关于字符串的签到题c语言,[原创]第一题：无限流[签到题] Writeup

转载

杉木優子 于 2021-05-20 14:59:01 发布 72 收藏
文章标签： 关于字符串的签到题c语言

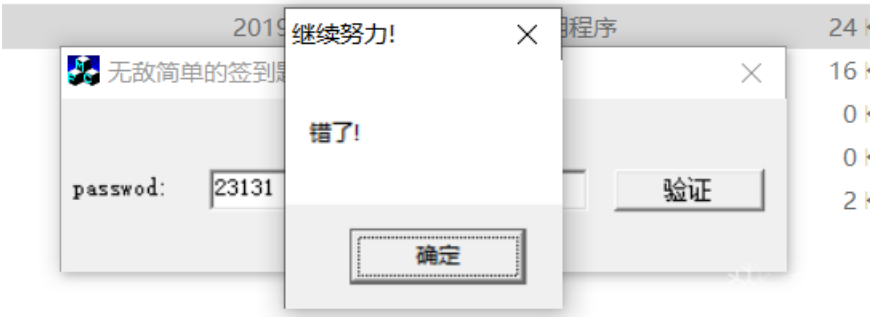
title: 看雪CFT第一题writeup date: 2019-12-02 15:53:59 tags: - ctf

IDA 版本: 6.8

题目连接:https://ctf.pediy.com/game-season_fight-125.htm

跟着思路写的。。。。有点乱，千万分析汇成最后一段python代码

输入成功失败



拖入IDA 中，搜索关键词

这段是输入错误时的提示，右键rename成,方便查看

```
.text:004017B0
.text:004017B0 error           proc near               ; CODE XREF: checkresult:loc_401815↓p
.text:004017B0                                         ; sub_401830:loc_4018EB↓p
.text:004017B0
.text:004017B0 hProcess        = dword ptr -4
.text:004017B0
.text:004017B0                 push    ebp
.text:004017B1                 mov     ebp, esp
.text:004017B3                 sub     esp, 44h
.text:004017B6                 push    ebx
.text:004017B7                 push    esi
.text:004017B8                 push    edi
.text:004017B9                 push    0               ; uType
.text:004017BB                 push    offset asc_40358C ; "继
.text:004017C0                 push    offset aA       ; "错了!"
.text:004017C5                 push    0               ; hWnd
.text:004017C7                 call    ds:MessageBoxA
.text:004017CD                 call    ds:GetCurrentProcess
.text:004017D3                 mov     [ebp+hProcess], eax
.text:004017D6                 push    0               ; uExitCode
.text:004017D8                 mov     eax, [ebp+hProcess]
.text:004017DB                 push    eax             ; hProcess
.text:004017DC                 call    ds:TerminateProcess
.text:004017E2                 pop     edi
.text:004017E3                 pop     esi
.text:004017E4                 pop     ebx
.text:004017E5                 mov     esp, ebp
.text:004017E7                 pop     ebp
.text:004017E8                 retn
.text:004017E8 error           endp
.text:004017E8
```

他上面这段就是成功的提示，成success

找到调用他的地方，发现是通过strcmp字符串对比，对比成功就显示成功,失败就显示失败

```
.text:004017F0 ; int __cdecl checkresult(char *Str1)
.text:004017F0 checkresult     proc near               ; CODE XREF: sub_401830+D7↓p
.text:004017F0
.text:004017F0 Str1            = dword ptr  8
.text:004017F0
.text:004017F0                 push    ebp
.text:004017F1                 mov     ebp, esp        ; 两个寄存器ebp,esp都保存上个函数的ebp值
.text:004017F3                 sub     esp, 40h
.text:004017F6                 push    ebx
.text:004017F7                 push    esi
.text:004017F8                 push    edi
.text:004017F9                 push    offset Str2     ; "goluck!"
.text:004017FE                 mov     eax, [ebp+Str1]
.text:00401801                 push    eax             ; Str1
.text:00401802                 call    strcmp
.text:00401807                 add     esp, 8
.text:0040180A                 test    eax, eax
.text:0040180C                 jnz     short loc_401815
.text:0040180E                 call    success
.text:00401813                 jmp     short loc_40181A
.text:00401815 ; ---------------------------------------------------------------------------
.text:00401815
.text:00401815 loc_401815:                             ; CODE XREF: checkresult+1C↑j
.text:00401815                 call    error
.text:0040181A
.text:0040181A loc_40181A:                             ; CODE XREF: checkresult+23↑j
.text:0040181A                 pop     edi
.text:0040181B                 pop     esi
.text:0040181C                 pop     ebx
.text:0040181D                 mov     esp, ebp
.text:0040181F                 pop     ebp
.text:00401820                 retn
0000180E 000000000040180E: checkresult+1E (Synchronized with Hex View-1)
```

