lab5-report

- 实验概述

  本次实验是利用ollydbg进行程序的破解，在lab4中已经通过修改汇编代码实现了绕过登录验证。在lab5中则真正的分析了输入Name和Serial的匹配机制。并且输入姓名作为Name字符串，找到对应的Serial序列，分析整个计算的算法流程，并且进行登录验证。

  附近实验是分析一个输入序列并且进行验证的程序，在本次实验中我进行了尝试并且找到了正确的输入Serial并且分析了相应的算法。

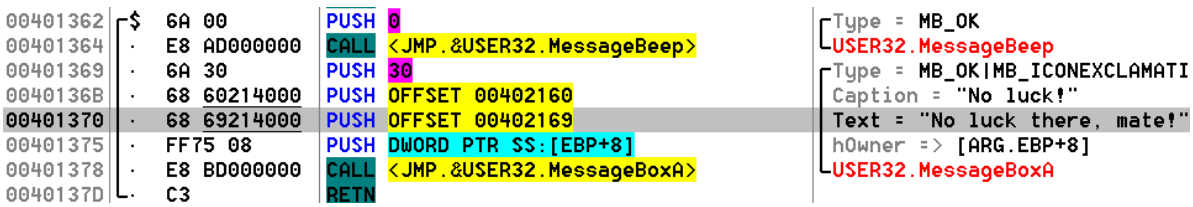
- 实验过程

  - 序列算法

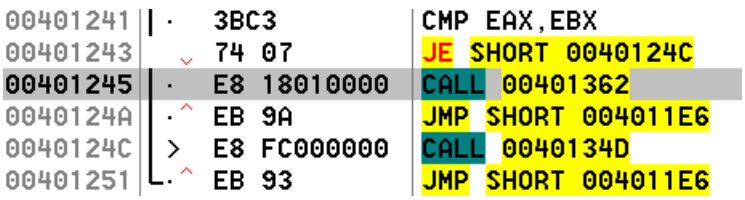
    该程序中输入字符串会生成有一个对应序列与之匹配，首先打开程序进行尝试，随便输入字符串以及序列之后会提示错误**No luck there, mate!**。所以这个展现出来的字符串可以作为突破口，开始硬编码序列号追踪。

    首先浏览一下strings信息，选择**Search for →All referenced text strings** 。

    可以看到该提示字符串在地址`0x00401370`处，跳转到此处。



    显然这是一个用于显示提示错误信息的函数，函数的入口地址为`0x00401362`，如果能找到从何处跳转到该提示函数，那么跳转语句所在的函数便是对序列和字符串进行判定的函数，所以跳转到`0x00401245`，此处为调用该函数的地址。



    在跳转处附近，还有一个`CALL 0040134D`，而该地址正是提示正确信息**Great work, mate!**所在的函数，结合上面的CMP语句，比较EAX和EBX寄存器的值，可以推断出是对两个输入值进行一系列变换运算之后，存放在EAX和EBX中，然后进行比较，如果能够匹配说明输入正确，否则错误。所以关键就是**找到编码序列算法**。

    为了分析编码序列算法，首先在上方设置一个断点，并且执行程序，输入的Name为"lishuangli"，Serial为"12345666"，得到：

```
00401223  .  83F8 00      CMP EAX,0
00401226  .^ 74 BE        JE SHORT 004011E6
00401228  .  68 8E214000  PUSH OFFSET 0040218E        ASCII "lishuangli"
0040122D  .  E8 4C010000  CALL 0040137E
00401232  .  50           PUSH EAX
00401233  .  68 7E214000  PUSH OFFSET 0040217E        ASCII "12345666"
00401238  .  E8 9B010000  CALL 004013D8
```

