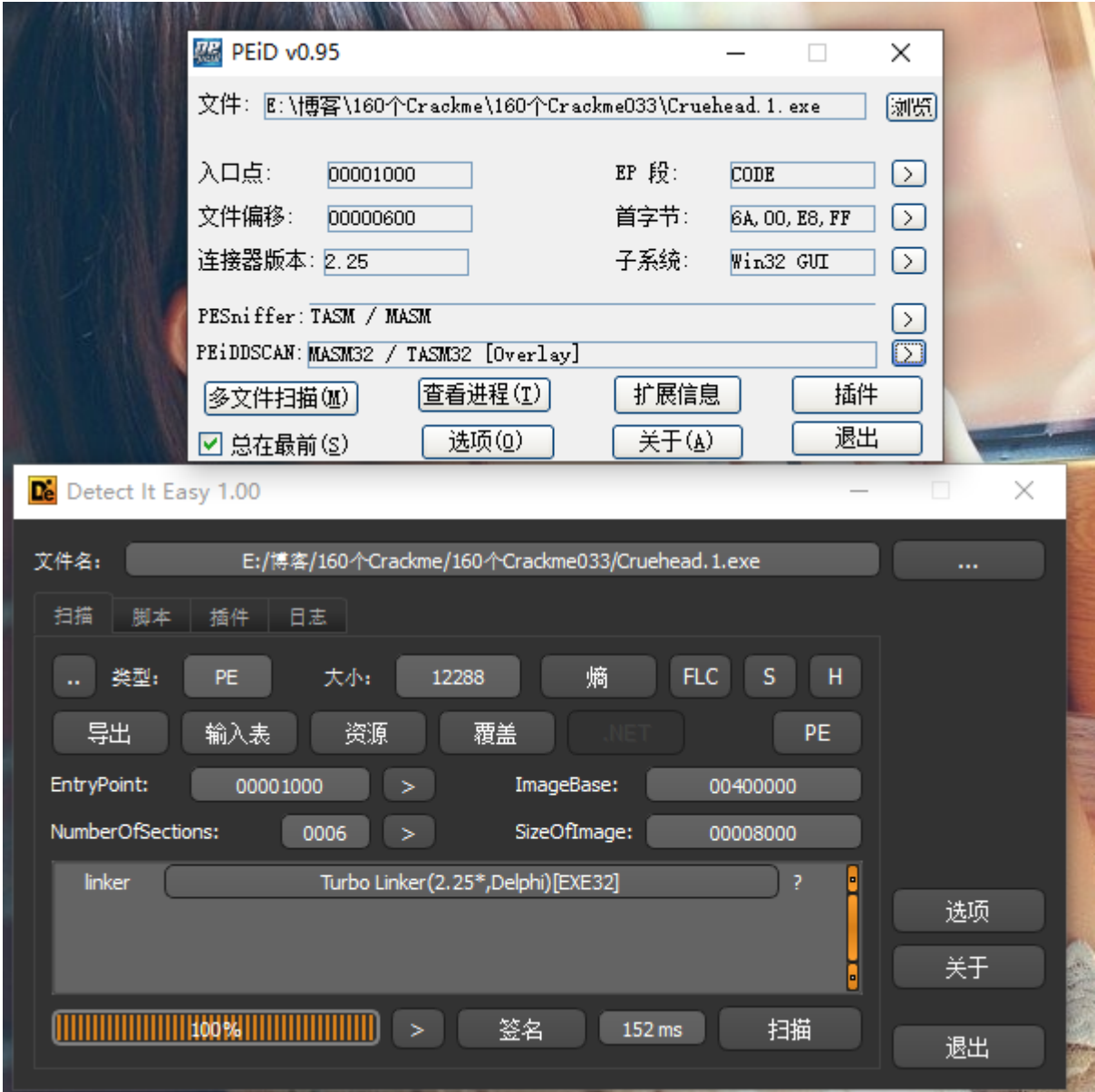


- 查壳
- 分析程序
- 用户名算法分析
- 序列号算法分析
- 写出注册机
- 校验结果

