

吉林省首届大学生网络安全大赛 初赛样题及解题步骤（仅供参考）

Crackme

一. 题目信息

1.1 使用工具

OllyDBG

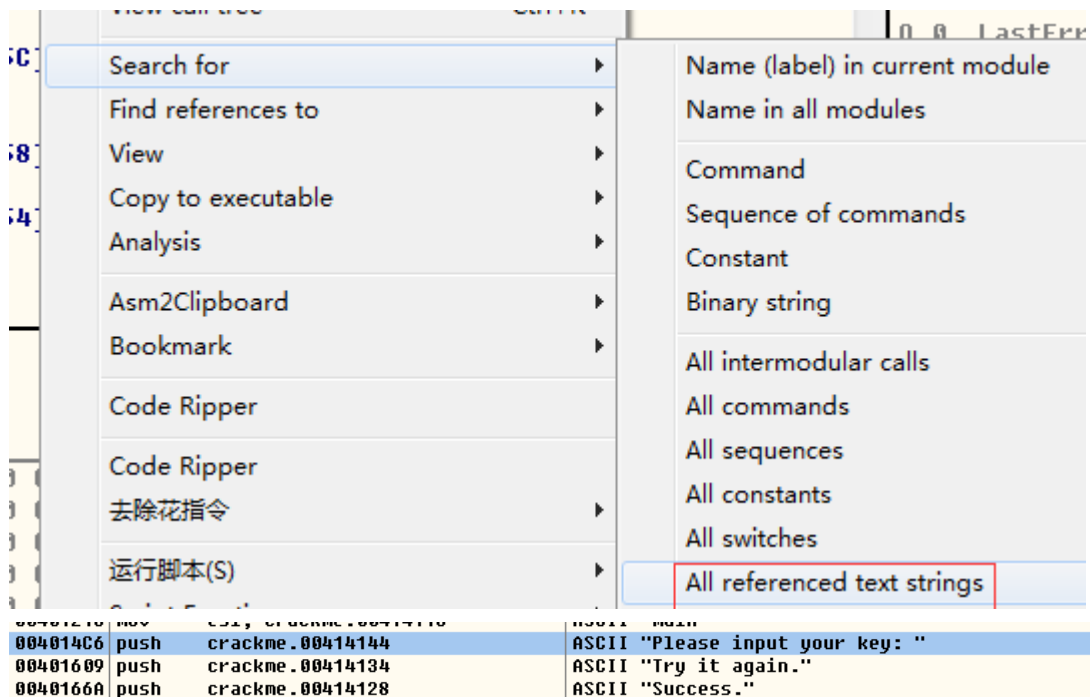
1.2 知识点

1. 对逆向基础知识的了解
2. 对 OllyDBG 的了解以及针对调试软件的使用
3. 对于异或算法的了解

1.3 解题步骤

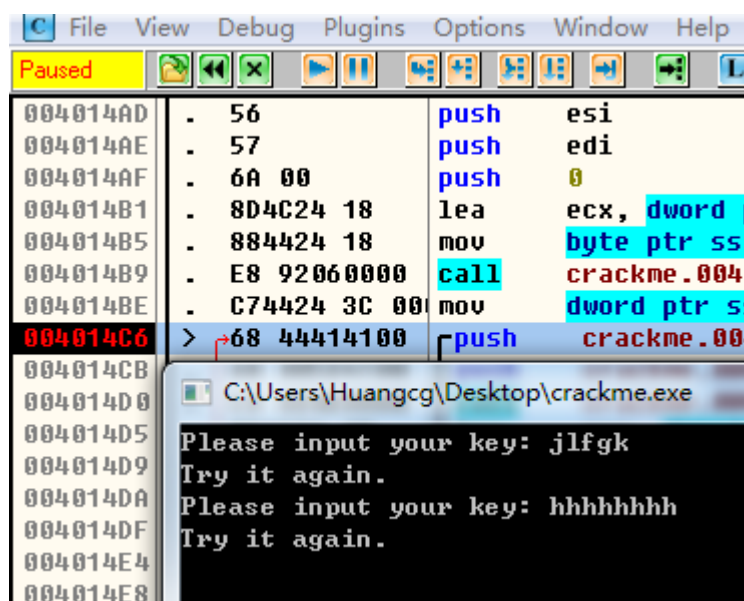
步骤一、 本关卡首先把要分析的 crackme.exe 文件加载到 OD 中。此时它暂停在程序的入口点，按 F9 让它运行起来，发现弹出一个控制台的窗口，由此可见这应该是一个基于控制台的程序。窗口中有一个 “Please input your key:” 的字符串，随便输入一个错误的 key 按回车，出来一个 “Try it again.” (再试一遍) 的字符串，接着提示 “Please input your key:”。

