

# [SUCTF2018]babyre [ACTF新生赛2020]fungame

原创

寻梦&之璐 于 2021-06-02 16:32:43 发布 282 收藏

分类专栏: [ctf](#)

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/CSNN2019/article/details/117443216>

版权



[ctf 专栏收录该内容](#)

145 篇文章 7 订阅

订阅专栏

## 文章目录

[\[SUCTF2018\]babyre](#)

[惯用思维](#)

[常人思维](#)

[GAMEOVER](#)

[\[ACTF新生赛2020\]fungame](#)

[int \\_\\_cdecl sub\\_401340\(int a1\)](#)

[int \\_\\_cdecl sub\\_4013BA\(char \\*Source\)](#)

## [SUCTF2018]babyre

### 惯用思维

首先呢, 是个bin文件, 需要用binwalk把文件提取出来。

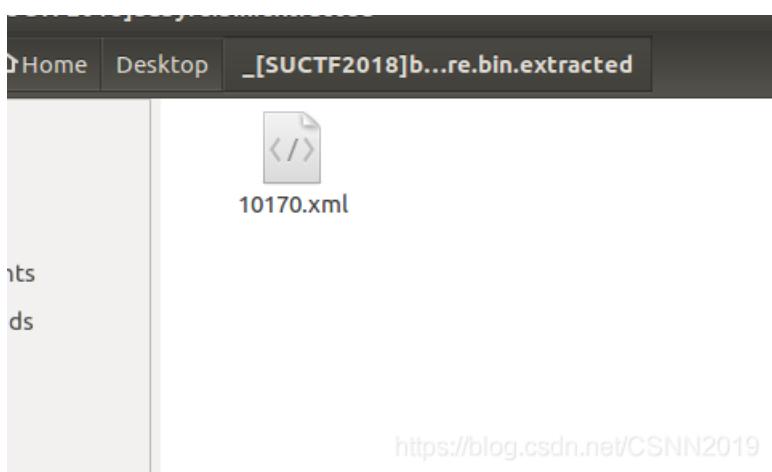
```
binwalk -e [SUCTF2018\]babyre.bin
```

```
ak666@ubuntu:~/Desktop$ cd Desktop
ak666@ubuntu:~/Desktop$ binwalk -e [SUCTF2018\babyre.bin]

DECIMAL      HEXADECIMAL      DESCRIPTION
-----      -----      -----
0            0x0      Microsoft executable, portable (PE)
65904        0x10170      XML document, version: "1.0"

ak666@ubuntu:~/Desktop$
```

<https://blog.csdn.net/CSNN2019>



<https://blog.csdn.net/CSNN2019>

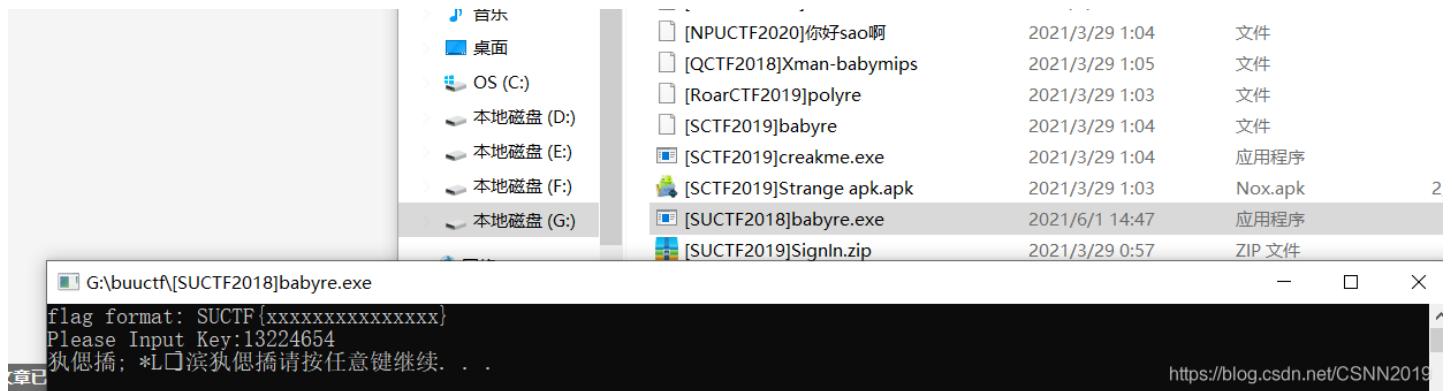
是一个 **xml** 文件，我打开后除了正常结构就是乱码。。。

然后就没法分析了。。。xml不能放在ida里面分析。。

## 常人思维

```
bash: ./[SUCTF2018]babyre.bin]: No such file or directory
ak666@ubuntu:~/Desktop$ file [SUCTF2018\babyre.bin]
[SUCTF2018\babyre.bin]: PE32+ executable (console) x86-64, for MS Windows
ak666@ubuntu:~/Desktop$
```

