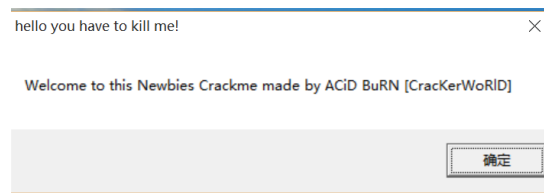


第一题：Acid burn.exe

- 法一：找正确用户名和序列号
- 法二：修改汇编指令绕过验证
- 法三：逆向出序列号生成算法

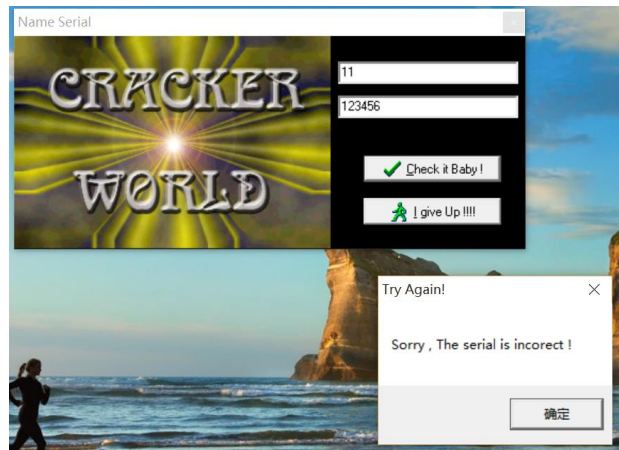
简介：Acid burn.exe 程序是一个 GUI 程序，要求关闭掉第一个弹出窗口，进入第二个窗有两种似登录验证方式：用户名+序列号、序列号，需要破解此处登录验证。




需要输入名字和序列号：



随便输入名称和序列号：显示序列号不正确。此处多试几次也是失败，那么要如何才能得到正确序列号呢？



当输入 name 和 serial 后点击 , 到弹出 serial 不正确的弹窗之间一定进行了用户提交数据和正确数据的验证过程。如果能通过调试定位到比较过程大概率就知道正确数据了。下面使用 X64dbg 进行程序动态调试。

动态调试：

1、预期：不弹出第一个弹窗。

通过调试器运行分析，发现运行下面函数就会弹出第一个窗口，尝试将 je 改为 jmp 跳过函数调用。将修改后的程序另保存为 exe 后运行[file->patch->修补文件保存为 exe 格式]，发现第一个弹窗消失了。

00425624	mov eax,dword ptr ss:[ebp-4]	[ebp-04]:&"d删除"
00425627	cmp word ptr ds:[eax+1CE],0	
0042562F	je acid.burn.425643	
00425631	mov ebx,dword ptr ss:[ebp-4]	[ebp-04]:&"d删除"
00425634	mov edx,dword ptr ss:[ebp-4]	[ebp-04]:&"d删除"
00425637	mov eax,dword ptr ds:[ebx+1D0]	
0042563D	call dword ptr ds:[ebx+1CC]	
00425643	xor eax,eax	
00425645	pop edx	

