

xctf攻防世界 REVERSE 高手进阶区 BABYRE

原创

[i8947943](#) 于 2022-04-13 18:14:01 发布 895 收藏

分类专栏: [攻防世界reverse之路](#) 文章标签: [reverse](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/i8947943/article/details/124153847>

版权



[攻防世界reverse之路](#) 专栏收录该内容

14 篇文章 0 订阅

订阅专栏

0x01. 进入环境，下载附件

给出的babyRE后缀文件，不懂是什么玩意，按照常规流程进行操作吧

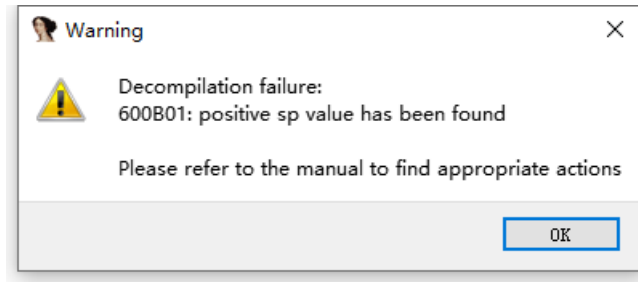
0x02. 问题分析

使用IDA打开，找到main函数，F5反编译，得到代码如下：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    char s; // [rsp+0h] [rbp-20h]