应该就是将输入的和str2 ()对比,对比成功就成功检查结果段的代码

.text:004017F0 ; int __cdecl checkresult(char *Str1)

.text:004017F0 checkresult proc near ; CODE XREF: sub_401830+D7•p

.text:004017F0

.text:004017F0 Str1 = dword ptr 8

.text:004017F0

下面这三步是在一起的似乎。。。

.text:004017F0 push ebp

.text:004017F1 mov ebp, esp ; 两个寄存器ebp,esp都保存上个函数的ebp值

.text:004017F3 sub esp, 40h

.text:004017F6 push ebx

.text:004017F7 push esi

.text:004017F8 push edi

.text:004017F9 push offset Str2 ; "goluck!"

.text:004017FE mov eax, [ebp+Str1]

```
.text:00401801              push    eax             ; Str1 eax 是Str1,就是第二个参数
.text:00401802              call    strcmp ;调用字符串比较函数，前面是将参数压入栈
.text:00401807              add     esp, 8 ;？？
.text:0040180A              test    eax, eax ;检查eax是不是0,检查的结果设置到zf标记位
.text:0040180C              jnz     short loc_401815 ; 如果zf不等于0,就跳到失败这里
.text:0040180E              call    success
.text:00401813              jmp     short loc_40181A
```

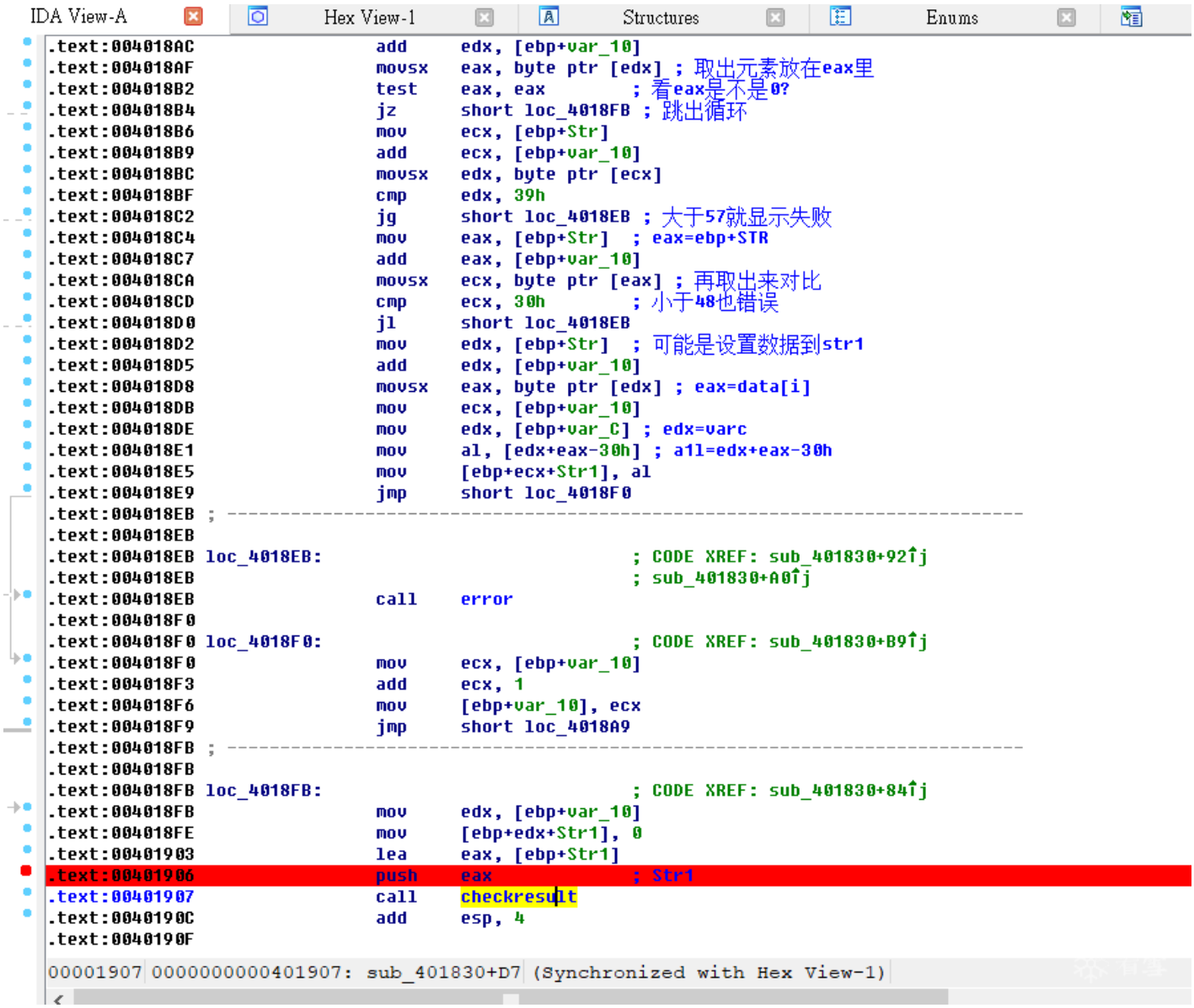
```
if(strcmp(str1,str2)!=0)
{
error()
}
else
success
```

