

第二十九章-P-CODE-Part1

(本章的 CrackMe 需要支持库 MSVBVM50.DLL)

前面章节我们已经介绍了 Visual Basic 破解相关的基础知识,如果大家还想更加深入的研究 VB 应用程序破解的话,可以学习高级篇中的关于 VB 破解的相关内容。(PS:就是我打包上传到百度网盘的西班牙破解文集系列)

COCO 写的 VB 破解教程就比较好,其中可能会涉及到比较复杂的处理技巧,拿来练习做好不过了。还有一些其他优秀教程可供我们更加深入的研究 VB Cracking。接着接下来我们将讨论下一个话题-P-CODE。

我们知道 VB 编写的应用程序有两种编译方式:一种是 Native 方式,我们前面章节已经讨论过了,另一种就是 P-CODE 方式-我们接下来将讨论的话题。

Native 编译的代码和 P-CODE 的主要区别在于:Native 是直接执行代码段中的代码的。而对于 P-CODE,如果我们使用之前那个 Patch 过的 OD 加载它,并且给代码段设置内存访问断点的话(实际上是内存执行断点),除非是直接调用 API 函数,否则不会断下来。说明,该区段没有直接可供 CPU 执行的代码。

P-code,实际上是一组自定义的指令集,必须通过基于堆栈的虚拟机翻译为 80X86 上的指令集才能执行,即通过 msbvm50.dll 和 msbvm60.dll 这两个动态库来解释执行。也可以理解为通过 P-CODE 告诉虚拟机将要进行什么操作。例如:

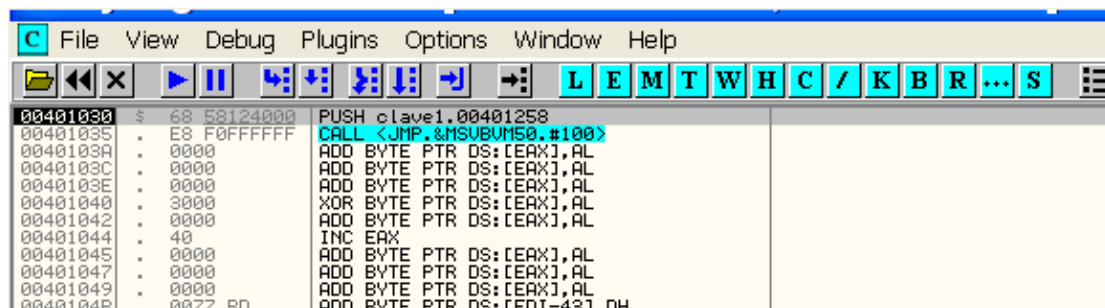
1e 表示 执行无条件跳转(JMP)

1e 意味着执行无条件跳转,该无条件跳转是通过虚拟机(msbvm50.dll,msbvm60.dll)来执行的,也就是说 P-CODE 程序会读取代码段中的值,然后由这些值来告诉虚拟机需要执行什么操作。这也就是为什么我们说 P-CODE 程序的代码段中没有可执行代码的缘故。

现在让我们一起拿起“手术刀”来跟踪,剖析一个 CrackMe 的 P-CODE 的奥秘吧。

我们实验的这个 CrackMe 名字叫做 clave1,这个 CrackMe 是我朋友 JB DUC 写的,用来介绍 P-CODE 相关的内容正好合适,我们需要找到该 CrackMe 的正确序列号。

很多人调试 P-CODE 可能喜欢用 WKT,如果大家想了解 WKT 怎么调试 P-CODE 的话,可以看 JB DUC 关于 P-CODE 的破解教程,我们这里还是用 OllyDbg 来调试了,由于微软官方并没有提供操作码的清单,所以还会用到另外一个工具 EXDEC-这个工具可以识别操作码的名称。

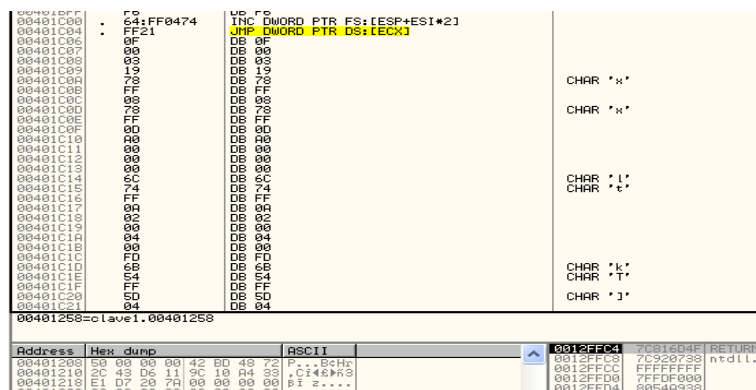


我们使用原版的 OD,配置好反反调试插件,加载该 CrackMe。可以看到停在了入口点处。

说明一点,之前介绍的用于剔除 Native 程序的 NAG 窗口的 4C 法同样适用于 P-CODE。

现在来看看跟 Native 的不同的地方:

我们从入口点往下看,会发现没有几行代码。



我们只看到了大片的字节码,这里不要试图去分析这些字节码(毫无意义),我们应该还记得对于 Native 方式编译的 VB 应用程序,入口点往下还会有大量的代码吧。

Address	Disassembly	Comment
00401D40	DB 00	
00401D4E	DB 00	
00401D4F	DB 00	
00401D50	FF	
00401D51	FF	
00401D52	FF	
00401D53	FF	
00401D54	00	
00401D55	00	
00401D56	00	
00401D57	00	
00401D58	5C1A4000	DD Flipi1.00401A5C
00401D5C	EC194000	DD Flipi1.004019EC
00401D60	D4324000	DD Flipi1.004032D4
00401D64	5C1A4000	DD Flipi1.00401A5C
00401D68	841A4000	DD Flipi1.00401A84
00401D6C	D8324000	DD Flipi1.004032D8
00401D70	5C1A4000	DD Flipi1.00401A5C
00401D74	9C1A4000	DD Flipi1.00401A9C
00401D78	DC324000	DD Flipi1.004032DC
00401D7C	5C1A4000	DD Flipi1.00401A5C
00401D80	B41A4000	DD Flipi1.00401AB4
00401D84	E0324000	DD Flipi1.004032E0
00401D88	CC	INT3
00401D89	CC	INT3
00401D8A	CC	INT3
00401D8B	CC	INT3
00401D8C	CC	INT3

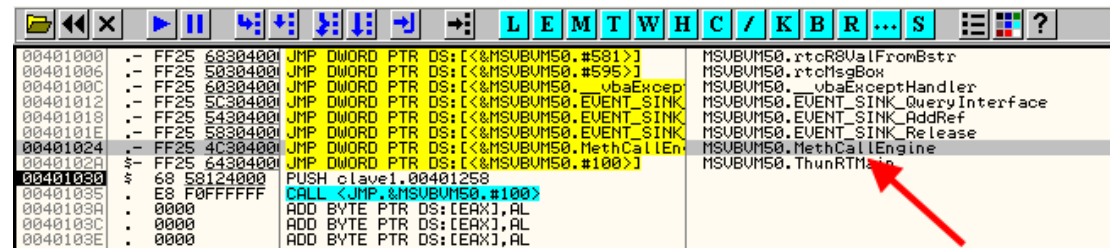
也有相似的字节码,我们继续往下。

Address	Disassembly	Comment
00401D98	CC	INT3
00401D9C	CC	INT3
00401D9D	CC	INT3
00401D9E	CC	INT3
00401D9F	CC	INT3
00401DA0	> 55	PUSH EBP
00401DA1	. 8BEC	MOV EBP,ESP
00401DA3	. 8BEC 0C	SUB ESP,0C
00401DA6	. 68 B6104000	PUSH <JMP.&MSUBUM160.__vbaExceptionHandler>
00401DAB	. 64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
00401DB1	. 50	PUSH EAX
00401DB2	. 64:8925 00000000	MOV DWORD PTR FS:[0],ESP
00401DB9	. 81EC A0000000	SUB ESP,0A0
00401DBF	. 53	PUSH EBX
00401DC0	. 56	PUSH ESI
00401DC1	. 57	PUSH EDI
00401DC2	. 8965 F4	MOV DWORD PTR SS:[EBP-C],ESP
00401DC5	. C745 F8 9810	MOV DWORD PTR SS:[EBP-8],Flipi1.00401098
00401DCC	. 8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
00401DCF	. 8BC6	MOV EAX,ESI
00401DD1	. 83E0 01	AND EAX,1
00401DD4	. 8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
00401DD7	. 83E6 FE	AND ESI,FFFFFFFE
00401DDA	. 56	PUSH ESI
00401DDB	. 8975 08	MOV DWORD PTR SS:[EBP+8],ESI
00401DDE	. 8B0E	MOV ECX,DWORD PTR DS:[ESI]
00401DE0	. FF51 04	CALL DWORD PTR DS:[ECX+4]
00401DE3	. 8B16	MOV EDX,DWORD PTR DS:[ESI]
00401DE5	. 33DB	XOR EBX,EBX
00401DE7	. 56	PUSH ESI
00401DE8	. 895D E4	MOV DWORD PTR SS:[EBP-1C],EBX
00401DEB	. 895D E0	MOV DWORD PTR SS:[EBP-20],EBX
00401DEE	. 895D D0	MOV DWORD PTR SS:[EBP-30],EBX
00401DF1	. 895D C0	MOV DWORD PTR SS:[EBP-40],EBX
00401DF4	. 895D B0	MOV DWORD PTR SS:[EBP-50],EBX
00401DF7	. 895D A0	MOV DWORD PTR SS:[EBP-60],EBX
00401DFA	. 895D 90	MOV DWORD PTR SS:[EBP-70],EBX
00401DFD	. 895D 80	MOV DWORD PTR SS:[EBP-80],EBX
00401E00	. FF92 00030000	CALL DWORD PTR DS:[EDX+3000]
00401E06	. 50	PUSH EAX
00401E07	. 8D45 E0	LEA EAX,DWORD PTR SS:[EBP-20]
00401E0A	. 50	PUSH EAX
00401E0B	. FF15 20104000	CALL DWORD PTR DS:[<&MSUBUM160.__vbaObjSet
00401E11	. 8BF8	MOV EDI,EAX
00401E13	. 8D55 E4	LEA EDX,DWORD PTR SS:[EBP-1C]
00401E16	. 52	PUSH EDX
00401E17	. 57	PUSH EDI
00401E18	. 8B0F	MOV ECX,DWORD PTR DS:[EDI]
00401E1A	. FF91 A0000000	CALL DWORD PTR DS:[ECX+A00]
00401E20	. 3BC3	CMP EAX,EBX
00401E22	. DBE2	FCLEX

这里我们可以看到大段的代码,而 P-CODE 的程序的话 OllyDbg 可能也会显示处少量的代码,OD 识别成了指令,但其实这些地方也

是纯字节码。

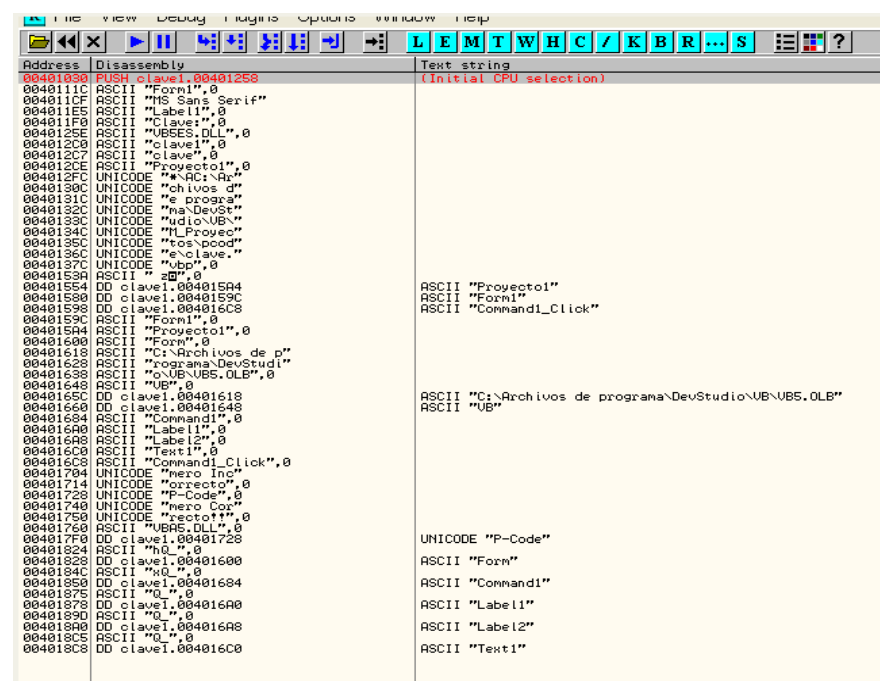
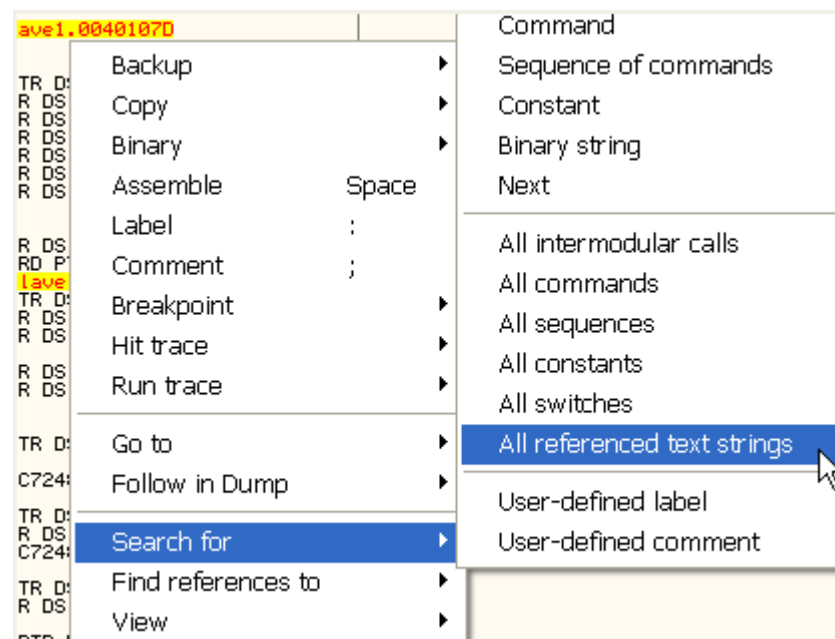
我们继续来看刚刚那个 P-CODE 的 CrackMe。



```
00401000  FF25 68304000 JMP DWORD PTR DS:[<&MSUBUM50.#581>] MSUBUM50.rtcR8UalFromBstr
00401006  FF25 50304000 JMP DWORD PTR DS:[<&MSUBUM50.#595>] MSUBUM50.rtcMsgBox
0040100C  FF25 60304000 JMP DWORD PTR DS:[<&MSUBUM50._vbaExcept MSUBUM50._vbaExceptionHandler
00401012  FF25 5C304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK MSUBUM50.EVENT_SINK_QueryInterface
00401018  FF25 54304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK MSUBUM50.EVENT_SINK_AddRef
0040101E  FF25 58304000 JMP DWORD PTR DS:[<&MSUBUM50.EVENT_SINK MSUBUM50.EVENT_SINK_Release
00401024  FF25 4C304000 JMP DWORD PTR DS:[<&MSUBUM50.MethCallEn MSUBUM50.MethCallEngine
0040102A  FF25 64304000 JMP DWORD PTR DS:[<&MSUBUM50.#100>] MSUBUM50.ThunRTTh
00401030  68 30124000 PUSH clave1.00401258
00401036  E8 F0FFFFFF CALL <JMP.<MSUBUM50.#100>
0040103A  0000 ADD BYTE PTR DS:[EAX],AL
0040103C  0000 ADD BYTE PTR DS:[EAX],AL
0040103E  0000 ADD BYTE PTR DS:[EAX],AL
```

另外一个特征就是 MethCallEngine 这个 API 函数,该函数我们在 P-CODE 方式编译的 VB 应用程序中都能看到,所以我们甄别一个 VB 应用程序是不是以 P-CODE 方式编译的,一般有两步:1:看代码段中入口点以下是不是大片的字节码 2:看有没有 MethCallEngine 这个函数。

好了,现在来看看看字符串列表中有没有什么有价值的字符串。



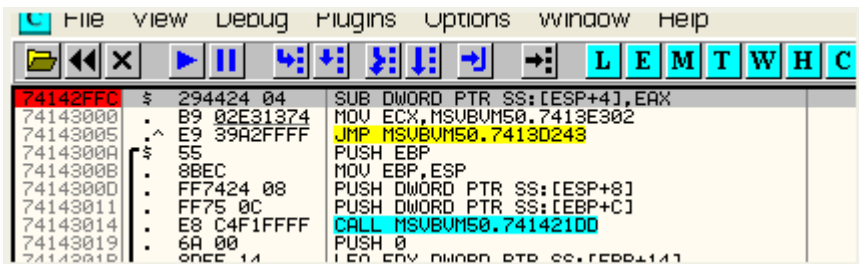
貌似没看到什么有用的字符串。

好,那我们直接给 JMP MethCallEngine 这一行下一个断点吧。

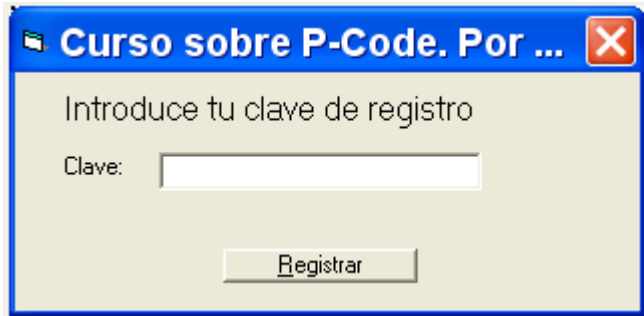
0040100C	.- FF25 60304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.__vbaExceptionHandler]	MSUBUM50.__vbaExceptionHandler
00401012	.- FF25 5C304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.EVENT_SINK_QueryInterface]	MSUBUM50.EVENT_SINK_QueryInterface
00401018	.- FF25 54304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.EVENT_SINK_AddRef]	MSUBUM50.EVENT_SINK_AddRef
0040101E	.- FF25 58304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.EVENT_SINK_Release]	MSUBUM50.EVENT_SINK_Release
00401024	.- FF25 4C304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.MethCallEngine]	MSUBUM50.MethCallEngine
0040102A	.- FF25 64304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.#100]	MSUBUM50.ThunRTMain
00401030	.\$ 68 58124000	PUSH clave1.00401258	
00401035	.- E8 F0FFFFFF	CALL <JMP.&MSUBUM50.#100>	
0040103A	.- 0000	ADD BYTE PTR DS:[EAX],AL	
0040103C	.- 0000	ADD BYTE PTR DS:[EAX],AL	
0040103E	.- 0000	ADD BYTE PTR DS:[EAX],AL	
00401040	.- 3000	XOR BYTE PTR DS:[EAX],AL	

接下来我们选中 JMP MethCallEngine 这一行,单击鼠标右键选择-Follow,转入 MethCallEngine 内部,接着在 MethCallEngine 入口点处设置一个断点。

00401018	.- FF25 54304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.EVENT_SINK_AddRef]	MSUBUM50.EVENT_SINK_AddRef
0040101E	.- FF25 58304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.EVENT_SINK_Release]	MSUBUM50.EVENT_SINK_Release
00401024	.- FF25 4C304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.MethCallEngine]	MSUBUM50.MethCallEngine
0040102A	.- FF25 64304000	JMP NEAR DWORD PTR DS:[&MSUBUM50.ThunRTMain]	MSUBUM50.ThunRTMain
00401030	.\$ 68 58124000	PUSH clave1.00401258	
00401035	.- E8 F0FFFFFF	CALL <JMP.&MSUBUM50.#100>	
0040103A	.- 0000	ADD BYTE PTR DS:[EAX],AL	
0040103C	.- 0000	ADD BYTE PTR DS:[EAX],AL	
0040103E	.- 0000	ADD BYTE PTR DS:[EAX],AL	
00401040	.- 3000	XOR BYTE PTR DS:[EAX],AL	
00401042	.- 0000	ADD BYTE PTR DS:[EAX],AL	
00401044	.- 40	INC EAX	
00401045	.- 0000	ADD BYTE PTR DS:[EAX],AL	
00401047	.- 0000	ADD BYTE PTR DS:[EAX],AL	
00401049	.- 0000	ADD BYTE PTR DS:[EAX],AL	
0040104B	.- 0077 BD	ADD BYTE PTR DS:[EDI],AL	
0040104D	.- 40	INC EAX	