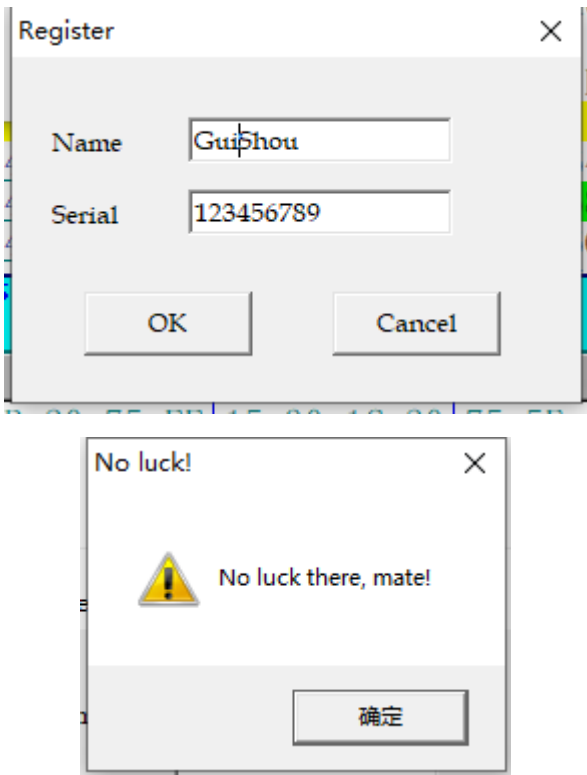
这个Crackme的作者是《使用OllyDbg从零开始Cracking》那部教程的作者在教程中使用的配套教程，当初也是看的这篇教程入门的，没想到会在这里遇见。

