

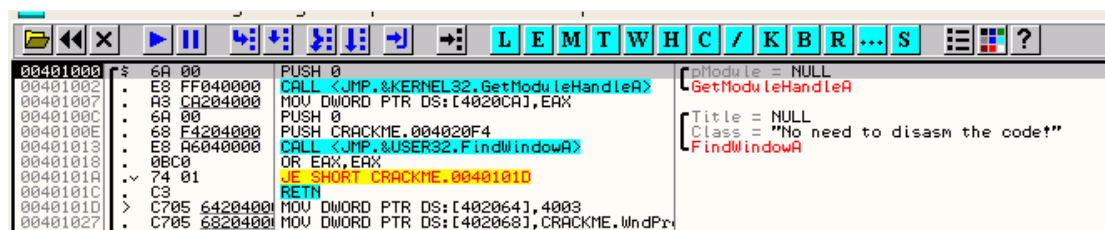
第 16 章-序列号生成算法分析-Part1

本章,我们分析的 CrackMe 与之前的不同之处在于序列号是基于名称变化的,也就是说我们将讨论序列号生成算法。

尽管分析的技巧和之前的很相似,我们还是得来看几个例子巩固一下。

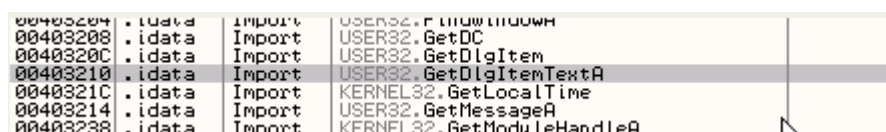
我们先来分析一下 CrueHead'a 的 CrackMe 的序列号生成算法。

用 OD 加载它。

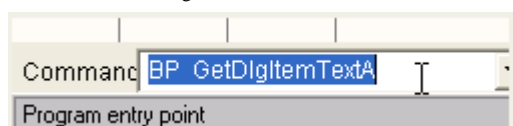


我们停在了入口点处。

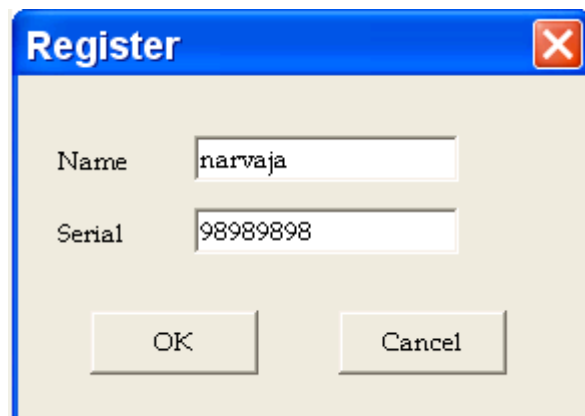
我们来看看该程序使用什么 API 函数来获取用户输入的序列号。



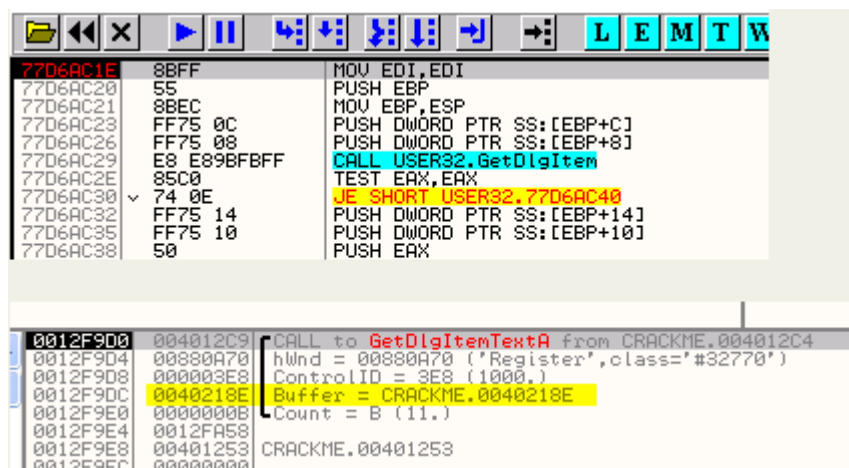
我们可以看到 GetDlgItemTextA 这个 API 函数,我们给这个 API 函数设置断点,输入用户名和正确的序列号看看是否会中断下来。



运行起来。



我们单击 OK 按钮,断在了刚刚设置的断点处,我们来看看堆栈中的参数情况。



Buffer 指向的缓冲区存放用户输入的文本,起始地址为 40218E,我们在 Buffer 参数上面单击鼠标右键选择-Follow in Dump 来在数据窗口中定位到该缓冲区。

Address	Hex dump	ASCII
0040218E	00 00 00 00 00 00 00 00
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00
004021A6	00 00 00 00 00 00 00 00
004021AE	00 00 00 00 00 00 00 00
004021B6	00 00 00 00 00 00 00 00

由于此函数还没有执行,所有缓冲区里面是空的,我们选择主菜单中的 Debug-Execute till return,我们就到了 ret 指令处,我们按 F7 键返回到主模块中。

Address	Hex dump	ASCII
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00
004021A6	00 00 00 00 00 00 00 00

我们可以看到缓冲区中保存了我们输入的名字,程序会进行相应的处理生成正确的序列号。如果我们想编写注册机的话,我们得分析由名称生成序列号的算法,但是现在我们暂时不关心生成算法,我们只是想看看生成的序列号是多少。

0012F9D0	004012E9	CALL to GetDlgItemTextA from CRACKME.004012E4
0012F9D4	00880A70	hWnd = 00880A70 ('Register',class='#32770')
0012F9D8	000003E9	ControlID = 3E9 (1001.)
0012F9DC	0040217E	Buffer = CRACKME.0040217E
0012F9E0	0000000B	Count = B (11.)
0012F9E4	0012FA58	
0012F9E8	00401253	CRACKME.00401253
0012F9EC	00000000	
0012F9F0	0012F9F0	

第二次断在了 GetDlgItemTextA 处,新的缓冲区保存用户输入的序列号,起始地址为 40217E,我们在数据窗口中定位到该缓冲区。

Address	Hex dump	ASCII
0040217E	00 00 00 00 00 00 00 00
00402186	00 00 00 00 00 00 00 00
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00
004021A6	00 00 00 00 00 00 00 00

同样由于函数还没有执行,缓冲区里面是空的,我们通过选择主菜单中的 Debug-Execute till return 执行到返回,然后按 F7 键返回到主模块中。

Address	Hex dump	ASCII
0040217E	39 38 39 38 39 38 39 38	98989898
00402186	00 00 00 00 00 00 00 00
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00
004021A6	00 00 00 00 00 00 00 00

好了,现在缓冲区里面存放了我们输入的错误序列号,从程序的角度出发,程序就会取用户输入的错误的序列号与根据名称生成的正确序列号进行比较,所以我们可以对错误序列号设置内存访问断点,看看程序的哪些地方使用了。

Address	Hex dump	ASCII
0040217E	39 38 39 38 39 38 39 38	98989898
00402186	00 00 00 00 00 00 00 00
0040218E	6E 61 72 76 61 6A 61 00	narvaja.
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00
004021A6	00 00 00 00 00 00 00 00
004021AE	00 00 00 00 00 00 00 00
004021B6	00 00 00 00 00 00 00 00
004021BE	00 00 00 00 00 00 00 00
004021C6	00 00 00 00 00 00 00 00
004021CE	00 00 00 00 00 00 00 00
004021D6	00 00 00 00 00 00 00 00
004021DE	00 00 00 00 00 00 00 00
004021E6	00 00 00 00 00 00 00 00