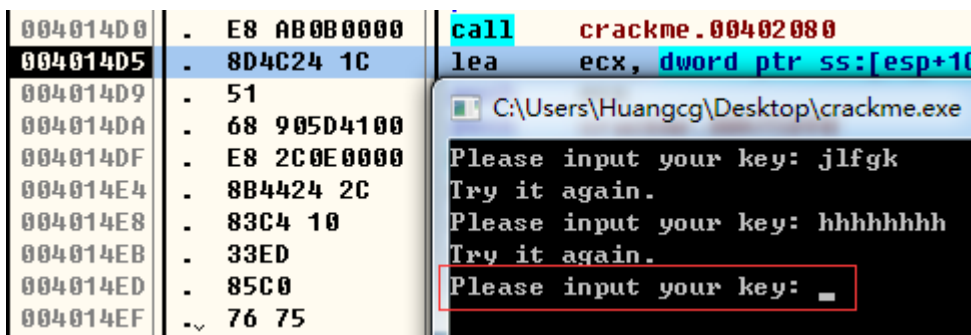
步骤二、 我们可以先尝试从字符串作为入口点去分析该程序，在反汇编窗口中右键--Search for—All referenced text strings



步骤三、在字符串表中能看到我们感兴趣的字符串，双击“Please input your key:”去到在反汇编窗口中调用到该字符串的地址。

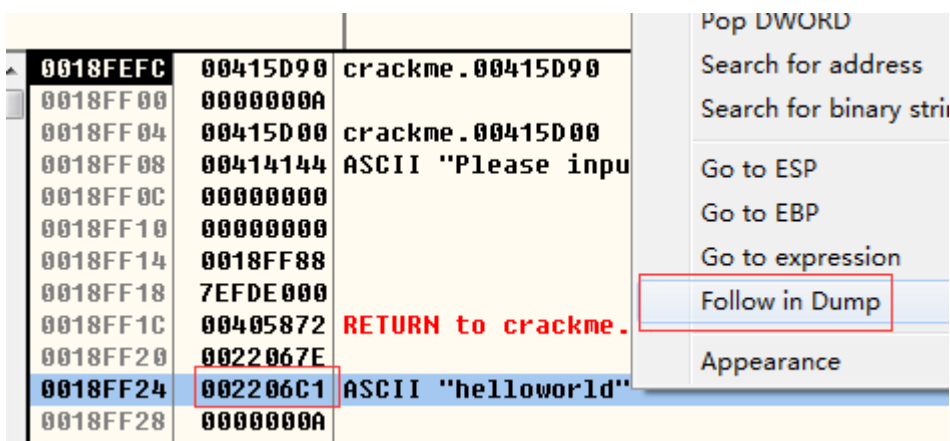
我们可以看到调用该字符串的地方是把字符串地址押入堆栈，紧跟着再往堆栈押入一个参数接着就是一个 `call` 的函数调用，如果我把这个 `call` 看成是一个往控制台输出字符串的函数，那么根据该程序可以推出后面会跟着一个接收键盘输入的函数。我们输入“hhhhhhh”来验证一下猜测是否正确。按下回车的时候程序已经暂停下来，再看窗口中并没有请输入你的 key 的提示。