## 查壳



这个程序看着有点像Delphi写的，但其实是汇编写的。作者在OEP处故意制造了一个和Delphi一模一样的入口特征来迷惑查壳器。

## 分析程序

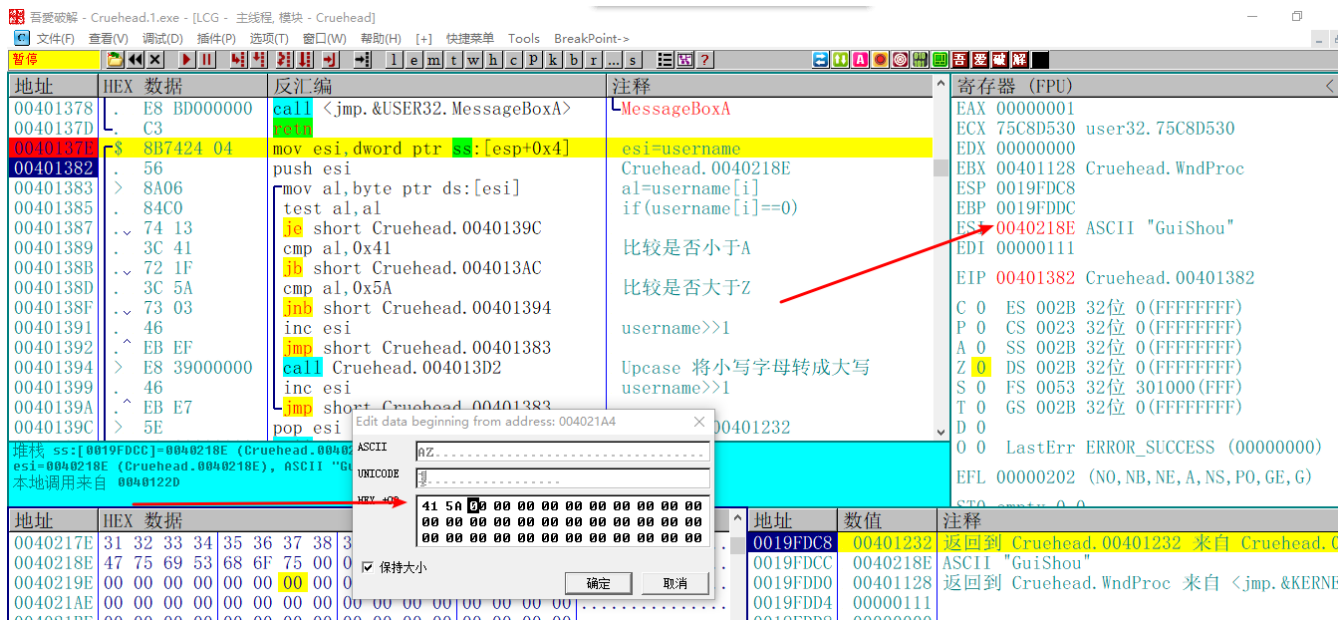


随便输入一个序列号和密码，根据错误提示，来到附近代码，开始分析算法。这个程序的算法在我跟踪完了之后发现是由两个部分组成

地址	HEX 数据	反汇编	注释
00401226	. ^ 74 BE	je short Cruehead.004011E6	
00401228	. 68 8E214000	push Cruehead.0040218E	GUISHOU
0040122D	. E8 4C010000	call Cruehead.0040137E	用户名的计算函数
00401232	. 50	push eax	
00401233	. 68 7E214000	push Cruehead.0040217E	123456789
00401238	. E8 9B010000	call Cruehead.004013D8	序列号的计算函数
0040123D	. 83C4 04	add esp,0x4	
00401240	. 58	pop eax	Cruehead.0040218E
00401241	. 3BC3	cmp eax,ebx	比较用户名和序列号计算的结果
00401243	. ^ 74 07	je short Cruehead.0040124C	相等则提示正确
00401245	. E8 18010000	call Cruehead.00401362	错误提示
0040124A	. ^ EB 9A	jmp short Cruehead.004011E6	
0040124C	. > E8 FC000000	call Cruehead.0040134D	正确提示
00401251	. ^ EB 93	jmp short Cruehead.004011E6	

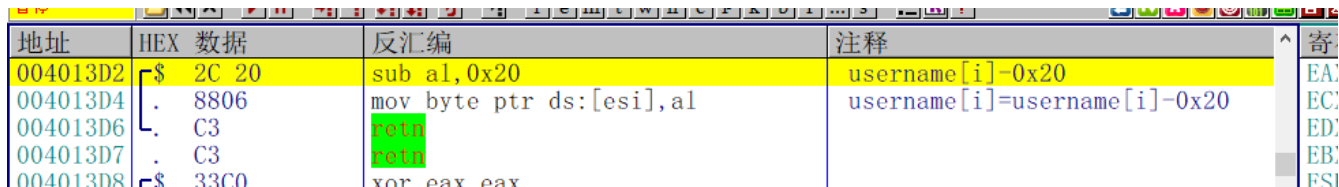
首先是一个用户名的计算函数，然后是一个序列号的计算函数，接着比较两个函数计算的结果是否相等，相等则提示正确。