显然分别取得两个字符串之后，会调用一个相应的函数进行处理。继续追踪。在`0x0040218E`处的函数多次单步执行之后，经过一个loop得到了全部大写处理后的字符串：

```
ESP 0014FE34 PTR to ASCII "LISHUANGLI"
EBP 0014FE4C
ESI 00402197 CRACKME.00402197
EDI 00401128 CRACKME.WndProc
```

随后又调用`0x004013C2`处的函数，经过追踪后发现其作用是对处理后的字符串（全部大写）依次取出每个字符并且将其ascii码累加存放在EDI寄存器中，最终串"LISHUANGLI"经过累加后得到的值为0x2F0，之后再xor 0x5678，代码如下：

```
8A1E      MOV BL,BYTE PTR DS:[ESI]
84DB      TEST BL,BL
74 05     JZ SHORT 004013D1
03FB      ADD EDI,EBX
46        INC ESI
EB F5     JMP SHORT 004013C6
```

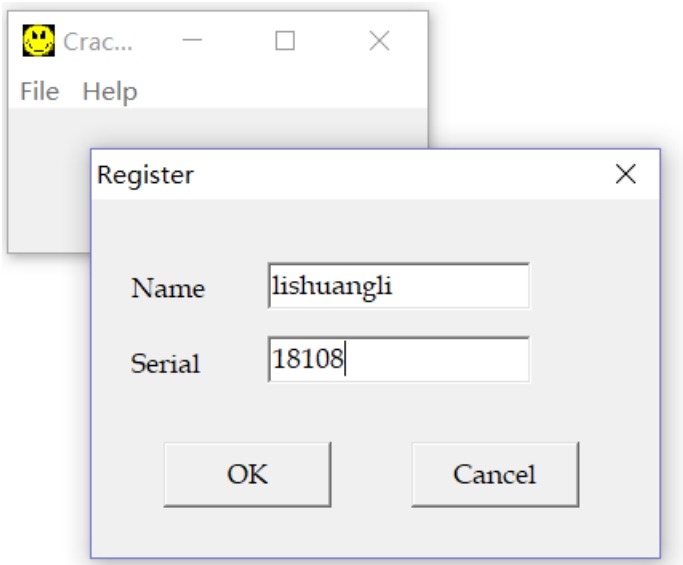
然后就是调用`0x004013D8`处的函数。在这个函数中对输入的Serial“12345666”进行每次乘0xA然后加每一位的值（ascii码减30后的值即为其字面值）的循环并放在EDI寄存器中，最后再对EDI寄存器xor 0x1234，得到的值为0xBC7376。

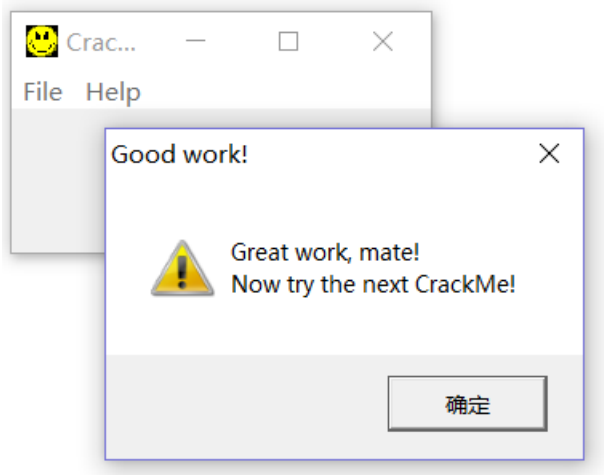
这样对Name和Serial的处理全部完成，然后就是对其处理后的两个数进行比较，随便输入了"12345666"显然不符合，但是通过以上分析过程可以推断出对一个字符串的编码序列算法如下：

- 输入的Name字符串全部转换为大写得到S1
- 对S1每一位的ascii码累加得到N1
- N1 xor 0x5678得到N2
- N2 xor 0x1234得到最终的正确Serial