str2是固定的,str1是输入的，但是直接将goluck!输入进去是不行的，说明是输入的正确答案,经过某种处理就变成goluck!,说明上面有个过程是

输入->str1

点击 CODE XREF: sub_401830+D7•p 跳到引用的地方

定位到这里，应该是上面有个循环



检查转换的完整代码

判断字符串长度是不是0，0的话直接显示失败

循环输入的字符串 3

.text:00401830 ; Attributes: bp-based frame

.text:00401830

.text:00401830 sub_401830 proc near ; DATA XREF: .rdata:00403548•o

.text:00401830

.text:00401830 Str1 = byte ptr -18h

.text:00401830 var_10 = dword ptr -10h

.text:00401830 var_C = dword ptr -0Ch

.text:00401830 Str = dword ptr -8

.text:00401830 var_4 = dword ptr -4

```
.text:00401830
.text:00401830                push    ebp
.text:00401831                mov     ebp, esp
.text:00401833                sub     esp, 58h
.text:00401836                push    ebx
.text:00401837                push    esi
.text:00401838                push    edi
.text:00401839                mov     [ebp+var_4], ecx
.text:0040183C                mov     eax, [ebp+var_4]
.text:0040183F                add     eax, 64h
.text:00401842                push    eax             ; struct CString *
.text:00401843                push    3EAh            ; int
.text:00401848                mov     ecx, [ebp+var_4] ; this
.text:0040184B                call    ?GetDlgItem@CWnd@@QBEPAV1@H@Z ; CWnd::GetDlgItem(int)
.text:00401850                mov     ecx, eax        ; this
.text:00401852                call    ?GetWindowTextA@CWnd@@QBEXAAVCString@@@Z ;
CWnd::GetWindowTextA(CString &)
.text:00401857                mov     ecx, [ebp+var_4]
.text:0040185A                add     ecx, 64h
.text:0040185D                call    sub_401970
.text:00401862                push    eax             ; int
.text:00401863                mov     ecx, [ebp+var_4]
.text:00401866                add     ecx, 64h        ; this
.text:00401869                call    ?GetBuffer@CString@@QAEPADH@Z ; CString::GetBuffer(int)
.text:0040186E                mov     [ebp+Str], eax
.text:00401871                mov     ecx, [ebp+Str]
.text:00401874                push    ecx             ; Str
.text:00401875                call    strlen          ; 判断字符串长度是不是0,是0就提示输入
.text:0040187A                add     esp, 4
.text:0040187D                test    eax, eax
.text:0040187F                jnz     short loc_401894
```

```
.text:00401881                push   0               ; unsigned int
.text:00401883                push   0               ; char *
.text:00401885                push   offset aIFIpassword ; "请输入password!"
.text:0040188A                mov    ecx, [ebp+var_4] ; this
.text:0040188D                call   ?MessageBoxA@CWnd@@QAEHPBD0I@Z ; CWnd::MessageBoxA(char
const *,char const *,uint)
.text:00401892                jmp    short loc_40190F
.text:00401894 ; ---------------------------------------------------------------------------
.text:00401894
.text:00401894 loc_401894:                            ; CODE XREF: sub_401830+4F•j
.text:00401894                mov    [ebp+var_C], offset aCukOgl ; "cuk!ogl"
.text:0040189B                mov    [ebp+var_10], 0
.text:004018A2                mov    [ebp+var_10], 0
.text:004018A9
```