## 用户名算法分析

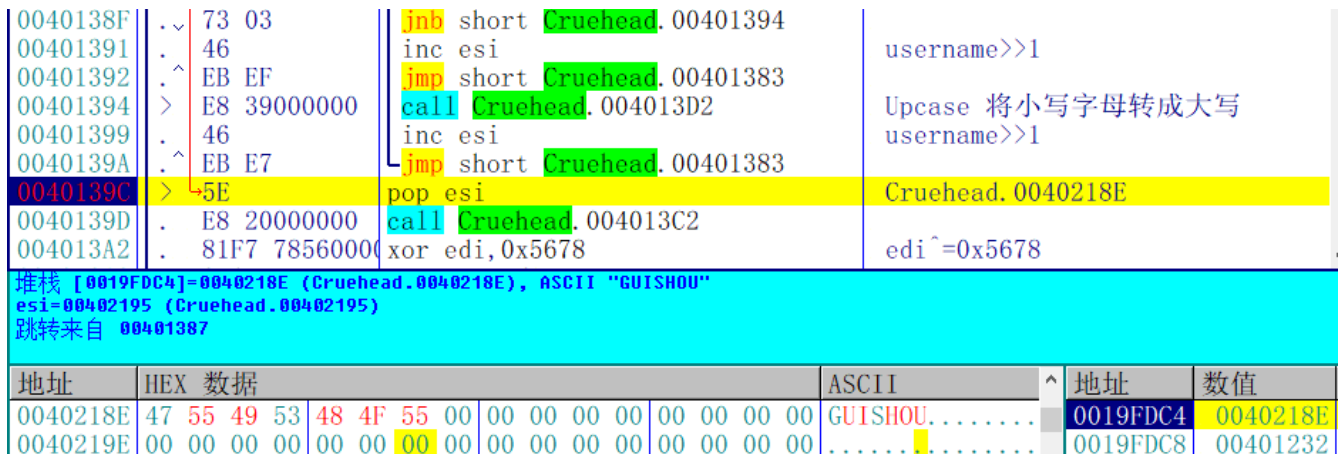


首先取出用户名，然后比较依次取出每一位的ASCII值，比较是否小于0x41，是否大于0x5A，41和5A这两个ASCII值其实就是大写的A-Z，如果不在这个区间就会进入sub4013D2这个函数

这个函数的作用也相当简单



就是把ASCII值减去0x20，也就是转成大写字母



整个循环的作用其实就是把用户名转为全部大写，然后就进入4013C2这个函数了

文件(F) 查看(V) 调试(D) 插件(P) 选项(O) 窗口(W) 帮助(H) [+] 快速菜单 Tools Breakpoint->									
暂停									
地址 HEX 数据 反汇编 注释									
004013C2		\$ 33FF	xor edi,edi		寄存器 (FPU) EAX 00000000 ECX 75C8D530 user32.75C8D530 EDX 00000000 EBX 00000000 ESP 0019FDC4 EBP 0019FDDC ESI 00402195 Cruehead.00402195 EDI 00000224 EIP 004013D1 Cruehead.004013D1 C 0 ES 002B 32位 0(FFFFFFFF) P 1 CS 0023 32位 0(FFFFFFFF) A 0 SS 002B 32位 0(FFFFFFFF) Z 1 DS 002B 32位 0(FFFFFFFF) S 0 FS 0053 32位 301000(FFF) T 0 GS 002B 32位 0(FFFFFFFF) D 0 O 0 LastErr ERROR_SUCCESS (00000000) EFL 00000246 (NO, NB, E, BE, NS, PE, GE, I ST0 empty 0.0				
004013C4		\$ 33DB	xor ebx,ebx						
004013C6		> 8A1E	mov bl,byte ptr ds:[esi]	bl=username[i]					
004013C8		\$ 84DB	test bl,bl	if(username[i]==0)					
004013CA		74 05	je short Cruehead.004013D1						
004013CC		\$ 03FB	add edi,ebx	edi+=username[i]					
004013CE		\$ 46	inc esi	username<<1					
004013CF		\$ EB F5	jmp short Cruehead.004013C6						
004013D1		> C3	ret						
004013D2		\$ 2C 20	sub al,0x20	username[i]-0x20					
004013D4		\$ 8806	mov byte ptr ds:[esi],al	username[i]=username[i]-0x20					
004013D6		\$ C3	ret						
004013D7		\$ C3	ret						
004013D8		\$ 33C0	xor eax,eax						
004013DA		\$ 33FF	xor edi,edi						
004013DC		\$ 33DB	xor ebx,ebx						
004013DE		\$ 8B7424 04	mov esi,dword ptr ss:[esp+0x4]	esi=Serial					
返回到 004013A2 (Cruehead.004013A2) 跳转来自 004013CA									
地址 HEX 数据 ASCII 地址 数值 注释									
0040218E	47 55 49 53	48 4F 55 00	00 00 00 00	00 00 00 00	GUI\$HOU.....	0019FDC4 004013A2 返回到 Cruehead.004013A2 来自 Cruehead.004013A2			
0040219E	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....	0019FDC8 00401232 返回到 Cruehead.00401232 来自 Cruehead.00401232			
004021AE	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	.....	0019FDCC 0040218E ASCII "GUI\$HOU"			