Backup

Copy

Binary

Label

Breakpoint

Search for

Memory, on access

Memory, on write

```

004013DC . 33DB      XOR EBX,EBX
004013DE . 8B7424 04  MOV ESI,DWORD PTR SS:[ESP+4]
004013E2 > B0 0A      MOV AL,0A
004013E4 . 8A1E      MOV BL,BYTE PTR DS:[ESI]
004013E6 . 84DB      TEST BL,BL
004013E8 . 74 0B      JE SHORT CRACKME.004013F5
004013EA . 80EB 30    SUB BL,30
004013ED . 0AFF8      IMUL EDI,EAX
004013F0 . 03FB      ADD EDI,EBX
004013F2 . 46         INC ESI
004013F3 . ^ EB ED     JMP SHORT CRACKME.004013E2
004013F5 . 81F7 341200 XOR EDI,1234
004013FB . 8BDF      MOV EBX,EDI
004013FD . C3        RETN
004013FE . FF25 8431400 JMP DWORD PTR DS:[&USER32.KillTimer>]
00401404 . FF25 8C31400 JMP DWORD PTR DS:[&USER32.GetSystemMetrics>]
0040140A . FF25 8C31400 JMP DWORD PTR DS:[&USER32.LoadCursorA>]
00401410 . FF25 9031400 JMP DWORD PTR DS:[&USER32.LoadAccelerator>]
00401416 . FF25 9431400 JMP DWORD PTR DS:[&USER32.MessageBeep>]
0040141C . FF25 9831400 JMP DWORD PTR DS:[&USER32.GetWindowRect>]
00401422 . FF25 9C31400 JMP DWORD PTR DS:[&USER32.LoadStringA>]
00401428 . FF25 A031400 JMP DWORD PTR DS:[&USER32.LoadIconA>]
0040142E . FF25 A431400 JMP DWORD PTR DS:[&USER32.LoadBitmap>]

DS:[0040217E]=39 ('9')
BL=00

```

Address	Hex dump	ASCII
0040217E	39 38 39 38 39 38 39 38	98989898
00402186	00 00 00 00 00 00 00 00
0040218E	4E 41 52 56 41 4A 41 00	NARUVAJA.
00402196	00 00 00 00 00 00 00 00
0040219E	00 00 00 00 00 00 00 00

Registers (FPU)

EAX	0000000A	
ECX	0012FDE4	
EDX	7C91EB94	ntldr
EBX	00000039	
ESP	0012FE98	
EBP	0012FEB4	
ESI	0040217E	ASCII
EDI	00000000	
EIP	004013E6	CRAC
C 0	ES 0023	32b

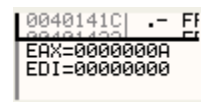
004013E2	8A1E	MOV BL, BYTE PTR DS:[ESI]
004013E4	840B	TEST BL, BL
004013E8	74 0B	JE SHORT CRACKME.004013F5
004013EA	80EB 30	SUB BL, 30
004013ED	0FAFF8	IMUL EDI, EAX

004013E6	. 840B	TEST BL,BL	
004013E8	74 0B	JE SHORT CRACKME.004013F5	
004013EA	. 80EB 30	SUB BL,30	
004013ED	. 0FAFF8	IMUL EDI,EAX	
004013F0	. 03FB	ADD EDI,EBX	
004013F2	. 46	INC ESI	
004013F3	EB ED	JMP SHORT CRACKME.004013E2	
004013F5	> 81F7 34120000	XOR EDI,1234	
004013F8	. 8BDF	MOV EBX,EDI	
004013FD	. C3	RETN	
004013FE	- FF25 84314000	JMP DWORD PTR DS:[<&USER32.KillTimer>]	USER32

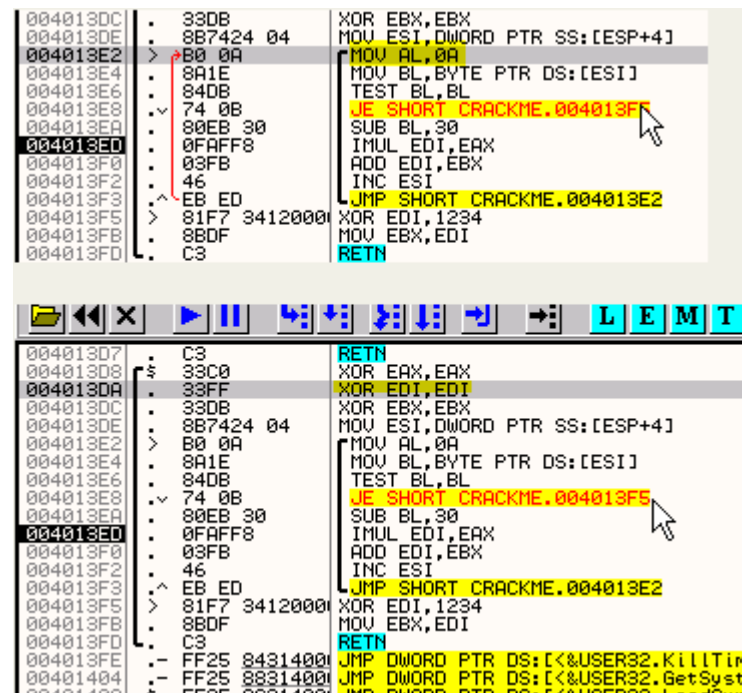
Registers (FPU)	
EAX	00000000
ECX	0012F0E4
EDX	7C91EB94
EBX	00000009
ESP	0012FE98
EBP	0012FEB4
ESI	0000217E
EDI	00000000
EIP	004013ED
C 0	ES 0023 32B
C 1	CS 001B 32B

BL 的值减去 30 等于 9,即错误序列号第一个字符的值。

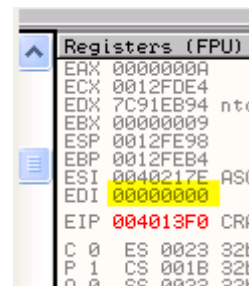
下一条指令 EDI 乘以 EAX。



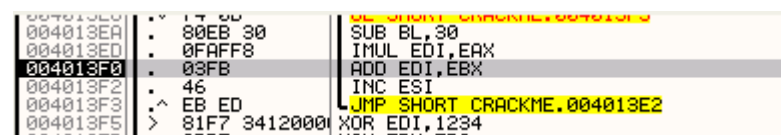
这两个寄存器被初始化为以下值:EAX 寄存器循环体开始处被初始化为 0A,EDI 寄存器在循环体之前被 XOR EDI,EDI 指令初始化为零了。



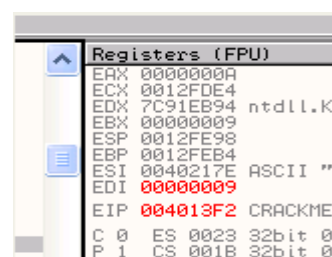
按 F7 键,我们可以看到两个操作数相乘,并且 IMUL 指令会考虑符号位,并且结果被保存到第一个操作数中。



相乘的结果为零,保存在 EDI 中。



接着 EDI 加上 EBX。



EDI 的结果为 9,下一条指令 INC ESI,ESI 递增 1,然后跳转到循环开始处,读取错误序列号的下一个字节。

0040130C	. 330B	XOR EBX,EBX						
0040130E	> 8B7424 04	MOV ESI,DWORD PTR SS:[ESP+4]						
004013E2	. B0 0A	MOV AL,0A						
004013E4	. 8A1E	MOV BL,BYTE PTR DS:[ESI]						
004013E6	. 840B	TEST BL,BL						
004013E8	^ 74 0B	JE SHORT CRACKME.004013F5						
004013EA	. 80EB 30	SUB BL,30						
004013ED	. 0FAFF8	IMUL EDI,EAX						
004013F0	. 03FB	ADD EDI,EBX						
004013F2	. 46	INC ESI						
004013F3	^ EB ED	JMP SHORT CRACKME.004013E2						
004013F5	> 81F7 34120000	XOR EDI,1234						
004013F8	. 8BDF	MOV EBX,EDI						
004013FD	. C3	RETN						
004013FE	.- FF25 84314000	JMP DWORD PTR DS:[&USER32.KillTim						
00401404	.- FF25 88314000	JMP DWORD PTR DS:[&USER32.GetSyst						
0040140A	.- FF25 8C314000	JMP DWORD PTR DS:[&USER32.LoadCur						
00401410	.- FF25 90314000	JMP DWORD PTR DS:[&USER32.LoadAcc						
00401416	.- FF25 94314000	JMP DWORD PTR DS:[&USER32.Message						
0040141C	.- FF25 98314000	JMP DWORD PTR DS:[&USER32.GetWinc						
00401422	.- FF25 9C314000	JMP DWORD PTR DS:[&USER32.L...						
DS:[0040217F]=38 ('8')								
BL=09 (TAB)								
<table border="1"> <thead> <tr> <th>Address</th><th>Hex dump</th><th>ASCII</th></tr> </thead> <tbody> <tr> <td>004013F5</td><td>00 00 00 00 00 00 00 00</td><td></td></tr> </tbody> </table>			Address	Hex dump	ASCII	004013F5	00 00 00 00 00 00 00 00	
Address	Hex dump	ASCII						
004013F5	00 00 00 00 00 00 00 00							