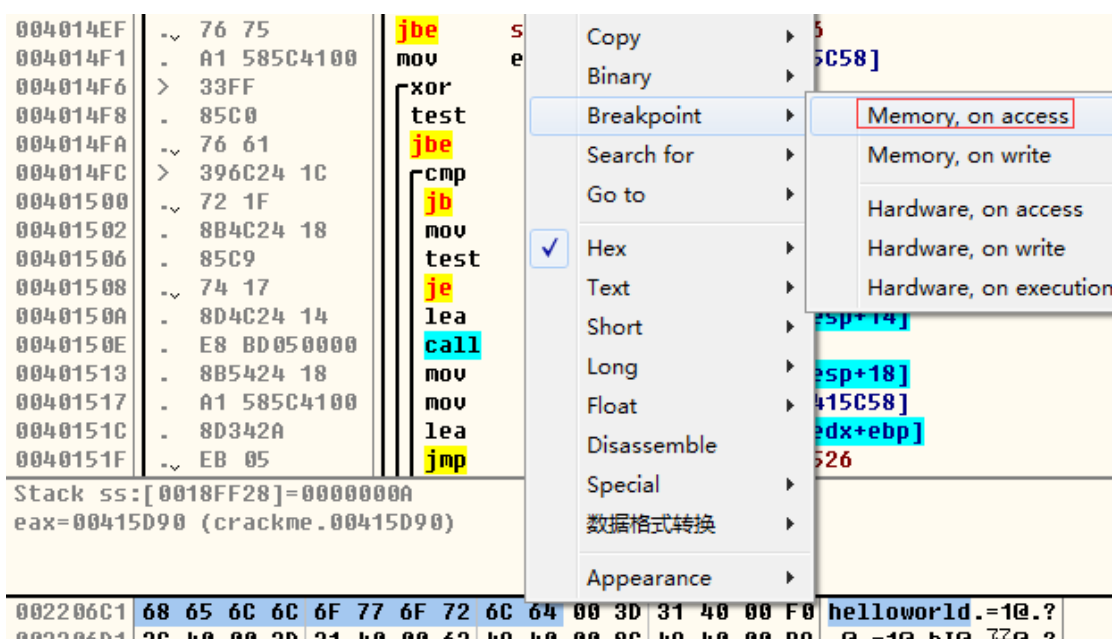


步骤四、一直按 F8 步过该函数，窗口中的提示已经出来了，接着 F8 步过下一个函数，活动状态转到控制台窗口中，等待输入的状态。

步骤五、输入“helloworld”，回车。程序再次暂停下来了，在堆栈窗口 18FF24 处(你们的机器不一定一样)已经看到我输入“helloworld”，现在已经可以确定之前的猜测是正确的了。右键该字符串—Follow in Dump 在数据窗口中定位到 2206C1 的内存单元。



步骤六、给输入的这 10 个字节长的字符串下内存访问断点，选中这 10 个字节，右键—Breakpoint—Memory, on access。按 F9 运行起来。下图



步骤七、 下图,程序断下来的地方是从 esi 读一个字节的数据传送给 al 寄存器(输入字符串的首字母), F8 一下, 把 al 跟 cl 异或运算, 下来的命令再把 al 传送到 esi 保存着的地址的内存单元中。综合信息窗口看到的值, 就是把输入的字符 ‘h’ 跟 ‘m’ 的 ASCII 作异或运算。接着 F9 运行。

0040154B	>	8A08	mov	cl, byte ptr ds:[eax]
0040154D	.	8A06	mov	al, byte ptr ds:[esi]
0040154F	.	32C1	xor	al, cl
00401551	.	47	inc	edi
00401552	.	8806	mov	byte ptr ds:[esi], al
00401554	.	A1 585C4100	mov	eax, dword ptr ds:[415C58]
00401559	.	3BF8	cmp	edi, eax
0040155D	^	72 05	jb	short ex200000_0040155C
<div> <div>cl=6D ('m')</div> <div>al=68 ('h')</div> </div>				
002206C1	68 65 6C 6C 6F 77 6F 72 6C 64 00 3D 31 40 00 F0	helloworld.		
002206D1	7C 40 00 3D 31 40 00 62 40 40 00 8C 40 40 00 B0	a = 1a h1a		

步骤八、 下图,断点位置还是在 40154D 处, F8 运行到异或指令, 从数据窗口能看出来上一步的异或运算结果已经把第一个字符 ‘h’ 十六进制的 ASCII 改成了 05。信息窗口可以看出来, 上一步的运算结果再跟 ASCII 为 61 的字符 ‘a’ 作异或运算, 继续 F9 运行起来。

0040154B	>	8A08	mov	cl, byte ptr ds:[eax]
0040154D	.	8A06	mov	al, byte ptr ds:[esi]
0040154F	.	32C1	xor	al, cl
00401551	.	47	inc	edi
00401552	.	8806	mov	byte ptr ds:[esi], al
00401554	.	A1 585C4100	mov	eax, dword ptr ds:[415C58]
00401559	.	3BF8	cmp	edi, eax
0040155D	^	72 05	jb	short ex200000_0040155C
<div> <div>cl=61 ('a')</div> <div>al=05</div> </div>				
002206C1	05 65 6C 6C 6F 77 6F 72 6C 64 00 3D 31 40 00 F0	helloworld.		
002206D1	7C 40 00 3D 31 40 00 62 40 40 00 8C 40 40 00 B0	a = 1a h1a		

步骤九、 下图,当再次断在读数据的时候, F8 运行一步, 这次发现还是异或, 把运算的结果 ‘d’ 跟 ‘i’ 异或。

0040154B	>	8A08	mov	cl, byte ptr ds:[eax]
0040154D	.	8A06	mov	al, byte ptr ds:[esi]
0040154F	.	32C1	xor	al, cl
00401551	.	47	inc	edi
00401552	.	8806	mov	byte ptr ds:[esi], al
00401554	.	A1 585C4100	mov	eax, dword ptr ds:[415C58]
00401559	.	3BF8	cmp	edi, eax
0040155D	^	72 05	jb	short ex200000_0040155C
<div> <div>cl=69 ('i')</div> <div>al=64 ('d')</div> </div>				
002206C1	64 65 6C 6C 6F 77 6F 72 6C 64 00 3D 31 40 00 F0	delloworld.		
002206D1	7C 40 00 3D 31 40 00 62 40 40 00 8C 40 40 00 B0	a = 1a h1a		