查看一下文件类型，居然是64位的exe程序，任何东西都保持怀疑的态度。。笑死人。。。



如下就是数据库，可以利用这些字符进行运算得到一些其它的字符进行输出

```
v7[0] = 2;
v7[1] = 3;
v7[2] = 2;
v7[3] = 1;
v7[4] = 4;
v7[5] = 7;
v7[6] = 4;
v7[7] = 5;
v7[8] = 10;
v7[9] = 11;
v7[10] = 10;
v7[11] = 9;
v7[12] = 14;
v7[13] = 15;
v7[14] = 12;
v7[15] = 13;
v7[16] = 16;
v7[17] = 19;
v7[18] = 16;
v7[19] = 17;
v7[20] = 20;
v7[21] = 23;
v7[22] = 22;
v7[23] = 19;
v7[24] = 28;
v7[25] = 25;
v7[26] = 30;
v7[27] = 31;
v7[28] = 28;
v7[29] = 25;
v7[30] = 26;
v7[31] = 31;
qmemcpy(v8, "$!``$!\"#().+$-&/81:;4=>7092;<567HIBBDDFGHIJJMMONPPRSUTVWYYZ[\\"\\]^`^`ccdeggiiKKlmnnpprstuwwxy{{}}",
", 94);
v8[94] = 127;
v8[95] = 127;
v8[96] = -127;
v8[97] = -127;
v8[98] = -125;
v8[99] = -125;
v8[100] = -116;
v8[101] = -115;
v8[102] = -114;
```

```
v8[103] = -113;
v8[104] = -120;
v8[105] = -119;
v8[106] = -118;
v8[107] = -117;
v8[108] = -116;
v8[109] = -115;
v8[110] = -114;
v8[111] = -121;
v8[112] = -104;
v8[113] = -111;
v8[114] = -110;
v8[115] = -109;
v8[116] = -108;
v8[117] = -107;
v8[118] = -106;
v8[119] = -105;
v8[120] = -104;
v8[121] = -103;
v8[122] = -102;
v8[123] = -102;
v8[124] = -100;
v8[125] = -100;
v8[126] = -98;
v8[127] = -98;
v8[128] = -96;
v8[129] = -96;
v8[130] = -94;
v8[131] = -94;
v8[132] = -92;
v8[133] = -92;
v8[134] = -90;
v8[135] = -90;
v8[136] = -88;
v8[137] = -88;
v8[138] = -86;
v8[139] = -86;
v8[140] = -84;
v8[141] = -84;
v8[142] = -82;
v8[143] = -82;
v8[144] = -80;
v8[145] = -79;
v8[146] = -78;
v8[147] = -77;
```