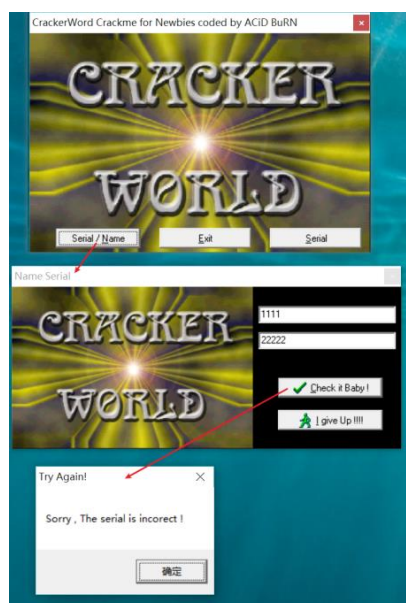
CrackerWorld Crackme for Newbies coded by ACID BuRN	0042560F	jmp acid.burn.403114	
	00425614	jmp acid.burn.425604	
	00425616	xor eax,eax	eax:&"d删除"
	00425618	push ebp	
	00425619	push acid.burn.42564D	
	0042561C	push dword ptr [eax]	
	00425621	mov dword ptr [eax],esp	
	00425624	mov eax,dword ptr ss:[ebp-4]	[ebp-04]:&"d删除"
	00425627	cmp word ptr ds:[eax+1CE],0	
	0042562F	je acid.burn.425643	
	00425631	mov ebx,dword ptr ss:[ebp-4]	[ebp-04]:&"d删除"
	00425634	mov edx,dword ptr ss:[ebp-4]	[ebp-04]:&"d删除"
	00425637	mov eax,dword ptr ds:[ebx+1D0]	eax:&"d删除", [ebx+1D0]:&"d删除"
	0042563D	call dword ptr ds:[ebx+1CC]	
	00425643	xor eax,eax	eax:&"d删除"
	00425645	pop edx	
	00425646	pop ecx	
	00425647	mov dword ptr [eax],edx	
	00425648	jmp acid.burn.425664	
	0042564D	jmp acid.burn.402F18	
	00425652	mov edx,dword ptr ss:[ebp-4]	
	00425655	mov eax,dword ptr ds:[431680]	eax:&"d删除", 00431680:&"d删除"
	0042565A	call acid.burn.42A0C0	