    int v5; // [rsp+18h] [rbp-8h]
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 181; ++i )
    {
        envp = (const char **)((unsigned __int8 *)judge + i) ^ 0xCu;
        *((_BYTE *)judge + i) ^= 0xCu;
    }
    printf("Please input flag:", argv, envp);
    __isoc99_scanf("%20s", &s);
    v5 = strlen(&s);
    if ( v5 == 14 && (unsigned int)judge(&s) )
        puts("Right!");
    else
        puts("Wrong!");
    return 0;
}
```

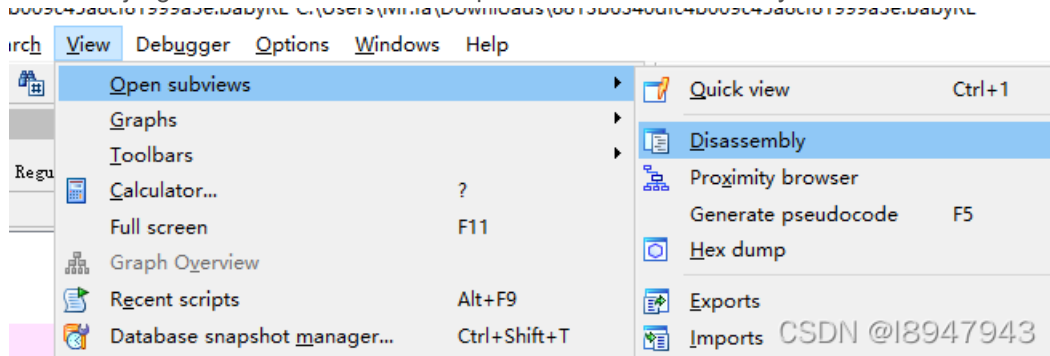