针对字符串“lishuangli”计算可得Serial = 0x2F0 xor 0x5678 xor 0x1234 = 0x46BC = 18108

输入进行验证可知正确！





- 附加实验**CRACKME1.exe**

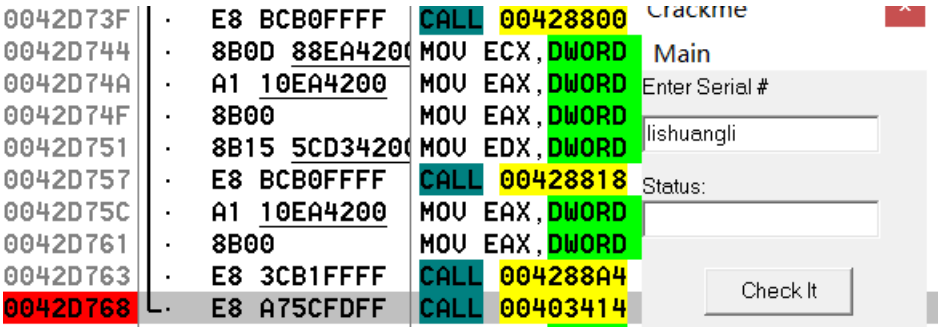
  首先分析这个程序，要求是输入一串Serial，如果正确的话会返回提示信息，否则会返回**Wrong Code DUDE**，所以目标就是能够找到正确的Serial输入进行破解。

  用ollydbg打开之后程序的入口在地址`0042D728`，从此处开始执行，然后分析汇编代码

```
0042D728  $  55              PUSH EBP
0042D729  .  8BEC            MOV EBP,ESP
0042D72B  .  83C4 F4         ADD ESP,-0C
0042D72E  .  B8 40D64200     MOV EAX,0042D640
0042D733  .  E8 2078FDFF     CALL 00404F58
0042D738  .  A1 10EA4200     MOV EAX,DWORD PTR DS:[42EA10]
0042D73D  .  8B00            MOV EAX,DWORD PTR DS:[EAX]
0042D73F  .  E8 BCB0FFFF     CALL 00428800
0042D744  .  8B0D 88EA4200   MOV ECX,DWORD PTR DS:[42EA88]
0042D74A  .  A1 10EA4200     MOV EAX,DWORD PTR DS:[42EA10]
0042D74F  .  8B00            MOV EAX,DWORD PTR DS:[EAX]
0042D751  .  8B15 5CD34200   MOV EDX,DWORD PTR DS:[42D35C]
0042D757  .  E8 BCB0FFFF     CALL 00428818
0042D75C  .  A1 10EA4200     MOV EAX,DWORD PTR DS:[42EA10]
0042D761  .  8B00            MOV EAX,DWORD PTR DS:[EAX]
0042D763  .  E8 3CB1FFFF     CALL 004288A4
0042D768  L. E8 A75CFDFF     CALL 00403414
```

  可以看到从入口开始，中间会多次调用函数去执行相关的任务，可以推测程序首先是调用许多GUI相关的函数进行可视化，然后通过用户IO交互输入信息，并且最后进行序列正确性判定。所以我在最后一个调用，也就是`CALL 00403414`处设置一个断点，并执行。

  此时会弹出输入 窗口，输入字符串"lishuangli"



  但继续执行后程序直接返回了**Wrong Code DUDE**的提示，说明直接执行到了结束。所以接下来的思路是找到字符串"Wrong Code DUDE"所在地址。在ollydbg中打开列有所有字符串的表然后可以发现"Wrong Code"字符串所在地址为`0x0042D543`处。

```
0042D543 MOV EDX,0042D5A4                ASCII "Wrong Code DUDE"
0042D555 MOV EDX,0042D5BC                ASCII "Thanks you made it"
```