2、继续使用新保存的文件进行调试，有两种验证方式：用户名+序列号、序列号。下面先找出用户名+序列号过验证：随便输入用户名和序列号都是报序列号不正确。

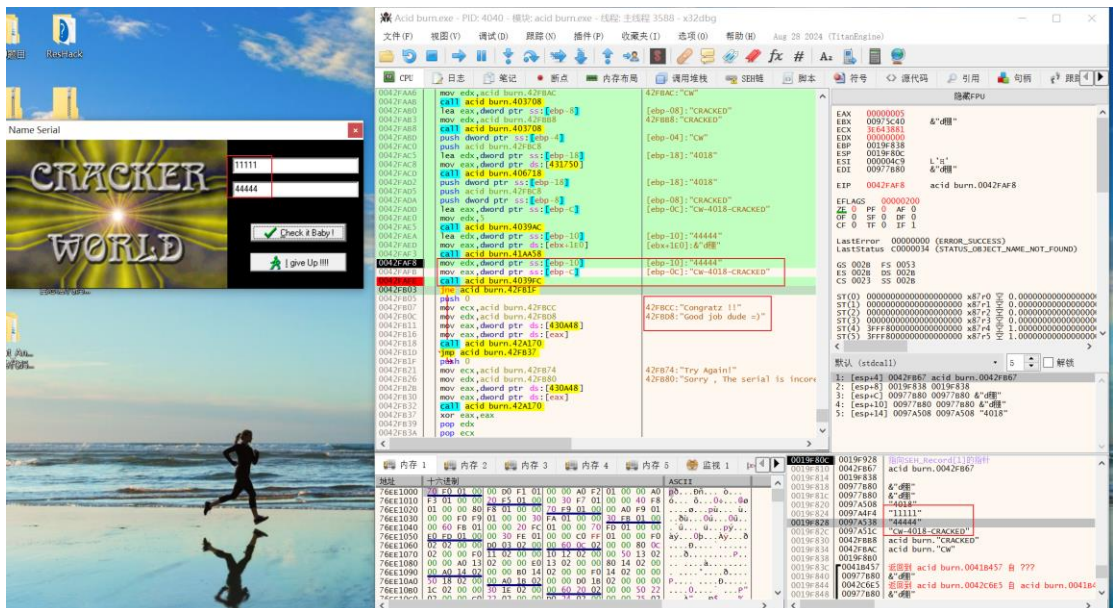


法一：找正确用户名和序列号

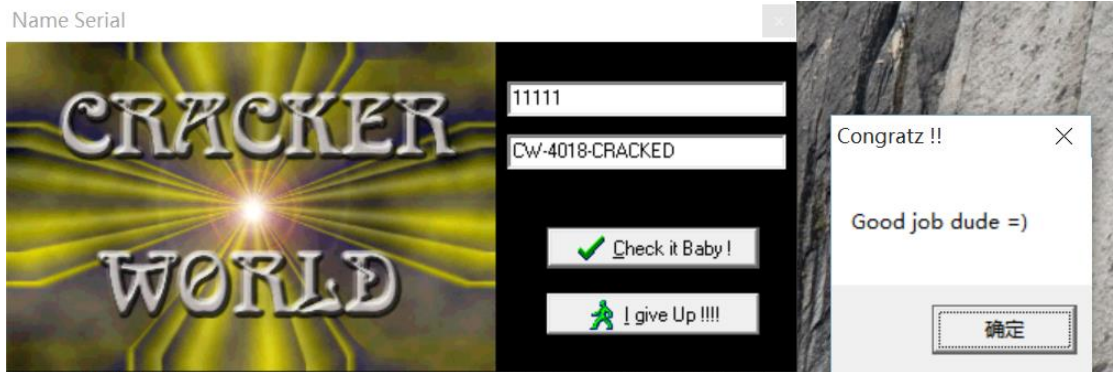
- 用户名+序列号过验证：



开始调试前先通过调试器字符串搜索找到 Good job 等字符串，双击进入反汇编界面，在字符串相关函数前面一些打下断点，方便调式时快速到达附近，然后可以单步调试看堆栈情况。【针对要分析现象提取出关键信息，通过字符串定位到附近打断断点，能加快分析速度】随后 Ctrl+F2 重新加载程序，随便输入用户名和序列号，一步步调试跟进发现似序列号对比函数和正确序列号"CW-4018-CRACKED"，构造登录名和序列号：111111/ CW-4018-CRACKED，可以将此输入验证界面。



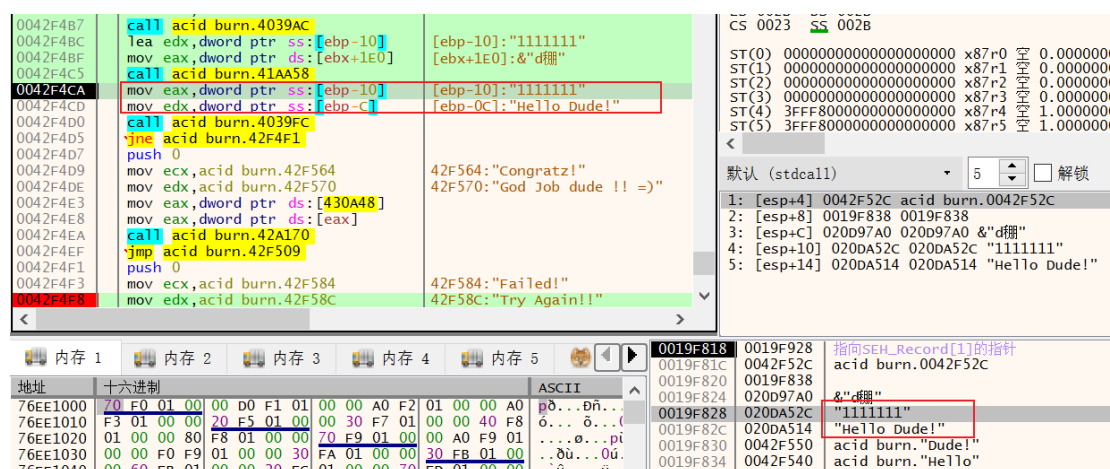
成功截图：说明发现的字符串就是序列号。



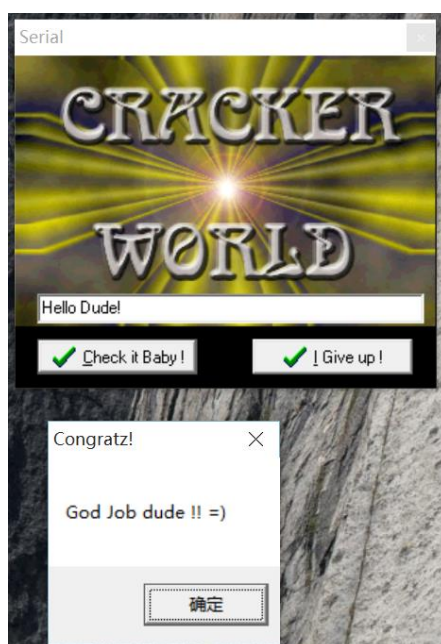
- 序列号过验证：



序列号界面使用上面一样的方法，定位到 Try Again 字符串前面附近指令打上断点，然后重新加载程序随便输入序列号然后一步步调试，最后发现和前面相同的比较函数：复制 Hello Dude!进行验证，成功。