步骤十、 再次运行起来, 这次跟 ‘n’ 异或, 下图。

0040154B	> 8A08	mov	cl, byte ptr ds:[eax]
0040154D	. 8A06	mov	al, byte ptr ds:[esi]
0040154F	. 32C1	xor	al, cl
00401551	. 47	inc	edi
00401552	. 8806	mov	byte ptr ds:[esi], al
00401554	. A1 585C4100	mov	eax, dword ptr ds:[415C58]
00401559	. 3BF8	cmp	edi, eax
0040155B	. 72 0F	jb	short crackme.004014FC

cl=6E ('n')
 al=0D (Carriage Return)

002206C1	6D 65 6C 6C 6F 77 6F 72 6C 64 00 3D 31 40 00 F0	.elloworl
----------	---	-----------

步骤十一、我们耐心地跟踪下去，看看最终运算的结果。F9 继续。下图程序还是断在原来的位置，不过这次有点不一样了，我们看寄存器窗口，esi 的值指向了 2206C2，也就是输入字符串的第二个字符，把这个字符读出来跟 ‘m’ (ASCII 为 6D) 作异或运算。一直按 F9 运行程序，最终发现我们输入的字符串每一个字符都依次跟 “main” 字符串的每一个字符都作异或运算。

0040154F	. 32C1	xor	al, cl
00401551	. 47	inc	edi
00401552	. 8806	mov	byte ptr ds:[esi], al
00401554	. A1 585C	mov	eax, dword ptr ds:[415C58]
00401559	. 3BF8	cmp	edi, eax
0040155B	. 72 0F	jb	short crackme.004014FC

cl=6D ('m')
 al=65 ('e')

002206C1	63 65 6C 6C 6F 77 6F 72 6C 64 00 3D 31 40 00 F0	elloworld.=10.?
----------	---	-----------------

步骤十二、下图是每个字符经过四个异或运算后的结果，已经变成了 “cnggd|dygo”。这里 ecx 为 2，rep 执行两次，把字符串的前 8 个字节拷贝到 edi 的内存地址，下面命令把字符串长度 0A 跟 3 作与运算，结果为 2，接下来还有一个 rep 也执行两次，这次是字节传送，加起来刚好是字符串的总长度。

00401A83	. B8 0011	mov	eax, crackme.00411100
00401A88	> 8B7D 04	mov	edi, dword ptr ss:[ebp+4]
00401A8B	. 8BCB	mov	ecx, ebx
00401A8D	. 03F0	add	esi, eax
00401A8F	. 8BC1	mov	eax, ecx
00401A91	. C1E9 02	shr	ecx, 2
00401A94	. F3:A5	rep	movs dword ptr es:[edi], dword ptr ds:[esi]
00401A96	. 8BC8	mov	ecx, eax
00401A98	. 83E1 03	and	ecx, 3
00401A9B	. F3:A4	rep	movs byte ptr es:[edi], byte ptr ds:[esi]
00401A9D	. 8B4D 04	mov	ecx, dword ptr ss:[ebp+4]

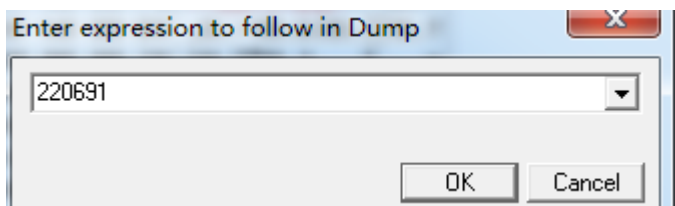
ecx=00000002 (decimal 2.)
 ds:[esi]=002206C1=67676E63
 es:[edi]=00220691=F0000000

002206C1	63 6E 67 67 64 7C 64 79 67 6F 00 3D 31 40 00 F0	cnggd dygo.=10.?
----------	---	------------------

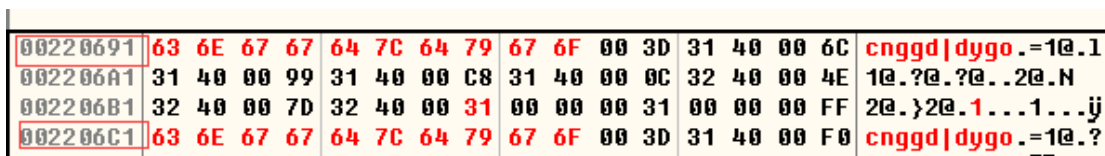
步骤十三、下图，按 F8 一直运行到两个 rep 指令执行完毕，在数据窗口点右键--Go to—Expression 或者按 Ctrl+G。

