130F	
C 0	ES 0023	32bit 0(FFFFFFFF)	
P 1	CS 001B	32bit 0(FFFFFFFF)	
A 0	SS 0023	32bit 0(FFFFFFFF)	
7 0	DS 0023	32bit 0(FFFFFFFF)	

Address	Hex dump	ASCII
00401222	31 30 34 34 35 36 37 38	10445678
0040122A	39 35 31 00 43 6F 72 72	951.Corr
00401232	65 63 74 21 00 49 6E 76	ect!.Inv
0040123A	61 6C 69 64 21 00 55 89	alid!.Ue
00401242	E5 83 EC 68 8B 45 0C 83	83ghIE.ä
0040124A	F8 10 0F 84 C3 01 00 00	°*ä}0..
00401252	83 F8 10 77 0E 83 F8 02	ä°wä°0
0040125A	0F 84 B5 01 00 00 E9 C3	*ä0..ut
00401262	01 00 00 3D 10 01 00 00	0..=0..t
0040126A	74 0C 3D 11 01 00 00 74	t.=0..t
00401272	2D F9 00 01 00 00 02 04	-i0000..

单步一行,EAX 就等于 401222 了。

MOV EDX,DWORD PTR DS:[EAX]

等价于:

MOV EDX,DWORD PTR DS:[401222]

0040130A	. B8 22124000	MOV EAX,crakmeea.00401222	ASC
0040130F	. 8B10	MOV EDX,DWORD PTR DS:[EAX]	
00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX	
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]	
00401317	. 8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX	
0040131A	. 8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]	
0040131D	. 8945 D8	MOV DWORD PTR SS:[EBP-28],EAX	
00401320	. 8D45 DC	LEA EAX,DWORD PTR SS:[EBP-24]	
00401323	. 83C4 FC	ADD ESP,-4	

该指令将 401222 地址处内容保存到 EDX 中,OD 中的提示窗口中提示为 10445678951 字符串的首 4 个字节。

0040137C	. 8D45 D0	LEA EAX,DWORD PTR SS:
0040137E	. 8B55 E0	MOV EDX,DWORD PTR SS:
DS:[00401222]=34343031		
EDX=00040608		
Address	Hex dump	ASCII
00401222	31 30 34 34 35 36 37 38	10445678
0040122A	39 35 31 00 43 6F 72 72	951.Corr
00401232	45 62 74 21 00 49 65 76	get+1ou

我们按 F7 键将 401222 处内容保存 EDX 中(寄存器中保存的数值和内存中的存放顺序是相反的)。

Registers (FPU)	
EAX	00401222 ASCII
ECX	77D32190 USER32
EDX	34343031
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakme
EDI	0240FA7C
EIP	00401311 crakme
C 0	ES 0023 32bit

下一条指令将 EDX 的内容保存到[EBP-30]堆栈空间中。

0040130A	. B8 22124000	MOV EAX,crakmeea.00401222
0040130F	. 8B10	MOV EDX,DWORD PTR DS:[EAX]
00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]
00401317	. 8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX

OD 的提示窗口显示,[EBP-30]在我的机器上是 240F9e4,我们在数据窗口中转到该地址。

0040137C	. 8D45 D0	LEA EAX
0040137E	. 8B55 E0	MOV EDX
EDX=34343031		
Stack SS:[0240F9E4]=0004443C		
Address	Hex dump	ASCII
0240F9E4	3C 44 04 00 00 02 00 00	<D+.0..
0240F9EC	03 00 00 00 03 00 00 00
0240F9F4	57 00 00 00 14 00 00 00	W...l...
0240F9FC	11 00 00 00 3C 44 04 00	!...<D+.
0240FA04	00 00 00 00 B8 4B 04 00	...@K+.

我们按 F7 键将 EDX 的内容保存到[EBP-30]内存单元中。

Address	Hex dump	ASCII
0240F9E4	31 30 34 34 00 02 00 00	1044.0..
0240F9EC	03 00 00 00 03 00 00 00
0240F9F4	57 00 00 00 14 00 00 00	W...l...