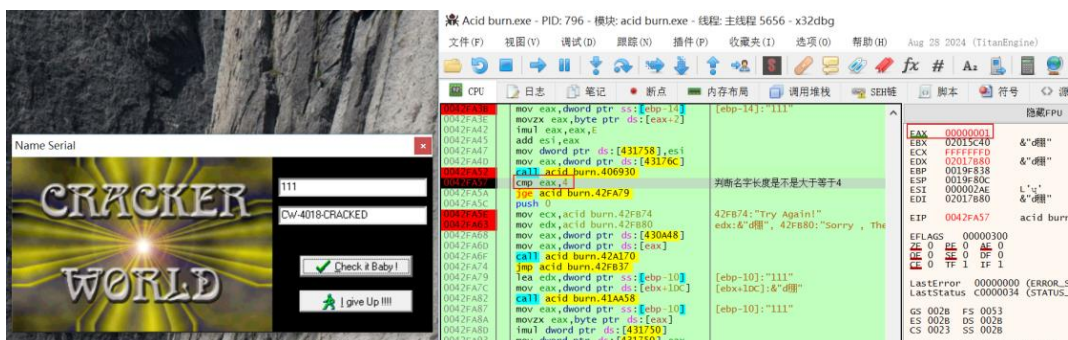


成功截图：

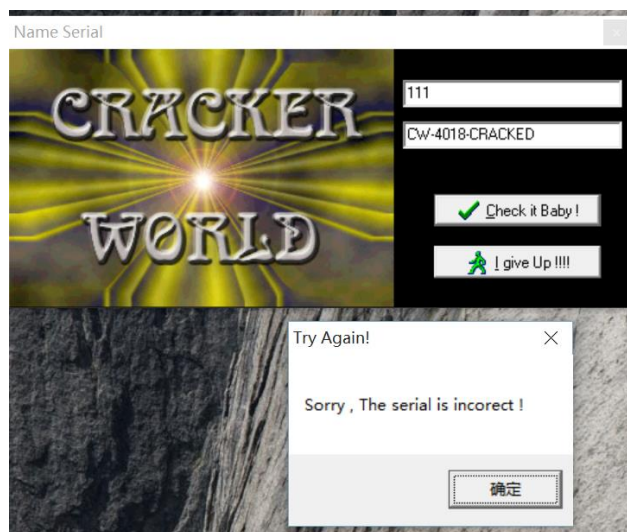


法二：修改汇编指令绕过验证

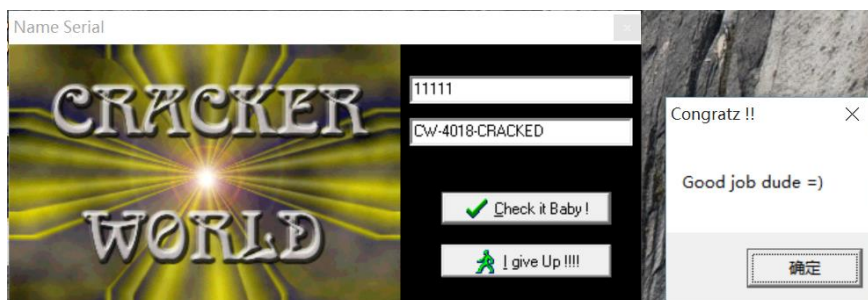
当输入不同长度的名字和相同序列号，结果是不同的。想要搞清楚 name 对结果的影响可以先字符串定位法搜“Try Again”在报错弹窗指令前面些打上断点（如果怕运行跳过了所打断点，就间隔多打几个），然后仔细分析流程会发现用户名长度会与 4 进行比较，若小于 4 则报错，大于等于 4 则正常执行。