AL 依然被初始化为 0A,错误序列号的下一个字节值为 38,不为零,所以将减去 30,然后执行 IMUL 指令。

004013ED	. 0FAFF8	IMUL EDI,EAX
004013F0	. 03FB	ADD EDI,EBX
004013F2	. 46	INC ESI
004013F3	^ EB ED	JMP SHORT CRACKME.004013E2
004013F5	> 81F7 34120000	XOR EDI,1234
004013F8	. 8BDF	MOV EBX,EDI
004013FD	. C3	RETN
004013FE	.- FF25 84314000	JMP DWORD PTR DS:[&USER32.Kil
00401404	.- FF25 88314000	JMP DWORD PTR DS:[&USER32.Get
0040140A	.- FF25 8C314000	JMP DWORD PTR DS:[&USER32.Lo
00401410	.- FF25 90314000	JMP DWORD PTR DS:[&USER32.Lo
00401416	.- FF25 94314000	JMP DWORD PTR DS:[&USER32.Mes
0040141C	.- FF25 98314000	JMP DWORD PTR DS:[&USER32.Get
00401422	.- FF25 9C314000	JMP DWORD PTR DS:[&USER32.L...
EAX=0000000A		
EDI=00000009		

可以看到上次循环的结果 EDI 乘以 EAX 的值(依然是初始化为 0A),结果依然保存在 EDI 中。

Registers (FPU)	
EAX	0000000A
ECX	0012FDE4
EDX	7C91EB94 ntd
EBX	00000008
ESP	0012FE98
EBP	0012FEB4
ESI	0040217F ASC
EDI	0000005A
EIP	004013F0 CRA
C 0	ES 0023 32b
P 0	CS 001B 32b

现在 EDI 的值为 5A,下一条指令,EDI 将加上 EBX。

Registers (FPU)	
EAX	0000000A
ECX	0012FDE4
EDX	7C91EB94 ntd
EBX	00000008
ESP	0012FE98
EBP	0012FEB4
ESI	0040217F ASC
EDI	00000062
EIP	004013F2 CRA
C 0	ES 0023 32b
P 0	CS 001B 32b

以上结果依然保存到 EDI 中,一步步跟踪这个循环是烦的,如果略过这个过程呢?我们接下来将介绍。

```

004013D7  .  C3                RETN
004013D8  .  33C0              XOR EAX,EAX
004013DA  .  33FF              XOR EDI,EDI
004013DC  .  33DB              XOR EBX,EBX
004013DE  .  8B7424 04         MOV ESI,DWORD PTR SS:[ESP+4]
004013E2  >  B0 0A             MOV AL,0A
004013E4  .  8A1E              MOV BL,BYTE PTR DS:[ESI]
004013E6  .  84DB              TEST BL,BL
004013E8  .  74 0B             JE SHORT CRACKME.004013F5
004013EA  .  80EB 30           SUB BL,30
004013ED  .  0FAFF8            IMUL EDI,EAX
004013F0  .  03FB              ADD EDI,EBX
004013F2  .  46                INC ESI
004013F3  .  EB ED             JMP SHORT CRACKME.004013E2
004013F5  >  81F7 34120000     XOR EDI,1234
004013F6  .  8BDF              MOV EBX,EDI
004013FD  .  C3                RETN

```

我们只需要在循环的下一条指令处设置一个断点,然后按 F9 键,就会中断下来,我们看看 EDI 的值为多少。

```

Registers (FPU)
EAX 00000000
ECX 0012FDE4
EDX 7C91EB94 ntc
EBX 00000000
ESP 0012FE98
EBP 0012FEB4
ESI 00402186 CR
EDI 00402186 CR
EIP 004013F5 CR
C 0 ES 0023 32t
P 1 CS 001B 32t

```

我们双击 EDI 寄存器。

Modify EDI

Hexadecimal: 05E6774A

Signed: 98989898

Unsigned: 98989898

OK Cancel

EDI 的值为十六进制形式,第二行显示的是有符号的十进制值,也就是我们输入的错误序列号的值,即以上循环结束 EDI 保存了错误序列号的十六进制值。

概括来说就是:“98989898”序列号将被转化为十进制的 98989898 或者十六进制的 5E6774A。

```

004013F2  .  46                INC ESI
004013F3  .  EB ED             JMP SHORT CRACKME.004013E2
004013F5  >  81F7 34120000     XOR EDI,1234
004013F6  .  8BDF              MOV EBX,EDI
004013FD  .  C3                RETN
004013FE  .  FF25 84314000     JMP DWORD PTR DS:[&USI]
00401404  .  FF25 88314000     JMP DWORD PTR DS:[&USI]

```

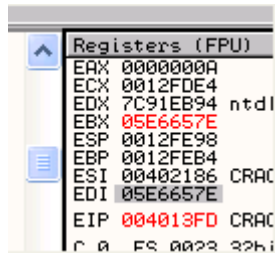
下一条指令,EDI 异或 1234。

```

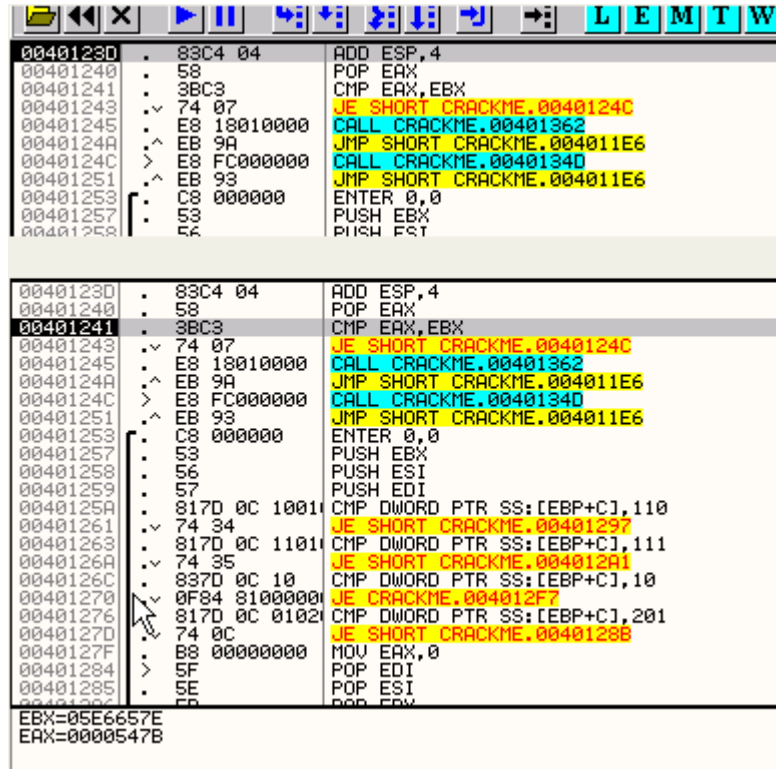
Registers (FPU)
EAX 00000000
ECX 0012FDE4
EDX 7C91EB94 n
EBX 00000000
ESP 0012FE98
EBP 0012FEB4
ESI 00402186 C
EDI 05E6657E C
EIP 004013FB C
C 0 FS 0023 32t
P 1 CS 001B 32t