41 40	Breakpoint	03 07 00	SEKNAME=HuangC
00 00	Search for	00 00 00 55!...1..
4F 40			
53 50	Go to	Expression	Ctrl+G
AA 00			

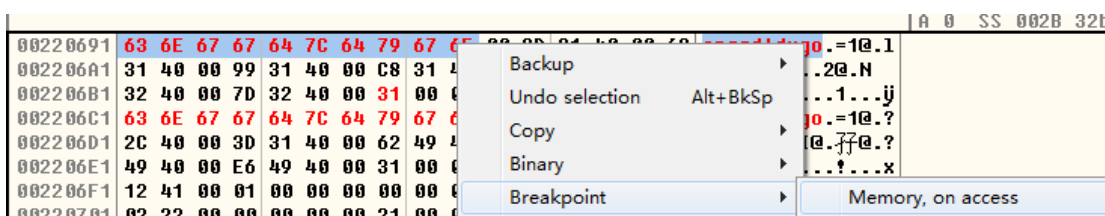
在弹出来的窗口中输入 220691，点 OK。



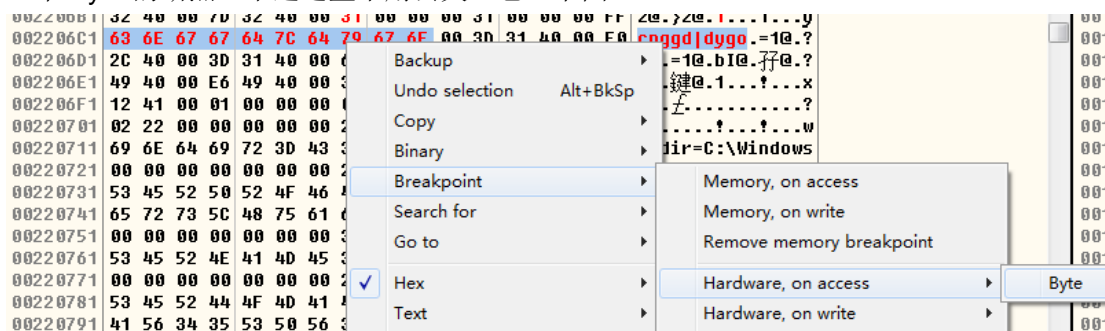
步骤十四、 下图，可以看到经过运算的字符串已经拷贝到了 220691，现在已经有两份同样的数据，我们要继续跟踪程序，看程序对字符串的最终操作的话，理论上应该在相同的数据都下断点，但是内存断点只允许存在一个。



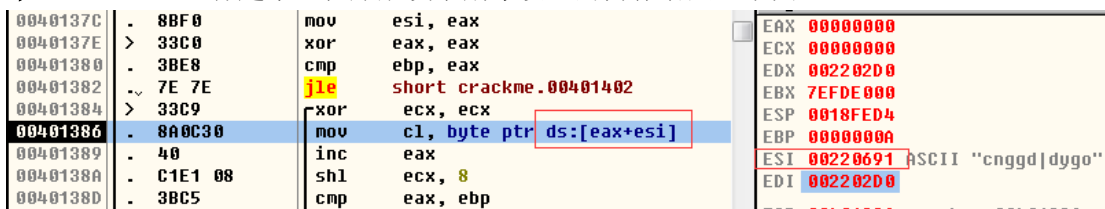
步骤十五、 因此，在 220691 处的字符串下内存访问断点，选中 -- 右键 —Breakpoint—Memory,on access, 下图。



步骤十六、 在 2206C1 的内存处下硬件访问断点，选中—右键--Breakpoint--Hardware,on access--Byte,因为硬件断点都是以 4 为边界取整的，2206C1 余 4 等于 1，所以，只能下一个 Byte 的断点，不过这里不用去关心它。下图。



步骤十七、 F9 运行起来，程序触发了刚才设置的内存断点， 下图



步骤十八、 这段代码看似又是读出字符串来来计算，看起来不简单，先别管它怎么运算的，一直 F8 单步下来，主要观察计算结果保存的地方。

步骤十九、 下图，可以看到整个计算的代码块，都只对 edi 进行赋值，由此可以判定经过这一系列的运算，或者说是加密更贴切一点，edi 中的内存单元保存的就是加密后的字符串。