名字为：111（3 位），验证结果错误。



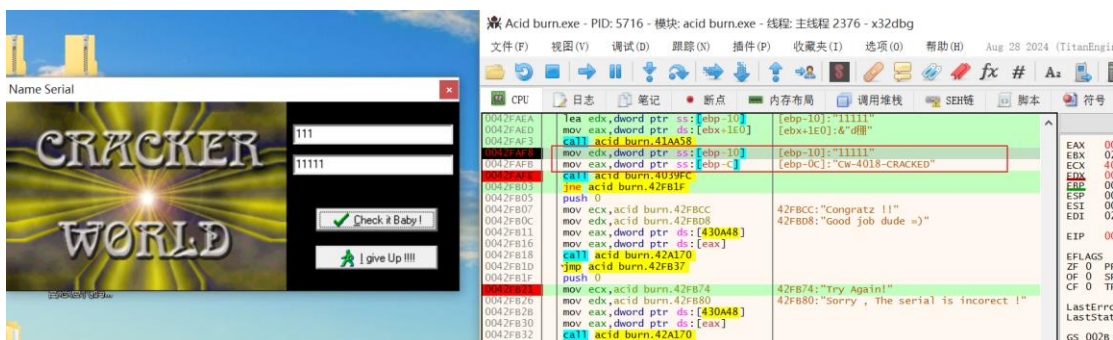
名字为：11111（5 位），验证成功。



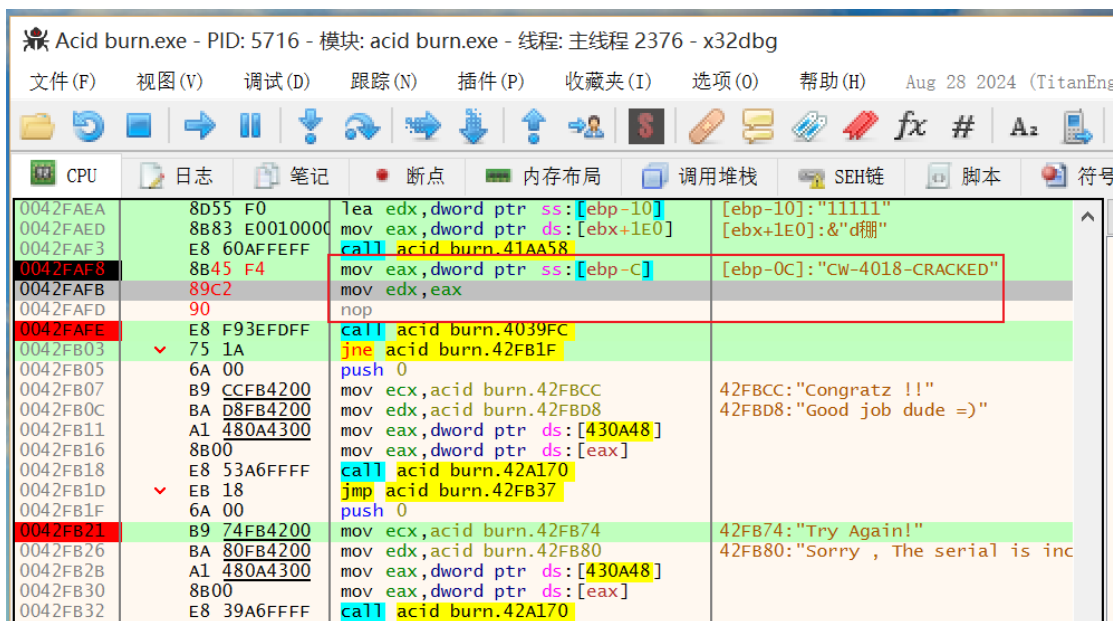
如上面图所示想要不受名字长度影响，可以使用 nop 指令将此比较指令替换，如下：