```

结果为 5E6657E,然后保存到 EBX 寄存器中,接着就到了 RET 指令。



我们可以看到 RET 返回以后,EAX 与 EBX 进行比较,根据比较的结果来决定是否跳转到正确序列号部分。



我们可以看到 EBX 与 EAX 进行比较,EAX 的值为 547B。

由于 EAX 的由程序计算出来的,EBX 是由我们输入的错误序列号计算出来的,由于我们输入的序列号是错误的,所以这两个寄存器不相等。

如果 EBX 等于 EAX 的话,将跳转到正确的序列号提示窗口处,现在两寄存器的值不相等,我们需要分析原因。

笔记如下:

EBX(错误序列号的十六进制值) XOR 1234

我们需要的是 EAX 与 EBX 相等。

如果 EAX=EBX

用 EAX 替换 EBX

也就是说

EAX(正确序列号的十六进制值) XOR 1234

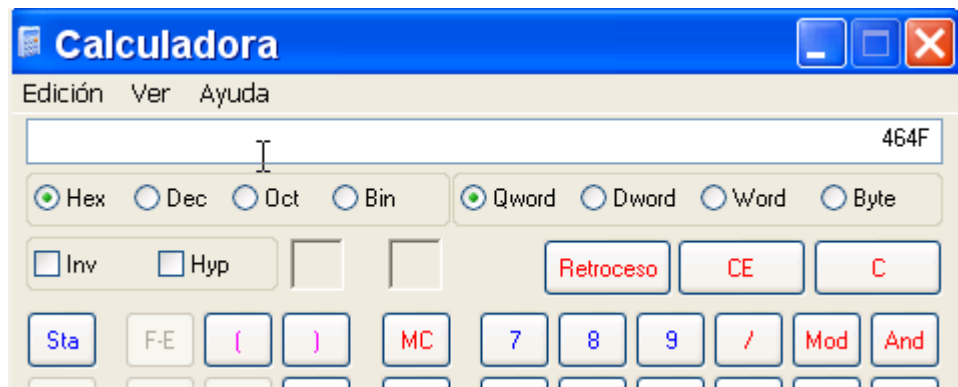
此时 EAX 的值为 547B。我们用这个值替换 EAX。

547B XOR 1234 =

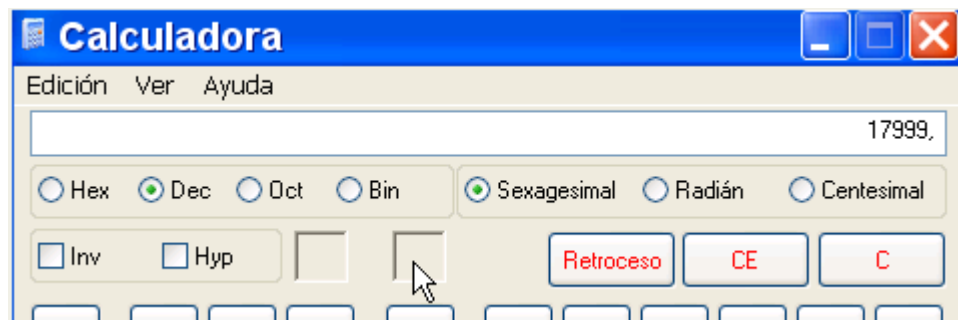
异或运算的结果为:

464F

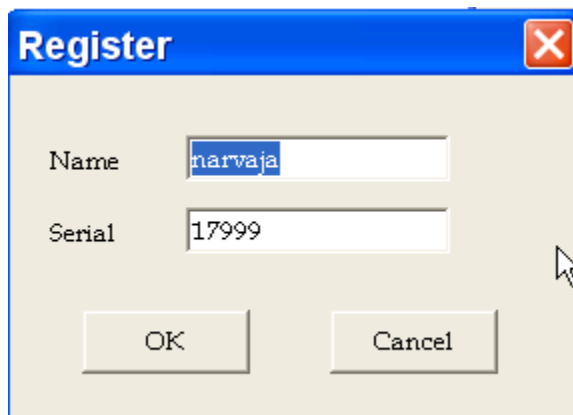
464F 为十六进制值,对应的十进制值为:



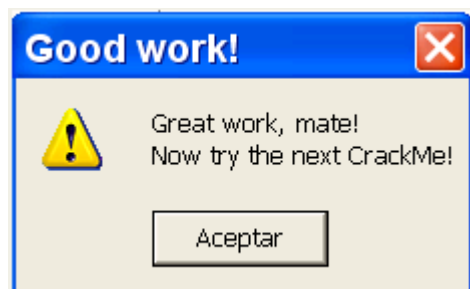
转化为十进制为:



17999 就是"narvaja"这个名称对应的正确序列号。我们删除之前设置的所有断点。



单击 OK 按钮。



这样我们就得到了我们输入名称所对应的正确序列号了。

这是一种解决方案,另一种解决方案如下:

如果我们对错误序列号进行的一系列操作称之为函数 F。

$F(\text{错误序列号}) = \text{EBX}$

对错误序列号进行操作的结果被存放到 EBX 中。

与之进行比较的 EAX 寄存器,我们的公式如下:

F(正确序列号) = EAX

正确序列号进行一系列操作结果保存到 EAX 中。

为了得到正确的序列号,我们对 EAX 进行反向的操作。

正确序列号 = 反向 F(EAX)

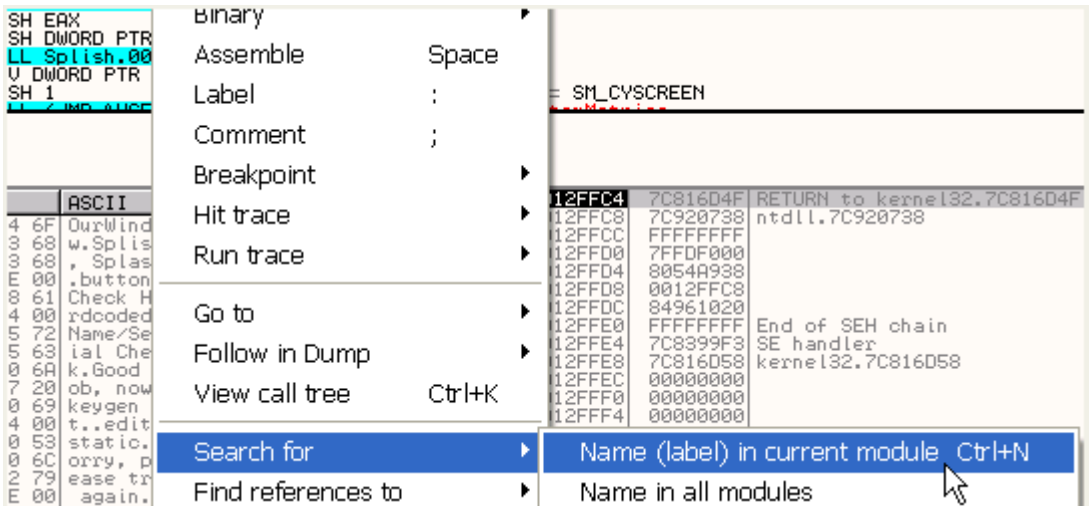
当前情况下,反向操作依然是异或。

因此 XOR EAX 的结果就是正确序列号的十六进制值,我们将其转化为十进制值就得到了正确的序列号。

因此,我们也可以利用这种方法来获取正确的序列号,除非反向操作是不等价的,我们来看一个这样的例子。

接下来的例子是 Splish,这个 CrackMe 的第一部分是找到硬编码序列号,现在我们来看看它的第二部分。

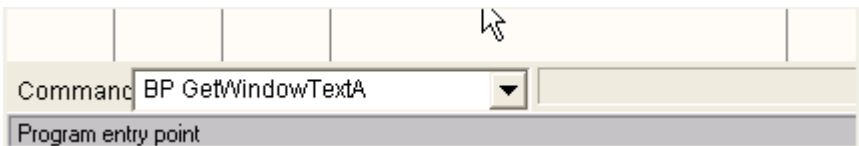
加载 Splish,断在入口点处。



看看程序使用了哪些 API 函数。

00402010	.rdata	Import	KERNEL32.GetModuleHandleA	
00402034	.rdata	Import	USER32.GetSystemMetrics	
0040200C	.rdata	Import	KERNEL32.GetTickCount	
00402024	.rdata	Import	USER32.GetWindowTextA	
00402020	.rdata	Import	USER32.LoadBitmapA	
00402038	.rdata	Import	USER32.LoadCursorA	
00402028	.rdata	Import	USER32.LoadIconA	