004013B0	. 8A9B B0404100	mov bl, byte ptr ds:[ebx+4140B0]
004013B6	. 881F	mov byte ptr ds:[edi], bl
004013B8	. 8BD9	mov ebx, ecx
004013BA	. C1FB 0C	sar ebx, 0C
004013BD	. 83E3 3F	and ebx, 3F
004013C0	. 8A9B B0404100	mov bl, byte ptr ds:[ebx+4140B0]
004013C6	. 885F 01	mov byte ptr ds:[edi+1], bl
004013C9	. 8BD9	mov ebx, ecx
004013CB	. C1FB 06	sar ebx, 6
004013CE	. 83E3 3F	and ebx, 3F
004013D1	. 83E1 3F	and ecx, 3F
004013D4	. 3BC5	cmp eax, ebp
004013D6	. 8A9B B0404100	mov bl, byte ptr ds:[ebx+4140B0]
004013DC	. 885F 02	mov byte ptr ds:[edi+2], bl
004013DF	. 8A89 B0404100	mov cl, byte ptr ds:[ecx+4140B0]
004013E5	. 884F 03	mov byte ptr ds:[edi+3], cl
004013E8	. 7E 04	jle short crackme.004013EE
004013EA	. C647 03 3D	mov byte ptr ds:[edi+3], 3D
004013EE	> 8D4D 01	lea ecx, dword ptr ss:[ebp+1]
004013F1	. 3BC1	cmp eax, ecx
004013F3	. 7E 04	jle short crackme.004013F9
004013F5	. C647 02 3D	mov byte ptr ds:[edi+2], 3D
004013F9	> 83C7 04	add edi, 4

步骤二十、 在 edi 上右键--Follow in Dump,跟踪到数据窗口。下图

EDX	002202D0	ASCII "Y25"
EBX	00000035	
ESP	0018FED4	
EBP	0000000A	
ESI	00220691	ASCII "cnggd dygo"
EDI	002202D0	ASCII "Y25"
EIP	004013E8	Increment
C 1	ES	Decrement
P 1	CS	Zero
A 1	SS	Set to 1
Z 0	DS	Modify
S 1	FS	
T 0	GS	Copy selection to clipboard
D 0		Copy all registers to clipboard
O 0	Last	
EFL	00000000	Follow in Dump
ST0	empty	View MMX registers
ST1	empty	View 3DNow! registers
ST2	empty	
ST3	empty	View debug registers

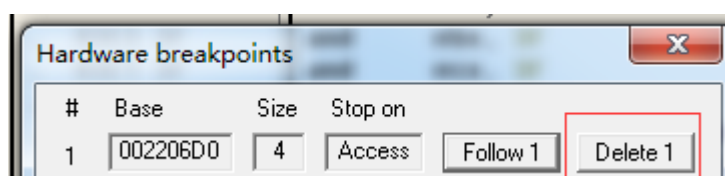
步骤二十一、 下图，执行到 4013DF 的时候，已经在 2202D0 处写入了 3 个字节的数据。

我们可以看到整个加密运算代码块到 4013FE 就结束了，在结束的下一行代码 401400 处下个断点（按 F2），

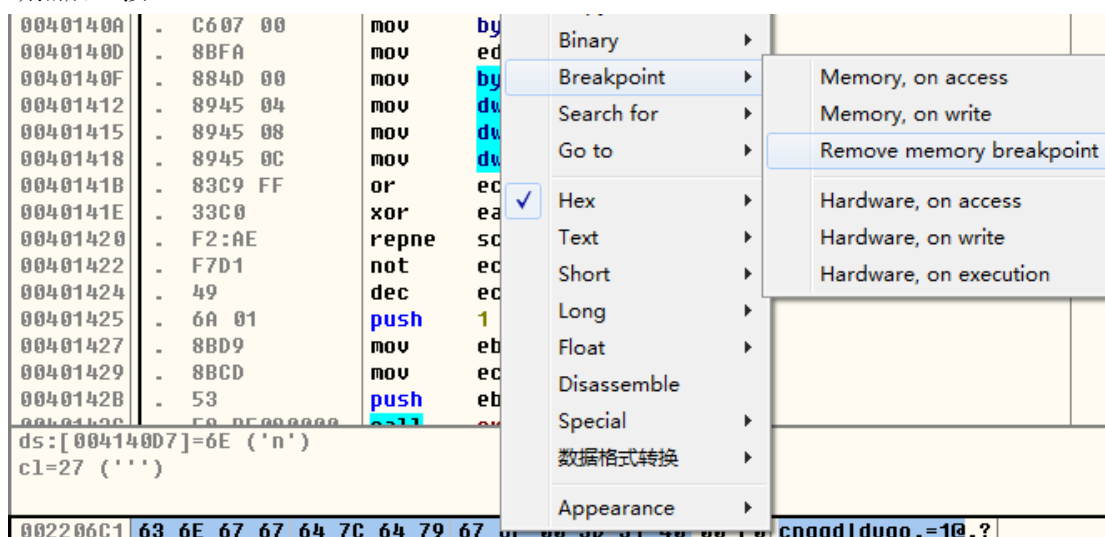
004013DC	. 885F 02	mov	byte ptr ds:[edi+2], bl
004013DF	. 8A89 B0404100	mov	cl, byte ptr ds:[ecx+4140B0]
004013E5	. 884F 03	mov	byte ptr ds:[edi+3], cl
004013E8	~ 7E 04	jle	short crackme.004013EE
004013EA	. C647 03 3D	mov	byte ptr ds:[edi+3], 3D
004013EE	> 8D4D 01	lea	ecx, dword ptr ss:[ebp+1]
004013F1	. 3BC1	cmp	eax, ecx
004013F3	~ 7E 04	jle	short crackme.004013F9
004013F5	. C647 02 3D	mov	byte ptr ds:[edi+2], 3D
004013F9	> 83C7 04	add	edi, 4
004013FC	. 3BC5	cmp	eax, ebp
004013FE	^ 7C 84	j1	short crackme.00401384
00401400	. 33C0	xor	eax, eax
00401402	> 8B6C24 24	mov	ebp, dword ptr ss:[esp+24]
00401404	004024 24	mov	cl, byte ptr ss:[esp+24]