这个函数的作用也很简单，就是将用户名每一位的ASCII值进行相加，

地址	HEX	数据	反汇编	注释	寄存器 (FPU)
00401399	46		inc esi	username>>1	EAX 00000000
0040139A	EB E7		jmp short Cruehead.00401383		ECX 75C8D530 user32.75C8D530
0040139C	5E		pop esi	Cruehead.00401232	EDX 00000000
0040139D	E8 20000000		call Cruehead.004013C2		EBX 00000000
004013A2	81F7 78560000		xor edi,0x5678	edi ^=0x5678	ESP 0019FDC8
004013A8	8BC7		mov eax,edi		EBP 0019FDDC
004013AA	EB 15		jmp short Cruehead.004013C1		ESI 00402195 Cruehead.00402195
004013AC	5E		pop esi	Cruehead.00401232	EDI 00000224
004013AD	6A 30		push 0x30	Cruehead.00401232	EIP 004013A2 Cruehead.004013A2
004013AF	68 60214000		push Cruehead.00402160	Style = MB_OK MB_ICONEXCLAMATION	C 0 ES 002B 32位 0(FFFFFFFF)
004013B4	68 69214000		push Cruehead.00402169	No luck!	P 1 CS 0023 32位 0(FFFFFFFF)
004013B9	FF75 08		push [arg.1]	No luck there, mate!	A 0 SS 002B 32位 0(FFFFFFFF)
004013BC	E8 79000000		call < jmp. &USER32.MessageBoxA>	hOwner = 3BE71008 ('CrackMe v1.0')	Z 1 DS 002B 32位 0(FFFFFFFF)
004013C1	C3		ret	MessageBoxA	