给我们熟悉的 GetWindowTextA 设置断点。



我们按 F9 键运行起来。



我们随便输入一个错误的名称和序列号,然后单击 Name/Serial Check 按钮。

77D3213C	6A 0C	PUSH 0C
77D3213E	68 A021D377	PUSH USER32.77D321A0
77D32143	E8 7864FEFF	CALL USER32.77D185C0
77D32148	8B7D 0C	MOV EDI,DWORD PTR SS:[EBP+C]
77D3214B	33DB	XOR EBX,EBX
77D3214D	3BFB	CMP EDI,EBX

断在了 GetWindowTextA 这个 API 函数入口处,我们来看看缓冲区。

0012FC64	004015F6	CALL to GetWindowTextA from Splish.004015F1
0012FC68	002D063E	hWnd = 002D063E (class='Edit',parent=00410624)
0012FC6C	00403242	Buffer = Splish.00403242
0012FC70	00000020	Count = 20 (32.)
0012FC74	0012FC84	
0012FC78	00401407	RETURN to Splish.00401407 from Splish.004015E4

我们在数据窗口中定位到该缓冲区。

0012FC64	004015F6	CALL to GetWindowTextA from Splish.004015F1
0012FC68	002D063E	hWnd = 002D063E (class='Edit',parent=00410624)
0012FC6C	00403242	Buffer = Splish.00403242
0012FC70	00000020	Count =
0012FC74	0012FC84	
0012FC78	00401407	RETURN to
0012FC7C	001206A2	
0012FC80	002D063E	
0012FC84	0012FCB0	
0012FC88	77D18734	RETURN to
0012FC8C	00410624	Splish.00
0012FC90	00000111	
0012FC94	00000002	
0012FC98	00190648	
0012FC9C	00401178	Splish.00
0012FCA0	DCBAABCD	
0012FCA4	00000000	
0012FCA8	0012FCEC	
0012FCAC	00401178	Splish.00
0012FCB0	0012FD18	
0012FCB4	77D18816	RETURN to
0012FCB8	00401178	Splish.00
0012FCBC	00410624	Splish.00
0012FCC0	00000111	
0012FCC4	00000002	
0012FCC8	00190648	
0012FCCC	00000111	
0012FCD0	00733058	
0012FCD4	0073306C	
0012FCD8	00000014	
0012FCDC	00000001	
0012FCE0	00000000	
0012FCE4	00000000	
0012FCE8	00000010	
0012FCEC	00000000	
0012FCF0	77D1B4C0	RETURN to
0012FCF4	00000001	
0012FCF8	00000000	
0012FCFC	00000000	
0012FD00	0012FCCC	

Address

Show ASCII dump

Show UNICODE dump

Lock stack

Copy to clipboard

Ctrl+C

Modify

Edit

Ctrl+E

Push DWORD

Pop DWORD

Search for address

Search for binary string

Ctrl+B

Go to ESP

*

Go to EBP

Go to expression

Ctrl+G

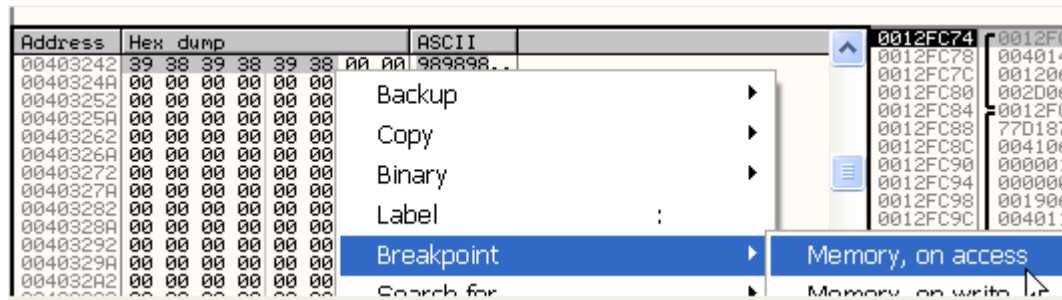
Follow in Dump

Address	Hex dump	ASCII
00403242	00 00 00 00 00 00 00 00
00403244	00 00 00 00 00 00 00 00
00403246	00 00 00 00 00 00 00 00
00403248	00 00 00 00 00 00 00 00
0040324A	00 00 00 00 00 00 00 00
0040324C	00 00 00 00 00 00 00 00
0040324E	00 00 00 00 00 00 00 00
00403250	00 00 00 00 00 00 00 00
00403252	00 00 00 00 00 00 00 00
00403254	00 00 00 00 00 00 00 00
00403256	00 00 00 00 00 00 00 00
00403258	00 00 00 00 00 00 00 00
0040325A	00 00 00 00 00 00 00 00
0040325C	00 00 00 00 00 00 00 00
0040325E	00 00 00 00 00 00 00 00
00403260	00 00 00 00 00 00 00 00
00403262	00 00 00 00 00 00 00 00
00403264	00 00 00 00 00 00 00 00
00403266	00 00 00 00 00 00 00 00
00403268	00 00 00 00 00 00 00 00
0040326A	00 00 00 00 00 00 00 00

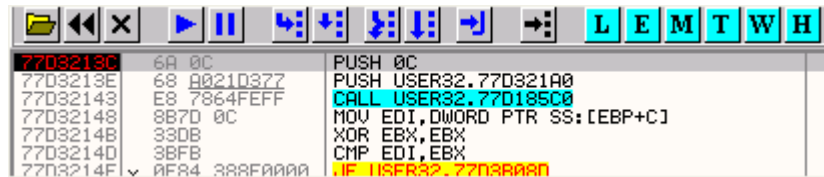
缓冲区里面是空的,我们选择主菜单中的 Debug-Execute till return。就会停在 ret 指令处,接着我们按 F7 键返回到主程序模块中。

Address	Hex dump	ASCII
00403242	39 38 39 38 39 38 00 00	989898..
00403244	00 00 00 00 00 00 00 00
00403246	00 00 00 00 00 00 00 00
00403248	00 00 00 00 00 00 00 00
0040324A	00 00 00 00 00 00 00 00
0040324C	00 00 00 00 00 00 00 00
0040324E	00 00 00 00 00 00 00 00
00403250	00 00 00 00 00 00 00 00
00403252	00 00 00 00 00 00 00 00
00403254	00 00 00 00 00 00 00 00
00403256	00 00 00 00 00 00 00 00
00403258	00 00 00 00 00 00 00 00
0040325A	00 00 00 00 00 00 00 00
0040325C	00 00 00 00 00 00 00 00
0040325E	00 00 00 00 00 00 00 00
00403260	00 00 00 00 00 00 00 00
00403262	00 00 00 00 00 00 00 00
00403264	00 00 00 00 00 00 00 00
00403266	00 00 00 00 00 00 00 00
00403268	00 00 00 00 00 00 00 00
0040326A	00 00 00 00 00 00 00 00