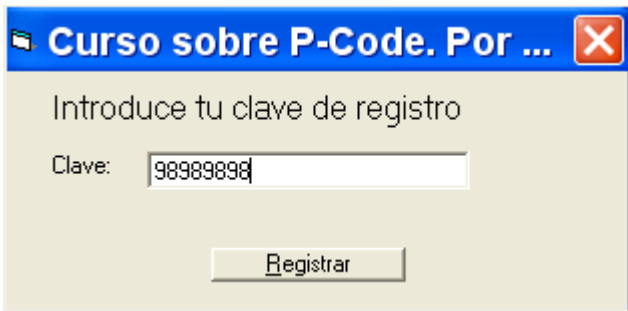


运行起来。

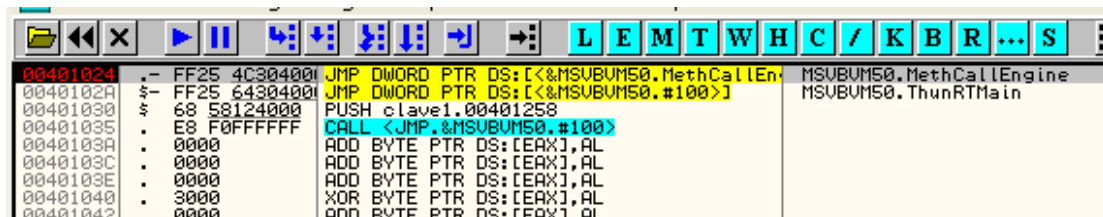


弹出了注册窗口,等待我们输入序列号。对于 Native 的 VB 程序,我们可以断 API 函数,但是 P-CODE 就搞不定了,但 4C 法对 P-CODE 程序依然有用。

现在我们随便输入一个错误的序列号。

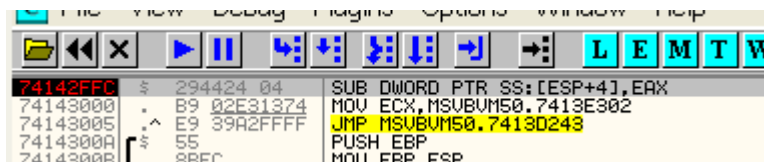


我们单击 Registrar(西班牙语:注册)按钮。



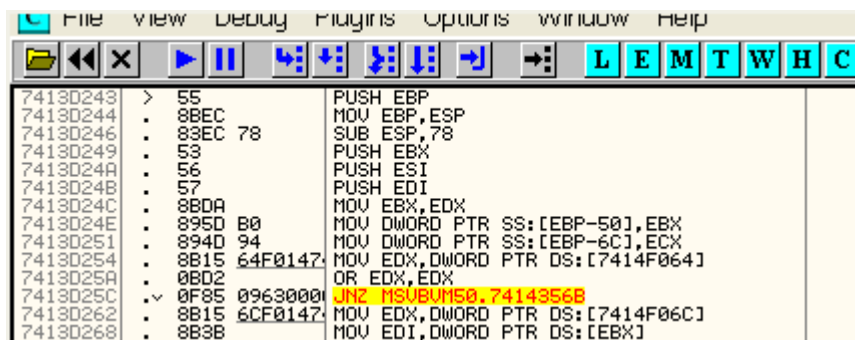
```
00401024  FF25 4C304001 JMP DWORD PTR DS:[&MSUBUM50.MethCallEn- MSUBUM50.MethCallEngine
0040102A  FF25 64304001 JMP DWORD PTR DS:[&MSUBUM50.#100>] MSUBUM50.ThunRTMain
00401030  68 58124000 PUSH clave1.00401258
00401035  E8 F0FFFFFF CALL <JMP.&MSUBUM50.#100>
0040103A  0000 ADD BYTE PTR DS:[EAX],AL
0040103C  0000 ADD BYTE PTR DS:[EAX],AL
0040103E  0000 ADD BYTE PTR DS:[EAX],AL
00401040  3000 XOR BYTE PTR DS:[EAX],AL
00401042  0000 ORN BYTE PTR DS:[EAX],01
```

断在了 JMP MethCallEngine 处,MethCallEngine 函数对 P-CODE 进行初始化。



```
74143024  294424 04 SUB DWORD PTR SS:[ESP+4],EAX
74143028  B9 02E31374 MOV ECX,MSUBUM50.7413E302
7414302C  E9 39A2FFFF JMP MSUBUM50.7413D243
74143030  55 PUSH EBP
74143032  8BEC MOV EBP,ESP
```

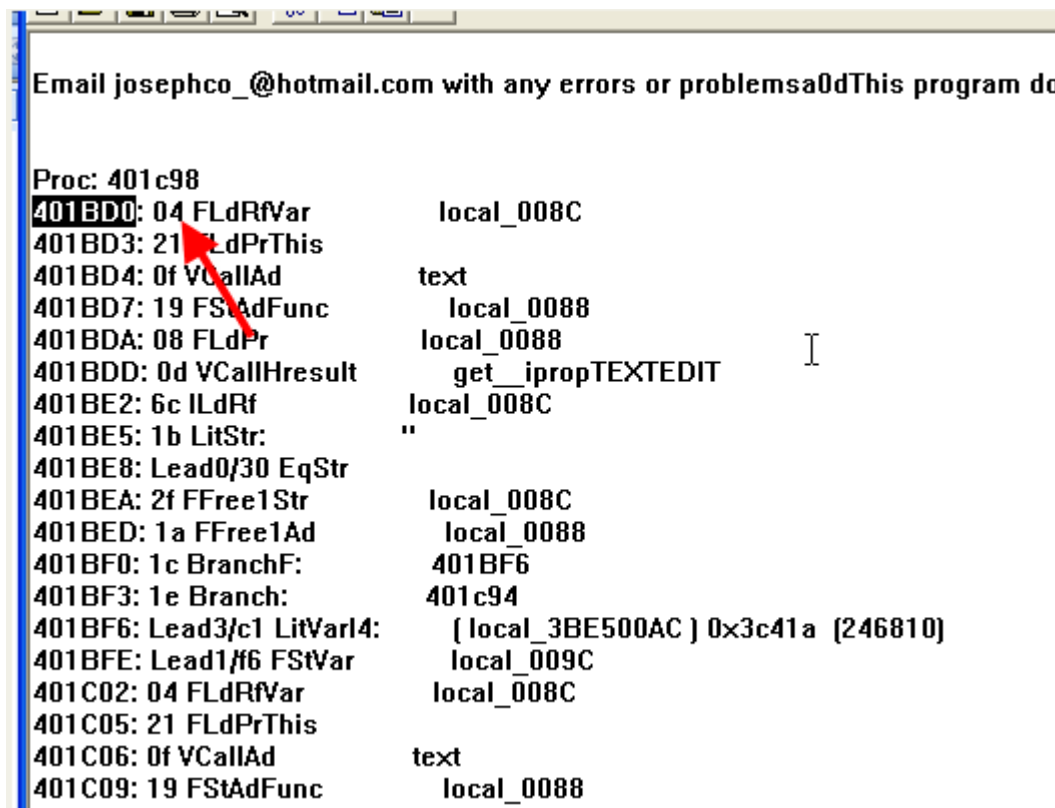
我们继续运行,断在了 MethCallEngine 的入口处,我们来看看它做了些什么。



```
7413D243  55 PUSH EBP
7413D244  8BEC MOV EBP,ESP
7413D246  83EC 78 SUB ESP,78
7413D249  53 PUSH EBX
7413D24A  56 PUSH ESI
7413D24B  57 PUSH EDI
7413D24C  8BDA MOV EBX,EDX
7413D24E  895D B0 MOV DWORD PTR SS:[EBP-50],EBX
7413D251  894D 94 MOV DWORD PTR SS:[EBP-6C],ECX
7413D254  8B15 64F0147 MOV EDX,DWORD PTR DS:[7414F064]
7413D25A  0BD2 OR EDX,EDX
7413D25C  0F85 09630000 JNZ MSUBUM50.74143568
7413D262  8B15 6CF0147 MOV EDX,DWORD PTR DS:[7414F06C]
7413D268  8B3B MOV EDI,DWORD PTR DS:[EBX]
```

我们可以看到跳转到了 7413D243 处。

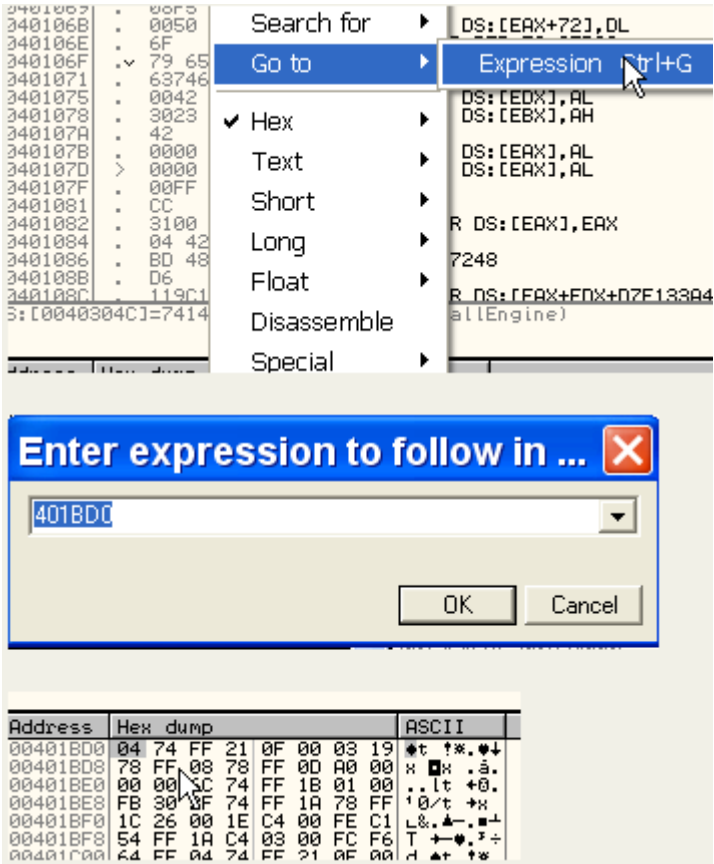
现在我们用 ExDec 打开该 CrackMe。ExDec 是一款专门针对 P-CODE 的反编译器。我们来看看它显示了些什么。



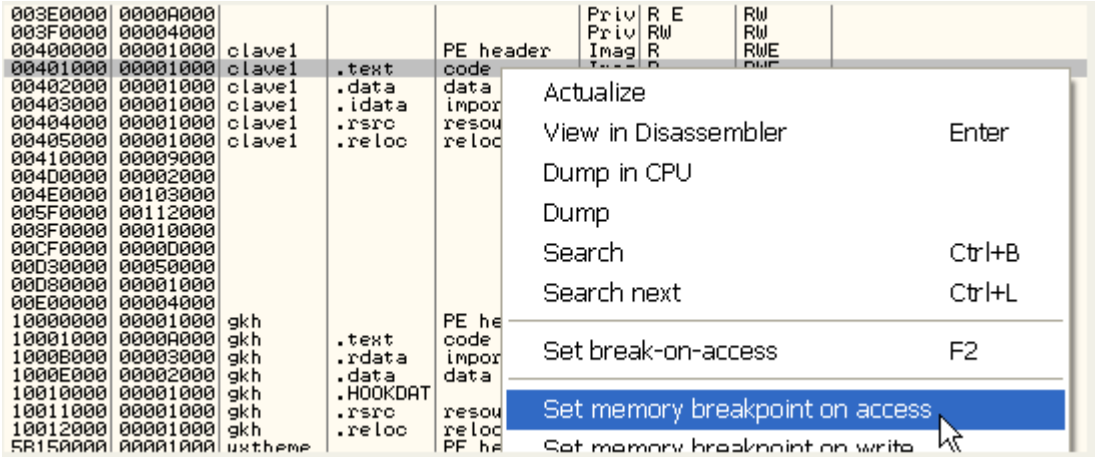
```
Email josephco_@hotmail.com with any errors or problemsa0dThis program do

Proc: 401c98
401BD0: 04 FLdRfVar local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd text
401BD7: 19 FStAdFunc local_0088
401BDA: 08 FLdPr local_0088
401BDD: 0d VCallHresult get_ipropTEXTEDIT
401BE2: 6c ILdRf local_008C
401BE5: 1b LitStr: ""
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str local_008C
401BED: 1a FFree1Ad local_0088
401BF0: 1c BranchF: 401BF6
401BF3: 1e Branch: 401c94
401BF6: Lead3/c1 LitVarl4: [ local_3BE500AC ] 0x3c41a [246810]
401BFE: Lead1/f6 FStVar local_009C
401C02: 04 FLdRfVar local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd text
401C09: 19 FStAdFunc local_0088
```

我们可以看到将被读取的第一个字节是 04,位于 401BD0 地址处,应该在第一次读取代码段指令的附近,我们可以给该字节设置一个内存访问断点。



我们单击 Registrar 按钮后就会断在了 MethCallEngine 处,我们给代码段设置内存访问断点。运行起来的话,将断在了读取代码段的指令处,读取 401BD0 内存单元中 04 的指令应该就在附近,所以我们接着给该字节设置内存访问断点,继续运行,就能马上定位到。



我们按 F9 运行起来。



断了下来,这个时候我们给刚刚那个 04 字节设置内存访问断点,运行起来,又断了下来,我们可以看到 ESI 指向的就是 401BD0 内存单元。(PS:这里下断点的顺序我换了次序,一次就可以定位到,作者 10 次才定位到)

```

7413D31C . 8A06 MOV AL, BYTE PTR DS:[ESI]
7413D31E . 46 INC ESI
7413D31F . FF2485 94ED11 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D326 > 66:F743 0C 11 TEST WORD PTR DS:[EBX+C],10
7413D32C > 0F85 59620001 JNZ MSUBUM50.7414358B

```

这里读取[ESI]的 04 字节值保存到 AL 中,这里是读取到的第一个 P-CODE 操作码。

接着我们来看看 ExDec 中显示的其他操作码。

Proc: 401c98
401BD0: 04 FLdRfVar local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd text
401BD7: 19 FStAdFunc local_0088
401BDA: 08 FLdPr local_0088
401BDD: 0d VCallHresult get_ipropTEXTEDIT
401BE2: 6c iLdRf local_008C
401BE5: 1b LitStr: ""
401BE8: Lead0/30 EqStr

Address	Hex dump	ASCII
00401BD0	04 74 FF 21 0F 00 03 19	4t !*,.4
00401BD8	78 FF 08 78 FF 00 A0 00	x 8x .a.
00401BE0	00 00 6C 74 FF 18 01 00	..lt +0.
00401BE8	FB 30 2F 74 FF 1A 78 FF	!0/t +x
00401BF0	1C 26 00 1E C4 00 FE C1	l&.A.-.u+
00401BF8	54 FF 1A C4 03 00 FC F6	T +-.f÷
00401C00	64 FF 04 74 FF 21 0F 00	d 4t !*,.
00401C08	03 19 78 FF 08 78 FF 00	4+x 8x .
00401C10	A0 00 00 00 6C 74 FF 0A	a...lt .
00401C18	02 00 04 00 FD 68 54 FF	0..kT
00401C20	5D 04 64 FF FB 40 2F 74	!d !0/t

我们将 ExDec 跟 OD 的数据窗口显示的内容对应起来看,会发现这些操作码并不连续,这是因为中间夹杂着操作码需要的参数。

```

7413D312 . 03F3 ADD ESI,EBX
7413D314 . 8975 A8 MOV DWORD PTR SS:[EBP-58],ESI
7413D317 . 8975 EC MOV DWORD PTR SS:[EBP-14],ESI
7413D31A > 33C0 XOR EAX,EAX
7413D31C . 8A06 MOV AL, BYTE PTR DS:[ESI]
7413D31E . 46 INC ESI
7413D31F . FF2485 94ED11 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D326 > 66:F743 0C 11 TEST WORD PTR DS:[EBX+C],10
7413D32C > 0F85 59620001 JNZ MSUBUM50.7414358B
7413D332 > 66:F743 0C 21 TEST WORD PTR DS:[EBX+C],20
7413D338 > 75 1D JNZ SHORT MSUBUM50.7413D357
7413D33A . 0BC0 OR EAX,EAX
7413D33C . B8 00E00000 MOV EAX,0E000
7413D341 > 74 14 JE SHORT MSUBUM50.7413D357

```

正如你所看到的,这里正在读取第一个字节。

Registers (FPU)

EAX	00000000
ECX	0012FA1C
EDX	740569D0 MSUBUM50.740569D0
EBX	00401C98 clave1.00401C98
ESP	0012F3C4
EBP	0012F4E0
ESI	00401BD0 clave1.00401BD0
EDI	00159888
EIP	7413D31C MSUBUM50.7413D31C
C 0	ES 0023 32bit 0(FFFFFFFF)

我们可以看到当前 ESI 指向了 401BD0,下一行,ESI 值递增 1,以便读取操作码的参数。

```

7413D317 . 8975 EC      MOV DWORD PTR SS:[EBP-14],ESI
7413D319 > 33C0        XOR EAX,EAX
7413D31C . 8A06        MOV AL,BYTE PTR DS:[ESI]
7413D31E . 4E         INC ESI
7413D31F . FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D326 > 66:F743 0C 1 TEST WORD PTR DS:[EBX+C],10
7413D32C > 0F85 5962000 JNZ MSUBUM50.7414358B
7413D332 > 66:F743 0C 2 TEST WORD PTR DS:[EBX+C],20

```

接着我们就到了间接跳转 JMP 指令这里,这一行将去执行这个操作码(我们在 ExDec 中看到的 04)。

```

401BD0: 04 FLdRfVar      local_008C

```

我们可以看到一个陌生的操作码。

```

7413D9A2 > 0FBF06      MOV SX EAX,WORD PTR DS:[ESI]
7413D9A5 . 03C5        ADD EAX,EBP
7413D9A7 . 50         PUSH EAX
7413D9A8 . 33C0        XOR EAX,EAX
7413D9AA . 8A46 02     MOV AL,BYTE PTR DS:[ESI+2]
7413D9AD . 83C6 03     ADD ESI,3
7413D9B0 . FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]

```

这里我们可以看到将执行操作码 04(即 FLdRfVar),就只有几行代码,也没实现什么很神奇的操作,嘿嘿。还可以看到 XOR EAX,EAX,然后就是读取后面操作码。

这里首先读取紧跟在 04 后面两个字节的参数。

```

7413D9A2 > 0FBF06      MOV SX EAX,WORD PTR DS:[ESI]
7413D9A5 . 03C5        ADD EAX,EBP
7413D9A7 . 50         PUSH EAX
7413D9A8 . 33C0        XOR EAX,EAX
7413D9AA . 8A46 02     MOV AL,BYTE PTR DS:[ESI+2]
7413D9AD . 83C6 03     ADD ESI,3
7413D9B0 . FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]

```

DS:[00401BD1]=FF74
EAX=00000004
Jumps from 7413D31F, 7413D3A9, 7413D3F6, 7413D420, 7413D442, 7413D454,

Address	Hex dump	ASCII
00401BD0	04 74 FF 21 0F 00 03 19	.*.*.*.*.*.*.*.*
00401BD8	78 FF 08 78 FF 00 A0 00	..*.*.*.*.*.*.*.*
00401BE0	00 00 6C 74 FF 1B 01 00	..*.*.*.*.*.*.*.*

通过 MOV SX 指令将 FF74(这是个负数,前面汇编章节介绍过)保存到 EAX 中,我们继续跟踪。

Registers (FPU)

EAX	FFFFFFFF74
ECX	0012FA1C
EDX	740569D0 MSUBUM50.740569D0
EBX	00401C98 clavel.00401C98
ESP	0012F3C4
EBP	0012F4E0
ESI	00401BD1 clavel.00401BD1
EDI	00159888
EIP	7413D9A5 MSUBUM50.7413D9A5
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)

EAX 的值为 -8C(十六进制),我们双击 EAX 值的话可以看到:

Modify EAX

Hexadecimal

FFFFFFFF74

Signed

-140

Unsigned

4294967156

Char

\xFF

\xFF

\xFF

t

OK

Cancel

我们可以看到 FF74 对应的十进制是 -140 也就是十六进制的 -8C。我们可以看到 ExDec 中显示的是 8C。

```

401BD0: 04 FLdRfVar      local_008C

```



```
Registers (FPU)
EAX 0012F454
ECX 0012FA1C
EDX 740569D0 MSUBUM50.7
EBX 00401C98 clave1.004
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BD1 clave1.004
EDI 00159888
EIP 7413D9A7 MSUBUM50.7
C 1 ES 0023 32bit 0xF
P 0 CS 001B 32bit 0xF
```

0012F3C0	0012F454
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000
0012F3D0	00000000
0012F3D4	00000000
0012F3D8	00000000
0012F3DC	00000000
0012F3E0	00000000
0012F3E4	00000000