0042FA4D	mov	eax, dword ptr ds:[43176C]	
0042FA52	call	acid burn.406930	
0042FA57	nop		判断名字长度是不是大于等于4
0042FA58	nop		
0042FA59	nop		
0042FA5A	jge	acid burn.42FA79	

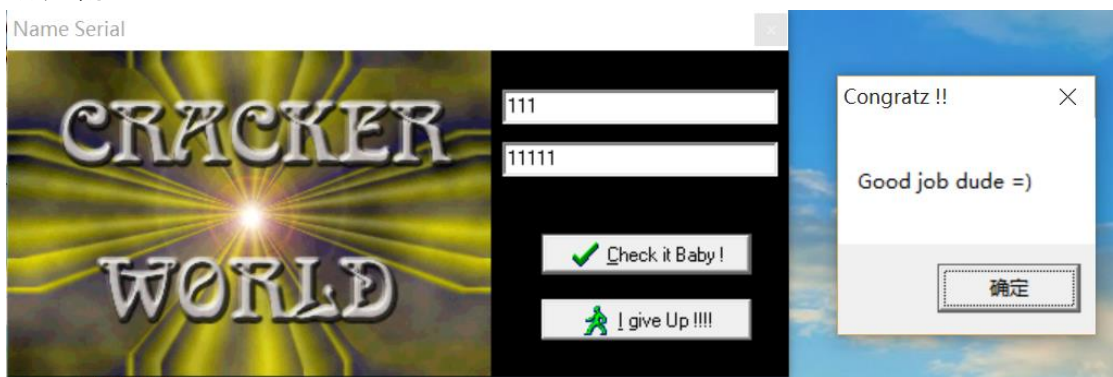
序列号判断处的指令是先将用户输入序列号存入 edx，然后再将正确序列号存到 eax。这里可以改一下存值逻辑：先存正确序列号，再将寄存器中正确的序列号替换掉用户输入的序列号。修改序列号判断处的指令，修改前如下：



修改后如下图：



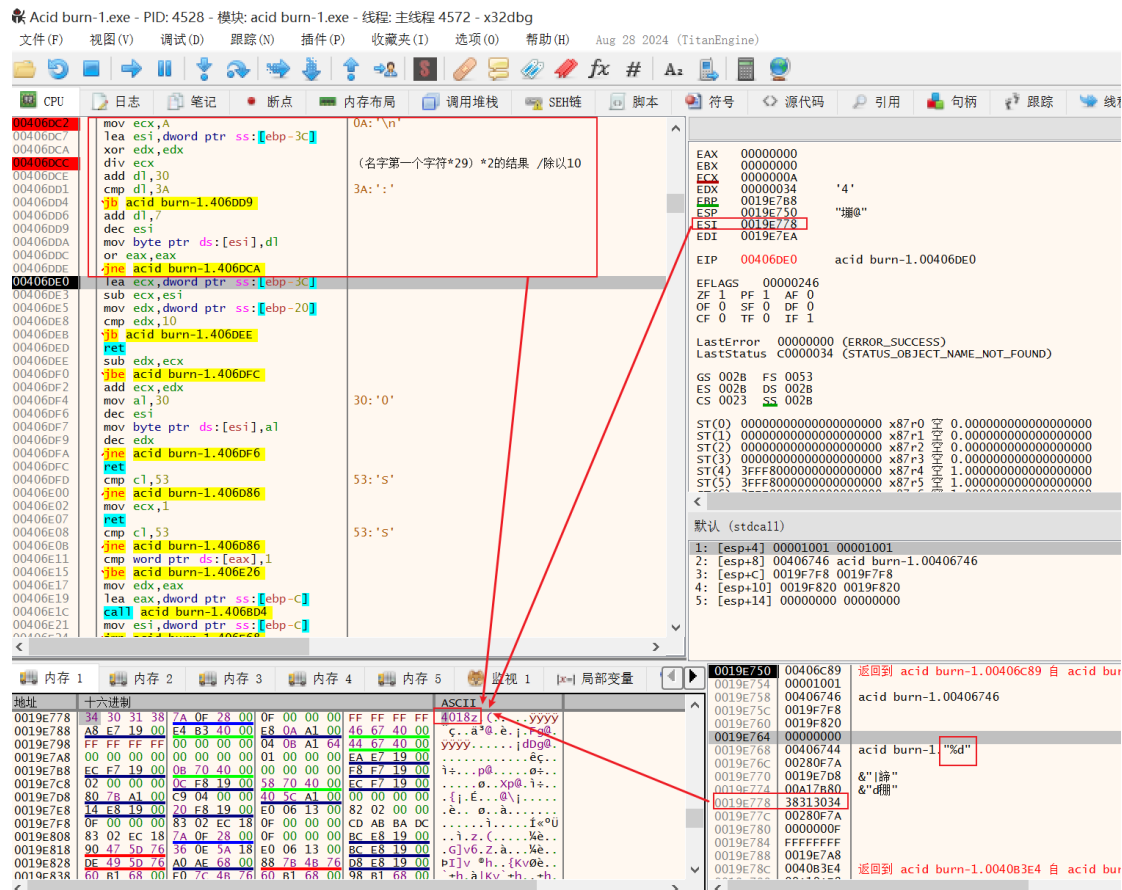
将修改后的文件另存为 exe 文件，接着随便输入用户名和序列号运行结果如下：
成功截图：