002202D0	59 32 35 00	00 00 00 00	00 00 00 00	00 00 00 00	Y25.....
----------	-------------	-------------	-------------	-------------	----------

步骤二十二、 把之前的内存断点以及硬件断点都删除了，在菜单栏点 Debug--Hardware breakpoints,在弹出来的窗口中点击 Delete，删除硬件断点。



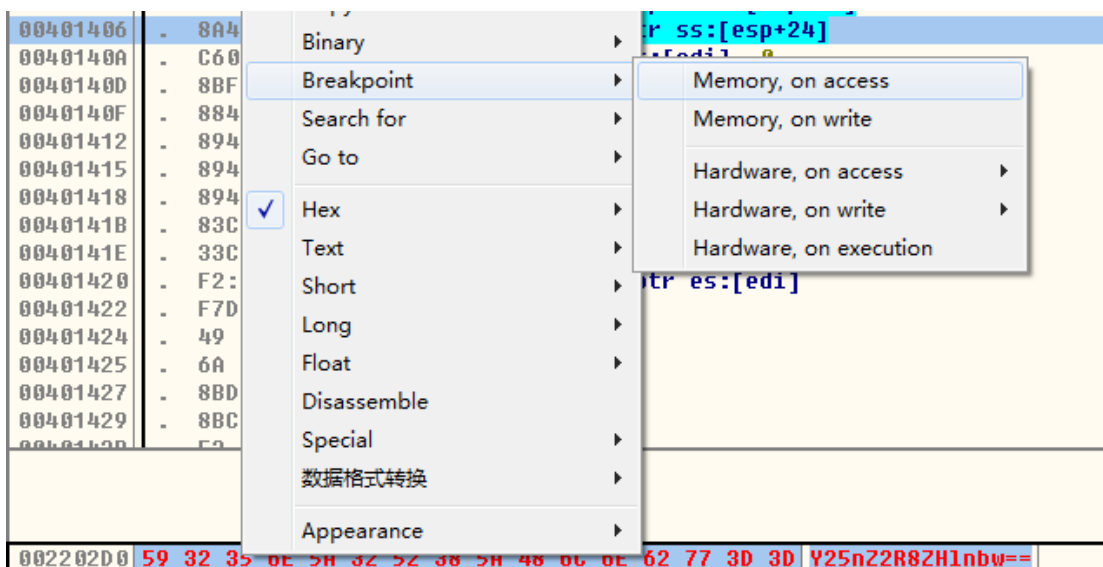
步骤二十三、 在数据窗口中右键—Go to—Expression,在弹窗中输入之前内存断点的地址 220691，选中内存单元，右键—Breakpoint—Remove memory breakpoint,下图。删除了断点后，按 F9。



步骤二十四、 程序断在了 401400 处，在寄存器窗口中能看到，最终加密过后的字符串变成了“Y25nZ2R8ZHlnbw==”，看这加密后的字符串像是 base64 加密的结果，通过 base64 在线解解密可以确定，这是 base64 加密算法。

Registers (FPU)	
EAX	0000000C
ECX	0000000B
EDX	002202D0 ASCII "Y25nZ2R8ZH1nbw=="
EBX	00000041
ESP	0018FED4
EBP	0000000A
ESI	00220691 ASCII "cnggd dygo"
EDI	002202E0