Command: ? 12f4e0 - 8c HEX: 12F454 - DEC: 1242196 - ASCII:         

Registers (FPU)

EAX	0012F454
ECX	0012FA1C
EDX	740569D0 MSUBUM50.740569D0
EBX	00401C98 clave1.00401C98
ESP	0012F3C4
EBP	0012F4E0
ESI	00401BD1 clave1.00401BD1
EDI	00159888
EIP	7413D9A7 MSUBUM50.7413D9A7

C 1 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)

7413D9A2	>	0FBF06	MOVX EAX,WORD PTR DS:[ESI]								
7413D9A5	.	03C5	ADD EAX,EBP								
7413D9A7	.	50	PUSH EAX								
7413D9A8	.	33C0	XOR EAX,EAX								
7413D9AA	.	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]								
7413D9AD	.	83C6 03	ADD ESI,3								
7413D9B0	.	FF2485 94ED1	JMP DWORD PTR DS:[EAX*4+7413ED94]								
7413D9B7	>	8F45 08	MOV EAX,DWORD PTR SS:[EBP+8]								

我们可以看到通过 XOR EAX,EAX 指令将 EAX 清零了,这就意味着操作码被清零了,该操作完成了,重置寄存器的值,然后接下来一行就可以读取下一个操作码了。

Email josephco_@hotmail.com with any errors or prot

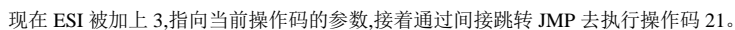
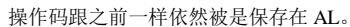
```

Proc: 401c98
401BD0: 04 FLdRfVar      local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd        text
401BD7: 19 FStAdFunc        local_0088
401BDA: 08 FLdPr             local_0088
401BDD: 0d VCallHresult    get_ipropTEXTEDIT
401BE2: 6c ILdRf            local_008C
401BE5: 1b LitStr:           ""
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str        local_008C
401BED: 1a FFree1Ad        local_0088
401BF0: 1c BranchF:          401BF6
401BF3: 1e Branch:           401c94
401BF6: Lead3/c1 LitVarl4: { local_3BE500AC } 0>
401BFE: Lead1/f6 FStVar  local_009C
401C02: 04 FLdRfVar        local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd        text
401C09: 19 FStAdFunc        local_0088

```

第二个操作码是 21,在接下来的一行读取它。

7413D9A7	. 50	PUSH EAX
7413D9A8	. 33C0	XOR EAX,EAX
7413D9AA	. 8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D9AD	. 83C6 03	ADD ESI,3
7413D9B0	. FF2485 94ED11	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9B7	> 8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA	. 8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX
7413D9BD	. 33C0	XOR EAX,EAX
7413D9BF	. 8A06	MOV AL,BYTE PTR DS:[ESI]
7413D9C1	. 46	INC ESI
7413D9C2	. FF2485 94ED11	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9C9	> 0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413D9CC	. 5B	POP EBX
7413D9CD	. 66:891C28	MOV WORD PTR DS:[EAX+EBP],BX
7413D9D1	. 33C0	XOR EAX,EAX
7413D9D3	. 8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D9D6	. 83C6 03	ADD ESI,3
7413D9D9	. FF2485 94ED11	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9E0	> 0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413D9E3	. 8F0428	POP DWORD PTR DS:[EAX+EBP]
7413D9E6	. 33C0	XOR EAX,EAX
7413D9E8	. 8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D9EB	. 83C6 03	ADD ESI,3
7413D9EE	. FF2485 94ED11	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9F5	> 0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413D9F8	. D91C28	FSTP DWORD PTR DS:[EAX+EBP]
7413D9FB	. DFE0	FSTSW AX
7413D9FD	. A8 0D	TEST AL,0D
7413D9FF	. 0F85 A1C70000	JNZ MSUBUM50.7414A1A6
7413DA05	. 33C0	XOR EAX,EAX
7413DA07	. 8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413DA0A	. 83C6 03	ADD ESI,3
7413DA0D	. FF2485 94ED11	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413DA14	> 8B3C24	MOV EDI,DWORD PTR SS:[ESP]
7413DA17	> 66:8B07	MOV AX,WORD PTR DS:[EDI]
7413DA1A	> 80E4 BF	AND AH,0BF
7413DA1D	. 66:83F8 09	CMP AX,9
7413DA21	. 0F85 C06A0000	JNZ MSUBUM50.741444E7
7413DA27	> 33C0	XOR EAX,EAX
7413DA29	. 8A06	MOV AL,BYTE PTR DS:[ESI]
DS:[00401BD3]=21 ('*')		
AL=00		
Address	Hex dump	ASCII
		0012F3C0
		0012F3C1



我们来 Google 一下它的含义。

'21, FLdPrThis

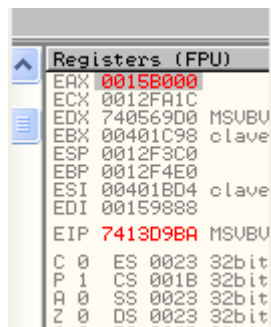
```
FLdPrThis      : [SR]=[stack2]
```

好,这里我们可以看到有些前辈做了注释,虽然我们不知道它具体是干什么用的,但是根据字面的意思来理解就是加载一个指针,并且指向一个数据项。

我们继续往下跟。

这里是将 EBP+8 指向内存单元的内容读取出来并保存到 EBP-4C 指向的内存单元中。

这个值在我的机器上是 15B000.我们在数据窗口中定位到这个地址。



Address	Hex dump	ASCII
0015B000	E8 22 40 00 00 00 00 00	b"0.....
0015B008	1C B0 15 00 7C B1 CF 00	..S.!
0015B010	0C B1 CF 00 00 00 00 00	..S.!
0015B018	00 00 00 00 60 A8 06 74t
0015B020	04 00 00 00 05 00 00 00t
0015B028	00 00 00 00 0F 00 00 00*
0015B030	00 00 00 00 E0 18 40 00t
0015B038	74 19 40 00 D0 19 40 00	t..t..t..t
0015B040	30 1A 40 00 90 1A 40 00	t..t..t..t
0015B048	00 00 00 00 00 00 00 00
0015B050	AB AB AB AB AB AB AB AB
0015B058	00 00 00 00 00 00 00 00
0015B060	07 00 00 00 ED 07 18 00t
0015B068	00 00 00 00 7C B8 CF 00t
0015B070	68 BD 15 00 70 AE 15 00	h..S..p..S..
0015B078	00 00 00 00 00 00 00 00
0015B080	00 F0 AD BA 00 F0 AD BA	..- ..-
0015B088	AB AB AB AB AB AB AB AB
0015B090	00 00 00 00 00 00 00 00
0015B098	19 00 07 00 F2 07 18 00	..- ..-

该地址中保存的是 4022E8,我们继续在数据窗口中定位到 4022E8。

Address	Hex dump	ASCII
004022C8	00 00 00 00 00 00 00 00
004022D0	00 00 00 00 00 00 00 00
004022D8	00 00 00 00 00 00 00 00
004022E0	00 00 00 00 6C 17 40 00t
004022E8	08 4C 05 74 FE 25 04 74	..L..t..t..t
004022F0	6F DC 05 74 C1 F8 10 74	..t..t..t..t
004022F8	D0 F8 10 74 A8 40 09 74	..t..t..t..t
00402300	92 47 09 74 80 BD 14 74	..t..t..t..t
00402308	88 BD 14 74 90 BD 14 74	..t..t..t..t
00402310	98 BD 14 74 A0 BD 14 74	..t..t..t..t
00402318	A8 BD 14 74 B0 BD 14 74	..t..t..t..t
00402320	B8 BD 14 74 C0 BD 14 74	..t..t..t..t
00402328	C8 BD 14 74 D0 BD 14 74	..t..t..t..t
00402330	D8 BD 14 74 E0 BD 14 74	..t..t..t..t
00402338	E8 BD 14 74 F0 BD 14 74	..t..t..t..t
00402340	F8 BD 14 74 00 BE 14 74	..t..t..t..t
00402348	08 BE 14 74 10 BE 14 74	..t..t..t..t
00402350	18 BE 14 74 20 BE 14 74	..t..t..t..t
00402358	28 BE 14 74 30 BE 14 74	..t..t..t..t
00402360	38 BE 14 74 40 BE 14 74	..t..t..t..t

这里我们可以推断出 15B000 其实是一个指针。该指针指向了一张表,虽然对我们的破解起不到什么实质性的帮助,但起码我们还是看出一点门道了。

还有一点就是可以看出该操作码没有参数。

我们继续跟。

Address	Hex dump	Disassembly
7413D9B0	FF2485 94ED1	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9B7	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX
7413D9BD	33C0	XOR EAX,EAX
7413D9BF	8A06	MOV AL,BYTE PTR DS:[ESI]
7413D9C1	46	INC ESI
7413D9C2	FF2485 94ED1	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9C9	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]

这里 EAX 又被清零了,下一行读取第三个操作码。

Address	Hex dump	ASCII
00401BCC	D8 22 40 00 04 74 FF 21	i"0.♦t †
00401BD4	0F 00 03 19 78 FF 08 78	*.♦↓x □x
00401BDC	FF 0D A0 00 00 00 6C 74	.ā...lt
00401BE4	FF 1B 01 00 FB 30 2F 74	+0.'0/t
00401BEC	FF 1A 78 FF 1C 26 00 1E	+x L&.▲
00401BF4	C4 00 FE C1 54 FF 1A C4	-..±T +
00401BFC	03 00 FC F6 64 FF 04 74	♦.±d ♦t
00401C04	FF 21 0F 00 03 19 78 FF	†*.♦↓x
00401C0C	08 78 FF 3D A0 00 00 00	□x .ā...
00401C14	6C 74 FF 07 02 00 04 00	lt .0.♦.
00401C1C	FD 68 54 FF 5D 04 64 FF	²kT J♦d
00401C24	FB 40 2F 74 FF 1A 78 FF	'0/t +x
00401C2C	1C 93 00 27 E1 FE 27 04	L0.'&±'♦
00401C34	FF 3A 34 FF 03 00 4E 24	:4 ♦.N\$
00401C3C	FF 04 24 FF F5 40 00 00	♦\$ S0..
00401C44	00 3A 54 FF 04 00 4E 44	.:T ♦.ND
00401C4C	FF 04 44 FF 0A 05 00 14	♦D .±.¶
00401C54	00 36 08 00 44 FF 24 FF	.C0.D \$
00401C5C	04 FF E4 FE 1E C4 00 27	♦ &±▲-.'
00401C64	E4 FE 27 04 FF 3A 34 FF	&±'♦ :4
00401C6C	03 00 4E 24 FF 04 24 FF	*.N\$ ±\$

Command: ? ebp-4c HEX: 12F4

从 ExDec 中我们可以看出该操作码是 0F。

Proc: 401c98	
401BD0: 04 FLdPrVar	local_008C
401BD3: 21 FLdPrThis	
401BD4: 0F VCallAd	text
401BD7: 19 FStAdFunc	local_0088
401BDA: 08 FLdPr	local_0088
401BDD: 0d VCallHresult	get_ipropTEXTEDIT
401BE2: 6c ILdRf	local_008C
401BE5: 1b LitStr:	"
401BE8: Lead0/30 EaStr	

VcallAd

'0F, VCallAd, FC, 02
'Access an item's method.
'Parameter 1 = 2 bytes.
'Parameter 1 is offset into item's Descriptor table.
'Offset = &h2FC.
'Method at offset in item's Descriptor table is accessed.
'Method's return value is pushed onto stack.
'Stack operations: Method dependent + Push x1.

以上是 0F 这个操作码具体的解释,我们可以看到它有一个占两个字节的参数。

Address	Hex dump	ASCII
00401BCC	D8 22 40 00 04 74 FF 21	i"0.♦t †
00401BD4	0F 00 03 19 78 FF 08 78	*.♦↓x □x
00401BDC	FF 0D A0 00 00 00 6C 74	.ā...lt
00401BE4	FF 1B 01 00 FB 30 2F 74	+0.'0/t
00401BEC	FF 1A 78 FF 1C 26 00 1E	+x L&.▲
00401BF4	C4 00 FE C1 54 FF 1A C4	-..±T +
00401BFC	03 00 FC F6 64 FF 04 74	♦.±d ♦t

该参数我这里显示的 0300,其表示句柄表中数据元素的偏移。接下来是一个间接跳转 JMP,我们跟进去看看。

Address	Hex dump	Disassembly
7413E89B	> 8B5D B4	MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E	53	PUSH EBX
7413E89F	0FB706	MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2	0303	ADD EAX,DWORD PTR DS:[EBX]
7413E8A4	FF10	CALL DWORD PTR DS:[EAX]
7413E8A6	50	PUSH EAX
7413E8A7	33C0	XOR EAX,EAX
7413E8A9	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413E8AC	83C6 03	ADD ESI,3
7413E8AF	FF2485 94ED1	JMP DWORD PTR DS:[EAX*4+7413ED94]

又是读取 EBP - 4C 的内容,保存到 EBX 中。

Registers (FPU)	
EAX	0000000F
ECX	0012FA1C
EDX	740569D0 MSUBUM
EBX	0015B000
ESP	0012F3C0
EBP	0012F4E0
ESI	00401B05 clave1
EDI	00159888
EIP	7413E89E MSUBUM
C 0	ES 0023 32bit
P 0	CS 001B 32bit
A 0	SS 0023 32bit
Z 0	DS 0023 32bit

这里我们可以看到是 15B000,并使用 PUSH 指令压入到堆栈中。

0012F3BC	0015B000
0012F3C0	0012F454
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000
0012F3D0	00000000
0012F3D4	00000000

7413E89B	> 8B5D B4	MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E	. 53	PUSH EBX
7413E89F	. 0FB706	MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2	. 0303	ADD EAX,DWORD PTR DS:[EBX]
7413E8A4	. FF10	CALL DWORD PTR DS:[EAX]
7413E8A6	. 50	PUSH EAX
7413E8A7	. 33C0	XOR EAX,EAX
7413E8A9	. 8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413E8AB	. 83C6 03	ADD ESI,3
7413E8AF	. FF2485 94ED1	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E8B4	. 8EBC3C	MOV EBT,WORD PTR DS:[ESI]

接着是将参数值 300 保存到 EAX 中。

Registers (FPU)	
EAX	00000300
ECX	0012FA1C
EDX	740569D0 MSUB
EBX	0015B000
ESP	0012F3BC
EBP	0012F4E0
ESI	00401B05 clav
EDI	00159888
EIP	7413E8A2 MSUB
C 0	ES 0023 32bit

7413E89B	> 8B5D B4	MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E	. 53	PUSH EBX
7413E89F	. 0FB706	MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2	. 0303	ADD EAX,DWORD PTR DS:[EBX]
7413E8A4	. FF10	CALL DWORD PTR DS:[EAX]
7413E8A6	. 50	PUSH EAX
7413E8A7	. 33C0	XOR EAX,EAX
7413E8A9	. 8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413E8AB	. 83C6 03	ADD ESI,3
7413E8AF	. FF2485 94ED1	JMP DWORD PTR DS:[EAX*4+7413ED94]

这里 EBX 的值为 15B000(我们已经知道了它指向了一张表),该表起始地址为 4022E8,我们姑且将这张表称之为 Description Item Table。

Address	Hex dump	ASCII
004018CC	D8 22 40 00 04 74 FF 21	! " \$ % & ' () * + , - . / : ;
004018CD	00 00 00 00 00 00 00 00
004018CE	00 00 00 00 00 00 00 00
004018CF	00 00 00 00 00 00 00 00
004018D0	00 00 00 00 00 00 00 00
004018D1	00 00 00 00 00 00 00 00
004018D2	00 00 00 00 00 00 00 00
004018D3	00 00 00 00 00 00 00 00
004018D4	00 00 00 00 00 00 00 00
004018D5	00 00 00 00 00 00 00 00
004018D6	00 00 00 00 00 00 00 00
004018D7	00 00 00 00 00 00 00 00
004018D8	00 00 00 00 00 00 00 00
004018D9	00 00 00 00 00 00 00 00
004018DA	00 00 00 00 00 00 00 00
004018DB	00 00 00 00 00 00 00 00
004018DC	00 00 00 00 00 00 00 00
004018DD	00 00 00 00 00 00 00 00
004018DE	00 00 00 00 00 00 00 00
004018DF	00 00 00 00 00 00 00 00
004018E0	00 00 00 00 00 00 00 00
004018E1	00 00 00 00 00 00 00 00
004018E2	00 00 00 00 00 00 00 00
004018E3	00 00 00 00 00 00 00 00
004018E4	00 00 00 00 00 00 00 00
004018E5	00 00 00 00 00 00 00 00
004018E6	00 00 00 00 00 00 00 00
004018E7	00 00 00 00 00 00 00 00
004018E8	00 00 00 00 00 00 00 00
004018E9	00 00 00 00 00 00 00 00
004018EA	00 00 00 00 00 00 00 00
004018EB	00 00 00 00 00 00 00 00
004018EC	00 00 00 00 00 00 00 00
004018ED	00 00 00 00 00 00 00 00
004018EE	00 00 00 00 00 00 00 00
004018EF	00 00 00 00 00 00 00 00
004018F0	00 00 00 00 00 00 00 00
004018F1	00 00 00 00 00 00 00 00
004018F2	00 00 00 00 00 00 00 00
004018F3	00 00 00 00 00 00 00 00
004018F4	00 00 00 00 00 00 00 00
004018F5	00 00 00 00 00 00 00 00
004018F6	00 00 00 00 00 00 00 00
004018F7	00 00 00 00 00 00 00 00
004018F8	00 00 00 00 00 00 00 00
004018F9	00 00 00 00 00 00 00 00
004018FA	00 00 00 00 00 00 00 00
004018FB	00 00 00 00 00 00 00 00
004018FC	00 00 00 00 00 00 00 00
004018FD	00 00 00 00 00 00 00 00
004018FE	00 00 00 00 00 00 00 00
004018FF	00 00 00 00 00 00 00 00

