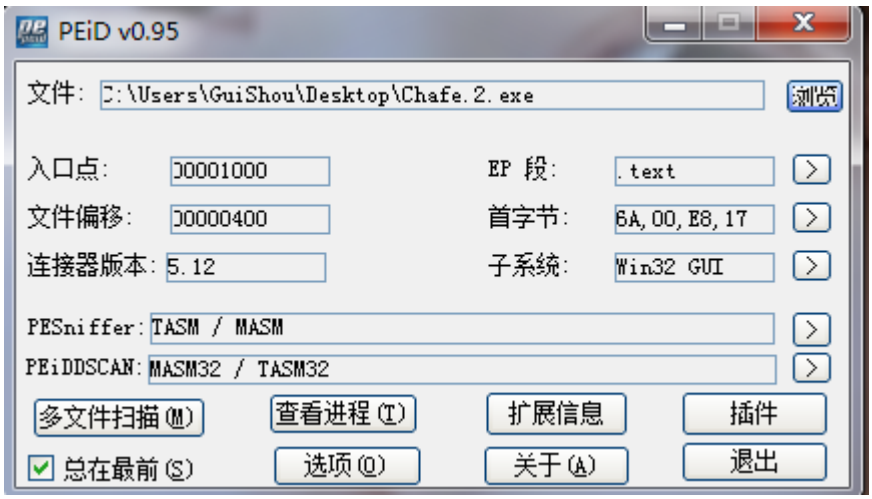
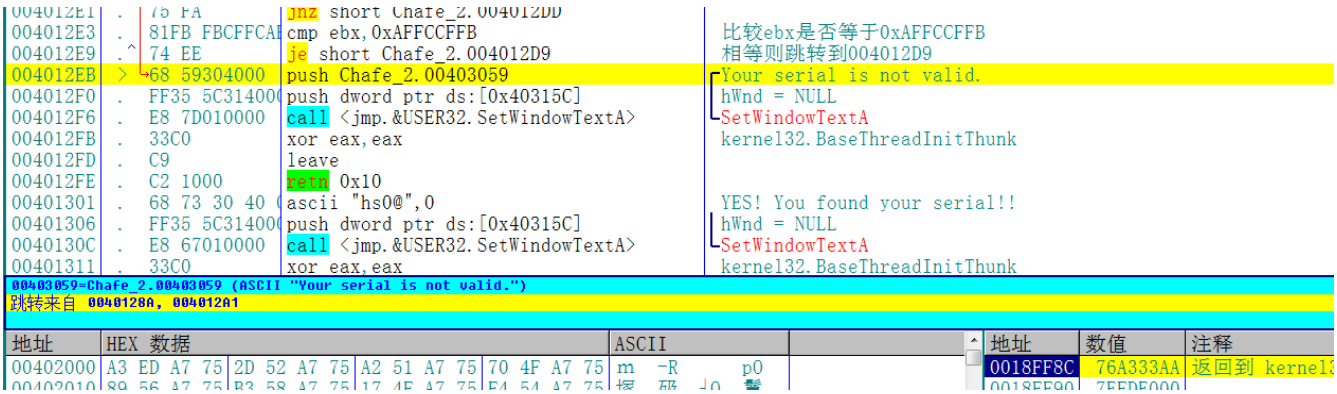
查壳