于是

00401311	. 8955 D0	MOV DWORD PTR SS:[EBP-30],EDX
00401314	. 8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]
00401317	. 8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX
0040131A	. 8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]
0040131D	. 8945 D8	MOV DWORD PTR SS:[EBP-28],EAX
00401320	. 8D45 DC	LEA EAX,DWORD PTR SS:[EBP-24]
00401323	. 83C4 FC	ADD ESP,-4
00401326	. 6A 08	PUSH 8
00401329	. 6A 08	PUSH 8

将常量数字字符串的接下来 4 个字节保存到 EDX 中。

00401317	.	8B55 E8	MOV EDX,DWORD PTR SS:
0040132E	.	8B55 E8	MOV EDX,DWORD PTR SS:
DS:[00401226]=38373635			
EDX=34343031			
Address	Hex dump	ASCII	
00401222	31 30 34 34 35 36 37 38	10445678	
0040122A	39 35 31 00 43 6F 72 72	951.Corr	
00401232	65 63 74 21 00 49 6E 76	ect!.Inv	
0040123A	61 6C 69 64 21 00 55 89	alid!.Is	

OD 的显示窗口显示,[EAX+4]的对应的地址为 401226,按 F7 键,接下来的 4 个字节被保存到 EDX 中。

Registers (FPU)	
EAX	00401222 ASC
ECX	77D3219D USE
EDX	38373635
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 cre
EDI	0240FA7C
EIP	00401317 cre
C 0	ES 0023 32b
D 1	CS 0010 32b

然后和之前的赋值一样。

Address	Hex dump	ASCII	
0240F9E0	00 00 00 00 31 30 34 341044	
0240F9E8	35 36 37 38 03 00 00 00	5678*...	
0240F9F0	03 00 00 00 57 00 00 00	*...W...	
0240F9F8	14 00 00 00 11 00 00 00	l...L...	

实际上就是将 4 个字节的内容从一块内存区域拷贝到另一块内存区域。

00401314	.	8B50 04	MOV EDX,DWORD PTR DS:[EAX+4]
00401317	.	8955 D4	MOV DWORD PTR SS:[EBP-2C],EDX
0040131A	.	8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]
0040131D	.	8945 D8	MOV DWORD PTR SS:[EBP-28],EAX

现在是最后的 4 个字节的拷贝。

Address	Hex dump	ASCII	
0240F9E0	00 00 00 00 31 30 34 341044	
0240F9E8	35 36 37 38 39 35 31 00	5678951.	
0240F9F0	03 00 00 00 57 00 00 00	*...W...	

拷贝完毕。

接下来是调用 memset,我们在 OD 中来看看其参数。

00401323	.	83C4 FC	ADD ESP,-4
00401326	.	6A 08	PUSH 8
00401328	.	6A 00	PUSH 0
0040132A	.	50	PUSH EAX
0040132B	.	E8 F0020000	CALL <JMP.&msvcrt.memset>
00401330	.	83C4 10	ADD ESP,10

n = 8
c = 00
s
memset

这里有(n,c,s)3 个值。

s:待填充的内存单元的起始地址

n:需要填充的字节数

c:待填充的值

0240F99C	0240F9F0	s = 0240F9F0
0240F9A0	00000000	c = 00
0240F9A4	00000008	n = 8
0240F9A8	00000009	
0240F9AC	05011BD1	

我们在堆栈中来看看上述参数:以 240F9F0 为起始地址长度为 8 的内存单元填充零。

Address	Hex dump	ASCII	
0240F9E0	00 00 00 00 31 30 34 341044	
0240F9E8	35 36 37 38 39 35 31 00	5678951.	
0240F9F0	00 00 00 00 00 00 00 00	
0240F9F8	14 00 00 00 11 00 00 00	l...L...	
0240FA00	3C 44 04 00 00 00 00 00	<D*....	

按 F8 键单步,我们可以看到 240F9F0 为起始地址的 8 个字节的内存区域被填充零了。

接下来调用的是 strlen 函数,用于计算字符串的长度。

00401343	8D45 D0	LEA EAX,DWORD PTR SS:[EBP-30]	
00401346	50	PUSH EAX	
00401347	E8 DC020000	CALL <JMP.&msvcrt.strlen>	strlen
0040134C	83C4 10	ADD ESP,10	

堆栈中的内容如下:

0240F99C	0240F9E4	00000000	00000000
0240F9A0	00000000	00000000	00000000

是计算起始地址为 240F9E4 的字符串的长度(我们知道这个字符串地址)。

按 F8 键执行 strlen,EAX 中保存字符串的长度。

Registers (FPU)	
EAX	0000000B ASCII "10445678951"
ECX	0240F9E4 ASCII "10445678951"
EDX	7F303438
EBX	00000000
ESP	0240F99C
EBP	0240FA14
ESI	00401240 crakmeea.00401240
EDI	0240FA7C
EIP	0040134C crakmeea.0040134C

我们可以看到长度为 0B,即十进制的 11,就是常量数字字符串的长度。

0040134F	89C0	MOV EAX,EAX	
00401351	8D50 FF	LEA EDX,DWORD PTR DS:[EAX-1]	
00401354	3955 F0	CMP DWORD PTR SS:[EBP-10],EDX	
00401357	72 07	JB SHORT crakmeea.00401360	
00401359	EB 35	JMP SHORT crakmeea.00401390	

这里将 EAX 的值减去 1 保存到 EDX 中,EDX 就等于 0A 了。

接下来将 EDX(值为 0A)与[EBP - 10](值为 0)进行比较。

0040134F	89C0	MOV EAX,EAX	
00401351	8D50 FF	LEA EDX,DWORD PTR DS:[EAX-1]	
00401354	3955 F0	CMP DWORD PTR SS:[EBP-10],EDX	
00401357	72 07	JB SHORT crakmeea.00401360	
00401359	EB 35	JMP SHORT crakmeea.00401390	
0040135B	90	NOP	
0040135C	8D7426 00	LEA ESI,DWORD PTR DS:[ESI]	
0040135D	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
0040135E	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]	

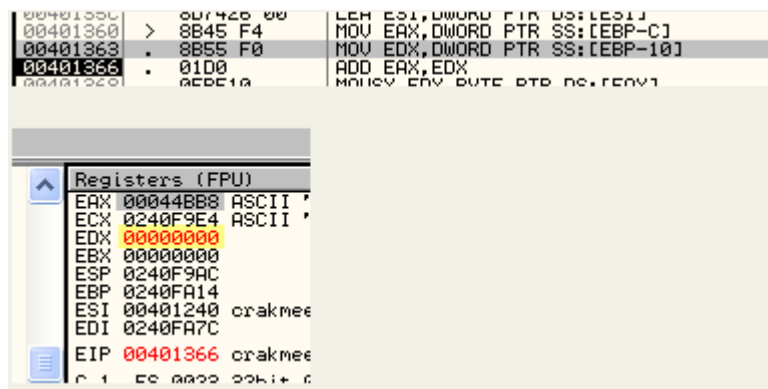
如果 0 小于 0A,就会跳转到 401360。

0040135C	8D7426 00	LEA ESI,DWORD PTR DS:[ESI]	
0040135D	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
0040135E	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]	
0040135F	01D0	ADD EAX,EDX	

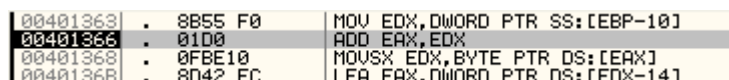
这一行,我们将输入的错误序列号保存到 EAX 中,我们按 F7 键执行,可以看到 EAX 中保存了错误的序列号"98989898"。

Registers (FPU)	
EAX	00044BB8 ASCII "98989898"
ECX	0240F9E4 ASCII "10445678951"
EDX	0000000A
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakmeea.00401240
EDI	0240FA7C
EIP	00401363 crakmeea.00401363
C 1	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)