这里由该表的起始地址偏移 300。

Address	Hex dump	ASCII
004022D8	00 00 00 00 00 00 00 00
004022E0	00 00 00 00 00 00 00 00
004022E8	08 4C 05 74 FE 25 04 74
004022F0	6F DC 05 74 C1 F8 10 74
004022F8	D0 F8 10 74 A8 40 09 74
00402300	92 47 09 74 80 BD 14 74
00402308	88 BD 14 74 90 BD 14 74
00402310	98 BD 14 74 A0 BD 14 74
00402318	A8 BD 14 74 B0 BD 14 74
00402320	B8 BD 14 74 C0 BD 14 74
00402328	C8 BD 14 74 D0 BD 14 74
00402330	D8 BD 14 74 E0 BD 14 74
00402338	E8 BD 14 74 F0 BD 14 74
00402340	F8 BD 14 74 00 BE 14 74
00402348	08 BE 14 74 10 BE 14 74
00402350	18 BE 14 74 20 BE 14 74
00402358	28 BE 14 74 30 BE 14 74
00402360	38 BE 14 74 40 BE 14 74
00402368	48 BE 14 74 50 BE 14 74
00402370	58 BE 14 74 60 BE 14 74
00402378	68 BE 14 74 70 BE 14 74
00402380	78 BE 14 74 80 BE 14 74
00402388	88 BE 14 74 90 BE 14 74
00402390	98 BE 14 74 A0 BE 14 74
00402398	A8 BE 14 74 B0 BE 14 74
004023A0	B8 BE 14 74 C0 BE 14 74
004023A8	C8 BE 14 74 D0 BE 14 74
004023B0	D8 BE 14 74 E0 BE 14 74
004023B8	E8 BE 14 74 F0 BE 14 74
004023C0	F8 BE 14 74 00 BF 14 74
004023C8	08 BF 14 74 10 BF 14 74
004023D0	18 BF 14 74 20 BF 14 74
004023D8	28 BF 14 74 30 BF 14 74
004023E0	38 BF 14 74 40 BF 14 74
004023E8	48 BF 14 74 50 BF 14 74
004023F0	58 BF 14 74 60 BF 14 74
004023F8	68 BF 14 74 70 BF 14 74
00402400	78 BF 14 74 80 BF 14 74
00402408	88 BF 14 74 90 BF 14 74
00402410	98 BF 14 74 A0 BF 14 74
00402418	A8 BF 14 74 B0 BF 14 74
00402420	B8 BF 14 74 C0 BF 14 74
00402428	C8 BF 14 74 D0 BF 14 74
00402430	D8 BF 14 74 E0 BF 14 74
00402438	E8 BF 14 74 F0 BF 14 74
00402440	F8 BF 14 74 00 C0 14 74
00402448	08 C0 14 74 10 C0 14 74
00402450	18 C0 14 74 20 C0 14 74
00402458	28 C0 14 74 30 C0 14 74
00402460	38 C0 14 74 40 C0 14 74
00402468	48 C0 14 74 50 C0 14 74
00402470	58 C0 14 74 60 C0 14 74
00402478	68 C0 14 74 70 C0 14 74
00402480	78 C0 14 74 80 C0 14 74
00402488	88 C0 14 74 90 C0 14 74
00402490	98 C0 14 74 A0 C0 14 74
00402498	A8 C0 14 74 B0 C0 14 74
004024A0	B8 C0 14 74 C0 C0 14 74
004024A8	C8 C0 14 74 D0 C0 14 74
004024B0	D8 C0 14 74 E0 C0 14 74
004024B8	E8 C0 14 74 F0 C0 14 74
004024C0	F8 C0 14 74 00 C1 14 74
004024C8	08 C1 14 74 10 C1 14 74
004024D0	18 C1 14 74 20 C1 14 74
004024D8	28 C1 14 74 30 C1 14 74
004024E0	38 C1 14 74 40 C1 14 74
004024E8	48 C1 14 74 50 C1 14 74
004024F0	58 C1 14 74 60 C1 14 74
004024F8	68 C1 14 74 70 C1 14 74
00402500	78 C1 14 74 80 C1 14 74
00402508	88 C1 14 74 90 C1 14 74
00402510	98 C1 14 74 A0 C1 14 74
00402518	A8 C1 14 74 B0 C1 14 74
00402520	B8 C1 14 74 C0 C1 14 74
00402528	C8 C1 14 74 D0 C1 14 74
00402530	D8 C1 14 74 E0 C1 14 74
00402538	E8 C1 14 74 F0 C1 14 74
00402540	F8 C1 14 74 00 C2 14 74
00402548	08 C2 14 74 10 C2 14 74
00402550	18 C2 14 74 20 C2 14 74
00402558	28 C2 14 74 30 C2 14 74
00402560	38 C2 14 74 40 C2 14 74
00402568	48 C2 14 74 50 C2 14 74
00402570	58 C2 14 74 60 C2 14 74
00402578	68 C2 14 74 70 C2 14 74
00402580	78 C2 14 74 80 C2 14 74
00402588	88 C2 14 74 90 C2 14 74
00402590	98 C2 14 74 A0 C2 14 74
00402598	A8 C2 14 74 B0 C2 14 74
004025A0	B8 C2 14 74 C0 C2 14 74
004025A8	C8 C2 14 74 D0 C2 14 74
004025B0	D8 C2 14 74 E0 C2 14 74
004025B8	E8 C2 14 74 F0 C2 14 74
004025C0	F8 C2 14 74 00 C3 14 74
004025C8	08 C3 14 74 10 C3 14 74
004025D0	18 C3 14 74 20 C3 14 74
004025D8	28 C3 14 74 30 C3 14 74
004025E0	38 C3 14 74 40 C3 14 74
004025E8	48 C3 14 74 50 C3 14 74
004025F0	58 C3 14 74 60 C3 14 74
004025F8	68 C3 14 74 70 C3 14 74
00402600	78 C3 14 74 80 C3 14 74
00402608	88 C3 14 74 90 C3 14 74
00402610	98 C3 14 74 A0 C3 14 74
00402618	A8 C3 14 74 B0 C3 14 74
00402620	B8 C3 14 74 C0 C3 14 74
00402628	C8 C3 14 74 D0 C3 14 74
00402630	D8 C3 14 74 E0 C3 14 74
00402638	E8 C3 14 74 F0 C3 14 74
00402640	F8 C3 14 74 00 C4 14 74
00402648	08 C4 14 74 10 C4 14 74
00402650	18 C4 14 74 20 C4 14 74
00402658	28 C4 14 74 30 C4 14 74
00402660	38 C4 14 74 40 C4 14 74
00402668	48 C4 14 74 50 C4 14 74
00402670	58 C4 14 74 60 C4 14 74
00402678	68 C4 14 74 70 C4 14 74
00402680	78 C4 14 74 80 C4 14 74
00402688	88 C4 14 74 90 C4 14 74
00402690	98 C4 14 74 A0 C4 14 74
00402698	A8 C4 14 74 B0 C4 14 74
004026A0	B8 C4 14 74 C0 C4 14 74
004026A8	C8 C4 14 74 D0 C4 14 74
004026B0	D8 C4 14 74 E0 C4 14 74
004026B8	E8 C4 14 74 F0 C4 14 74
004026C0	F8 C4 14 74 00 C5 14 74
004026C8	08 C5 14 74 10 C5 14 74
004026D0	18 C5 14 74 20 C5 14 74
004026D8	28 C5 14 74 30 C5 14 74
004026E0	38 C5 14 74 40 C5 14 74
004026E8	48 C5 14 74 50 C5 14 74
004026F0	58 C5 14 74 60 C5 14 74
004026F8	68 C5 14 74 70 C5 14 74
00402700	78 C5 14 74 80 C5 14 74
00402708	88 C5 14 74 90 C5 14 74
00402710	98 C5 14 74 A0 C5 14 74
00402718	A8 C5 14 74 B0 C5 14 74
00402720	B8 C5 14 74 C0 C5 14 74
00402728	C8 C5 14 74 D0 C5 14 74
00402730	D8 C5 14 74 E0 C5 14 74
00402738	E8 C5 14 74 F0 C5 14 74
00402740	F8 C5 14 74 00 C6 14 74
00402748	08 C6 14 74 10 C6 14 74
00402750	18 C6 14 74 20 C6 14 74
00402758	28 C6 14 74 30 C6 14 74
00402760	38 C6 14 74 40 C6 14 74
00402768	48 C6 14 74 50 C6 14 74
00402770	58 C6 14 74 60 C6 14 74
00402778	68 C6 14 74 70 C6 14 74
00402780	78 C6 14 74 80 C6 14 74
00402788	88 C6 14 74 90 C6 14 74
00402790	98 C6 14 74 A0 C6 14 74
00402798	A8 C6 14 74 B0 C6 14 74
004027A0	B8 C6 14 74 C0 C6 14 74
004027A8	C8 C6 14 74 D0 C6 14 74
004027B0	D8 C6 14 74 E0 C6 14 74
004027B8	E8 C6 14 74 F0 C6 14 74
004027C0	F8 C6 14 74 00 C7 14 74
004027C8	08 C7 14 74 10 C7 14 74
004027D0	18 C7 14 74 20 C7 14 74
004027D8	28 C7 14 74 30 C7 14 74
004027E0	38 C7 14 74 40 C7 14 74
004027E8	48 C7 14 74 50 C7 14 74
004027F0	58 C7 14 74 60 C7 14 74
004027F8	68 C7 14 74 70 C7 14 74
00402800	78 C7 14 74 80 C7 14 74
00402808	88 C7 14 74 90 C7 14 74
00402810	98 C7 14 74 A0 C7 14 74
00402818	A8 C7 14 74 B0 C7 14 74
00402820	B8 C7 14 74 C0 C7 14 74
00402828	C8 C7 14 74 D0 C7 14 74
00402830	D8 C	

我们可以看到堆栈中保存了结果,我们需要弄明白它表示什么意思。

Proc: 401c98	
401BD0: 04 FLdRfVar	local_008C
401BD3: 21 FLdPrThis	
401BD4: 0f VCallAd	text
401BD7: 19 FStAdFunc	local_0088
401BDA: 08 FLdPr	local_0088
401BDD: 0d VCallHresult	get__ipropTEXTEDIT

接下来的操作码是 19,参数值是 88,代表一个局部变量。

7413E89F	0FB706	MOVZX EAX,WORD PTR DS:[ESI]	
7413E8A2	0303	ADD EAX,DWORD PTR DS:[EBX]	
7413E8A4	FF10	CALL DWORD PTR DS:[EAX]	
7413E8A6	50	PUSH EAX	
7413E8A7	33C0	XOR EAX,EAX	
7413E8A9	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413E8AC	83C6 03	ADD ESI,3	
7413E8AF	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	MSUBUM50.7413E50A
7413E8B6	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]	
7413E8B9	0FBF46 02	MOVSX EAX,WORD PTR DS:[ESI+2]	
7413E8BD	83C6 04	ADD ESI,4	

这里我们直接跟进。

7413E50A	BB FFFFFFFF	MOV EBX,-1	
7413E50F	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E512	83C6 02	ADD ESI,2	
7413E515	03C5	ADD EAX,EBP	
7413E517	59	POP ECX	
7413E518	53	PUSH EBX	
7413E519	50	PUSH EAX	
7413E51A	51	PUSH ECX	
7413E51B	E8 301E0000	CALL MSUBUM50.74140350	
7413E520	33C0	XOR EAX,EAX	
7413E522	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E524	46	INC ESI	
7413E525	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	

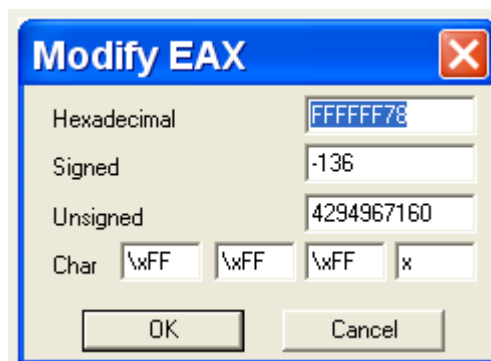
我们看到这里。

Address	Hex dump	ASCII
00401BD0	04 74 FF 21 0F 00 03 19	!t !*.!↓
00401BD8	78 FF 08 78 FF 0D A0 00	x 8x .a.
00401BE0	00 00 6C 74 FF 1B 01 00	..lt +0.
00401BE8	FB 30 2F 74 FF 1A 78 FF	'0/t +x.

通过 MOVSX 指令读取出占两个字节的参数值,将其保存到 EAX 中。FF 开头表明该参数值是一个负数。

Registers (FPU)	
EAX	FFFFFFFF78
ECX	0012FA1C
EDX	00CF4BA0
EBX	FFFFFFFF
ESP	0012F38C
EBP	0012F4E0
ESI	00401BD8 clav
EDI	00159888
EIP	7413E512 MSUE
C 0	ES 0023 32bi
D 4	CS 001D 00bi

该值对应的十六进制为-88,跟 ExDec 中显示的刚好对应起来了。



十进制的-139 正好等于十六进制的-88。

7413E50A	BB FFFFFFFF	MOV EBX,-1	
7413E50F	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E512	83C6 02	ADD ESI,2	
7413E515	03C5	ADD EAX,EBP	
7413E517	59	POP ECX	
7413E518	53	PUSH EBX	
7413E519	50	PUSH EAX	
7413E51A	51	PUSH ECX	
7413E51B	E8 301E0000	CALL MSUBUM50.74140350	
7413E520	33C0	XOR EAX,EAX	
7413E522	8A06	MOV AL,BYTE PTR DS:[ESI]	

接着 ESI 加 2,然后刚刚计算出的-88 加上 EBP 的值,即将 EBP - 88 保存到 EAX 中。

Registers (FPU)			
EAX	0012F458		
ECX	0012FA1C		
EDX	00CF4BA0		
EBX	FFFFFFFF		
ESP	0012F3BC		
EBP	0012F4E0		
ESI	00401BDA	cl	
EDI	00159888		
EIP	7413E517	MSU	
C 1	ES 0023	32b	
D 0	FS 001B	33b	

7413E50A	BB FFFFFFFF	MOV EBX,-1	
7413E50F	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E512	83C6 02	ADD ESI,2	
7413E515	03C5	ADD EAX,EBP	
7413E517	59	POP ECX	
7413E518	53	PUSH EBX	
7413E519	50	PUSH EAX	
7413E51A	51	PUSH ECX	
7413E51B	E8 301E0000	CALL MSUBUM50.74140350	
7413E520	33C0	XOR EAX,EAX	

这里我们可以看到到达了一个 CALL 处,根据堆栈的来看其有三个参数。

0012F3B4	00CFBCA4	Arg1 = 00CFBCA4
0012F3B8	0012F458	Arg2 = 0012F458
0012F3BC	FFFFFFFF	Arg3 = FFFFFFFF
0012F3C0	0012F454	
0012F3C4	00000000	

第一个参数是前一个操作码执行的结果,第二个参数我这里是 12F458,即 EBP - 88-表示一个局部变量。第三个参数是-1。这里我们不跟进这个 CALL,直接按 F8 键单步步过这个 CALL,看看会发生什么。

Address	Hex dump	ASCII
0012F458	A4 BC CF 00 EC F4 12 00	â.â.
0012F460	C8 F5 12 00 3C BB CF 00	.â.
0012F468	F8 F4 12 00 FF FF FF FF	b.â.

我们会发现堆栈发生了变化,ECX 被清零了。

EBP-88 内存单元保存了前一个操作码执行的结果。

```

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd             text
401BD7: 19 FStAdFunc            local_0088
401BDA: 08 FLdPr                local_0088
401BDD: 0d VCallHresult        get__ipropTEXTEDIT
401BE2: 6c ILdRf                local_008C
401BE5: 1b LitStr:              ""
401BF8: 1 Lead0/30 FnStr

```

接下来一个操作码是 08,它也将局部变量 EBP - 88 作为参数。

Address	Hex dump	Disassembly	Comment
7413E512	83C6 02	ADD ESI,2	
7413E515	03C5	ADD EAX,EBP	
7413E517	59	POP ECX	
7413E518	53	PUSH EBX	
7413E519	50	PUSH EAX	
7413E51A	51	PUSH ECX	
7413E51B	E8 301E0000	CALL MSUBUM50.74140350	
7413E520	33C0	XOR EAX,EAX	
7413E522	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E524	46	INC ESI	
7413E525	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	MSUBUM50.7413E3D0
7413E52C	FF3424	PUSH DWORD PTR SS:[ESP]	
7413E52E	CC	POP EAX	

我们跟进这个 JMP。

Address	Hex dump	Disassembly	Comment
7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413E3D3	83C6 02	ADD ESI,2	
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]	
7413E3D9	0BC0	OR EAX,EAX	
7413E3DB	0F84 34760000	JE MSUBUM50.74145A15	
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX	
7413E3E4	33C0	XOR EAX,EAX	
7413E3E6	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E3E8	46	INC ESI	
7413E3E9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413E3F0	8B7D B4	MOV EDI,DWORD PTR SS:[EBP-4C]	

这下面并不是我们之前看到的 XOR EAX,EAX 结束,而是 OR EAX,EAX,接着使用条件跳转判断 EAX 是否为零。我们来看看它具体干了些什么。

首先将操作码的参数 FF78 保存到 EAX 中,注意这里使用的是 MOVSX,FF 开头表示是负数,十六进制值为-88。

Address	Hex dump	ASCII
00401BD3	21 0F 00 03 19 78 FF 08	!*.~x
00401BD8	78 FF 00 A0 00 00 00 6C	x .ä...l
00401BE3	1B 01 00 FB 30 2F	t +0.'0/
00401BE8	74 FF 1A 78 FF 1C 26 00	t +x L&.
00401BF3	1E C4 00 FE C1 54 FF 1A	▲-.-T +

Registers (FPU)		
EAX	FFFFFF78	
ECX	00000000	
EDX	00CF4BA0	
EBX	FFFFFFFF	
ESP	0012F3C0	
EBP	0012F4E0	
ESI	00401BDB	clave1
EDI	00159888	
EIP	7413E3D3	MSUBUM
C 0	ES 0023	32bit
P 1	CS 001B	32bit

7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413E3D3	83C6 02	ADD ESI,2
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]
7413E3D9	0BC0	OR EAX,EAX
7413E3DB	0F84 34760000	JE MSUBUM\$0.74145A15
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX
7413E3E4	33C0	XOR EAX,EAX
7413E3E6	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E3E8	46	INC ESI

这一行是将 EAX + EBP 指向内存单元的值保存到 EAX 中,即 EBP - 88 这个局部变量的值。

Registers (FPU)		
EAX	00CFBCA4	
ECX	00000000	
EDX	00CF4BA0	
EBX	FFFFFFFF	
ESP	0012F3C0	
EBP	0012F4E0	
ESI	00401BDD	cl
EDI	00159888	
EIP	7413E3D9	MS
C 0	ES 0023	32
D 1	CS 001B	32

7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413E3D3	83C6 02	ADD ESI,2
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]
7413E3D9	0BC0	OR EAX,EAX
7413E3DB	0F84 34760000	JE MSUBUM\$0.74145A15
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX
7413E3E4	33C0	XOR EAX,EAX
7413E3E6	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E3E8	46	INC ESI
7413E3E9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E3F0	8B7D B4	MOV EDI,DWORD PTR SS:[EBP-4C]
7413E3F0	8B7D B4	MOV EDI,DWORD PTR DS:[EBP-4C]

这里判断 EAX 是否为零,如果为零就跳转到 74145A15 地址处。如果不为零就继续往下执行。

7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413E3D3	83C6 02	ADD ESI,2
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]
7413E3D9	0BC0	OR EAX,EAX
7413E3DB	0F84 34760000	JE MSUBUM\$0.74145A15
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX
7413E3E4	33C0	XOR EAX,EAX

这里将 EAX 的值保存到 EBP - 4C 中。

我们应该还记得之前读取 EBP + 8 的内容,接着将其保存到 EBP - 4C 中。所以说 EBP - 4C 的值不为零。

所以我们将 EBP - 4C 称为指针数据元素。

7413E3D0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413E3D3	83C6 02	ADD ESI,2
7413E3D6	8B0428	MOV EAX,DWORD PTR DS:[EAX+EBP]
7413E3D9	0BC0	OR EAX,EAX
7413E3DB	0F84 34760000	JE MSUBUM\$0.74145A15
7413E3E1	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX
7413E3E4	33C0	XOR EAX,EAX
7413E3E6	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E3E8	46	INC ESI
7413E3E9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E3F0	8B7D B4	MOV EDI,DWORD PTR SS:[EBP-4C]

接下来是下一个操作码。

Proc: 401c98
401BD0: 04 FLdRfVar local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd text
401BD7: 19 FStAdFunc local_0088
401BDA: 08 FLdPr local_0088
401BDD: 0d VCallHresult get_ipropTEXTEDIT
401BE2: 6c iLdRf local_008C
401BE5: 1b LitStr: "
401BE8: 1ad0/30 FcStr

7413E3E6 8A06 MOV AL, BYTE PTR DS:[ESI]
7413E3E8 46 INC ESI
7413E3E9 FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94] MSUBUM50.7413E829
7413E3F0 8B7D B4 MOV EDI, DWORD PTR SS:[EBP-4C]
7413E3F3 0FB706 MOVZX EAX, WORD PTR DS:[ESI]

这里我们来 Google 一下这个操作码 0d VCallHresult。

表示获取文本框中输入的文本。

这里将读取我们输入的错误序列号,我们继续跟,看看是不是这样。

File View Debug Plugins Options Window Help

7413E829 8B45 B4 MOV EAX, DWORD PTR SS:[EBP-4C]
7413E82C 50 PUSH EAX
7413E82D 0FB73E MOVZX EDI, WORD PTR DS:[ESI]
7413E830 8B00 MOV EAX, DWORD PTR DS:[EAX]
7413E832 03C7 ADD EAX, EDI
7413E834 FF10 CALL DWORD PTR DS:[EAX]
7413E836 8B55 BC MOV EDX, DWORD PTR SS:[EBP-44]
7413E839 66:F742 76 0201 TEST WORD PTR DS:[EDX+76], 2
7413E83F 75 13 JNZ SHORT MSUBUM50.7413E854
7413E841 0BC0 OR EAX, EAX
7413E843 78 19 JS SHORT MSUBUM50.7413E85E
7413E845 33C0 XOR EAX, EAX
7413E847 8A46 04 MOV AL, BYTE PTR DS:[ESI+4]
7413E849 83C6 05 ADD ESI, 5

我们可以看到该操作码跟之前一样还是以 XOR EAX,EAX 结束。

首先读取 EBP - 4C 的内容(指向数据项的指针)保存到 EAX 中。

Registers (FPU)	
EAX	00CFBCA4
ECX	00000000
EDX	00CF4BA0
EBX	FFFFFFFF
ESP	0012F3C0
EBP	0012F4E0
ESI	00401BDE c:\ave
EDI	00159888
EIP	7413E82C MSUBUM
C 0	ES 0023 32bit
P 1	CS 001B 32bit

接下来将这个值压入堆栈。

0012F3BC	00CFBCA4
0012F3C0	0012F454
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000
0012F3D0	00000000
0012F3D4	00000000

接着读取操作码的参数。

Assembly List:

7413E829	8B45 B4	MOV EAX,DWORD PTR SS:[EBP-4C]
7413E82C	50	PUSH EAX
7413E82D	0FB73E	MOVZXL EDI,WORD PTR DS:[ESI]
7413E830	8B00	MOV EAX,DWORD PTR DS:[EAX]
7413E832	03C7	ADD EAX,EDI
7413E834	FF10	CALL DWORD PTR DS:[EAX]
7413E836	8B55 BC	MOV EDX,DWORD PTR SS:[EBP-44]
7413E839	66:F742 76 0201	TEST WORD PTR DS:[EDX+76].2