024这个Crackme跟023是同一个作者，难度两颗星，程序设计的很巧妙，非常有意思

分析程序



首先根据字符串的错误提示找到错误跳转到这个地址的位置，总共有两处，两处校验的地方，地址相差不远，随便找一个跟过去

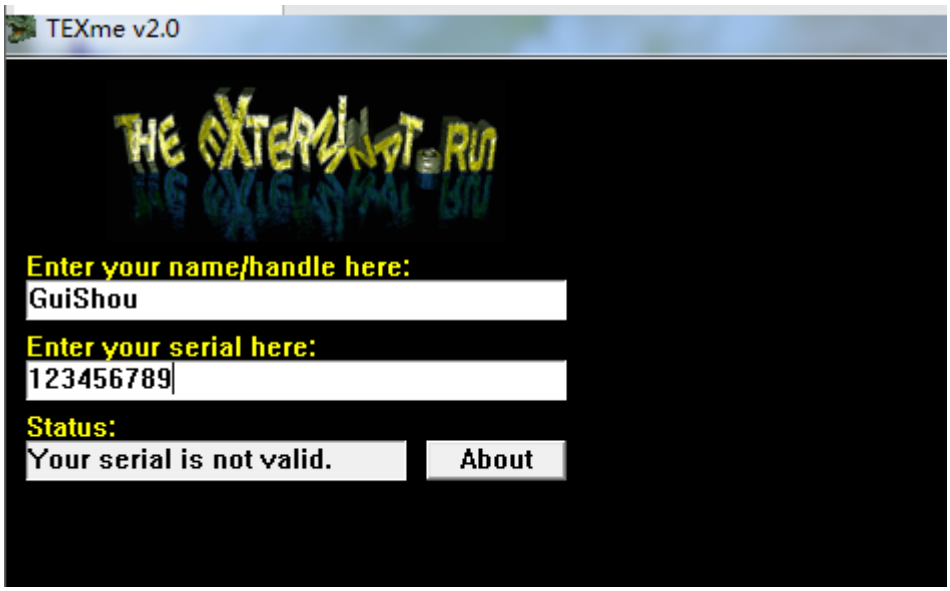
```
0040127B  .  FF35 5031400(push dword ptr ds:[0x403150]          hWnd = NULL
00401281  .  E8 BC010000  call <jmp.&USER32.GetDlgItemInt>      GetDlgItemInt
00401286  .  837D FC 00   cmp dword ptr ss:[ebp-0x4],0x0
0040128A  .v 74 5F        je short Chafe_2.004012EB             检测序列号是否有效
0040128C  .  50           push eax                              kernel32.BaseThreadInitThunk
0040128D  .  6A 14        push 0x14                             Count = 14 (20.)
0040128F  .  68 6C314000  push Chafe_2.0040316C                 Buffer = Chafe_2.0040316C
00401294  .  FF35 5431400(push dword ptr ds:[0x403154]          hWnd = NULL
0040129A  .  E8 AF010000  call <jmp.&USER32.GetWindowTextA>     GetWindowTextA
0040129F  .  85C0         test eax,eax                          kernel32.BaseThreadInitThunk
004012A1  .v 74 48        je short Chafe_2.004012EB             检测用户名长度是否有效
```

两处校验分别是校验序列号和用户名是否有效，校验完了之后就是程序的核心算法

分析算法

这个程序的算法分为三个部分，每个部分之间都相互关联，环环相扣，首先来看第一部分

第一部分 计算用户名和序列号



先输入用户名和序列号，然后下断点，这个程序的校验算法部分检测的是键盘事件，而不是定时器，所以打完断点之后随便按一个键程序就能断下来

```
004012A3  .  A1 0B304000  mov eax,dword ptr ds:[0x40300B]   eax=(int)CTEX
004012A8  .  BB 6C314000  mov ebx,Chafe_2.0040316C          ebx=username
004012AD  > 0303          add eax,dword ptr ds:[ebx]        eax+=[username]
004012AF  .  43           inc ebx                           username>>1
004012B0  .  81FB 7C314000cmp ebx,Chafe_2.0040317C          检测username是否为零
004012B6  .^ 75 F5        jnz short Chafe_2.004012AD        eax的值跟用户名关联
004012B8  .  5B           pop ebx                           ebx=Serial
004012B9  .  03C3         add eax,ebx                       eax+=Serial
```

这个就是第一部分的算法，过程如下：

1. eax的值第一次固定为CTEX的ASCII值
2. ebx为指向用户名的指针
3. 将eax和ebx的内容相加 结果保存在eax
4. 然后用户名左移1位
5. 检测用户名是否到了结尾，即循环次数为用户名的长度
6. 用户名的结果计算完毕之后再加上序列号