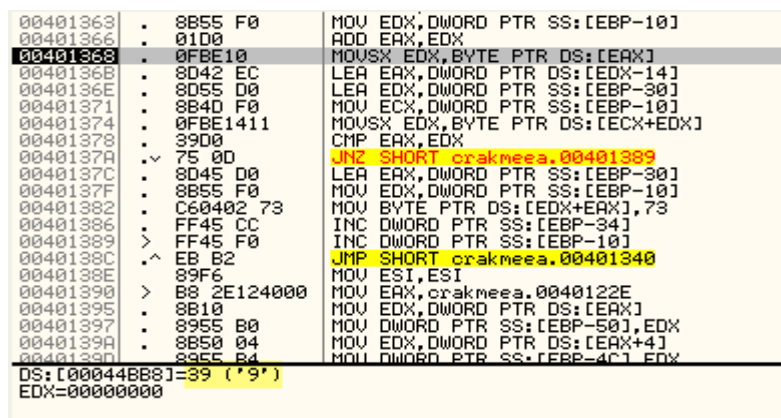
下一条指令,EDX 清零。



接下来一行

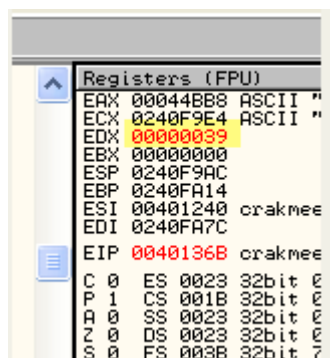


EAX 保存了我们输入的错误序列号的首地址,EDX 的初始值为零,现在创建一个循环,EAX + EDX,然后 EDX 依次递增至获取我们输入序列号的每一个字节。

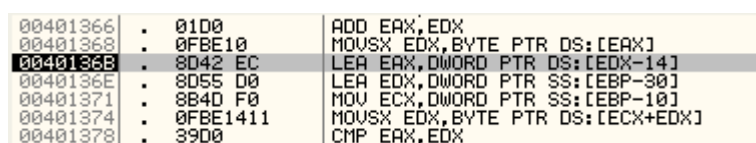


我们知道 MOV SX 指令将指定字节保存到 EDX 中,如果该字节是正数,高位补零,如果该字节为负数高位补 1。

这里,将错误序列号的第一个字节保存 EDX 中,我们单步执行,可以看到 EDX 值变成了 39。



下一条指令是 LEA。



EDX 的值为 39,减去 14,然后将结果保存到 EAX 中。

Registers (FPU)	
EAX	00000025
ECX	0240F9E4 ASCII
EDX	00000039
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 cr-
EDI	0240FA7C
EIP	0040136E cr-
C 0	ES 0023 32b
P 1	CS 001B 32b
A 0	SS 0023 32b

EAX 中保存的结果为 25。

00401368	. 0FBE10	MOVSX EDX, BYTE PTR DS:[EAX]
0040136B	. 8D42 EC	LEA EAX, DWORD PTR DS:[EDX-14]
0040136E	. 8D55 D0	LEA EDX, DWORD PTR SS:[EBP-30]
00401371	. 8B4D F0	MOV ECX, DWORD PTR SS:[EBP-10]
00401374	. 0FBE1411	MOVSX EDX, BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX, EDX

下一条指令是将 EBP-30 的值(我的机器为 240f9e4)保存到 EDX 中。

0040139A	. 8B50 04	MOV EDX, DWORD PTR DS:[EAX+4]
0040139D	. 8B55 B4	MOV DWORD PTR SS:[EBP-4C], EDI
Stack address=0240F9E4, (ASCII "10445678951")		
EDX=00000039		

按 F7 键。

Registers (FPU)	
EAX	00000025
ECX	0240F9E4 ASCII "10445678951"
EDX	0240F9E4 ASCII "10445678951"
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakmeea.00401240
EDI	0240FA7C
EIP	00401371 crakmeea.00401371
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 0	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FDF000(FFF)

EDX 就保存了常量数字字符串的首地址。

00401368	. 0FBE10	MOVSX EDX, BYTE PTR DS:[EAX]
0040136B	. 8D42 EC	LEA EAX, DWORD PTR DS:[EDX-14]
0040136E	. 8D55 D0	LEA EDX, DWORD PTR SS:[EBP-30]
00401371	. 8B4D F0	MOV ECX, DWORD PTR SS:[EBP-10]
00401374	. 0FBE1411	MOVSX EDX, BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX, EDX
0040137A	. 75 0D	JNZ SHORT crakmeea.00401389

我们可以看到 ECX 被作为一个循环变量初始化为零,

0040136E	. 8D55 D0	LEA EDX, DWORD PTR SS:[EBP-30]
00401371	. 8B4D F0	MOV ECX, DWORD PTR SS:[EBP-10]
00401374	. 0FBE1411	MOVSX EDX, BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX, EDX
0040137A	. 75 0D	JNZ SHORT crakmeea.00401389