Address	Hex dump	ASCII
00401BD6	03 19 78 FF 08 78 FF 0D	␣␣␣␣␣␣␣␣␣␣
00401BDE	A0 00 00 00 6C 74 FF 1B	␣␣␣␣␣␣␣␣␣␣
00401BE6	01 00 FB 30 2F 74 FF 1A	␣␣␣␣␣␣␣␣␣␣
00401BEE	78 FF 1C 26 00 1E C4 00	␣␣␣␣␣␣␣␣␣␣
00401BF6	FE C1 54 FF 1A C4 03 00	␣␣␣␣␣␣␣␣␣␣

Registers (FPU):

EAX	00CFBCA4
ECX	00000000
EDX	00CF4BA0
EBX	FFFFFFFF
ESP	0012F3BC
EBP	0012F4E0
ESI	00401BDE clc
EDI	000000A0
EIP	7413E830 MSU

这里将参数值保存到 EDI 中。

然后读取 EAX 指向的内存单元的内容,这是另一个表的起始内容。

Registers (FPU):

EAX	00CF4BA0
ECX	00000000
EDX	00CF4BA0
EBX	FFFFFFFF
ESP	0012F3BC
EBP	0012F4E0
ESI	00401BDE clc
EDI	000000A0
EIP	7413E832 MSU

Address	Hex dump	ASCII
00CF4B88	AB AB AB AB EE FE EE FE	␣␣␣␣␣␣␣␣␣␣
00CF4B90	00 00 00 00 00 00 00 00	␣␣␣␣␣␣␣␣␣␣
00CF4B98	49 00 0F 00 41 07 18 00	␣␣␣␣␣␣␣␣␣␣
00CF4BA0	95 5A 06 74 A2 E2 05 74	␣␣␣␣␣␣␣␣␣␣
00CF4BA8	58 E2 05 74 4A 68 0E 74	␣␣␣␣␣␣␣␣␣␣
00CF4BB0	55 0D 0A 74 80 43 06 74	␣␣␣␣␣␣␣␣␣␣
00CF4BB8	AB CD 06 74 47 26 09 74	␣␣␣␣␣␣␣␣␣␣
00CF4BC0	6F 37 07 74 09 D4 06 74	␣␣␣␣␣␣␣␣␣␣
00CF4BC8	55 1E 08 74 20 38 10 74	␣␣␣␣␣␣␣␣␣␣
00CF4BD0	A4 94 06 74 BE F7 0C 74	␣␣␣␣␣␣␣␣␣␣
00CF4BD8	C6 F7 0C 74 AC 40 09 74	␣␣␣␣␣␣␣␣␣␣
00CF4BE0	7E 79 0D 74 8E 79 0D 74	␣␣␣␣␣␣␣␣␣␣
00CF4BE8	AA 79 0D 74 BC 79 0D 74	␣␣␣␣␣␣␣␣␣␣
00CF4BF0	D0 79 0D 74 E2 79 0D 74	␣␣␣␣␣␣␣␣␣␣
00CF4BF8	F6 79 0D 74 08 7A 0D 74	␣␣␣␣␣␣␣␣␣␣
00CF4C00	1C 7A 0D 74 2E 7A 0D 74	␣␣␣␣␣␣␣␣␣␣
00CF4C08	42 7A 0D 74 54 7A 0D 74	␣␣␣␣␣␣␣␣␣␣
00CF4C10	68 7A 0D 74 7A 7A 0D 74	␣␣␣␣␣␣␣␣␣␣
00CF4C18	8E 7A 0D 74 A0 7A 0D 74	␣␣␣␣␣␣␣␣␣␣
00CF4C20	B4 7A 0D 74 C6 7A 0D 74	␣␣␣␣␣␣␣␣␣␣

Command: ? 88

我们看到该表的 00A0 偏移处。

Registers (FPU)			
EAX	00CF4C40		
ECX	00000000		
EDX	00CF4BA0		
EBX	FFFFFFFF		
ESP	0012F3BC		
EBP	0012F4E0		
ESI	00401BDE	cl	
EDI	000000A0		
EIP	7413E834	MS	

Address	Hex dump	ASCII
00CF4C30	00 7B 00 74 12 7B 00 74	.(.t#(.t
00CF4C38	26 7B 00 74 38 7B 00 74	%(.t8(.t
00CF4C40	B6 A5 09 74 32 00 09 74	AN.t2.t
00CF4C48	4C 7B 00 74 5E 7B 00 74	L(.t^(.t
00CF4C50	72 7B 00 74 84 7B 00 74	rt.t\$(.t
00CF4C58	98 7B 00 74 AA 7B 00 74	yt.t-(.t
00CF4C60	BE 7B 00 74 D0 7B 00 74	%(.t\$(.t
00CF4C68	E4 7B 00 74 F6 7B 00 74	%(.t+(.t
00CF4C70	0A 7C 00 74 1C 7C 00 74	.(.tL(.t
00CF4C78	30 7C 00 74 42 7C 00 74	0(.tB(.t
00CF4C80	56 7C 00 74 68 7C 00 74	U(.th(.t
00CF4C88	7C 7C 00 74 8E 7C 00 74	!(!.tA(.t
00CF4C90	A2 7C 00 74 B4 7C 00 74	o(!.tI(.t
00CF4C98	C8 7C 00 74 DA 7C 00 74	e(!.tr(.t
00CF4CA0	EE 7C 00 74 00 7D 00 74	-(!.t.)(.t
00CF4CA8	14 7D 00 74 26 7D 00 74	0).t&(.t
00CF4CB0	3A 7D 00 74 4C 7D 00 74	:).tL(.t
00CF4CB8	60 7D 00 74 72 7D 00 74	').tr(.t
00CF4CC0	86 7D 00 74 98 7D 00 74	\$).tY(.t
00CF4CC8	AC 7D 00 74 BE 7D 00 74	%).t#(.t
00CF4CD0	CC 7D 00 74 E4 7D 00 74	%).t#(.t

Command	788
---------	-----

这里依然是间接 CALL 表中内容,我们不跟进这个 CALL,直接按 F8 键单步步过这个 CALL,然后看堆栈的结果。

0012F3BC	00CFBCA4
0012F3C0	0012F454
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000

我们按 F8 键执行这个 CALL。

接着 EBP - 44 的内容保存到 EDX 中。

Registers (FPU)			
EAX	00000000		
ECX	7C92056D	nt	
EDX	00159888		
EBX	FFFFFFFF		
ESP	0012F3C4		
EBP	0012F4E0		
ESI	00401BDE	cl	
EDI	000000A0		
EIP	7413E839	MS	
C 0	ES 0023	32	
P 1	CS 001B	32	

7413E834	FF10	CALL DWORD PTR DS:[EAX]
7413E836	8B55 BC	MOV EDX,DWORD PTR SS:[EBP-44]
7413E839	66:F742 76 020	TEST WORD PTR DS:[EDX+76],2
7413E83F	75 13	JNZ SHORT MSUBUM50.7413E854
7413E841	0BC0	OR EAX,EAX
7413E843	78 19	JS SHORT MSUBUM50.7413E85E
7413E845	33C0	XOR EAX,EAX

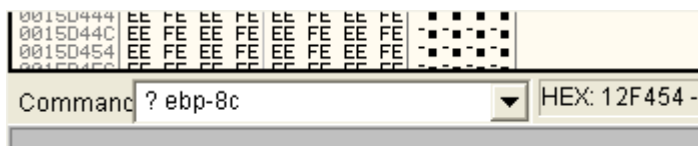
这里判断某个值,接着读取下一个操作码。这里你可能会问读取的是什么,是我们输入的错误序列号吗?我们看看 ExDec 先。

```

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd             text
401BD7: 19 FStAdFunc             local_0088
401BDA: 08 FLdPr                local_0088
401BDD: 0d VCallHresult          get_ipropTEXTEDIT
401BE2: 6c ILdRf                 local_008C
401BE5: 1b LitStr:               "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str           local_008C
401BED: 15 FF---111             1---1 0000

```

我们可以看到该操作码的参数是 8C,也就是 EBP - 8C,我们看看 EBP - 8C 的值是多少。



这里我们可以看到是 12F454。

Address	Hex dump	ASCII
0012F454	BC D3 15 00 A4 BC CF 00	#E\$.R"q.
0012F45C	EC F4 12 00 C8 F5 12 00	y"q.,"\$q.
0012F464	3C BB CF 00 E8 F4 12 00	<"q."b"q.
0012F46C	FF FF FF FF 15 20 00 00	\$...
0012F474	02 E3 13 74 E8 F4 12 00	@b!!t"q.
0012F47C	00 00 00 00 00 00 00 00
0012F484	00 00 00 00 00 1B 40 00s+@.
0012F48C	E4 17 40 00 98 1C 40 00	q"q."yL@.
0012F494	A4 BC CF 00 00 E0 00 00	R"q."0..
0012F49C	88 98 15 00 00 00 00 00	ey\$.....
0012F4A4	00 00 00 00 00 00 04 00d.
0012F4AC	00 00 00 00 00 F4 12 00s"q.
0012F4B4	80 AF 15 00 1C FA 12 00	C>\$."L."q.
0012F4BC	D0 69 05 74 02 E3 13 74	s!t@b!!t
0012F4C4	C4 F3 12 00 E0 F4 12 00	-q".q"q.
0012F4CC	D0 1B 40 00 88 98 15 00	s+@."ey\$.
0012F4D4	98 1C 40 00 00 00 00 00	yL@.....
0012F4DC	00 00 00 00 EC F4 12 00y"q.
0012F4E4	A9 E5 05 74 00 B0 15 00	@q!t."\$.
0012F4EC	FC F4 12 00 F8 1A 40 00	"q".q+@.
0012F4F4	00 00 15 00 00 00 00 00

其保存的是 15D3BC 是我们输入的错误序列号的指针。

EAX=00000000

Address	Hex dump	ASCII
0015D3BC	39 00 38 00 39 00 38 00	9.8.9.8.
0015D3C4	39 00 38 00 39 00 38 00	9.8.9.8.
0015D3CC	00 00 AD BA 00 F0 AD BA	.. .-
0015D3D4	00 F0 AD BA AB AB AB AB	.- %%%%
0015D3DC	AB AB AB AB 00 00 00 00	%%%%....
0015D3E4	00 00 00 00 2D 00 07 00-...
0015D3EC	EE 04 EE 00 F0 02 15 00	..-."@\$.
0015D3F4	00 00 15 00 00 00 00 00

嘿嘿,终于找到了我们输入的错误序列号。接下来一个操作码是 6C ILdRf。

Email josephco_@hotmail.com with any errors or problems

Proc: 401c98
401BD0: 04 FLdRfVar local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd text
401BD7: 19 FStAdFunc local_0088
401BDA: 08 FLdPr local_0088
401BDD: 0d VCallHresult get__ipropTEXTEDIT
401BE2: 6c ILdRf local_008C
401BE5: 1b LitStr: "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str local_008C

7413E843	78 19	JS SHORT MSUBUM50.7413E85E	
7413E845	33C0	XOR EAX,EAX	
7413E847	8A46 04	MOV AL,BYTE PTR DS:[ESI+4]	
7413E84A	83C6 05	ADD ESI,5	
7413E84D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413E854	B8 689C0000	MOV EAX,9C68	
7413E859	E9 3AEBFFFF	JMP MSUBUM50.7413D398	
7413E85E	66:3D 689C	CMP AX,9C68	
7413E862	0F84 D57A0000	JE MSUBUM50.7414633D	
7413E868	E7	DISC ENT	

'6C, ILdRf, 0C, 00
'Load reference value.
'Parameter 1 = 2 bytes.
'Parameter 1 is offset into local Frame.
'Offset = &hC.
'Address pointer is retrieved from local Frame at offset.
'Address pointer is pushed onto stack.
'Stack operations: Push x1.

这里的解释是该操作码加载一个引用的值。我们跟进这个操作码看看。

7413E841	0BC0	OR EAX,EAX	
7413E843	78 19	JS SHORT MSUBUM50.7413E85E	
7413E845	33C0	XOR EAX,EAX	
7413E847	8A46 04	MOV AL,BYTE PTR DS:[ESI+4]	
7413E84A	83C6 05	ADD ESI,5	
7413E84D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	MSUBUM50.7413D947
7413E854	B8 689C0000	MOV EAX,9C68	
7413E859	E9 3AEBFFFF	JMP MSUBUM50.7413D398	

这里。

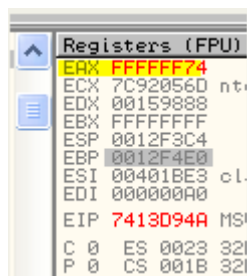
7413D947	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413D94A	FF3428	PUSH DWORD PTR DS:[EAX+EBP]	
7413D94D	33C0	XOR EAX,EAX	
7413D94F	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413D952	83C6 03	ADD ESI,3	
7413D955	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	

这里通过 MOVSX 指令读取参数的值,是个负数。

7413D947	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]	
7413D94A	FF3428	PUSH DWORD PTR DS:[EAX+EBP]	
7413D94D	33C0	XOR EAX,EAX	
7413D94F	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413D952	83C6 03	ADD ESI,3	
7413D955	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	

Address	Hex dump	ASCII
00401BD8	78 FF 0D A0 00 00 00 6C	x .ä...l
00401BE3	74 FF 1B 01 00 FB 30 2F	t +0.'0/
00401BEB	74 FF 1A 78 FF 1C 26 00	t +x L&.
00401BF3	1E C4 00 FE C1 54 FF 1A	▲.-.T +
00401BFB	C4 03 00 FC F6 64 FF 04	→.÷d ♦
00401C03	74 FF 21 0F 00 03 19 78	t !*.♦+x

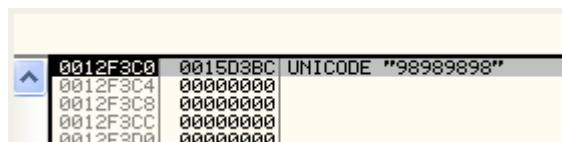
参数是 FF74,所以保存到 EAX 中是:



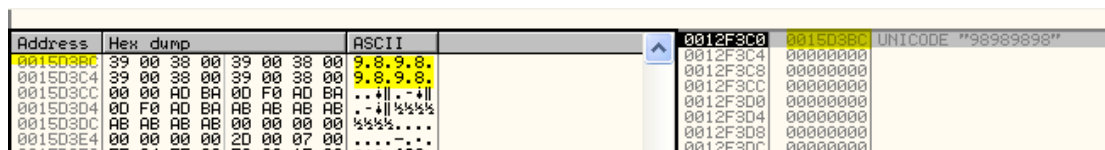
对应的十六进制是-8C。



这里将 EAX + EBP 指向的内容压入堆栈,实际上是将 EBP - 8C 的内容压入堆栈。



这里我们可以看到 EBP - 8C 指向了我们输入的错误序列号,堆栈中也保存了这个指针。

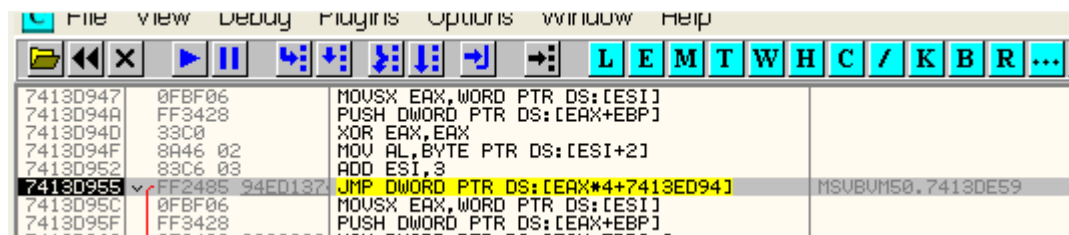


我们在数据窗口中清楚地看到指向了我们输入的错误序列号。

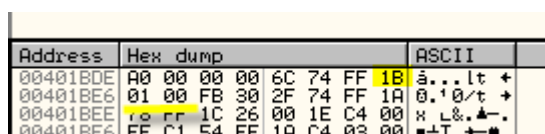
下一个操作码是:

1b LitStr,根据字面上的意思来理解是"字符串"。

我们来看看它会干些什么。



我们跟进该操作码。



可以看到参数为 0001,将被保存到 EAX 中,是个正数。

Registers (FPU)	
EAX	00000001
ECX	7C92056D n
EDX	00159888
EBX	FFFFFFFF
ESP	0012F3C0
EBP	0012F4E0
ESI	00401BE6 c
EDI	000000A0
EIP	7413DE5C M

7413DE59	0FB706	MOVZX EAX,WORD PTR DS:[ESI]	
7413DE5C	8B55 AC	MOV EDX,DWORD PTR SS:[EBP-54]	clave1.004017E4
7413DE5F	FF3482	PUSH DWORD PTR DS:[EDX+EAX*4]	
7413DE62	33C0	XOR EAX,EAX	
7413DE64	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413DE67	83C6 03	ADD ESI,3	
7413DE6A	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413DE71	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]	

接下来一行我们可以看到 4017E4 被保存到了 EDX 中,这个值是什么,我们暂时还无从知晓。

7413DE59	0FB706	MOVZX EAX,WORD PTR DS:[ESI]	
7413DE5C	8B55 AC	MOV EDX,DWORD PTR SS:[EBP-54]	
7413DE5F	FF3482	PUSH DWORD PTR DS:[EDX+EAX*4]	clave1.004016F8
7413DE62	33C0	XOR EAX,EAX	
7413DE64	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413DE67	83C6 03	ADD ESI,3	
7413DE6A	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	

压入这个 4016F8 是干嘛的呢?

```

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis            local_008C
401BD4: 0f VCallAd              text
401BD7: 19 FStAdFunc            local_0088
401BDA: 08 FLdPr                local_0088
401BDD: 0d VCallHresult         get_ipropTEXTEDIT
401BE2: 6c ILdRf                local_008C
401BE5: 1b LitStr:              "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str           local_008C

```

ExDec 中只显示了两个单引号,表明将一个空字符串压入堆栈。

Address	Hex dump	ASCII			
004016F8	00 00 00 00 22 00 00 00"		0012F3BC	004016F8
00401700	4E 00 FA 00 6D 00 65 00	N...m...		0012F3C0	0015D3BC
00401708	72 00 6F 00 20 00 49 00	r.o..l.		0012F3C4	00000000
00401710	6E 00 63 00 6F 00 72 00	n.c.o.r.		0012F3C8	00000000
00401718	72 00 65 00 63 00 74 00	r.e.c.t.		0012F3CC	00000000
00401720	6F 00 00 00 0C 00 00 00	o.....		0012F3D0	00000000
00401728	50 00 2D 00 43 00 6F 00	P.-.C.o.		0012F3D4	00000000
00401730	64 00 65 00 00 00 00 00	d.e.....		0012F3D8	00000000
00401738	22 00 00 00 4E 00 FA 00	"...N..		0012F3DC	00000000
00401740	6D 00 65 00 72 00 6F 00	m.e.r.o.		0012F3E0	00000000
00401748	20 00 43 00 6F 00 72 00	.C.o.r.		0012F3E4	00000000
00401750	72 00 65 00 63 00 74 00	r.e.c.t.		0012F3E8	00000000
00401758	6F 00 21 00 21 00 00 00	o..f...		0012F3EC	00000000
00401760	56 42 41 35 2E 44 4C 4C	VBAS, DLL		0012F3F0	00000000
00401768	00 00 00 00 00 00 00 00		0012F3F4	00000000
00401770	00 00 00 00 00 00 00 00		0012F3F8	00000000

通过数据窗口我们也能看出是一个空字符串,即当我们单击注册按钮时,下一个操作码会检查我们输入的是否为空。

7413DE59	0FB706	MOVZX EAX,WORD PTR DS:[ESI]	
7413DE5C	8B55 AC	MOV EDX,DWORD PTR SS:[EBP-54]	
7413DE5F	FF3482	PUSH DWORD PTR DS:[EDX+EAX*4]	
7413DE62	33C0	XOR EAX,EAX	
7413DE64	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413DE67	83C6 03	ADD ESI,3	
7413DE6A	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413DE71	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]	
7413DE74	83C6 02	ADD ESI,2	

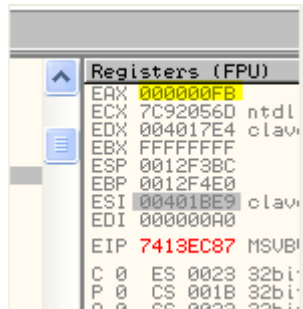
ExDec 中显示如下:

```
401BE2: bc ILdRt      local_008C
401BE5: 1b LitStr:      "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str    local_008C
```