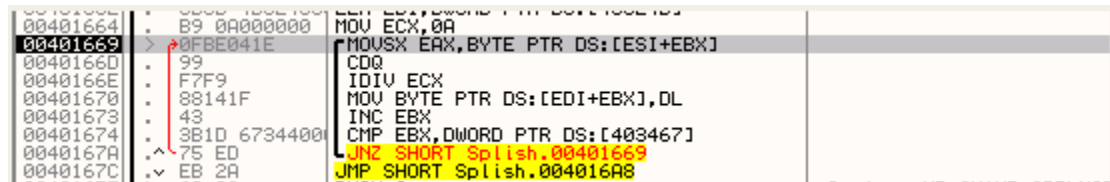
可以看到缓冲区中保存了错误的序列号,我们对错误的序列号设置内存访问断点。



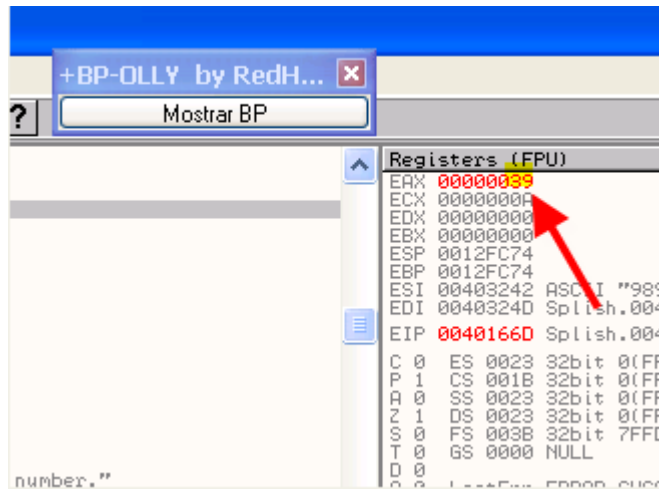
运行起来。



再次断在了 GetWindowTextA 这个 API 函数入口处,这里获取的是窗体的名称,我们不感兴趣,继续运行起来。



断在了这个函数里面,如果我们执行这行代码的话,会发现将错误序列号的第一个字节移动到 EAX 中了。



接下来我们看看下面的 CDQ 指令的解释,我们直接来看 Google 里面的解释,我们在谷歌中直接搜索 CDQ Assembler。

例如如下指令:

CDQ

IDIV ESI

CDQ 指令双字扩展,把 EAX 中的符号位扩展到 EDX 中去,然后 EDX:EAX 对应的值除以 ESI,商保存到 EAX 中,余数保存到 EDX 中。

EAX 符号位扩展到 EDX 中,EDX 的值应该变为零,相当于对 EDX 进行 XOR EDX,EDX 操作。现在不需要将 EDX 清零了,因为 CDQ 指令已经帮我们完成了该操作。

所以当前情况下我们不必每次循环之前将 EDX 赋值为零,我们只需要在 IDIV 指令之前加上一个 CDQ 指令即可。

EDX:EAX 除以 ECX,商存放在 EAX 中,余数存放到 EDX 中。好了,我们现在来看看具体的实现。

第一个字节为 39,除以 ECX(值为 0A)。

Registers (FPU)	
EAX	00000039
ECX	0000000A
EDX	00000000
EBX	00000000
ESP	0012FC74
EBP	0012FC74
ESI	00403242 ASC
EDI	0040324D Spli
EIP	0040166E Spli
C 0	ES 0023 32E
P 1	CS 001B 32E

看看发生了什么。

Registers (FPU)	
EAX	00000005
ECX	0000000A
EDX	00000007
EBX	00000000
ESP	0012FC74
EBP	0012FC74
ESI	00403242 ASCII
EDI	0040324D Spli
EIP	00401670 Spli

这里商 5 被保存到了 EAX 中,余数 7 被保存到了 EDX 中。

0040165E	. 8D3D 4D324001	LEA EDI,DWORD PTR DS:[40324D]
00401664	. B9 0A000000	MOV ECX,0A
00401669	> 0FBF041E	MOVSX EAX,BYTE PTR DS:[ESI+EBX]
0040166D	. 99	CDQ
0040166E	. F7F9	IDIV ECX
00401670	. 8B141F	MOV BYTE PTR DS:[EDI+EBX],DL
00401673	. 43	INC EBX
00401674	. 3B1D 67344001	CMP EBX,DWORD PTR DS:[403467]
0040167A	. ^ 75 ED	JNZ SHORT Splish.00401669
0040167C	. v EB 2A	JMP SHORT Splish.004016A8
0040167E	> 6A 00	PUSH 0

可以看到下一行将 DL 的值保存到 40324D 指向的内存单元中。在数据窗口中定位到 40324D 这个内存单元。

004016A8	>	EB 4D	JMP	
004016A8		8D35 4D324001	LEA	
004016AF		053D 50324001	LEA	
DL=07				
DS:[0040324D]=00				
Address Hex dump				
Address	Hex dump			ASCII
0040324D	00	00	00 00 00 00 00 00
00403255	00	00	00 02 08 08 03 05	...0000+
0040325D	05	03	00 00 00 00 00 00	+0.....
00403265	00	00	00 00 00 00 00 00

继续按 F7 键。

Address Hex dump			
Address	Hex dump		ASCII
0040324D	07 00 00 00 00 00 00 00	
00403255	00 00 00 02 08 08 03 05	...0000+	
0040325D	05 03 00 00 00 00 00 00	+0.....	
00403265	00 00 00 00 00 00 00 00	
0040326D	00 00 00 00 00 00 00 00	

7 被保存到了该内存单元中。

00401670	. 8B141F	MOV BYTE PTR DS:[EDI+EBX],DL
00401673	. 43	INC EBX
00401674	. 3B1D 67344001	CMP EBX,DWORD PTR DS:[403467]
0040167A	. ^ 75 ED	JNZ SHORT Splish.00401669
0040167C	. v EB 2A	JMP SHORT Splish.004016A8
004016A8	> 8D35 4D324001	LEA
004016AF	. 053D 50324001	LEA
DS:[00403467]=00000006		
EBX=00000001		

我们可以 EBX 的值为零,然后递增 1,接着与 6 进行比较,如果不相等将继续循环。

00401664	. 89 0A000000	MOV ECX,0A
00401665	> 0FBF041E	MOVSX EAX,BYTE PTR DS:[ESI+EBX]
0040166D	. 99	CDQ
0040166E	. F7F9	IDIV ECX
00401670	. 88141F	MOV BYTE PTR DS:[EDI+EBX],DL
00401673	. 43	INC EBX
00401674	. 3B1D 67344000	CMP EBX,DWORD PTR DS:[403467]
0040167A	. 75 ED	JNZ SHORT Splish.00401669
0040167C	. EB 2A	JMP SHORT Splish.004016A8
0040167E	> 6A 00	PUSH 0
00401680	. 68 0A304000	PUSH Splish.0040300A
004016A8	> 8D35 4D3240	LEA ESI,DWORD PTR DS:[403240]
004016AF	. 0000 00000005	MOV EAX,5
DS:[00403243]=38 ('8')		
EAX=00000005		
Jump from 0040167A		
Address	Hex dump	

现在我来看看第二个字节将发生什么。

Registers (FPU)	
EAX	00000038
ECX	0000000A
EDX	00000000
EBX	00000001
ESP	0012FC74
EBP	0012FC74
ESI	00403242 AS
EDI	0040324D Sp
EIP	0040166E Sp
C 1	ES 0023 32

通过是 EDX:EAX 除以 0A,商保存到 EAX 中,余数保存到 EDX 中。

Registers (FPU)	
EAX	00000005
ECX	0000000A
EDX	00000006
EBX	00000001
ESP	0012FC74
EBP	0012FC74
ESI	00403242 ASCII
EDI	0040324D Splish
EIP	00401670 Splish
C 1	ES 0023 32bit
P 0	CS 001B 32bit
A 1	SS 0023 32bit