之后将用户名的ASCII值之和0x5678进行异或。这个就是完整的用户名的算法了。接下来分析序列号部分

## 序列号算法分析

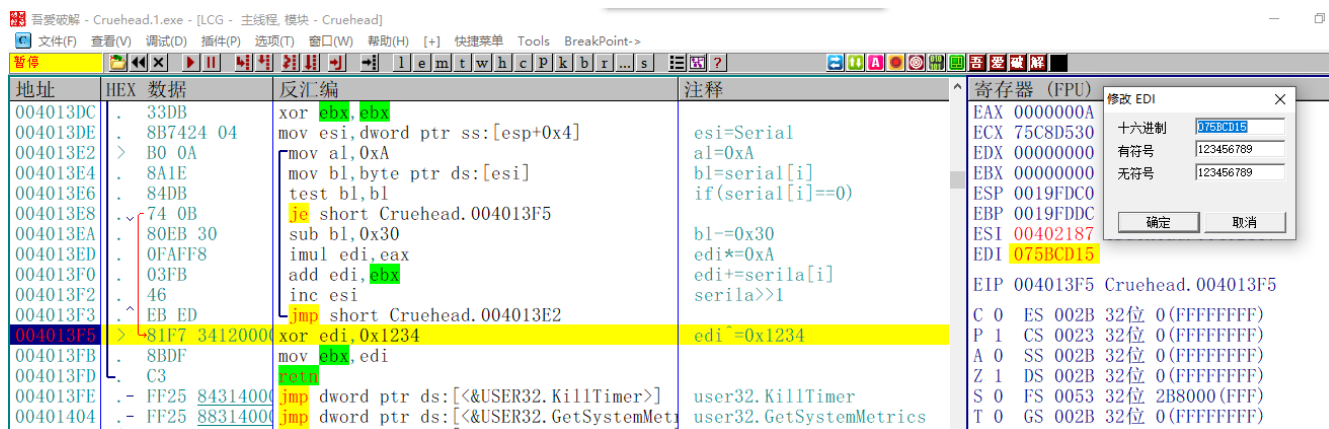
[File][Edit][View][Tools][Window][Help]   l e m t w h c p k b i r ... s [?] [F1-F12] [Print] [Save] [Load] [Exit]										
地址		HEX	数据	反汇编			注释			
004013D8		33C0		xor eax,eax						
004013DA		33FF		xor edi,edi						
004013DC		33DB		xor ebx,ebx						
004013DE		8B7424 04		mov esi,dword ptr ss:[esp+0x4]			esi=Serial			
004013E2		B0 0A		mov al,0xA			al=0xA			
004013E4		8A1E		mov bl,byte ptr ds:[esi]			bl=serial[i]			
004013E6		84DB		test bl,bl			if(serial[i]==0)			
004013E8		74 0B		je short Cruehead.004013F5						
004013EA		80EB 30		sub bl,0x30			bl-=0x30			
004013ED		0FAFF8		imul edi,eax			edi*=0xA			
004013F0		03FB		add edi,ebx			edi+=serila[i]			
004013F2		46		inc esi			serila>>1			
004013F3		EB ED		jmp short Cruehead.004013E2						
004013F5		81F7 34120000		xor edi,0x1234			edi ^=0x1234			
004013FB		8BDF		mov ebx,edi						
004013FD		C3		ret						
004013FE		FF25 84314000		jmp dword ptr ds:[&USER32.KillTimer]			user32.KillTimer			
堆栈 ss:[0019FDC4]=0040217E (Cruehead.0040217E), ASCII "123456789"										
esi=0040217E (Cruehead.0040217E), ASCII "123456789"										
寄存器 (FPU)										
EAX		00000000								
ECX		75C8D530 user32.75C8D530								
EDX		00000000								
EBX		00000000								
ESP		0019FDC0								
EBP		0019FDDC								
ESI		0040217E ASCII "123456789"								
EDI		00000000								
EIP		004013E2 Cruehead.004013E2								
C 0		ES 002B 32位 0(FFFFFFFF)								
P 1		CS 0023 32位 0(FFFFFFFF)								
A 0		SS 002B 32位 0(FFFFFFFF)								
Z 1		DS 002B 32位 0(FFFFFFFF)								
S 0		FS 0053 32位 301000(FFF)								
T 0		GS 002B 32位 0(FFFFFFFF)								
D 0										
O 0		LastErr ERROR_SUCCESS (00000000)								
EFL		00000246 (NO, NB, E, BE, NS, PE, GE, LE)								
ST0		smat0.0								
地址		HEX		数据		ASCII		地址	数值	注释
0040217E		31	32	33	34	35	36	37	38	39 00 0