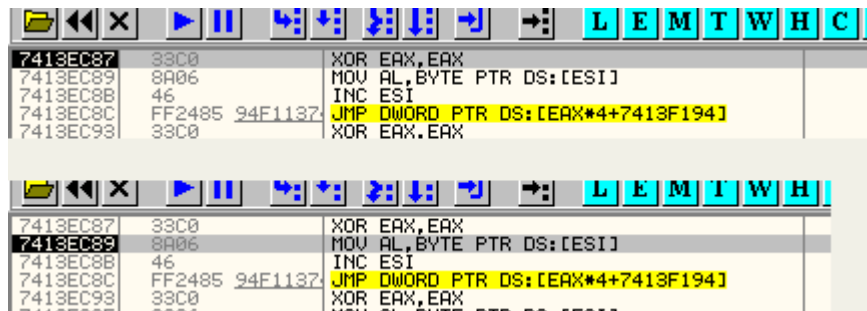
这里 Lead0 是第一个操作,30 EqStr 是第二个操作。我们来 Google 一下它的意思。

Lead0/30 EqStr - 比较两个字符串。

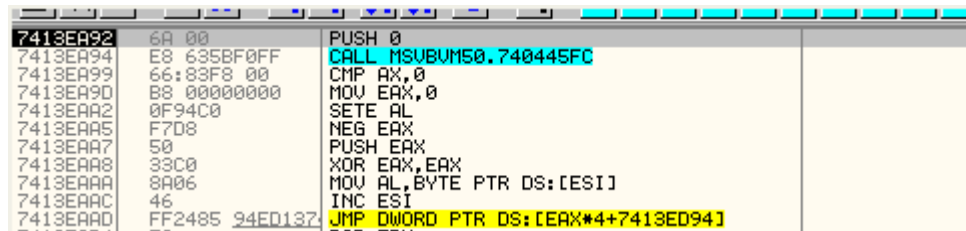
也就是说这里将比较两个字符串。这是一个双操作码的操作。第一个操作码的操作数是 FB,我们跟进这个操作码。



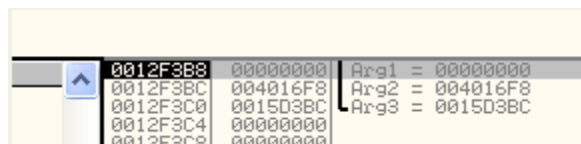
这里直接以 XOR EAX,EAX 结束,什么也没做,接着读取第二个操作码。



第二个操作码是 30,接着读取参数。操作执行完后以 XOR EAX,EAX 结束。我们跟进这个操作码。



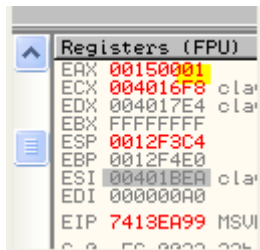
这第二个操作码 PUSH 0。



接下来是一个 CALL,有三个参数。我们按 F8 键单步步过这个 CALL,看看会发生什么。

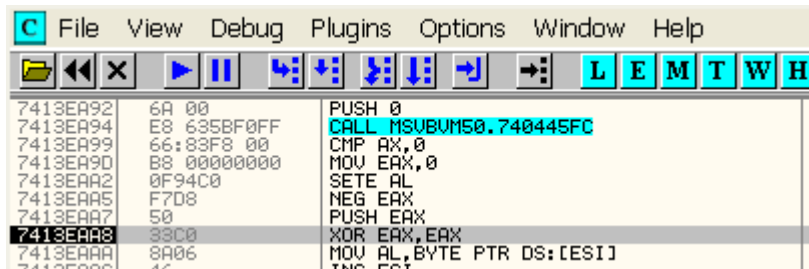
堆栈移动了,里面值没有变,只是堆栈被抬高了。

下一行 CMP AL,0,这里 AL 保存的是上一个 CALL 的结果,我这里的值是:

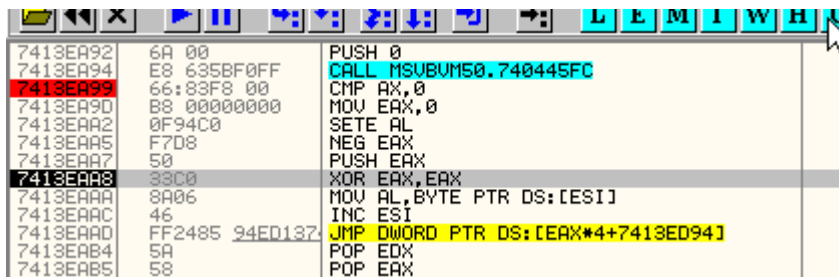


AL = 01

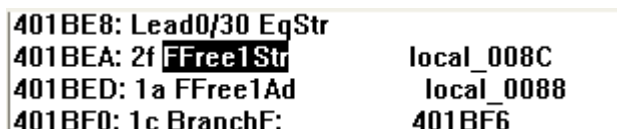
表示两个字符串不相等。



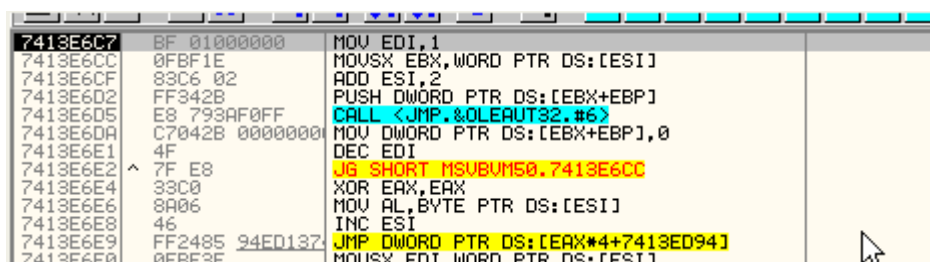
比较完以后首先将 EAX 置零。如果刚刚比较的结果不为零,就将零压入堆栈,如果比较的结果相等就将 FFFFFFFF 压入堆栈。说明在做检查,嘿嘿。



我们接着看下一个操作码。



我们 Google 一下它的意思会发现跟 SysFreeString 类似,就是释放字符串所占的内存空间。我们可以看到这里要释放的内存空间是 EBP - 8C。



这里首先将 EDX 赋值为 1,接着通过 MOVSB 将参数值保存到 EBX 中,接着将 EBX - FF74(十六进制的-8C)压入堆栈。

Registers (FPU)		
EAX	0000002F	
ECX	004016F8	clave
EDX	004017E4	clave
EBX	FFFFFF74	
ESP	0012F3BC	
EBP	0012F4E0	
ESI	00401BEB	clave
EDI	00000001	
EIP	7413E6CF	MSUBU
C 0	ES 0023	32bit
P 1	CS 001R	32bit

7413E6C0	0FBF1E	MOVSB EBX,WORD PTR DS:[ESI]
7413E6CF	83C6 02	ADD ESI,2
7413E6D2	FF342B	PUSH DWORD PTR DS:[EBX+EBP]
7413E6D5	E8 793AF0FF	CALL <JMP.&OLEAUT32.#6>
7413E6DA	C7042B 00000000	MOV DWORD PTR DS:[EBX+EBP],0
7413E6E1	4F	DEC EDI

这里 EBX + EBP 即 EBP - 8C,所以压入堆栈的是错误的序列号。

Address	Hex dump	ASCII		0012F3B8	0015D3BC	Arg1 = 0015D3BC
0015D3BC	39 00 38 00 39 00 38 00	9.8.9.8.		0012F3BC	00000000	
0015D3C4	39 00 38 00 39 00 38 00	9.8.9.8.		0012F3C0	00000000	
0015D3CC	00 00 AD BA 00 F0 AD BA	..+..+..		0012F3C4	00000000	
0015D3D4	00 F0 AD BA AB AB AB AB	..+..+..		0012F3C8	00000000	
0015D3DC	00 00 AD BA AB AB AB AB	..+..+..		0012F3CC	00000000	

74042153	- FF25 88190474	JMP DWORD PTR DS:[&OLEAUT32.#6>]	OLEAUT32.SysFreeString
74042159	E8 28000000	CALL MSUBUM50.74042186	
7404215E	83D0 64F01474	CMPL DWORD PTR DS:[7414F064],0	
74042165	74 18	JE SHORT MSUBUM50.7404217F	
74042167	A1 68F01474	MOV EAX,DWORD PTR DS:[7414F068]	
7404216C	50	PUSH EAX	
7404216D	FF15 84100474	CALL DWORD PTR DS:[&kernel32.TlsGetValue]	kernel32.TlsGetValue
74042173	8B48 18	MOV ECX,DWORD PTR DS:[EAX+18]	
74042176	8B41 14	MOV EAX,DWORD PTR DS:[ECX+14]	
74042179	8B10	MOV EDX,DWORD PTR DS:[EAX]	
7404217B	8951 14	MOV DWORD PTR DS:[ECX+14],EDX	
7404217E	C3	RETN	
7404217F	D1 6CF01474	MOVL FDX,DWORD PTR DS:[7414F06C]	

这里我们可以看到下面的 CALL 里面会调用一个 API 函数 SysFreeString,然后返回。

7413E6C7	BF 01000000	MOV EDI,1
7413E6CC	0FBF1E	MOVSB EBX,WORD PTR DS:[ESI]
7413E6CF	83C6 02	ADD ESI,2
7413E6D2	FF342B	PUSH DWORD PTR DS:[EBX+EBP]
7413E6D5	E8 793AF0FF	CALL <JMP.&OLEAUT32.#6>
7413E6DA	C7042B 00000000	MOV DWORD PTR DS:[EBX+EBP],0
7413E6E1	4F	DEC EDI
7413E6E2	7F E8	JG SHORT MSUBUM50.7413E6CC
7413E6E4	33C0	XOR EAX,EAX

这个时候我们输入的错误序列号被清空了。

Address	Hex dump	ASCII
0012F454	00 00 00 00 A4 BC CF 00#B.
0012F45C	EC F4 12 00 C8 F5 12 00	qT.5.
0012F464	3C BB CF 00 E8 F4 12 00	<T.5.
0012F46C	FF FF FF FF 15 20 00 00	3..
0012F474	02 E3 13 74 E8 F4 12 00	00!!tT.
0012F47C	00 00 00 00 00 00 00 00	

这里我们输入的错误序列号其实还在 15D3BC 这个内存单元中,只不过它的指针 EBP - 8C 被清空了而已。

7413E6D5	E8 793AF0FF	CALL <JMP.&OLEAUT32.#6>
7413E6DA	C7042B 00000000	MOV DWORD PTR DS:[EBX+EBP],0
7413E6E1	4F	DEC EDI
7413E6E2	7F E8	JG SHORT MSUBUM50.7413E6CC
7413E6E4	33C0	XOR EAX,EAX
7413E6E6	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E6E8	46	INC ESI
7413E6E9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E6F0	0FBF3E	MOVSB EDI,WORD PTR DS:[ESI]

接着看下一个操作码。

401BED: 1a FF	Free1Ad	local_0088
401BF0: 1c	BranchF:	401BF6
401BF3: 1e	Branch:	401c94
401BF5: 1d	BranchF:	401c94

这里将清除局部变量 EBP - 88 的内容。

EBP - 88 的值是多少？

00401C45	00 00 00 00	00 00 00 00	...
Command ? ebp - 88			
HEX: 12F458 - [
Address	Hex dump	ASCII	
0012F458	A4 BC CF 00	EC F4 12 00	...
0012F460	C8 F5 12 00	3C BB CF 00	...
0012F468	E8 F4 12 00	FF FF FF FF	...
0012F470	15 20 00 00	02 E3 13 74	...
0012F478	E8 F4 12 00	00 00 00 00	...
0012F480	00 00 00 00	00 00 00 00	...
0012F488	D0 1B 40 00	E4 17 40 00	...
0012F490	98 1C 40 00	A4 BC CF 00	...
0012F498	00 E0 00 00	88 98 15 00	...

这不是表中的数据项吗,将被清除,我们接着往下看。

7413E724	EB 07	MOV EDI,1
7413E726	BF 01000000	MOV EBX,WORD PTR DS:[ESI]
7413E728	0FBF1E	ADD ESI,2
7413E72E	83C6 02	OR EAX,DWORD PTR DS:[EBX+EBP]
7413E731	8B042B	OR EAX,EAX
7413E734	0BC0	JE SHORT MSUBUM50.7413E745
7413E736	74 0D	PUSH EAX
7413E738	50	MOV EAX,DWORD PTR DS:[EAX]
7413E739	8B00	CALL DWORD PTR DS:[EAX+8]
7413E73B	FF50 08	

这里是获取操作码的参数。

Address	Hex dump	ASCII
00401BE6	01 00 FB 30 2F 74 FF 1A	...
00401BEE	78 FF 1C 26 00 1E C4 00	...
00401BF6	FE C1 54 FF 1A C4 03 00	...
00401BFE	FC F6 64 FF 04 74 FF 21	...
00401C06	0F 00 03 19 78 FF 08 78	...

FF78 对应十六进制的-88。

7413E72E	83C6 02	ADD ESI,2
7413E731	8B042B	MOV EAX,DWORD PTR DS:[EBX+EBP]
7413E734	0BC0	OR EAX,EAX
7413E736	74 0D	JE SHORT MSUBUM50.7413E745
7413E738	50	PUSH EAX
7413E739	8B00	MOV EAX,DWORD PTR DS:[EAX]
7413E73B	FF50 08	CALL DWORD PTR DS:[EAX+8]

这里又是将 EBP - 88 的内容保存到 EAX 中,接着判断它是否为零。这里不为零,我们继续。

7413E739	8B00	MOV EAX,DWORD PTR DS:[EAX]	
7413E73B	FF50 08	CALL DWORD PTR DS:[EAX+8]	MSUBUM50.7405E258
7413E73E	C7042B 00000000	MOV DWORD PTR DS:[EBX+EBP],0	
7413E745	4F	DEC EDI	
7413E746	7F E3	JG SHORT MSUBUM50.7413E72B	
7413E748	33C0	XOR EAX,EAX	

这里调用这个 CALL 释放 EBP - 88 局部变量的内存空间。

Address	Hex dump	ASCII
0012F458	00 00 00 00	EC F4 12 00
0012F460	C8 F5 12 00	3C BB CF 00
0012F468	E8 F4 12 00	FF FF FF FF

这里 EAX 被清零了。

7413E746	7F E3	JG SHORT MSUBUM50.7413E72B
7413E748	33C0	XOR EAX,EAX
7413E74A	8A06	MOV AL,BYTE PTR DS:[ESI]
7413E74C	46	INC ESI
7413E74D	FF2485 24ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E754	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413E757	83C6 02	ADD ESI,2
7413E75A	D1EF	SHR EDI,1

下一个操作码是:

401BEA: 2f FF	Free1Str	local_008C
401BED: 1a FF	Free1Ad	local_0088
401BF0: 1c	BranchF:	401BF6
401BF3: 1e	Branch:	401c94
401BF6: Lead3/c1	LitVarl4:	{ local_3BE500AC

该操作码是一个条件跳转,所有的 Branch 开头的都是跳转操作:

指令	操作码	跳转条件
Branch	1e	无条件跳转
BranchF	1c	栈顶数据为 false 则跳转
BranchT	1d	栈顶数据为 True 则跳转

这是一个条件跳转操作,如果栈顶数据为假就跳转,这里检查文本框中的序列号是否正确。

如果跳转了的话,下个操作码将是:

401BED: 1a FF	Free1Ad	local_0088
401BF0: 1c	BranchF:	401BF6
401BF3: 1e	Branch:	401c94
401BF6: Lead3/c1	LitVarL4:	[local_3BE500AC] 0x3c41a [246810]
401BFF: Lead1/#6	FstVar	local_009C

这里的条件跳转直接越过了 401BF3 处的无条件跳转。

C File View Debug Plugins Options Window Help			
[Icons]			
7413D524	0FB736	MOVZX ESI,WORD PTR DS:[ESI]	
7413D527	0375 A8	ADD ESI,DWORD PTR SS:[EBP-58]	
7413D52A	33C0	XOR EAX,EAX	
7413D52C	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D52E	46	INC ESI	
7413D52F	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D536	33C0	XOR EAX,EAX	
7413D538	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413D53B	83C6 03	ADD ESI,3	
7413D53E	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D545	59	POP ECX	
7413D546	66:0BC9	OR CX,CX	
7413D549	74 D9	JE SHORT MSUBUM50.7413D524	
7413D54B	33C0	XOR EAX,EAX	
7413D54D	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413D550	83C6 03	ADD ESI,3	
7413D553	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D55A	58	POP EAX	
7413D55B	0FBFD0	MOVZX EDX,AX	
7413D55E	3BC2	CMP EAX,EDX	
7413D560	0F85 5E050000	JNZ MSUBUM50.7413DAC4	
7413D566	50	PUSH EAX	
7413D567	2E:8BC0	MOV EAX,EAX	
7413D56A	33C0	XOR EAX,EAX	
7413D56C	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D56E	46	INC ESI	
7413D56F	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D576	D9FC	FRNDINT	
7413D578	83EC 04	SUB ESP,4	
7413D57B	DF1C24	FISTP WORD PTR SS:[ESP]	
7413D57E	DFE0	FSTSW AX	
7413D580	A8 00	TEST AL,00	
7413D582	0F85 1ECC0000	JNZ MSUBUM50.7414A1A6	
7413D588	33C0	XOR EAX,EAX	
7413D58A	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D58C	46	INC ESI	
7413D58D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D594	E8 41F4F1FF	CALL MSUBUM50.0ba12Var	
DS:[00401BF6]=FE			
AL=00			

这个操作就结束了,我们继续看下一个。

7413D527	0375 A8	ADD ESI,DWORD PTR SS:[EBP-58]	
7413D52A	33C0	XOR EAX,EAX	
7413D52C	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D52E	46	INC ESI	
7413D52F	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D536	33C0	XOR EAX,EAX	
7413D538	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413D53B	83C6 03	ADD ESI,3	
7413D53E	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D545	59	POP ECX	
7413D546	66:0BC9	OR CX,CX	
7413D549	74 D9	JE SHORT MSUBUM50.7413D524	
7413D54B	33C0	XOR EAX,EAX	
7413D54D	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]	
7413D550	83C6 03	ADD ESI,3	
7413D553	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D55A	58	POP EAX	
7413D55B	0FBFD0	MOVZX EDX,AX	
7413D55E	3BC2	CMP EAX,EDX	
7413D560	0F85 5E050000	JNZ MSUBUM50.7413DAC4	
7413D566	50	PUSH EAX	
7413D567	2E:8BC0	MOV EAX,EAX	
7413D56A	33C0	XOR EAX,EAX	
7413D56C	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D56E	46	INC ESI	
7413D56F	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D576	D9FC	FRNDINT	
7413D578	83EC 04	SUB ESP,4	
7413D57B	DF1C24	FISTP WORD PTR SS:[ESP]	
7413D57E	DFE0	FSTSW AX	
7413D580	A8 00	TEST AL,00	
7413D582	0F85 1ECC0000	JNZ MSUBUM50.7414A1A6	
7413D588	33C0	XOR EAX,EAX	
7413D58A	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413D58C	46	INC ESI	
7413D58D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413D594	E8 41F4F1FF	CALL MSUBUM50.0ba12Var	
DS:[00401BF6]=FE			
AL=00			

这里到达了 401BF6 这个分支,条件跳转成立了,越过了 401BF3 处的无条件跳转,嘿嘿。

Lead3/c1 LitVarL4

这是个双操作码的操作,我们来看一看。

[Icons]			
7413ECAB	33C0	XOR EAX,EAX	
7413ECAD	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413ECAE	46	INC ESI	
7413ECB0	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413FD94]	
7413ECB7	33C0	XOR EAX,EAX	
7413ECB9	8A06	MOV AL,BYTE PTR DS:[ESI]	

这里第一个操作码结束了,继续读取第二个操作码。

7413DEEF	BB 03000000	MOV EBX,3
7413DEF4	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413DEF7	66:891C2F	MOV WORD PTR DS:[EDI+EBP],BX
7413DEFB	8B46 02	MOV EAX,DWORD PTR DS:[ESI+2]
7413DEFE	83C6 06	ADD ESI,6
7413DF01	89442F 08	MOV DWORD PTR DS:[EDI+EBP+8],EAX
7413DF05	EB D9	JMP SHORT MSUBUM50.7413DEE0
7413DF07	BB 05000000	MOV EBX,5

我们继续跟踪。

Address	Hex dump	ASCII
00401BF0	1C 26 00 1E C4 00 FE C1	L&.▲-.■↓
00401BF8	54 FF 1A C4 03 00 FC F6	T→♦.¿÷
00401C00	64 FF 04 74 FF 21 0F 00	d♦t†*.
00401C08	03 19 78 FF 08 78 FF 0D	♦↓x 0x.
00401C10	A0 00 00 00 6C 74 FF 0A	á...lt.
00401C18	02 00 04 00 FD 6B 54 FF	0.♦.²kT