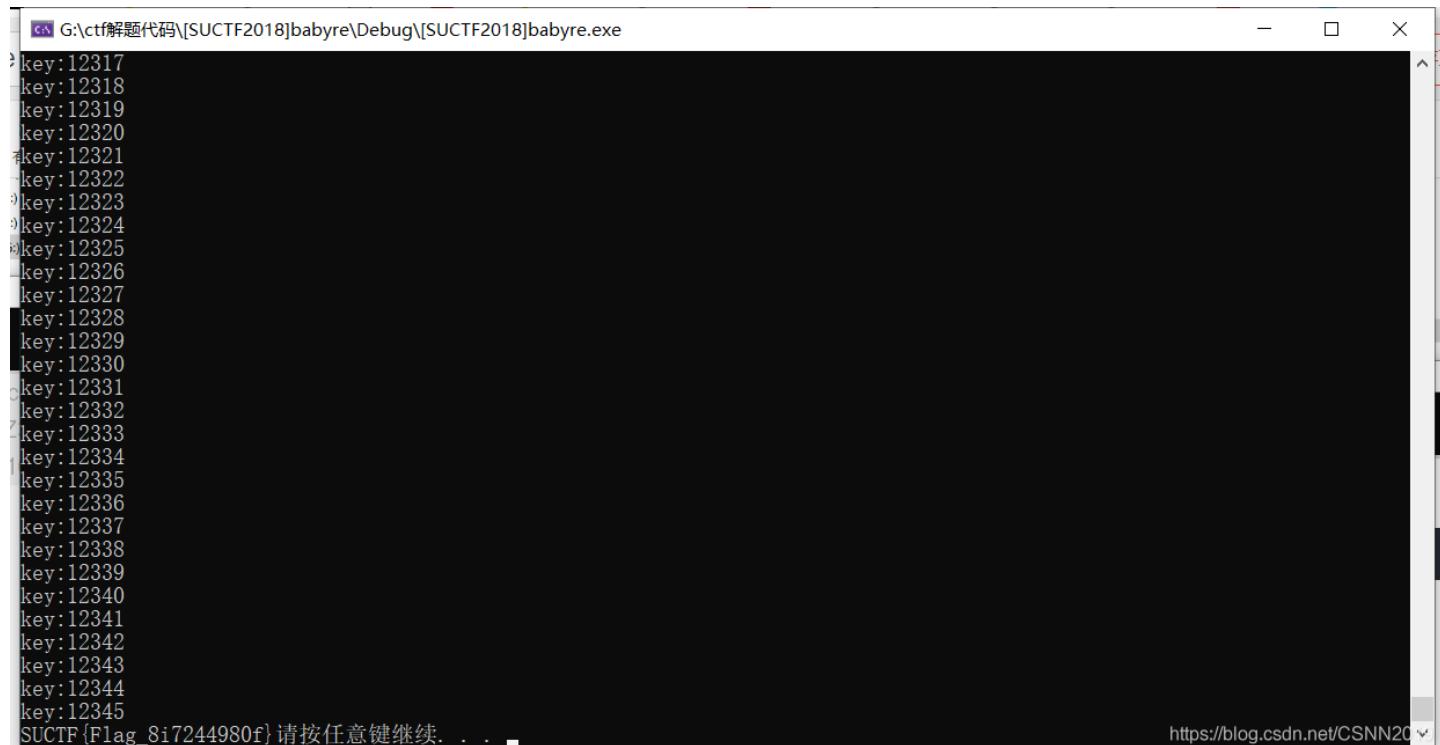
这里的话就是利用数据区的字符进行一系列运算，然后进行flag的赋值，然后接着一个函数进行字符输出。

```
while ( v9[30] )
{
    --v9[30];
    for ( j = 22; j; v9[j] |= v13 << v9[30] )
    {
        v12 = v7[22 * v9[30] + --j];
        v13 = (v12 >> ((v10[0] >> (2 * v9[30])) & 3)) & 1;
    }
}
```

输入不同的key然后就会有不同的字符输出，我们需要找到输出flag的key，而flag字符的特征就是 **SUCTF**，如果就这种特征，那么就进行输出

```
if (flag[0] == 'S' && flag[1] == 'U' && flag[2] == 'C' && flag[3] == 'T' && flag[4] == 'F')
{
    for (int i = 0; i < 22; i++)
        printf("%c", flag[i]);
    system("pause");
}
```

## GAMEOVER



```
G:\ctf解题代码\[SUCTF2018]babyre\Debug\[SUCTF2018]babyre.exe
key:12317
key:12318
key:12319
key:12320
key:12321
key:12322
key:12323
key:12324
key:12325
key:12326
key:12327
key:12328
key:12329
key:12330
key:12331
key:12332
key:12333
key:12334
key:12335
key:12336
key:12337
key:12338
key:12339
key:12340
key:12341
key:12342
key:12343
key:12344
key:12345
SUCTF{Flag_8i7244980f} 请按任意键继续. . .
```

SUCTF{Flag\_8i7244980f}

## [ACTF新生赛2020]fungame

int \_\_cdecl sub\_401340(int a1)

```
int __cdecl sub_401340(int a1)
{
    char i; // [esp+1Fh] [ebp-9h]
    printf("Please input:");
    scanf("%s", a1);
    for ( i = 0; i <= 15; ++i )
    {
        if ( (*(BYTE *) (i + a1)) ^ *((BYTE *) y1 + i)) != y2[i] )
            exit(0);
    }
    return 0;
}
```

<https://blog.csdn.net/CSNN2019>

```
flag=[None]*16
b=""
y1=[
    0x23, 0x61, 0x3E, 0x69, 0x54, 0x41, 0x18, 0x4D, 0x6E, 0x3B,
    0x65, 0x53, 0x30, 0x79, 0x45, 0x5B
]
y2=[
    0x71, 0x04, 0x61, 0x58, 0x27, 0x1E, 0x4B, 0x22, 0x5E, 0x64,
    0x03, 0x26, 0x5E, 0x17, 0x3C, 0x7A
]
for i in range(len(y1)):
    a=hex(y2[i]^y1[i])
    b+=chr(y2[i]^y1[i])
    flag[i]=a
print(b)
```

Re\_1s\_So0\_funny!