0040166E	. F7F9	IDIV ECX
00401670	. 88141F	MOV BYTE PTR DS:[EDI+EBX],DL
00401673	. 43	INC EBX
00401674	. 3B1D 67344000	CMP EBX,DWORD PTR DS:[403467]
0040167A	. 75 ED	JNZ SHORT Splish.00401669
0040167C	. EB 2A	JMP SHORT Splish.004016A8
0040167E	> 6A 00	PUSH 0
00401680	. 68 0A304000	PUSH Splish.0040300A
00401685	. 68 A0304000	PUSH Splish.004030A0
0040168A	. 6A 00	PUSH 0
0040168C	. E8 B7000000	CALL <JMP.&USER32.MessageBoxA>
00401691	. EB 62	JMP SHORT Splish.004016F5
00401693	> 6A 00	PUSH 0
00401695	. 68 0A304000	PUSH Splish.0040300A
0040169A	. 68 B8304000	PUSH Splish.004030B8
0040169F	. 6A 00	PUSH 0
004016A1	. E8 A2000000	CALL <JMP.&USER32.MessageBoxA>
004016A6	. EB 4D	JMP SHORT Splish.004016F5
004016A8	> 8D35 4D324000	LEA ESI,DWORD PTR DS:[403240]
004016AF	. 0000 00000005	MOV EAX,5

DL=06
DS:[0040324E]=00

Address	Hex dump	ASCII
0040324D	07 00 00 00 00 00 00 00
00403255	00 00 00 02 08 08 03 05	...00000002
0040325D	05 03 00 00 00 00 00 00	0000000503000000
00403265	00 00 00 00 00 00 00 00
0040326D	00 00 00 00 00 00 00 00
00403275	00 00 00 00 00 00 00 00

保存余数。

Address	Hex dump	ASCII
00403240	07 06 00 00 00 00 00 00	..+.....
00403255	00 00 00 02 08 08 03 05	...88888
0040325D	05 03 00 00 00 00 00 00	..+.....
00403265	00 00 00 00 00 00 00 00
0040326D	00 00 00 00 00 00 00 00
00403275	00 00 00 00 00 00 00 00

好了,对所有字节进行以上操作。

Address	Hex dump	ASCII
00403240	07 06 07 06 07 06 00 00	..+..+..
00403255	00 00 00 02 08 08 03 05	...88888
0040325D	05 03 00 00 00 00 00 00	..+.....
00403265	00 00 00 00 00 00 00 00
0040326D	00 00 00 00 00 00 00 00
00403275	00 00 00 00 00 00 00 00

接着 JNZ 指令跳转没有发生退出循环。

00401673	43	INC EBX	
00401674	3B1D 67344001	CMP EBX,DWORD PTR DS:[403467]	
0040167D	75 ED	JNZ SHORT Splish.00401683	
0040167C	EB 2A	JMP SHORT Splish.004016A8	
0040167E	6A 00	PUSH 0	
00401680	68 0A304000	PUSH Splish.0040300A	
00401685	68 A0304000	PUSH Splish.004030A0	
0040168A	6A 00	PUSH 0	
0040168C	E8 B7000000	CALL <JMP.&USER32.MessageBoxA>	Style = MB_OK!MB_APPLMODAL
00401691	EB 62	JMP SHORT Splish.004016F5	Title = "Splish, Splash"
00401693	6A 00	PUSH 0	Text = "Please enter your name."
00401695	68 0A304000	PUSH Splish.0040300A	hOwner = NULL
0040169A	68 B0304000	PUSH Splish.004030B8	MessageBoxA
0040169F	6A 00	PUSH 0	
004016A1	E8 A2000000	CALL <JMP.&USER32.MessageBoxA>	Style = MB_OK!MB_APPLMODAL
004016A6	EB 4D	JMP SHORT Splish.004016F5	Title = "Splish, Splash"
004016A8	8D35 4D324001	LEA ESI,DWORD PTR DS:[40324D]	Text = "Please enter your serial number."
004016AE	8D3D 58324001	LEA EDI,DWORD PTR DS:[403258]	hOwner = NULL

接下来我们看到正确序列号以及错误序列号的消息框代码,说明离找到正确序列号已经不远了。

004016A1	E8 A2000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
004016A6	EB 4D	JMP SHORT Splish.004016F5	
004016A8	8D35 4D324001	LEA ESI,DWORD PTR DS:[40324D]	
004016AE	8D3D 58324001	LEA EDI,DWORD PTR DS:[403258]	
004016B4	33DB	XOR EBX,EBX	
004016B6	3B1D 63344001	CMP EBX,DWORD PTR DS:[403463]	
004016BC	74 0F	JE SHORT Splish.004016C0	
004016BE	0FB8041F	MOVSB EBX,EBX	
004016C2	0FB80C1E	MOVSB ECX,ECX	
004016C6	3BC1	CMP EAX,ECX	
004016C8	75 18	JNZ SHORT Splish.004016E2	
004016CA	43	INC EBX	
004016CB	EB E9	JMP SHORT Splish.004016B6	
004016CD	6A 00	PUSH 0	
004016CF	68 0A304000	PUSH Splish.0040300A	
004016D4	68 42304000	PUSH Splish.00403042	
004016D9	6A 00	PUSH 0	
004016DB	E8 68000000	CALL <JMP.&USER32.MessageBoxA>	Style = MB_OK!MB_APPLMODAL
004016E0	EB 13	JMP SHORT Splish.004016F5	Title = "Splish, Splash"
004016E2	6A 00	PUSH 0	Text = "Good job, now keygen it."
004016E4	68 0A304000	PUSH Splish.0040300A	hOwner = NULL
004016E9	68 67304000	PUSH Splish.00403067	MessageBoxA
004016EE	6A 00	PUSH 0	
004016F0	E8 53000000	CALL <JMP.&USER32.MessageBoxA>	Style = MB_OK!MB_APPLMODAL
004016F5	C9	LEAVE	Title = "Splish, Splash"
004016F6	C2 0800	RET 8	Text = "Sorry, please try again."

接下来 LEA 指令分别将两个地址保存到 ESI,EDI 寄存器中。

Registers (FPU)	
EAX	00000005
ECX	0000000A
EDX	00000006
EBX	00000006
ESP	0012FC74
EBP	0012FC74
ESI	0040324D Sp
EDI	00403258 Sp
EIP	004016B4 Sp

ESI 指向了保存刚刚运行的结果,EDI 指向哪里呢?嘿嘿

Address	Hex dump	ASCII
00403240	07 06 07 06 07 06 00 00	..+..+..
00403255	00 00 00 02 08 08 03 05	...88888
0040325D	05 03 00 00 00 00 00 00	..+.....
00403265	00 00 00 00 00 00 00 00
0040326D	00 00 00 00 00 00 00 00
00403275	00 00 00 00 00 00 00 00
0040327D	00 00 00 00 00 00 00 00

继续

```

004016AE . 803D 58324000 LEA EDI,DWORD PTR DS:[403258]
004016B4 . 33DB XOR EBX,EBX
004016B6 > 3B1D 63344000 CMP EBX,DWORD PTR DS:[403463]
004016BC . 74 0F JE SHORT Splish.004016CD
004016BE . 0FBF041F MOVSX EAX,BYTE PTR DS:[EDI+EBX]
004016C2 . 0FBF0C1E MOVSX ECX,BYTE PTR DS:[ESI+EBX]
004016C6 . 3BC1 CMP EAX,ECX
004016C8 . 75 18 JNZ SHORT Splish.004016E2
004016CA . 43 INC EBX
004016CB . EB E9 JMP SHORT Splish.004016B6
004016CD > 6A 00 PUSH 0
004016CF . 68 0A304000 PUSH Splish.0040300A
004016D4 . 68 42304000 PUSH Splish.00403042
004016D9 . 6A 00 PUSH 0
004016DB . E8 68000000 CALL <JMP.&USER32.MessageBoxA>
004016E0 . EB 13 JMP SHORT Splish.004016F5
004016E2 > 6A 00 PUSH 0
004016E4 . 68 0A304000 PUSH Splish.0040300A
004016E9 . 68 67304000 PUSH Splish.00403067
004016EE . 6A 00 PUSH 0
004016F0 . E8 53000000 CALL <JMP.&USER32.MessageBoxA>
004016F5 > C9 LEAVE
004016F6 . C2 0800 RETN 8
004016F9 . CC INT3
004016FA $- FF25 70204000 JMP DWORD PTR DS:[<&USER32.BeginPal
DS:[00403463]=00000007
EBX=00000000
Jump from 004016CB