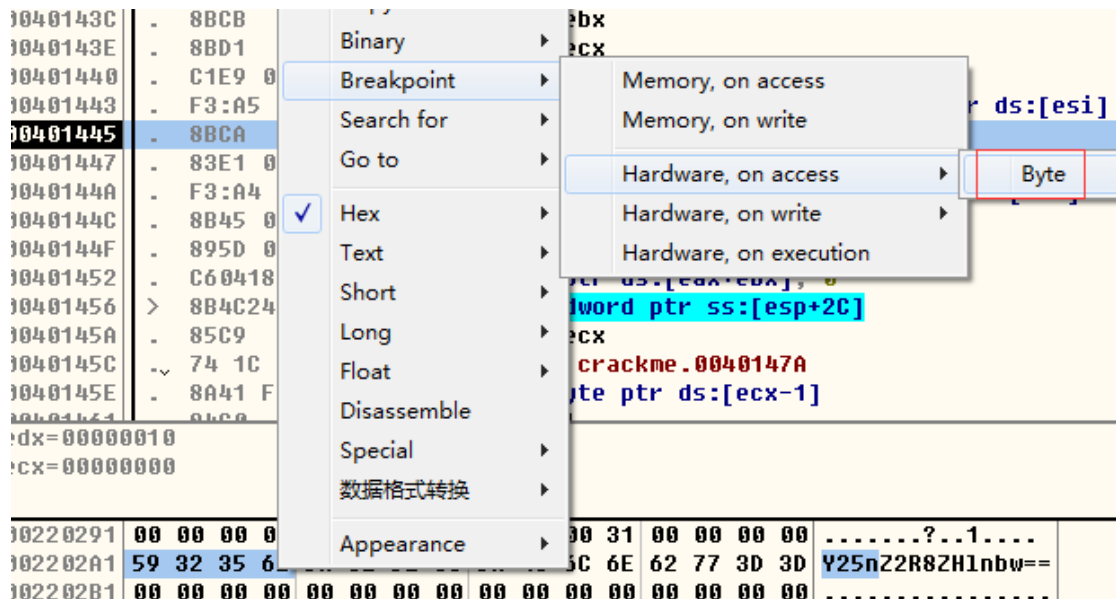
步骤二十五、回到程序中，继续给 2202D0 设置内存断点。接下来再次断下来是进行扫描，不用管它，F8 一下，再 F9 运行起来。



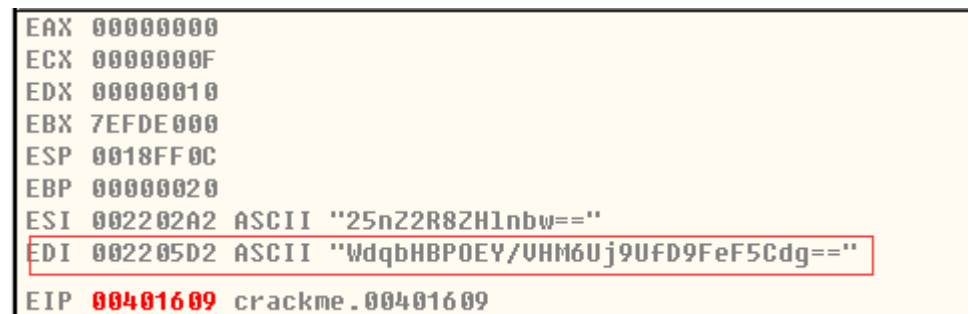
步骤二十六、再次断下来的地方发现是把安符串往 2202A1 处拷贝。

00401440	. C1E9 02	shr	ecx, 2		EAX 00000010
00401443	. F3:A5	rep	movs dword ptr es:[edi], dword ptr ds:[esi]		ESP 0018FED4
00401445	. 8BCA	mov	ecx, edx		EBP 0018FF30
00401447	. 83E1 03	and	ecx, 3		ESI 002202D0 ASCII "Y25nZ2R8ZH1nbw=="
00401448	. F3:A4	rep	movs byte ptr es:[edi], byte ptr ds:[esi]		EDI 002202A1

步骤二十七、 执行完拷贝操作，在 2202A1 处下硬件断点（在我的机器上，内存断点很不稳定）。



步骤二十八、 程序断在一个字符串比较之后（因为是硬件断点，所以是断在命令之后），我们看寄存器窗口，edi 也指向一个字符串，那么这个字符串是不是就是正确的 key 加密后的字符串呢？我们可以测试一下，把 edi 指向的字符串，根据刚才我们跟踪的加密顺序逆着加密一次，得到的字符串，应该就是正确的 key 了。



步骤二十九、 计算出来正确的 key 是 “flag{D3M4_x1Y4_w4NsUI}”，下图，到此，正确的 key 已经逆出来了。