然而却是错误，看第二个函数

### int \_\_cdecl sub\_4013BA(char \*Source)

```
int __cdecl sub_4013BA(char *Source)
{
    char Destination[12]; // [esp+1Ch] [ebp-Ch] BYREF

    strcpy(Destination, Source);
    strcpy(x, Source);
    return 0;
}

strcpy(Destination, Source);
```

这行代码导致栈溢出，然后导致返回地址被覆盖，紧接着返回的时候就没法正确返回，以至于程序过了第一个函数，第二个函数结束时，就自动gg。

我一直好奇和其它博主输入一样，然后栈溢出导致返回地址覆盖时，改变后却不一样，嗯哼？？？出题人强行整活。。

```
strcpy(x, Source);
```

这里有个全局变量x，要找这个线索，固定了前16字节字符，后面覆盖值需要自己算，然后跳入 `_sub_40233D` 这个函数。说白了就是一个pwn题，跟re无关。

```
.bss:0040604C          align 10h
.bss:00406050          public _x
.bss:00406050 ; char x[36]
.bss:00406050 _x        db 24h dup(?) ; DATA XREF: _sub_4013BA+1F↑o
.bss:00406050          ; _sub_40233D+B8↑o ...
.bss:00406074          public __bss_end__
.bss:00406074          end
```

`_sub_40233D` 这个函数是出题人自己搞的，它也不会自动覆盖跳，需要玩家手动算。欧了，base64，这题目对于re玩家毫无价值，对于pwn玩家有太简单。搞不懂re库为啥会有这种？？？

```
void __noreturn sub_40233D()
{
    char Str2[13]; // [esp+13h] [ebp-35h] BYREF
    char Str1[16]; // [esp+20h] [ebp-28h] BYREF
    char Str[12]; // [esp+30h] [ebp-18h] BYREF
    size_t v3; // [esp+3Ch] [ebp-Ch]

    printf("Please input again:");
    strcpy(Str2, "YTFzMF9wU24=");
    memset(Str, 0, sizeof(Str));
    memset(Str1, 0, sizeof(Str1));
    scanf("%s", Str);
    v3 = strlen(Str);
    sub_402421(Str, v3, Str1);
    if (!strcmp(Str1, Str2))
    {
        printf("%s%s", x, Str);
        exit(0);
    }
    exit(0);
}
```

<https://blog.csdn.net/CSNN2019>

输入: Re\_1s\_So0\_funny!, 返回地址如下:

```
.text:004013EB call    _strcpy
.text:004013D2 mov     eax, [ebp+Source]
.text:004013D5 mov     [esp+4], eax           ; Source
.text:004013D9 mov     dword ptr [esp], offset _x      ; Destination
.text:004013E0 call    _strcpy
.text:004013E5 mov     eax, 0
.text:004013EA leave
EIP: .text:004013EB retn
.text:004013EB _sub_4013BA endp
.text:004013EB
```

Path: G:\buuctf\[ACTF新生赛2020]fungame\tmp\fungame.exe  
Threads: 18836 4994 Ready fungame.exe

Decimal	Hex	State	Name
18836	4994	Ready	fungame.exe

Stack view: 0060FEDC 00402600 \_main+28  
0060FEF0 00090FD8 debug034:00090FD8  
0060FEE4 00000000  
0060FEE8 00000018  
0060FEEC 00402BDE \_\_do\_global\_ctors+2E  
0060FEF0 00402B80 \_\_do\_global\_dtors  
0060FEFA 0000003B  
0060FEFB 00000008  
0060FEFC 00090FD8 debug034:00090FD8  
UNKNOWN 0060FEDC: Stack[00004994]:00601 (Synchronized with ESP)

Hex View: 000007EB 004013EB: \_sub\_4013BA+31 (Synchronized with EIP)  
Output window

输入: Re\_1s\_So0\_funny!0, 返回地址如下:

The screenshot shows the IDA Pro interface with several windows open:

- IDA View-EIP**: Shows assembly code for the current function. The code includes:
 

```
.text:004013C7 lea    eax, [ebp+Destination]
.text:004013C9 mov    [esp], eax           ; Destination
.text:004013CB call   _strcpy
.text:004013D0 mov    eax, [ebp+Source]
.text:004013D2 mov    [esp+4], eax        ; Source
.text:004013D9 mov    dword ptr [esp], offset _x ; Destination
.text:004013E0 call   _strcpy
.text:004013E5 mov    eax, 0
.text:004013E9 leave
.text:004013EB ret
```
- Hex View-1**: Shows the raw memory dump of the program's memory.
- Stack view**: Shows the stack contents.
- General registers**: Shows the state of CPU registers (EAX, EBX, ECX, etc.).
- Modules**: Shows the loaded modules and their paths.
- Threads**: Shows the running threads.
- Memory dump**: A large pane showing the memory dump with annotations for global constructors and destructors.

这个题完全没必要继续研究，就是出题人强行整活，作为re题的话，直接没法做；作为pwn题的话，又太无聊，总结  
——凑数题（跳转不固定，flag固定，这咋做？脑洞？）