序列号的计算部分也很简单，首先取出每一位的ASCII值，然后减去0x30，减去0x30也就意味着将字母转成数字。

接着用edi乘以0xA，然后用edi再加上序列号，总结为下面的等式：

$$edi = serial + 0x10 * edi;$$

看着好像挺复杂的，实际上就是把字符串形式的序列号转成了十进制形式的序列号



接着再将edi和0x1234进行异或。到此序列号的算法部分就完成了



接着比较两个部分的结果是否相等

## 写出注册机

这种需要在用户名和序列号直接通过两种不同的算法需要等于同一个结果这种模式很像 Crackme029。那么这个Crackme的注册机编写思路也是一样，首先根据用户名计算出中间结果，然后再根据结果反向逆推出注册码，代码如下：

```
int calckey()
{
    char username[20] = { 0 };
    int result = 0;
```

```

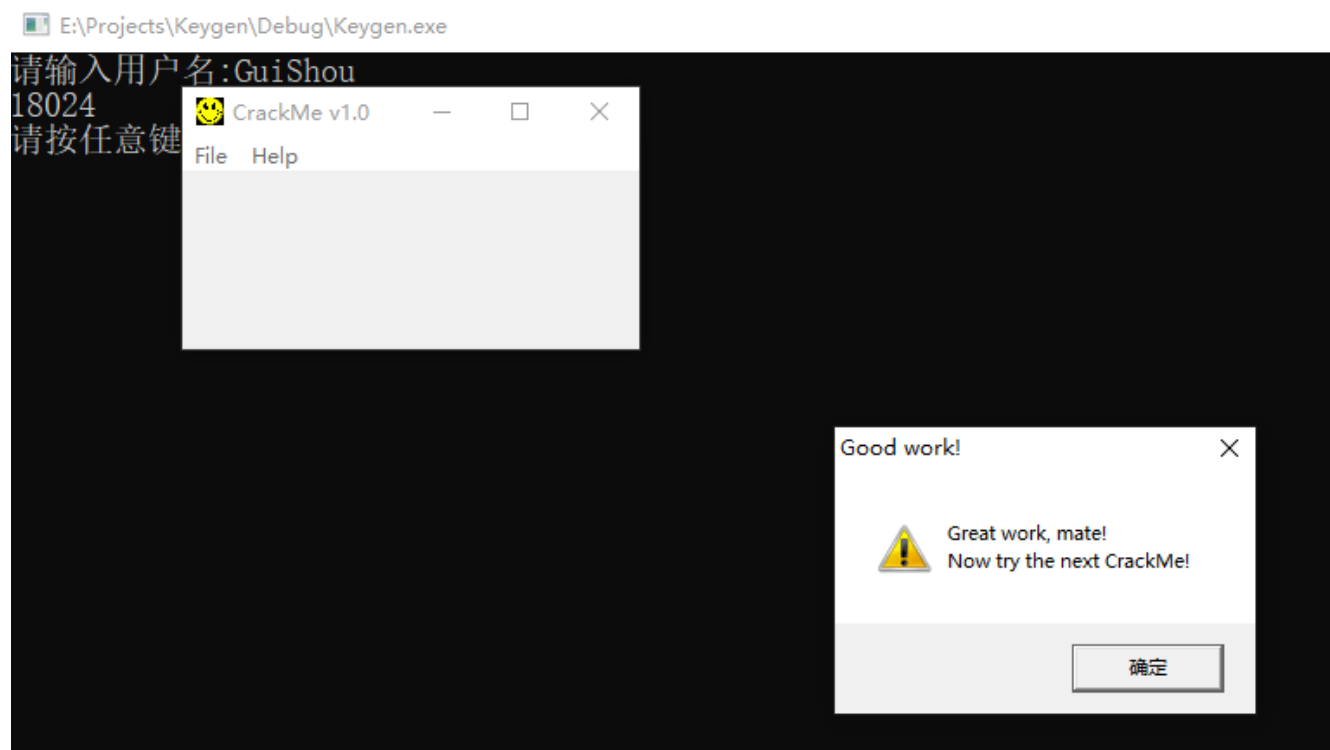
char serial[20] = { 0 };
printf("请输入用户名:");
scanf_s("%s", username, 20);

//计算用户名的结果
for (int i = 0; i < strlen(username); i++)
{
    if (username[i] < 0x41 || username[i]>0x5A)
    {
        username[i] -= 0x20;
    }
    result += username[i];
}
result ^= 0x5678;

result ^= 0x1234;
printf("%d\n", result);
return 0;
}

```

## 校验结果



输入用户名和计算的结果，提示正确，破解完成。

需要相关文件的可以到我的Github下载：<https://github.com/TonyChen56/160-Crackme>