这里我们看到 ECX 已经被赋值为了零,然后 EDX + ECX 就指向了常量数字字符串的第一个字节,我们来看看 OD 解释窗口的信息。

0040139A	. 8B50 04	MOV EDX, DWORD PTR DS:[EAX+4]
0040139D	. 8B55 B4	MOV DWORD PTR SS:[EBP-4C], EDI
Stack DS:[0240F9E4]=31 ('1')		
EDX=0240F9E4, (ASCII "10445678951")		

31 对应的 ASCII 码为 '1',为常量数字字符串的第一个字符。

00401371	. 8B4D F0	MOV ECX, DWORD PTR SS:[EBP-10]
00401374	. 0FBE1411	MOVSX EDX, BYTE PTR DS:[ECX+EDX]
00401378	. 39D0	CMP EAX, EDX
0040137A	. 75 0D	JNZ SHORT crakmeea.00401389
0040137C	. 8D45 D0	LEA EAX, DWORD PTR SS:[EBP-30]
0040137F	. 8B55 B4	MOV DWORD PTR SS:[EBP-4C], EDI

这里就进行比较了

00401397	. 8B55 B4	MOV DWORD PTR SS:[EBP-4C], EDI
0040139A	. 8B50 04	MOV EDX, DWORD PTR DS:[EAX+4]
0040139D	. 8B55 B4	MOV DWORD PTR SS:[EBP-4C], EDI
EDX=00000031		
EAX=00000025		

EAX 保存了我们输入的错误字符串的第一个字符 39 减去 14 的结果,这里为 25,EDX 保存了常量数字字符串的第一个字节 31。

因此,我们可以看到的

CMP EAX, EDX

实际上是

CMP 错误的序列号的第一个字节-14,常量数字字符串的第一个字节

CMP 25,31

由于这两个操作数的差值不为零,所以零标志位 Z 不会置 1,JNZ 跳转就会实现。

我们输入的是一个错误的序列号,如果输入的是一个正确的序列号的话,那么比较的结果就是正确的。

CMP (正确序列号的第一个字节值-14),31

判断两者相等的条件为:

正确序列号的第一个字节值-14 = 常量数字字符串的第一个字节

所以

正确的序列号的第一个字节值 = 常量数字字符串第一个字节值 + 14



正确的序列号的第一个字节值 = 31 + 14

正确的序列号的第一个字节值 = 45,该 ASCII 码对应的字符为"E"。

所以序列号的第一个字母为 E。

以上字符校验过程重复进行,直到所有字符都校验完毕。

第一个字节值 = 常量数字字符串第一个字节值+ 14

第二个字节值 = 常量数字字符串第二个字节值+14

第三个字节值 = 常量数字字符串第三个字节值+14

依次类推。

我们用这个公式来计算序列号(正确序列号字节值 = 常量数字字符串对应字节值 + 14)的每个字符。

31	30	34	34	35	36	37	38	10	44	56	78
39	35	31	00	00	00	00	00	95	1	.	.

31 + 14 = 45 对应字符 E

30 + 14 = 44 对应字符 D

34 + 14 = 48 对应字符 H

34 + 14 = 48 对应字符 H

35 + 14 = 49 对应字符 I

36 + 14 = 50 对应字符 J

37 + 14 = 51 对应字符 K

38 + 14 = 52 对应字符 L

39 + 14 = 53 对应字符 M

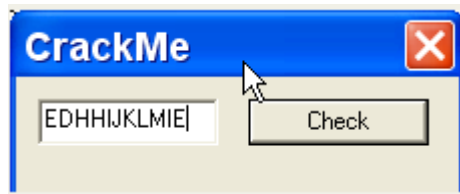
35 + 14 = 49 对应字符 I

31 + 14 = 45 对应字符 E

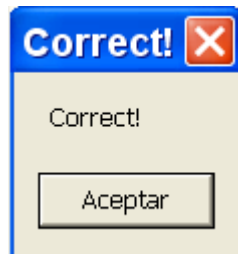
因此,正确的序列号为:

EDHHIJKLMIE

我们删除之前设置过的断点,然后在主窗口中输入正确的序列号。



单击 check 按钮。



这个 CrackMe 就完成了。

这个 CrackMe 相比之前的要复杂一点,但是我觉得稍加练习也是可以很容易的解决它的。

好了,这里有一个名为 Splish 的 CrackMe,大家尝试一下找到它的序列号。