```

可以 EBX 值为零,与 7 进行比较,如果它们相等...

```

004016AE . 803D 58324000 LEA EDI,DWORD PTR DS:[403258]
004016B4 . 33DB XOR EBX,EBX
004016B6 > 3B1D 63344000 CMP EBX,DWORD PTR DS:[403463]
004016BC . 74 0F JE SHORT Splish.004016CD
004016BE . 0FBF041F MOVSX EAX,BYTE PTR DS:[EDI+EBX]
004016C2 . 0FBF0C1E MOVSX ECX,BYTE PTR DS:[ESI+EBX]
004016C6 . 3BC1 CMP EAX,ECX
004016C8 . 75 18 JNZ SHORT Splish.004016E2
004016CA . 43 INC EBX
004016CB . EB E9 JMP SHORT Splish.004016B6
004016CD > 6A 00 PUSH 0
004016CF . 68 0A304000 PUSH Splish.0040300A
004016D4 . 68 42304000 PUSH Splish.00403042
004016D9 . 6A 00 PUSH 0
004016DB . E8 68000000 CALL <JMP.&USER32.MessageBoxA>
004016E0 . EB 13 JMP SHORT Splish.004016F5

```

Style = MB_OK!MB_APPLMODAL
 Title = "Splish, Splash"
 Text = "Good job, now keygen it."
 hOwner = NULL
 MessageBoxA

将跳转到正确序列号的提示框处。中间的循环体中还有另一个 JNZ 指令会跳转到错误序列号的消息框处。

```

004016AE . 803D 58324000 LEA EDI,DWORD PTR DS:[403258]
004016B4 . 33DB XOR EBX,EBX
004016B6 > 3B1D 63344000 CMP EBX,DWORD PTR DS:[403463]
004016BC . 74 0F JE SHORT Splish.004016CD
004016BE . 0FBF041F MOVSX EAX,BYTE PTR DS:[EDI+EBX]
004016C2 . 0FBF0C1E MOVSX ECX,BYTE PTR DS:[ESI+EBX]
004016C6 . 3BC1 CMP EAX,ECX
004016C8 . 75 18 JNZ SHORT Splish.004016E2
004016CA . 43 INC EBX
004016CB . EB E9 JMP SHORT Splish.004016B6
004016CD > 6A 00 PUSH 0
004016CF . 68 0A304000 PUSH Splish.0040300A
004016D4 . 68 42304000 PUSH Splish.00403042
004016D9 . 6A 00 PUSH 0
004016DB . E8 68000000 CALL <JMP.&USER32.MessageBoxA>
004016E0 . EB 13 JMP SHORT Splish.004016F5
004016E2 > 6A 00 PUSH 0
004016E4 . 68 0A304000 PUSH Splish.0040300A
004016E9 . 68 67304000 PUSH Splish.00403067
004016EE . 6A 00 PUSH 0
004016F0 . E8 53000000 CALL <JMP.&USER32.MessageBoxA>
004016F5 > C9 LEAVE
004016F6 . C2 0800 RETN 8

```

Style = MB_OK!MB_APPLMODAL
 Title = "Splish, Splash"
 Text = "Good job, now keygen it."
 hOwner = NULL
 MessageBoxA

Style = MB_OK!MB_APPLMODAL
 Title = "Splish, Splash"
 Text = "Sorry, please try again."
 hOwner = NULL
 MessageBoxA

好了,我们看看比较。

Address	Hex dump	ASCII
0040324D	07 06 07 06 07 06 00 00	..+..+..
00403255	00 00 00 02 08 08 03 05	...00000
0040325D	05 03 00 00 00 00 00 00	00000000
00403265	00 00 00 00 00 00 00 00	00000000
0040326D	00 00 00 00 00 00 00 00	00000000
00403275	00 00 00 00 00 00 00 00	00000000

EAX 将保存 EDI 指向的内存单元的第二个字节即 02,ECX 将保存之前计算结果的第一个字节 7。

```

004016BE . 0FBF041F MOVSX EAX,BYTE PTR DS:[EDI+EBX]
004016C2 . 0FBF0C1E MOVSX ECX,BYTE PTR DS:[ESI+EBX]
004016C6 . 3BC1 CMP EAX,ECX
004016C8 . 75 18 JNZ SHORT Splish.004016E2
004016CA . 43 INC EBX
004016CB . EB E9 JMP SHORT Splish.004016B6
004016CD > 6A 00 PUSH 0

```

004016FA \$- FF25 702
 ECX=00000007
 EAX=00000002

如果我们计算结果的第一个字节为 02 的话就好了。

我们当前情况是:

$$39-5*0A=7。$$

也就是说除法运算的结果是 $39/0A$ 商为 5,余数为 7。所以我们可以通过反向运算 5 乘以 0A 然后加上 7 得到 39。

$$39=5*0A+7$$

对于正确序列号的情况如下:

$$\text{正确的字节} = 5*0A + 2$$

$$\text{即正确字节} = 32 + 2 = 34 \text{ (注意是十六进制)}$$

34 这个 ASCII 码对应的字符'4'。

我们可以看到,

$$\text{正确字节值} = 5*0A + 2$$

得到是一个(30~39)之间的值,如果结果超出了这个范围,我们可以做如下变换:

$$\text{正确字节值} = 4*0A + \text{平衡值}$$

好了,我们的第一个字符是'4'。

接下来计算剩下的字节。

Address	Hex dump	ASCII
00403240	07 06 07 06 07 06 00 00	.*.*.*.
00403250	00 00 01 02 08 08 03 05	..0000
00403260	05 03 00 00 00 00 00 00	..*
00403270	00 00 00 00 00 00 00 00
00403280	00 00 00 00 00 00 00 00
00403290	00 00 00 00 00 00 00 00
004032A0	00 00 00 00 00 00 00 00
004032B0	00 00 00 00 00 00 00 00

02 已经计算过了

然后是 08。

$$\text{正确字节} = 5 * 0A + 8$$

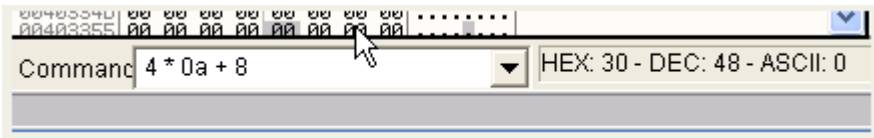
$$\text{即正确字节} = 32 + 8 \text{ (十六进制)}$$

等于 40,超过了 39('9'的 ASCII 码)这个上限,我们按变换的公式计算:

$$\text{正确字节} = 4 * 0A + 8$$

$$\text{即正确字节} = 28 + 8 = 30, \text{即'0'的 ASCII 码。}$$

你也可以在命令栏窗口中验证一下:



因此第二个字节等于 30,即字符'0'的 ASCII 码。

下一个字节依然是 08,所以结果依然是 30,即字符'0'的 ASCII 码。

然后是 03。

$$\text{正确字节} = 5 * 0A + 3$$

$$\text{即正确字节} = 32 + 3 = 35 \text{ (注意是十六进制),即字符'5'的 ASCII 码。}$$

然后是 05。

$$\text{正确字节} = 5 * 0A + 5$$

$$\text{即正确字节} = 32 + 5 = 37, \text{即字符'7'的 ASCII 码。}$$

接着还是 05,正确字节 = 32 + 5 = 37,即字符'7'的 ASCII 码。

最后是 03,前面计算过了字符'5'的 ASCII 码。

因此,名称"narvaja"对应的正确序列号是 4005775。删除之前设置的所有断点运行起来。



单击 Name/Serial Check 按钮。



嘿嘿,这个 CrackMe 就完成了。

留个练习的 CrackMe,名字叫 mexcrk1。