跳转到其所在地址之后在一个EBP入栈指令处（地址为`0x0042D510`）设置断点，然后再次执行，再次输入字符串"lishuangli"，然后单步执行，执行到地址`0x0042D534`处有：

```
0042D52F  ·  E8 54CCFEFF   CALL 0041A188                    [CRACK1_(1).0041A188
0042D534  ·  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]     ASCII "lishuangli"
0042D537  ·  BA 90D54200   MOV EDX,0042D590                 ASCII "Benadryl"
0042D53C  ·  E8 8F63FDFF   CALL 004038D0                    [CRACK1_(1).004038D0
0042D541  ·  74 12         JZ SHORT 0042D555
0042D543  ·  BA A4D54200   MOV EDX,0042D5A4                 ASCII "Wrong Code DUDE"
0042D548  ·  8B83 E8010000 MOV EAX,DWORD PTR DS:[EBX+1E8]
0042D54E  ·  E8 65CCFEFF   CALL 0041A1B8
0042D553  ·  EB 10         JMP SHORT 0042D565
0042D555  >  BA BCD54200   MOV EDX,0042D5BC                 ASCII "Thanks you made it"
```

分析以上汇编指令可知，首先是把用户输入的字符串“lishuangli”和另一未知字符串"Benadryl"移入寄存器EAX和EDX，然后调用`0x004038D0`处的函数进行相关的字符串处理，返回之后如果状态寄存器0标志指示结果为0，跳转到`0x0042D555`处，就是提示输入正确，所以要分析`0x004038D0`处函数的处理结果。进入处理函数之后单步执行数次之后发现相关的字符串移入了相关寄存器：

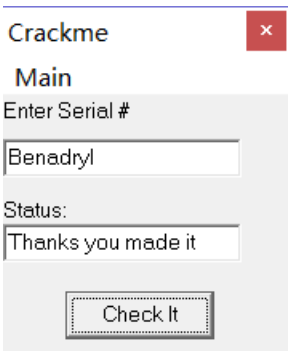
```
004038D3  ·  89C6          MOV ESI,EAX
004038D5  ·  89D7          MOV EDI,EDX
004038D7  ·  39D0          CMP EAX,EDX
004038D9  ·  0F84 8F00000( JE 0040396E
```

```
Registers (FPU)
EAX 01695A30 ASCII "lishuangli"
ECX 73F35792
EDX 0042D590 ASCII "Benadryl"
EBX 01693418
ESP 0014F8B0
EBP 0014F8D4
ESI 01695A30 ASCII "lishuangli"
EDI 0042D590 ASCII "Benadryl"
```

并且，CMP指令比较EAX和EDX的内容后如果相等则跳转到`0x0040396E`处，这样再跳转到相应的地址`0x0040396E`进行查看:

```
0040396E  >  5F            POP EDI
0040396F  ·  5E            POP ESI
00403970  ·  5B            POP EBX
00403971  ·  C3            RETN
```

直接出栈返回了！并且此时比较结果为相等，所以状态寄存器里保留着零标志为1，返回之后会跳转到提示成功的地址处，故只要输入"Benadryl"字符串即可破解成功。



除了直接判断之外，在此函数还有详细的算法流程对两个字符串进行比较判断，我阅读汇编代码后得到的程序中判定思路如下：

```
eax <- len(user_s) edx <- len(right_s)
eax1 = eax0 - edx0
if eax1>0 jump to 38F3
else
      edx1 = edx0 + eax1 = eax0 2,8
      edx1 = edx1 / 4 = eax0 /4 2,2
      if edx1 == 0 jump to 391F
      else{
            比较两个字符串前四个字节，不相等 jump to 3959
            edx--，if edx == 0 jump to 3919
            比较两个字符串下四个字节，不相等 jump to 3959
            esi += 8，edi += 8
            edx--，if edx ! = 0，go back
      }
....
3959：
比较ebx和ecx中第一个字节，不相等 return
比较ebx和ecx中第二个字节，不相等 return
ecx >> 0x10，ebx >> 0x10
比较前两个字节，相等则返回且提示正确，否则错误
```