第二部分 根据用户名和序列号的结果改写Opcode

```
004012BB   .  3105 D9124000  xor dword ptr ds:[0x4012D9],eax     根据用户名和序列号计算的结果
004012C1   .  C1E8 10        shr eax,0x10                       eax>>0x10
004012C4   .  66:2905 D912   sub word ptr ds:[0x4012D9],ax      对[0x4012D9]代码改写
004012CB   .  BE EC114000    mov esi,Chafe_2.004011EC           esi=83EC8B55
004012D0   .  B9 3E000000    mov ecx,0x3E                       i=0x3E
004012D5   .  33DB           xor ebx,ebx
004012D7   .v EB 04          jmp short Chafe_2.004012DD
004012D9   >  54             push esp                           ------------------------------------------------
004012DA      45             db 45                              CHAR 'E'
004012DB      58             db 58                              CHAR 'X'
004012DC      00             db 00                              ----------------------------------------
```

首先将eax的值，也就是第一部分计算出的结果跟[0x4012D9]这个地址进行异或，

```
0401269  > C705 D9124000  mov dword ptr ds:[0x4012D9],0x584554   [0x4012D9]=0x584554
0401273  .  6A 00         push 0x0                               IsSigned = FALSE
0401275  .  8D45 FC       lea eax,dword ptr ss:[ebp-0x4]
0401278  .  50            push eax                               pSuccess = 00ABA11C
0401279  .  6A 64         push 0x64                              ControlID = 64 (100.)
040127B  .  FF35 50314000 push dword ptr ds:[0x403150]           hWnd = 001607E8 ('TEXme v2.0',class='CTEX')
0401281  .  E8 BC010000   call <jmp.&USER32.GetDlgItemInt>       GetDlgItemInt
```

[0x4012D9]这个地址的值在算法一开始就被赋值了，所以这个值是固定的。

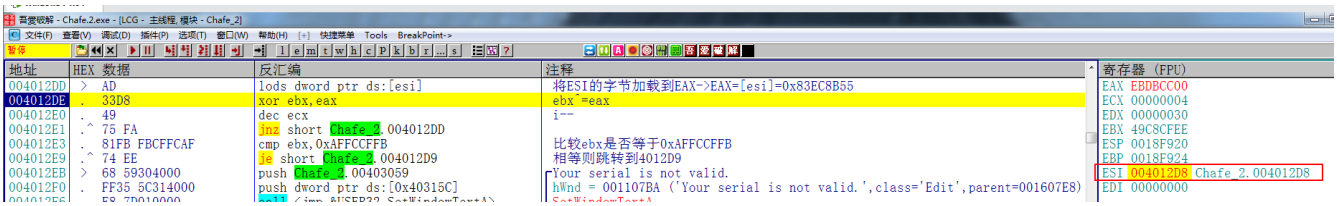
然后将eax右移0x10位，然后用[0x4012D9]的值减去ax，这两部分的操作都会改变[0x4012D9]处的代码，那么这么做到底是为了什么呢？现在还不知道，需要接着往下看

第三部分 关键校验

```
004012DC      00             db 00                              ----------------------------------------
004012DD   >  AD             lods dword ptr ds:[esi]            将ESI的字节加载到EAX->EAX=[esi]=0x83EC8B55
004012DE   .  33D8           xor ebx,eax                        ebx^=eax
004012E0   .  49             dec ecx                            i--
004012E1   .^ 75 FA          jnz short Chafe_2.004012DD
004012E3   .  81FB FBCFFCAF  cmp ebx,0xAFFCCFFB                 比较ebx是否等于0xAFFCCFFB
004012E9   .^ 74 EE          je short Chafe_2.004012D9          相等则跳转到4012D9
004012EB   >  68 59304000    push Chafe_2.00403059              Your serial is not valid.
004012F0   .  FF35 5C314000  push dword ptr ds:[0x40315C]       hWnd = 001107BA ('Your serial is not valid.',class='Edit',parent=001607E8)
004012F6   .  E8 7D010000    call <jmp.&USER32.SetWindowTextA>  SetWindowTextA
004012FB   .  33C0           xor eax,eax
004012FD   .  C9             leave
004012FE   .  C2 1000        retn 0x10
00401301   .  68 73 30 40 0  ascii "hs0@",0                     YES! You found your serial!!
00401306   .  FF35 5C314000  push dword ptr ds:[0x40315C]       hWnd = 001107BA ('Your serial is not valid.',class='Edit',parent=001607E8)
0040130C   .  E8 67010000    call <jmp.&USER32.SetWindowTextA>  SetWindowTextA
00401311   .  33C0           xor eax,eax
```