仅输入序列号方法也是同样的方法找到比较函数处，修改指令，另存为可执行文件，运行即可成功。此修改指令过验证法可以将以上需要修改指令的地方都修改完后另存为可执行文件运行即可。

法三：逆向出序列号生成算法

- (1) 上面步骤中可知 name 长度有限制，需大于等于 4



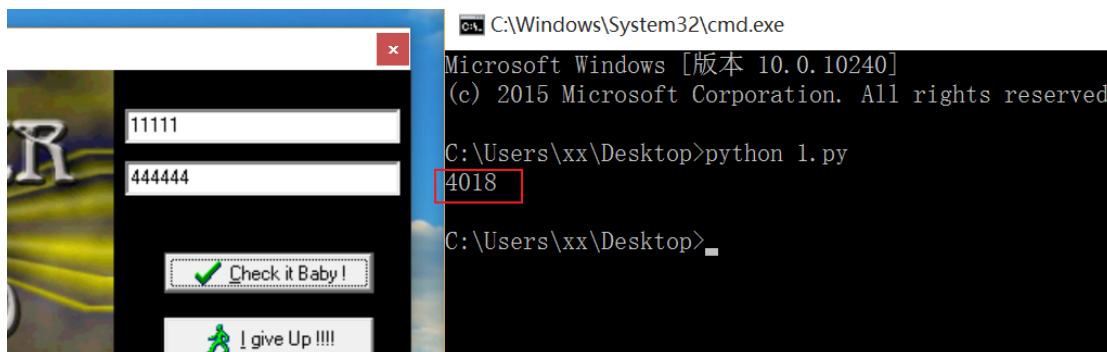
由上面可知算法核心公式，写一个脚本帮助计算出序列号中可变部分：

```
#将下面代码保存为 python 脚本
def calculate_result(input_string):
    return (ord(input_string[0]) * 0x29 * 0x02)

# 示例使用
result = calculate_result("11111") # 可以替换成其他字符串
print(result)
```

#ord () 取得的第一个字符 ASCII 码是 10 进制表示的，实际计算时 python 会将 16 进制表示的 x029 转换成 10 进制进行运算，0x02 同上。

名字字符串“11111”由脚本算出序列号可变部分结果如下：



总结：

- 1、找出关键字打断点：调试代码前针对调试对象抽取出针对性的字符串或者函数信息，方便打断点缩小分析调试范围。
- 2、调试器中改指令：调试器中修改指令的时候注意指令长度，短变长会造成覆盖可寻址法解决（换其他指令），长变短可用 nop 填充。如果修改指令后保存文件报错或者没有被修改，可考虑程序文件代码段是否有写入权限。程序代码段还要可写入属性才能更改。CCF 可以查看并修改段属性 0xE0000020（可读-可写-可执行）。一般调试也会备份原文件，不会直接修改原文件，修改指令后另存为新文件也是可行的，原文件没有写权限也是不影响的。