----------循环开始

根据输入的值计算 成str1

```
.text:004018A9 loc_4018A9:                            ; CODE XREF: sub_401830+C9•j
.text:004018A9                mov    edx, [ebp+Str]  ; edx指向str(输入的字符串)
.text:004018AC                add    edx, [ebp+var_10] ; var_10 作为数组索引
.text:004018AF                movsx  eax, byte ptr [edx] ; 取出第var_10个元素放在eax里
.text:004018B2                test   eax, eax        ; 看eax是不是0?
.text:004018B4                jz     short loc_4018FB ; 表示没取到?跳出循环
```

检查当前遍历的字符串在不在'0'到'9'之间，就是说答案最后只能是数字

```
.text:004018B6                mov    ecx, [ebp+Str] ecx = str1
.text:004018B9                add    ecx, [ebp+var_10]
.text:004018BC                movsx  edx, byte ptr [ecx]
.text:004018BF                cmp    edx, 39h 0x39对应的是'1'这个字符串
.text:004018C2                jg     short loc_4018EB ; 大于57就显示失败
```

```
.text:004018C4                mov     eax, [ebp+Str]  ; eax=ebp+STR
.text:004018C7                add     eax, [ebp+var_10]
.text:004018CA                movsx   ecx, byte ptr [eax] ; 再取出来对比
.text:004018CD                cmp     ecx, 30h        ; 小于48也错误
.text:004018D0                jl      short loc_4018EB
```

根据输入的值 转换

```
.text:004018D2                mov     edx, [ebp+Str]  ; 可能是设置数据到str1
.text:004018D5                add     edx, [ebp+var_10]
.text:004018D8                movsx   eax, byte ptr [edx] ; eax=data[i]
.text:004018DB                mov     ecx, [ebp+var_10]
.text:004018DE                mov     edx, [ebp+var_C] ;
```

al就是str1 的某个索引下的值

a1 = edx_eax-0x30

edx 是上面var_c 是上面 [ebp+var_C], offset aCukOgl ; "cuk!ogl"

就是edx是这个字符串

edx[offset],offset就输入的ascii码-0x30

假设输入是input

str1[index]=aCukOgl[input[index]-30h]

最后的str1要是goluck!

然后每个倒过来计算一下就行

比如第一个index=0时

'g'=aCukOgl[input[index]-30h]

g在aCukOgl里的index是5

input[index]-0x30=5

input[index]=0x30+5=53='5'

```
.text:004018E1                mov     al, [edx+eax-30h] ; a1l=edx+eax-30h
ecx上面设置成了var_10的值，就是 当前数组的索引, str[var_10]=al
.text:004018E5                mov     [ebp+ecx+Str1], al
.text:004018E9                jmp     short loc_4018F0
.text:004018EB ; ---------------------------------------------------------------------------
.text:004018EB
.text:004018EB loc_4018EB:                            ; CODE XREF: sub_401830+92•j
.text:004018EB                                        ; sub_401830+A0•j
.text:004018EB                call    error
.text:004018F0
.text:004018F0 loc_4018F0:                            ; CODE XREF: sub_401830+B9•j
.text:004018F0                mov     ecx, [ebp+var_10]
.text:004018F3                add     ecx, 1
.text:004018F6                mov     [ebp+var_10], ecx
.text:004018F9                jmp     short loc_4018A9
.text:004018FB ; ---------------------------------------------------------------------------
.text:004018FB
.text:004018FB loc_4018FB:                            ; CODE XREF: sub_401830+84•j
.text:004018FB                mov     edx, [ebp+var_10]
.text:004018FE                mov     [ebp+edx+Str1], 0
.text:00401903                lea     eax, [ebp+Str1]
.text:00401906                push    eax             ; Str1
.text:00401907                call    checkresult
.text:0040190C                add     esp, 4
.text:0040190F
.text:0040190F loc_40190F:                            ; CODE XREF: sub_401830+62•j
.text:0040190F                pop     edi
```

.text:00401910 pop esi

.text:00401911 pop ebx

.text:00401912 mov esp, ebp

.text:00401914 pop ebp

.text:00401915 retn

.text:00401915 sub_401830 endp

根据上面的算法，得到的解题脚本

```
aCukOgl = list('cuk!ogl')
answer = list('goluck!')
# 假设输入input
# answer[index]=aCukOgl[input[index]-0x30],index是input循环遍历的索引
# 要计算的是input
for c in answer:
index=aCukOgl.index(c)
x_data= index+0x30
# print(x_data)
print(chr(x_data))
```

得到结果

5461023