从内存地址[4011EC]开始，读入四个字节与EBX异或，结果保存在ebx，一直至内存地址[4012E4]，这段正好是关键算法部分，循环结束之后比较ebx是否等于0xAFFCCFFB，如果不相等则报错，如果相等的话就跳转0x4012D9处。

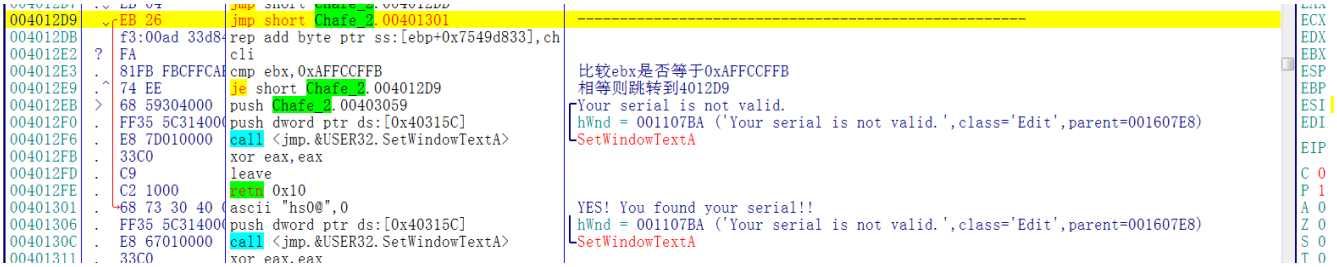
这里有一个坑，就算不能在这个校验的范围内下F2断点，如果下了的话会改变最后的值



揣摩作者意图

按照正常情况下来说，有一个提示错误的分支，另外一个就一定是提示正确的分支，而这个提示正确的分支却是指向一段动态改变的代码，那就是说，我们只能通过0x4012D9处的代码让程序提示注册成功，否则这个程序就会陷入无限的死循环或者是直接奔溃。

也就是说，0x4012D9这个位置必须带我们到正确的提示，也就是说这个地址的OpCode必须为EB 26



程序算法总结

1. 首先对用户名和注册码进行一系列的计算，得出Result
2. 通过Result改变0x4012D9处的代码
3. 循环0x3E次，校验最后的结果

写出注册机

过程分析清楚了，但是想写出注册机的话还是得费点劲的，这种逆推算法的事情我不太擅长，看见就脑阔疼，索性直接从看雪上面拷了一个，代码如下:

虽然这个注册机效率有点慢，但好歹能编译通过，而且计算正确，之前找了好几个要么编译不过，要么算的不对，我也是服了

```
#include <iostream>
#include <windows.h>
using namespace std;

int main(void)
{
    CHAR szName[20] = { 0 };
    cout << "Name:";
    cin >> szName;
    DWORD dwNum = 0x58455443;//程序实现填入的全局变量
    for (int i = 0; i < strlen(szName); i++)//用户名循环相加DWORD
    {
        void *p = &szName[i];
        _asm
        {
            mov edi, p;
            mov eax, dword ptr[edi];
            add dwNum, eax;
        }
    }
    ////////////////////////////////////////////////////////////////////////////
    DWORD dwPass = 0;
    while (1)
    {
        DWORD temp = (dwNum + dwPass) ^ 0x584554;
        temp -= (WORD)((dwNum + dwPass) >> 0x10);
        if (dwPass % 0x100000 == 0)  //这里if语句可以不要只是用来看看它在跑密码还是死机了
            cout << dwPass << endl;
        if (temp == 0x585426EB)//这就是上一个算法得到的正确的opcode
            break;
```

```
        dwPass++;
    }
    cout << "Pass:" << dwPass << endl;
    system("pause");
}
```

校验结果

输入用户名和计算出来的序列号，提示成功 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