这里读取操作码的参数。

Registers (FPU)	
EAX	000000C1
ECX	00000000
EDX	00000000
EBX	00000003
ESP	0012F3C4
EBP	0012F4E0
ESI	00401BF8
EDI	FFFFFF54
EIP	7413DEF7 MSUI
C 0	ES 0023 32b

FF54 以 FF 开头,所以是一个负数。

7413DEEF	BB 03000000	MOV EBX,3
7413DEF4	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413DEF7	66:891C2F	MOV WORD PTR DS:[EDI+EBP],BX
7413DEFB	8B46 02	MOV EAX,DWORD PTR DS:[ESI+2]
7413DEFE	83C6 06	ADD ESI,6
7413DF01	89442F 08	MOV DWORD PTR DS:[EDI+EBP+8],EAX
7413DF05	EB D9	JMP SHORT MSUBUM50.7413DEE0
7413DF07	BB 05000000	MOV EBX,5

这次参数占 4 个字节,被保存到 EAX 中。

Address	Hex dump	ASCII
00401BF0	1C 26 00 1E C4 00 FE C1	L&.▲-.■↓
00401BF8	54 FF 1A C4 03 00 FC F6	T→♦.¿÷
00401C00	64 FF 04 74 FF 21 0F 00	d♦t†*.
00401C08	03 19 78 FF 08 78 FF 0D	♦↓x 0x.
00401C10	A0 00 00 00 6C 74 FF 0A	á...lt.
00401C18	02 00 04 00 FD 6B 54 FF	0.♦.²kT

Registers (FPU)	
EAX	0003C41A
ECX	00000000
EDX	00000000
EBX	00000003
ESP	0012F3C4
EBP	0012F4E0
ESI	00401BF8
EDI	FFFFFF54
EIP	7413DEF7 MSUI

这是一个局部变量。

7413DEEF	BB 03000000	MOV EBX,3
7413DEF4	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413DEF7	66:891C2F	MOV WORD PTR DS:[EDI+EBP],BX
7413DEFB	8B46 02	MOV EAX,DWORD PTR DS:[ESI+2]
7413DEFE	83C6 06	ADD ESI,6
7413DF01	89442F 08	MOV DWORD PTR DS:[EDI+EBP+8],EAX
7413DF05	EB D9	JMP SHORT MSUBUM50.7413DEE0
7413DF07	BB 05000000	MOV EBX,5
7413DF0C	0FBF3E	MOVSX EDI,WORD PTR DS:[ESI]
7413DF0F	66:891C2F	MOV WORD PTR DS:[EDI+EBP],BX

EBP + FFFFFFF54 + 8 即第一个参数加上 8,结果是 12F43C,参数的值被保存到 12F43C 中。

7413DF61	58	POP EAX
7413DF62	66:290424	SUB WORD PTI
7413DF66	^ 0F80 58FBFFFF	JG MSUBUM50
EAX=0003C41A		
Stack DS:[0012F43C]=00000000		
Address	Hex dump	ASCII
0012F43C	1A C4 03 00 00 00 00 00	→♥.....
0012F444	00 00 00 00 00 00 00 00
0012F44C	00 00 00 00 00 00 00 00
0012F454	00 00 00 00 00 00 00 00
0012F45C	FC F4 12 00 C8 F5 12 00	úŕ\$.úŕ\$.

接着我们跳转到了这里。

7413DEE0	03FD	ADD EDI,EBP
7413DEE2	57	PUSH EDI
7413DEE8	33C0	XOR EAX,EAX
7413DEE5	8A06	MOV AL,BYTE PTR DS:[ESI]

这里将 12F434 压入堆栈。

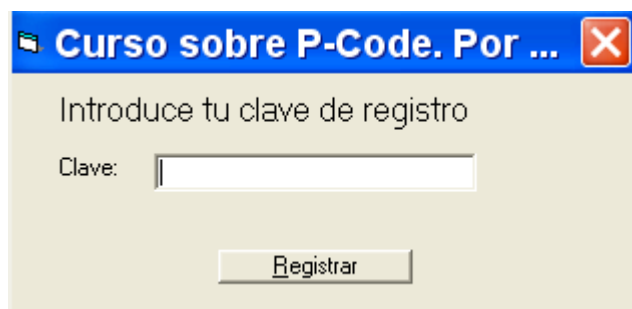
0012F3C0	0012F434
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000
0012F3D0	00000000

12F434 是一个结构体的指针。而这个结构体里面又保存了其他 3 个结构体。

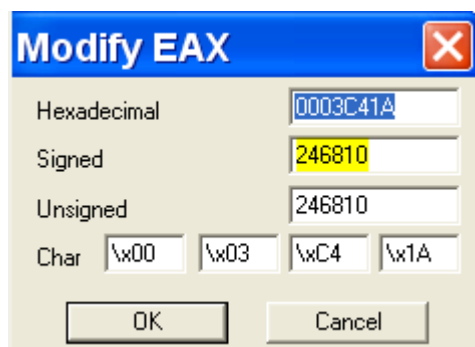
Address	Hex dump	ASCII
0012F434	03 00 00 00 00 00 00 00	♥.....
0012F43C	1A C4 03 00 00 00 00 00	→♥.....
0012F444	00 00 00 00 00 00 00 00

我们跟到了这里,嘿嘿。

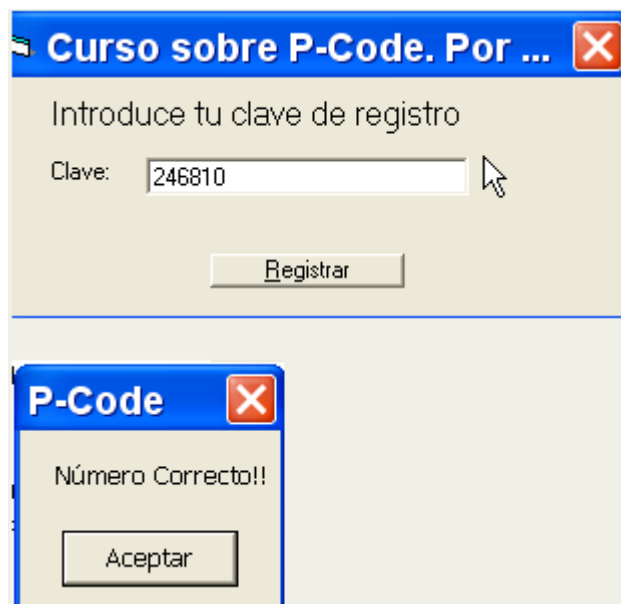
如果该 CrackMe 采用的是硬编码的话,我们可以切换到小数形式,看看是不是正确的序列号。我们直接再打开一个这个 CrackMe。



我们在 OD 中看看十进制值为多少。



我们可以看到十进制值为 246810,我们在 CrackMe 中输入它。



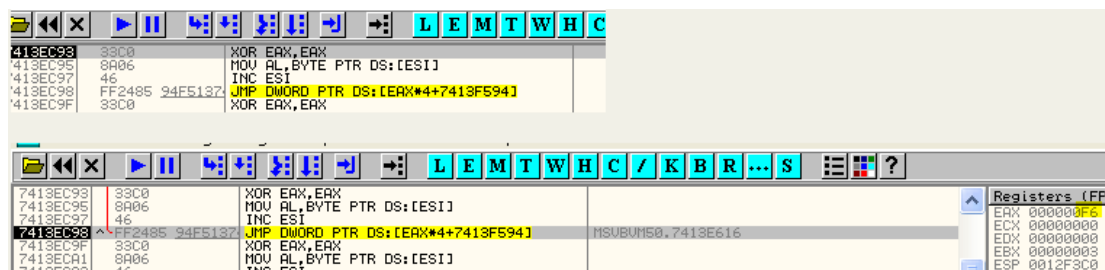
嘿嘿,提示输入的序列号正确。我们接着看比较的过程。

```

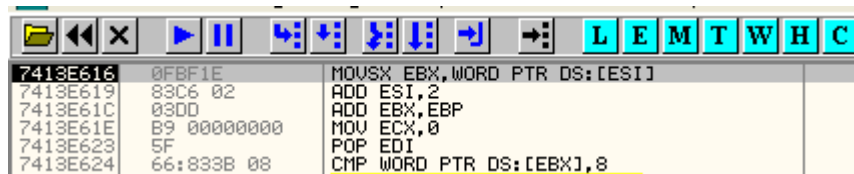
401BF3: 1e Branch:          401c94
401BF6: Lead3/c1 LitVarl4:  (local_3BE500AC) 0x3c41a (246810)
401BFE: Lead1/#6 FStVar      local_009C
401C02: 04 FLdRfVar          local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd           text
401C09: 19 FStAdFunc         local_0088
  
```

下一个操作是双操作码。

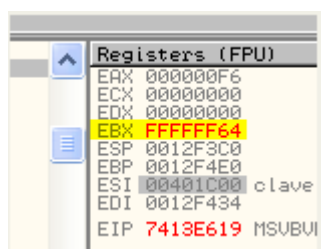
第一个操作码是 FC,我们跟进这个操作码,直接就结束了,接着读取第二个操作码。



该操作码是 F6,我们跟进去,ExDec 中显示的是 EBP - 9C,即 local_009c。



这里读取参数,是个负数。



对应十六进制的-9C。

7413E616	0FBF1E	MOVX EBX,WORD PTR DS:[ESI]
7413E619	83C6 02	ADD ESI,2
7413E61C	03DD	ADD EBX,EBP
7413E61E	B9 00000000	MOV ECX,0
7413E623	5F	POP EDI
7413E624	66:833B 08	CMP WORD PTR DS:[EBX],8
7413E628	72 08	JB SHORT MSUBUM50.7413E632
7413E62A	51	PUSH ECX

这里 EBP + EBX,即 EBP - 9C,值为 12F444,这里是空的。

Registers (FPU)		
EAX	000000F6	
ECX	00000000	
EDX	00000000	
EBX	0012F444	
ESP	0012F3C0	
EBP	0012F4E0	
ESI	00401C02	cl
EDI	0012F434	
EIP	7413E61E	MS

Address	Hex dump	ASCII
0012F444	00 00 00 00 00 00 00 00
0012F44C	00 00 00 00 00 00 00 00
0012F454	00 00 00 00 00 00 00 00
0012F45C	EC F4 12 00 C8 F5 12 00	q!\$.3\$.
0012F464	3C BB CF 00 E8 F4 12 00	<!\$.!\$.
0012F46C	FF FF FF FF 15 20 00 00	3 ..
0012F474	02 E3 13 74 E8 F4 12 00	00!!t!\$.
0012F47C	00 00 00 00 00 00 00 00
0012F484	00 00 00 00 D0 1B 40 00\$+0.

7413E61E	B9 00000000	MOV ECX,0
7413E623	5F	POP EDI
7413E624	66:833B 08	CMP WORD PTR DS:[EBX],8
7413E628	72 08	JB SHORT MSUBUM50.7413E632
7413E62A	51	PUSH ECX
7413E62B	53	PUSH EBX
7413E62C	E8 CFE9FFFF	CALL MSUBUM50.7413D000
7413E631	59	POP ECX
7413E632	8A07	MOV FAX,DWORD PTR DS:[EDI]

这里判断[EBX]是否小于 8,如果小于 8 则跳转。

413E632	8B07	MOV EAX,DWORD PTR DS:[EDI]
413E634	66:83F8 09	CMP AX,9
413E638	75 15	JNZ SHORT MSUBUM50.7413E64F
413E63A	8BD7	MOV EDX,EDI
413E63C	8BCB	MOV ECX,EBX
413E63E	E8 E0270000	CALL MSUBUM50.7413E64F
413E643	33C0	XOR EAX,EAX
413E645	8A06	MOV AL,BYTE PTR DS:[ESI]
413E647	46	INC ESI
413E648	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
413E64F	66:83F8 0D	CMP AX,0D
413E653	0F84 88750000	JE MSUBUM50.74145BE1
413E659	8903	MOV DWORD PTR DS:[EBX],EAX

接着又跳转,这里我们就不深究了,直接看到该操作码的最后一行。

7413E653	0F84 88750000	JE MSUBUM50.74145BE1	
7413E659	8903	MOV DWORD PTR DS:[EBX],EAX	
7413E65B	66:890F	MOV WORD PTR DS:[EDI],CX	
7413E65E	8B4F 04	MOV ECX,DWORD PTR DS:[EDI+4]	
7413E661	894B 04	MOV DWORD PTR DS:[EBX+4],ECX	
7413E664	8B4F 08	MOV ECX,DWORD PTR DS:[EDI+8]	
7413E667	894B 08	MOV DWORD PTR DS:[EBX+8],ECX	
7413E66A	8B4F 0C	MOV ECX,DWORD PTR DS:[EDI+C]	
7413E66D	894B 0C	MOV DWORD PTR DS:[EBX+C],ECX	
7413E670	33C0	XOR EAX,EAX	
7413E672	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E674	46	INC ESI	
7413E675	FF2485 24ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	

Address	Hex dump	ASCII
0012F444	03 00 00 00 00 00 00 00
0012F44C	00 00 00 00 00 00 00 00
0012F454	00 00 00 00 00 00 00 00
0012F45C	EC F4 12 00 C8 F5 12 00
0012F464	3C BB CF 00 E8 F4 12 00

这里将 3 保存到 EBP - 9C 中。

Address	Hex dump	ASCII
0012F434	03 00 00 00 00 00 00 00
0012F43C	1A C4 03 00 00 00 00 00
0012F444	03 00 00 00 00 00 00 00
0012F44C	00 00 00 00 00 00 00 00

执行后。

ECX=00000000			
Address	Hex dump	ASCII	
0012F434	00 00 00 00 00 00 00 00	
0012F43C	1A C4 03 00 00 00 00 00	
0012F444	03 00 00 00 00 00 00 00	
0012F44C	00 00 00 00 00 00 00 00	
0012F454	00 00 00 00 00 00 00 00	
0012F45C	EC F4 12 00 C8 F5 12 00	

下一行拷贝这里的内容到 EBP - 9C 中。

Address	Hex dump	ASCII
0012F444	03 00 00 00 00 00 00 00
0012F44C	1A C4 03 00 00 00 00 00
0012F454	00 00 00 00 00 00 00 00
0012F45C	EC F4 12 00 C8 F5 12 00

EBP - 9C 结构里保存了正确的序列号,还有这个数字 3。

7413E66D	894B 0C	MOV DWORD PTR DS:[EBX+C],ECX	
7413E670	33C0	XOR EAX,EAX	
7413E672	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E674	46	INC ESI	
7413E675	FF2485 24ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413E67C	E8 2FEDFFFF	CALL MSUBUM50.7413D3B0	
7413E681	C1E0 04	SHL EAX,4	
7413E684	03D8	ADD EBX,EAX	
7413E686	EB 96	JMP SHORT MSUBUM50.7413E61E	
7413E688	0FFB3F	MOVSX EAX,WORD PTR DS:[ESI]	

接着看下一个操作码。

401C02: 04 FLdRfVar	local_008C
---------------------	------------

这里跟前面的介绍基本上是一样的了。

401C02: 04 FLdRfVar	local_008C
401C05: 21 FLdPrThis	
401C06: 0f VCallAd	text
401C09: 19 FStAdFunc	local_0088
401C0C: 08 FLdPr	local_0088
401C0F: 0d VCallHresult	get__ipropTEXTEDIT
401C14: 6c ILdRf	local_008C

相同的地方我们直接略过,直接看到我们还没有跟过的操作码。

```

401C0F: 0d VCallHresult      get__ipropExIEDll
401C14: 6c ILdRf              local 008C
401C17: 0a ImpAdCallFPR4:    rtcR8ValFromBstr
401C1C: Lead2/6b CVarR8
401C20: 5d HardType

```

这里我们跳过前面的,直接给 401C17 这个操作码设置一个内存访问断点,当读取到这个操作码的时候就会断下来。

7413D94F	8A46 02	MOV AL, BYTE PTR DS:[ESI+2]
7413D952	83C6 03	ADD ESI, 3
7413D955	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D95C	0FBF06	MOVSX EAX, WORD PTR DS:[ESI]
7413D95F	FF3428	PUSH DWORD PTR DS:[EAX+EBP]
7413D962	C70428 00000000	MOV DWORD PTR DS:[EAX+EBP], 0
7413D969	92CB	VND ENV ENV

断在了这里,这里读取操作码 0A,我们看看 ExDec 中显示的:

ImpAdCallFPR4,表示调用一个 API 函数。

例如:

4017F5: 0a ImpAdCallFPR4: _rtcMsgBox

这个例子是调用__rtcMsgBox 这个 API 函数。

再来看

401C17: 0a ImpAdCallFPR4: _rtcR8ValFromBstr

7413E7A6	0FB70E	MOVZX ECX, WORD PTR DS:[ESI]
7413E7A9	8B55 AC	MOV EDX, DWORD PTR SS:[EBP-54]
7413E7AC	8B048A	MOV EAX, DWORD PTR DS:[EDX+ECX*4]
7413E7AF	0BC0	OR EAX, EAX

这里读取该操作码的参数。

Address	Hex dump	ASCII
00401C17	0A 02 00 04 00 FD 6B 54	.0..?kT
00401C1F	FF 5D 04 64 FF FB 40 2F	Jd'0/
00401C27	74 FF 1A 78 FF 1C 93 00	t *x L0.

Registers (FPU)	
EAX	00000000
ECX	00000002
EDX	00159888
EBX	FFFFFFFF
ESP	0012F3C0
EBP	0012F4E0
ESI	00401C18 clave
EDI	000000A0
EIP	7413E7A9 MSUBU
C 0	ES 0023 32bit

参数被保存到 ECX 中。

7413E7A6	0FB70E	MOVZX ECX, WORD PTR DS:[ESI]	
7413E7A9	8B55 AC	MOV EDX, DWORD PTR SS:[EBP-54]	
7413E7AC	8B048A	MOV EAX, DWORD PTR DS:[EDX+ECX*4]	
7413E7AF	0BC0	OR EAX, EAX	clave1.00401000
7413E7B1	0F84 78790000	JE MSUBU50.7414612F	
7413E7B7	0FB77E 02	MOVZX EDI, WORD PTR DS:[ESI+2]	
7413E7BD	83C6 04	ADD ESI, 4	

EAX 被赋值为 401000,接着判断 EAX 是否为零。

7413E7B7	0FB77E 02	MOVZX EDI, WORD PTR DS:[ESI+2]
7413E7BD	83C6 04	ADD ESI, 4
7413E7BE	03FC	ADD EDI, ESP
7413E7C0	FFD0	CALL EAX
7413E7C2	3BFC	CMP EDI, ESP
7413E7C4	0F85 58790000	JNZ MSUBU50.7414612F

接着读取第二个参数。

Address	Hex dump	ASCII
00401C17	0A 02 00 04 00 FD 6B 54	.0..?kT
00401C1F	FF 5D 04 64 FF FB 40 2F	Jd'0/
00401C27	74 FF 1A 78 FF 1C 93 00	t *x L0.