我们可以看到，使用scanf函数得到输入s后，其长度为14且经过judge函数后，才能得到Right! 结果。我们尝试双击judge函数，如图：



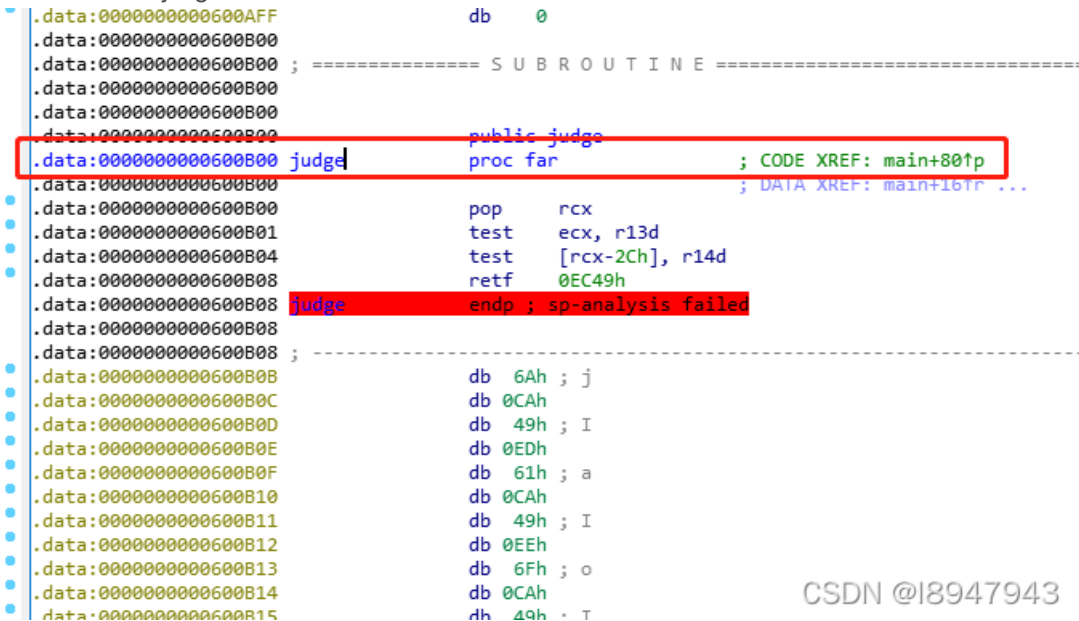
发现judge函数无法被反编译，TMD在这真的无语，到底出发点是什么？直接卷writeup，发现看不懂解释，直接写了一篇教程，先跟着做出来吧！

0x03. 模拟wp操作

我们观察完代码，需要找到judge函数的地址，点击view->open subviews->disassembly，如图：

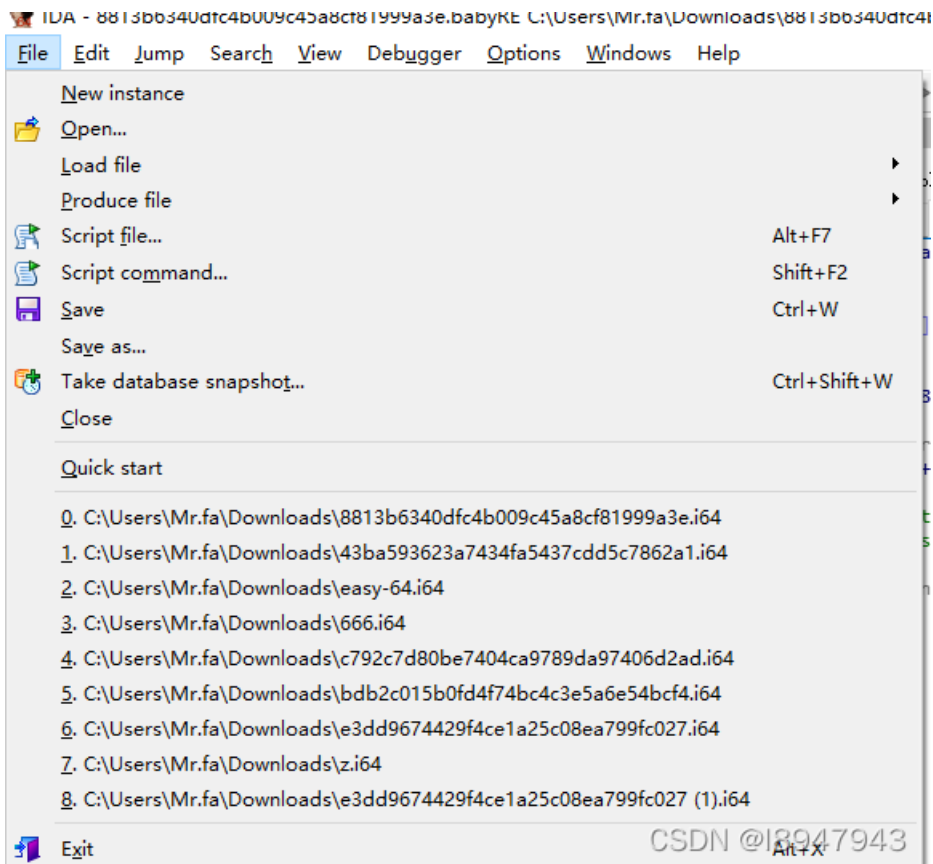


得到text类型的视图，我们找到judge函数的地址位置，如图：



CSDN @I8947943

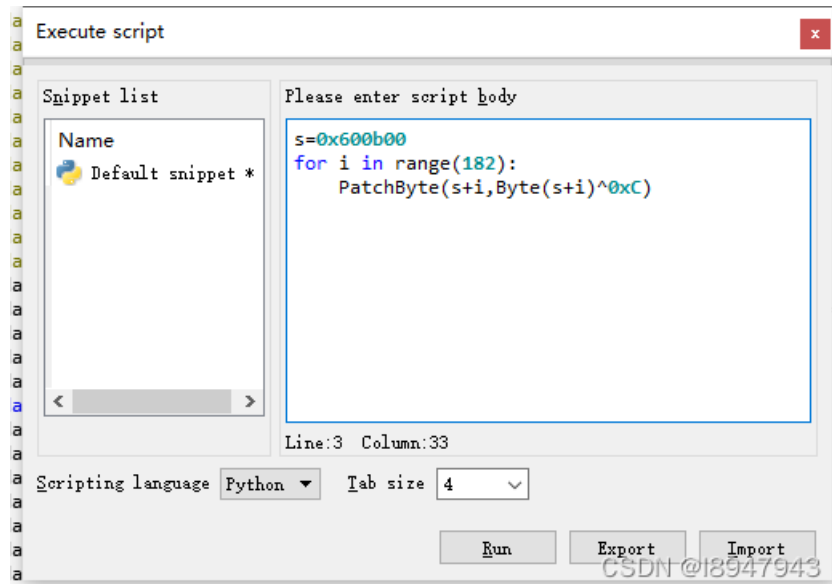
是从0x600B00开始的，我们使用脚本，将加密的judge函数还原出来。点击file->script command，使用python脚本，如图：



代码片段:

```
s=0x600b00
for i in range(182):
    PatchByte(s+i,Byte(s+i)^0xC)
```

如图:



注意: 这段代码只能运行一次

点击run会增加一大段代码，如图：

```
.data:0000000000600B00 judge public judge ; CODE XREF: main+80↑p
.data:0000000000600B00 proc far ; DATA XREF: main+16↑r ...
.data:0000000000600B00 push rbp
.data:0000000000600B01 mov rbp, rsp
.data:0000000000600B04 mov [rbp-28h], rdi
.data:0000000000600B08 mov byte ptr [rbp-20h], 66h
.data:0000000000600B08 judge endp ; sp-analysis failed
.data:0000000000600B08
.data:0000000000600B0C mov byte ptr [rbp-1Fh], 6Dh
.data:0000000000600B10 mov byte ptr [rbp-1Eh], 63h
.data:0000000000600B14 mov byte ptr [rbp-1Dh], 64h
.data:0000000000600B18 mov byte ptr [rbp-1Ch], 7Fh
.data:0000000000600B1C mov byte ptr [rbp-1Bh], 6Bh
.data:0000000000600B20 mov byte ptr [rbp-1Ah], 37h
.data:0000000000600B24 mov byte ptr [rbp-19h], 64h
.data:0000000000600B28 mov byte ptr [rbp-18h], 3Bh
.data:0000000000600B2C mov byte ptr [rbp-17h], 56h
.data:0000000000600B30 mov byte ptr [rbp-16h], 60h
.data:0000000000600B34 mov byte ptr [rbp-15h], 3Bh
.data:0000000000600B38 mov byte ptr [rbp-14h], 6Eh
.data:0000000000600B3C mov byte ptr [rbp-13h], 70h
.data:0000000000600B40 mov dword ptr [rbp-4], 0
.data:0000000000600B47 jmp short loc_600B71
-----
.data:0000000000600B49 loc_600B49: ; CODE XREF: .data:0000000000600B75↓j
.data:0000000000600B49 mov eax, [rbp-4]
.data:0000000000600B4C movsxd rdx, eax
.data:0000000000600B4F mov rax, [rbp-28h]
.data:0000000000600B53 add rax, rdx
.data:0000000000600B56 mov edx, [rbp-4]
.data:0000000000600B59 movsxd rcx, edx
.data:0000000000600B5C mov rdx, [rbp-28h]
.data:0000000000600B60 add rdx, rcx
.data:0000000000600B63 movzx edx, byte ptr [rdx]
.data:0000000000600B66 mov ecx, [rbp-4]
.data:0000000000600B69 xor edx, ecx
.data:0000000000600B6B mov [rax], dl
.data:0000000000600B6D add dword ptr [rbp-4], 1
.data:0000000000600B71
-----
.data:0000000000600B71 loc_600B71: ; CODE XREF: .data:0000000000600B47↑j
.data:0000000000600B71 cmp dword ptr [rbp-4], 0Dh
```

我们需要将上述红底色的代码，按U键（取消原来的定义），再按C（重新生成汇编代码），选中600B00-600BB5（judge 的起止位置）按P（重新生成 function）。这时就可以按F5生成judge 的伪代码了。代码如下：

```

signed __int64 __fastcall judge(__int64 a1)
{
    char v2; // [rsp+8h] [rbp-20h]
    char v3; // [rsp+9h] [rbp-1Fh]
    char v4; // [rsp+Ah] [rbp-1Eh]
    char v5; // [rsp+Bh] [rbp-1Dh]
    char v6; // [rsp+Ch] [rbp-1Ch]
    char v7; // [rsp+Dh] [rbp-1Bh]
    char v8; // [rsp+Eh] [rbp-1Ah]
    char v9; // [rsp+Fh] [rbp-19h]
    char v10; // [rsp+10h] [rbp-18h]
    char v11; // [rsp+11h] [rbp-17h]
    char v12; // [rsp+12h] [rbp-16h]
    char v13; // [rsp+13h] [rbp-15h]
    char v14; // [rsp+14h] [rbp-14h]
    char v15; // [rsp+15h] [rbp-13h]
    int i; // [rsp+24h] [rbp-4h]

    v2 = 102;
    v3 = 109;
    v4 = 99;
    v5 = 100;
    v6 = 127;
    v7 = 107;
    v8 = 55;
    v9 = 100;
    v10 = 59;
    v11 = 86;
    v12 = 96;
    v13 = 59;
    v14 = 110;
    v15 = 112;
    for ( i = 0; i <= 13; ++i )
        *(_BYTE *)(i + a1) ^= i;
    for ( i = 0; i <= 13; ++i )
    {
        if ( *(_BYTE *)(i + a1) != *(&v2 + i) )
            return 0LL;
    }
    return 1LL;
}

```

发现judge函数主要是通过14个变量，每次与i进行异或，那么逆向脚本也就可以出来了：

```

v = [102, 109, 99, 100, 127, 107, 55, 100, 59, 86, 96, 59, 110, 112]
flag = ''
for i in range(14):
    flag += chr(v[i]^i)

print(flag)

```

得到最终的结果为： `flag{n1c3_j0b}`

0x04. 总结

tmd太难了，这个题，实在想不到为啥要用python还原代码，而且写法还那么奇葩。这个题做了两天，真的各种试错。难！！！！