Registers (FPU)			
EAX	00401000	<	
ECX	00000002	<	
EDX	004017E4	c	
EBX	FFFFFFFF		
ESP	0012F3C0		
EBP	0012F4E0		
ESI	00401C18	c	
EDI	00000004		
EIP	7413E7B8	MS	

Address	Disassembly	Comment
7413E7B8	83C4 04	HLL ESI,4
7413E7BE	03FC	ADD EDI,ESP
7413E7C0	FFD0	CALL EAX
7413E7C2	3BFC	CMP EDI,ESP
7413E7C4	0F85 5B790000	JNZ MSUBVM50.74146125
7413E7C6	90	NOB EAX,EAX

我们到了 CALL EAX 这里,此时 EAX 值为 401000,我们看看这个地址是哪个 API 函数。

Address	Disassembly	Comment
00401000	FF25 68304000	JMP DWORD PTR DS:[<&MSUBVM50.#581>]
00401006	FF25 50304000	JMP DWORD PTR DS:[<&MSUBVM50.#595>]
0040100C	FF25 60304000	JMP DWORD PTR DS:[<&MSUBVM50.#595>]
00401012	FF25 5C304000	JMP DWORD PTR DS:[<&MSUBVM50.#595>]
00401018	FF25 54304000	JMP DWORD PTR DS:[<&MSUBVM50.#595>]
0040101E	FF25 58304000	JMP DWORD PTR DS:[<&MSUBVM50.#595>]
00401024	FF25 4C304000	JMP DWORD PTR DS:[<&MSUBVM50.#595>]
0040102A	FF25 64304000	JMP DWORD PTR DS:[<&MSUBVM50.#595>]
00401030	68 58124000	PUSH clave1.00401258
00401035	E8 F0FFFFFF	CALL <JMP.&MSUBVM50.#100>
0040103A	90	ADD BYTE PTR DS:[EAX+1],AL

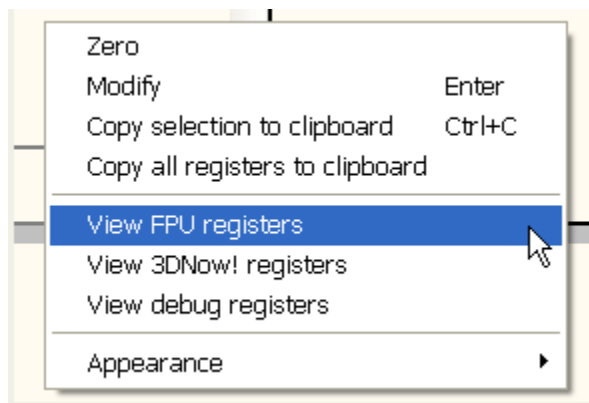
传递给该函数的参数在堆栈中:

Address	Value	Comment
0012F3C0	0015D39C	UNICODE "98989898"
0012F3C4	00000000	
0012F3C8	00000000	
0012F3CC	00000000	
0012F3D0	00000000	

可以看到是我们输入的错误序列号。

Registers (FPU)			
EAX	00000020		
ECX	00159710		
EDX	00000000		
EBX	FFFFFFFF		
ESP	0012F3C4		
EBP	0012F4E0		
ESI	00401C1C	clave1.00401C1C	
EDI	0012F3C4		
EIP	7413E7C2	MSUBVM50.7413E7C2	
C 0	ES 0023	32bit 0(FFFFFFFF)	
P 1	CS 001B	32bit 0(FFFFFFFF)	
A 0	SS 0023	32bit 0(FFFFFFFF)	
Z 1	DS 0023	32bit 0(FFFFFFFF)	
S 0	FS 003B	32bit 7FDF0F00(FFF)	
T 0	GS 0000	NULL	
D 0			
O 0	LastErr	ERROR_SUCCESS (0x00000000)	
EFL	00000246	(NO,NB,E,BE,NS,PE,GE,LE)	
ST0	valid	98989898.000000000000	
ST1	empty	+UNORM 73B4 5004111F 00000060	
ST2	empty	0.0000000078466614990e-4933	
ST3	empty	0.0656640730442726420e-4933	
ST4	empty	6.5358173580319098120e-3137	
ST5	empty	6.1411476247742629740e-2865	
ST6	empty	+UNORM 7CB0 00037C90 003FFFC0	
ST7	empty	-0.0	
FST	3900	Cond 0 0 0 1	
FCW	137F	Prec NEAR,64	
		Mask 1 1 1 1 1 1	

这里该操作序列号被装载到了浮点寄存器 ST0 中去了,浮点寄存器我们还没有介绍过,下面还有一些寄存器。如果你看不到浮点寄存器的话,你可以在寄存器窗口中单击鼠标右键选择-View FPU registers。



加载我们错误序列号的位置在这:

7413E7C2	3BFC	CMP EDI,ESP	
7413E7C4	0F85 5B790000	JNZ MSUBUM50.74146125	
7413E7CA	33C0	XOR EAX,EAX	
7413E7CC	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413E7CE	46	INC ESI	
7413E7CF	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]	
7413E7D6	0FB70E	MOVZX ECX,WORD PTR DS:[ESI]	
7413E7D9	8B55 AC	MOV EDX,DWORD PTR SS:[EBP-54]	
7413E7DC	8BA48A	MOV FAX,DWORD PTR DS:[FAX+FCX*4]	

我们看到下一个操作码。

401C1C:Lead2/6b CVarR8。

我们跟进这个操作码。



7413EC9F	33C0	XOR EAX,EAX	
7413ECA1	8A06	MOV AL,BYTE PTR DS:[ESI]	
7413ECA3	46	INC ESI	
7413ECA4	FF2485 94F9137	JMP DWORD PTR DS:[EAX*4+7413F994]	

这是一个双操作码的操作,先读取第一个操作码,接着读取第二个。

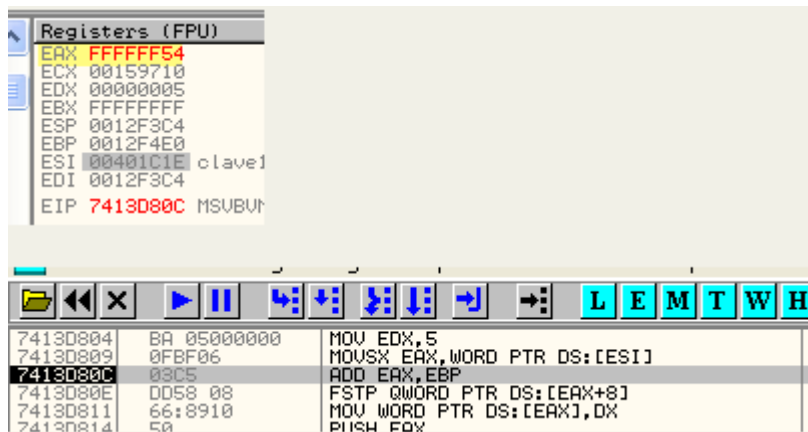
7413D804	BA 05000000	MOV EDX,5	
7413D809	0FBF06	MOVZX EAX,WORD PTR DS:[ESI]	
7413D80C	03C5	ADD EAX,EBP	
7413D80E	DD58 08	FSTP QWORD PTR DS:[EAX+8]	
7413D811	66:8910	MOV WORD PTR DS:[EAX],DX	
7413D814	50	PUSH EAX	
7413D815	DFF0	FSTSW AX	
7413D817	A8 0D	TEST AL,0D	
7413D819	0F85 87C90000	JNZ MSUBUM50.7414A1A6	
7413D81F	33C0	XOR EAX,EAX	

接下来,是一些浮点指令,我们还没有介绍过。

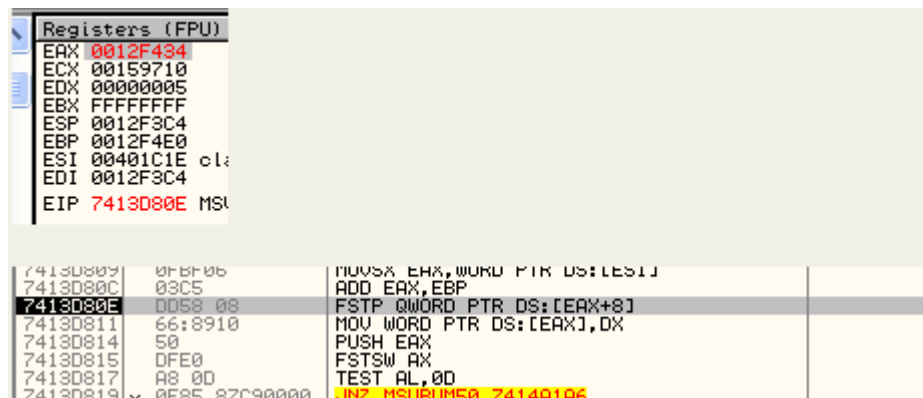
但是我们可以看到读取的参数。

Address	Hex dump	ASCII
00401C16	FF 0A 02 00 04 00 FD 68	.@.k
00401C1E	54 FF 5D 04 64 FF FB 40	TJd'0
00401C26	2F 74 FF 1A 78 FF 1C 93	/t +% L0
00401C2E	00 27 E4 FE 27 04 FF 3A	.%* :

这里:



EBP + EAX



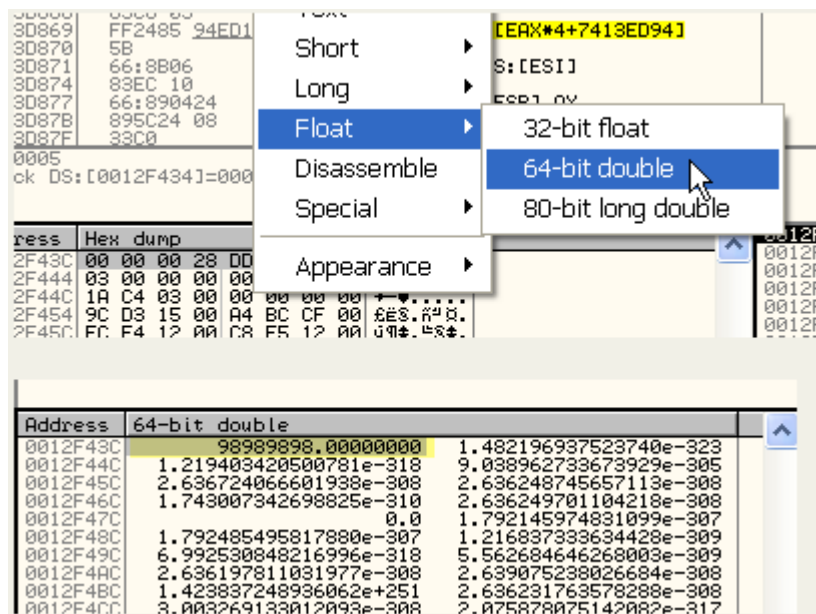
这里 FSTP 指令将 ST0 中的内容保存 EAX+8 指向的 1 内存单元中,即 12F43C 中。然后执行一次出栈操作。我们后面章节再详细讨论。

Address	Hex dump	ASCII
0012F43C	1A C4 03 00 00 00 00 00
0012F444	03 00 00 00 00 00 00 00
0012F44C	1A C4 03 00 00 00 00 00
0012F454	9C D3 15 00 A4 BC CF 00
0012F45C	EC F4 12 00 C8 F5 12 00

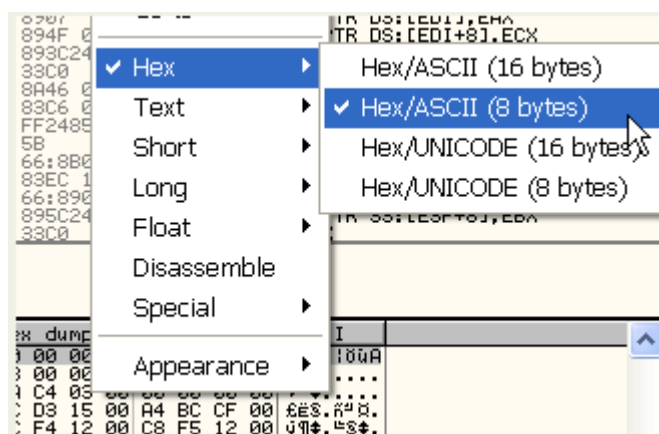
执行以后:

Address	Hex dump	ASCII
0012F43C	00 00 00 28 DD 99 97 41	...(!0uA
0012F444	03 00 00 00 00 00 00 00
0012F44C	1A C4 03 00 00 00 00 00
0012F454	9C D3 15 00 A4 BC CF 00
0012F45C	EC F4 12 00 C8 F5 12 00

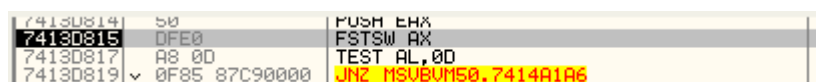
这里有可能是我们输入的错误序列号,我们将其转化为 64 位双精度小数看看,单击鼠标右键。



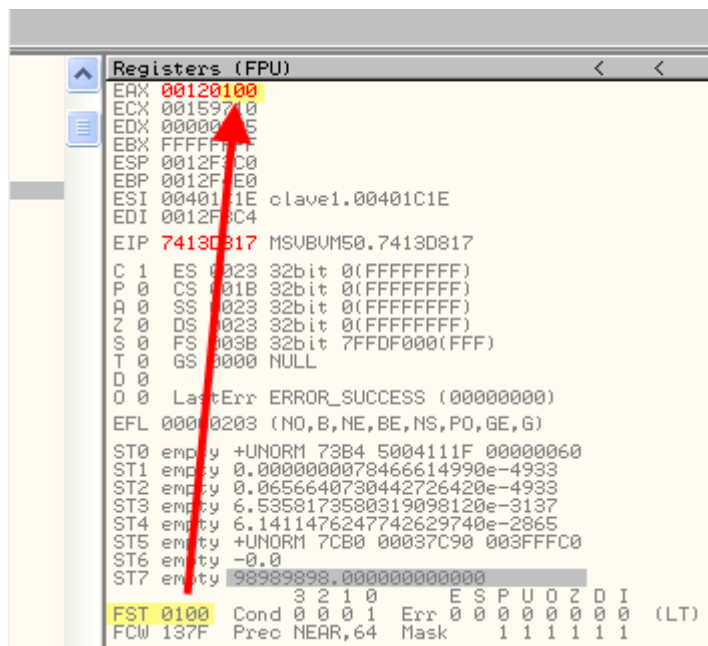
可以看到正好是我们输入的错误序列号,但是这里占的是 8 个字节,即 64 位。从逻辑上来讲,一个占 4 个字节,32 位,一个占 8 个字节,64 位,看起来不一样。



切换为正常模式显示。



该指令将浮点寄存器的状态字保存到 AX 中,我们执行这一行。



74130819	0F85 87C90000	JNZ MSUBUM50.741401A6
7413081F	33C0	XOR EAX,EAX
74130821	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
74130824	83C6 03	ADD ESI,3
74130827	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413082E	8B0424	MOV EAX,DWORD PTR SS:[ESP]
74130831	66:8108 0080	OR WORD PTR DS:[EAX],8000
74130836	33C0	XOR EAX,EAX
74130838	8A06	MOV AL,BYTE PTR DS:[ESI]
7413083A	46	INC ESI
7413083B	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
74130842	0F85 87C90000	JNZ MSUBUM50.741401A6

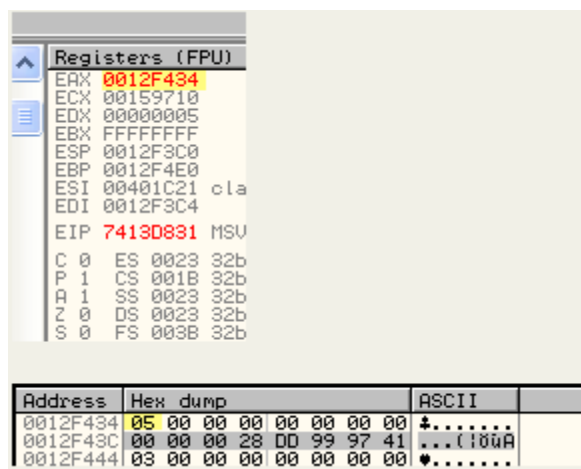
这里该操作就结束了。

401C20: 5d HardType

这个操作码我们不知道是干什么用的,我们还是跟一跟吧。

74130827	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413082E	8B0424	MOV EAX,DWORD PTR SS:[ESP]
74130831	66:8108 0080	OR WORD PTR DS:[EAX],8000
74130836	33C0	XOR EAX,EAX
74130838	8A06	MOV AL,BYTE PTR DS:[ESI]
7413083A	46	INC ESI
7413083B	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
74130842	0F85 87C90000	JNZ MSUBUM50.741401A6

这一行是将 ESP 指向的内容保存到 EAX 中。



这个内存单元位于转换后的错误序列号的上面。

Address	Hex dump	ASCII
0012F434	05 80 00 00 00 00 00 00	#C.....
0012F43C	00 00 00 28 00 99 97 41	...(!00A
0012F444	03 00 00 00 00 00 00 00
0012F44C	10 C4 03 00 00 00 00 00

下一个操作码

```
401C21: 04 FLdRfVar          local_009C
```

这里将 EBP-9C 压入堆栈:

Command ? ebp-9c

HEX: 12F444-

0012F3BC	0012F444
0012F3C0	0012F434
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000

接下来又是一个双操作码的操作。

```
401C24: Lead0/40 NeVarBool
```

7413EC87	33C0	XOR EAX,EAX
7413EC89	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC8B	46	INC ESI
7413EC8C	FF2485 94F1137	JMP DWORD PTR DS:[EAX*4+7413F194]
7413EC93	33C0	XOR EAX,EAX
7413EC95	8A06	MOV AL,BYTE PTR DS:[ESI]

这里读取第二个操作码。

7413EBFA	6A 00	PUSH 0
7413EBFC	BB 74ED1374	MOV EBX,MSUBUM50.7413ED74
7413EC01	E8 8E180000	CALL MSUBUM50.74140494
7413EC06	FF3483	PUSH DWORD PTR DS:[EBX+EAX*4]
7413EC09	22C8	VAR ENV ENV

这里我们到了一个 CALL 处,堆栈中参数如下:

0012F3B8	00000000	Arg1 = 00000000
0012F3BC	0012F444	Arg2 = 0012F444
0012F3C0	0012F434	Arg3 = 0012F434
0012F3C4	00000000	
0012F3C8	00000000	

其中一个指向了正确的序列号,一个指向了错误的序列号,将它们进行比较吗?

Address	Hex dump	ASCII
0012F444	03 00 00 00 00 00 00 00
0012F44C	1A C4 03 00 00 00 00 00
0012F454	9C D3 15 00 A4 BC CF 00
0012F45C	EC F4 12 00 C8 F5 12 00
0012F464	3C BB CF 00 E8 F4 12 00

Address	Hex dump	ASCII
0012F434	05 80 00 00 00 00 00 00	#C.....
0012F43C	00 00 00 28 00 99 97 41	...(!00A
0012F444	03 00 00 00 00 00 00 00
0012F44C	1A C4 03 00 00 00 00 00

7413EBFA	6A 00	PUSH 0
7413EBFC	BB 74ED1374	MOV EBX,MSUBUM50.7413ED74
7413EC01	E8 8E180000	CALL MSUBUM50.74140494
7413EC06	FF3483	PUSH DWORD PTR DS:[EBX+EAX*4]
7413EC09	33C0	XOR EAX,EAX
7413EC0B	8A06	MOV AL,BYTE PTR DS:[ESI]

这里我们下一个断点。

Registers (FPU)	
EAX	00000001
ECX	00000003
EDX	0012F444
EBX	7413ED74 MSU
ESP	0012F3C4
EBP	0012F4E0
ESI	00401C26 c1
EDI	0012F3C4
EIP	7413EC06 MSU
C 0	ES 0023 32t
P 0	CS 001B 32t
A 0	SS 0023 32t
7 0	DS 0023 32t

我们可以看到这个 CALL 返回的结果 EAX 值为 1。

0012F3C0	FFFFFFFF
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000

这里 FFFFFFFF 被压入堆栈,为了看到正确的序列号是多少,我们在比较这里设置一个断点。

Address	Hex dump	ASCII
0012F444	03 00 00 00 00 00 00 00
0012F44C	1A C4 03 00 00 00 00 00
0012F454	9C D3 15 00 A4 BC CF 00	SES. R# D.
0012F45C	EC F4 12 00 C8 F5 12 00	yq#.L\$#.

03C41A 对应的十进制值为 246810,我们在文本框中输入这个值。

Curso sobre P-Code. Por ...

Introduce tu clave de registro

Clave:

Registrar

我们按下 Registrar 按钮,断了下来,我们看到比较处。

Address	64-bit double
0012F434	1.619201341115517e-319 246810.0000000000
0012F444	1.482196937523740e-323 1.219403420500781e-318
0012F454	9.038962733674010e-305 2.636724066601938e-308
0012F464	2.636248745657113e-308 1.743007342698825e-310

246810 对应的十六进制,可能会以不同格式存储,在不同的 CALL 中会被转化然后进行比较,我们看看结果是什么。

Registers (FPU)	
EAX	00000000
ECX	00000003
EDX	0012F444
EBX	7413ED74 MSU
ESP	0012F3C4
EBP	0012F4E0
ESI	00401C26 c1
EDI	0012F3C4
EIP	7413EC06 MSU
C 0	ES 0023 32

我们可以看到 EAX = 0。

0012F3C0	00000000
0012F3C4	00000000
0012F3C8	00000000
0012F3CC	00000000
0012F3D0	00000000
0012F3D4	00000000

栈顶元素也被置零了。

接下来的 BranchF 操作码就会根据栈顶元素值来决定显示什么提示框了。

401C21: 04 FLdRfVar	local_009C
401C24: Lead0/40 NeVarBool	
401C26: 2f FFree1Str	local_008C
401C29: 1a FFree1Ad	local_0088
401C2C: 1c BranchF:	401C63
401C2F: 27 LitVar_Missing	
401C32: 27 LitVar_Missing	

你可能会说怎么这么长啊,因为这里是初次介绍 P-CODE,所以我们给大家逐一介绍了每个操作码,后面章节我们就不会这么赘述了。我们可以根据 ExDec 反编译器得知每个操作码的名称,然后用 OllyDbg 来定位调试。

下一章节,我们将介绍 clave2 这个例子,大家可以先试试。