

XCTF asong

原创

[pipixia233333](#) 于 2019-05-09 23:10:26 发布 613 收藏

分类专栏: [逆向之旅](#)

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/qq_41071646/article/details/90043300

版权



[逆向之旅](#) 专栏收录该内容

128 篇文章 2 订阅

订阅专栏

说实话 这个题 还是比较难受的

当我看到 有三个文件的时候 我有点慌 感觉又是什么玩意的时候

但是仔细分析了一下 感觉这个题 还有点好玩

一个是elf 文件 一个是 文本文件 一个是out out 里面是需要我们解密的数据

然后 我们先分析一下elf

```
int64 __fastcall main(int64 a1, char **a2, char **a3)
{
    char *that_girl_index; // ST00_8
    const char *input; // ST08_8

    that_girl_index = malloc(188uLL);
    input = malloc(80uLL);
    init_s();
    scanf_s(input); // 字符最多100个
    flag_ss(input);
    get_file_thatgirl_index("that_girl", that_girl_index);
    sub_400E54(input, that_girl_index);
    return 0LL;
}
```

https://blog.csdn.net/qq_41071646

这里没有什么好说的我们 `get_file_thatgirl_index` 这个函数其实就是存的频率

```
{
    int v2; // eax
    int *v4; // [rsp+0h] [rbp-20h]
    char buf; // [rsp+13h] [rbp-Dh]
    int fd; // [rsp+14h] [rbp-Ch]
    unsigned __int64 v7; // [rsp+18h] [rbp-8h]

    v7 = __readfsqword(0x28u);
    fd = open(a1, 0, a2, a1);
    while ( read(fd, &buf, 1uLL) == 1 )
    {
        v2 = get_index(buf); // 10-36 是'A'-'z' or 'a'-'z'
        ++v4[v2]; // 这里是一个表 统计了 不同字符的频率
    }
    return close(fd);
}
```

https://blog.csdn.net/qq_41071646

0-9 下标是0-9 小写大写的下标都一样

然后 我们看一下下面的函数

```
4 int v4, // [rsp+10h] [rbp-44h]
5 char v5[56]; // [rsp+20h] [rbp-40h]
6 unsigned __int64 v6; // [rsp+58h] [rbp-8h]
7
8 v6 = __readfsqword(0x28u);
9 v4 = strlen(a1);
0 for ( i = 0; i < v4; ++i )
1     v5[i] = a2[get_index(a1[i])];
2 sub_400D33(v5);
3 sub_400DB4(v5, v4);
4 sub_400CC0(v5, "out", v4);
5 return __readfsqword(0x28u) ^ v6;
6 }
```

https://blog.csdn.net/qq_41071646

这里就很重要了 如果get不到 v5的值 那么我们就没有办法 进行下一步了 重点 就是 a2 的值哪里来的

这里我跑了一个脚本 模仿了 get_file_thatgirl_index 这个函数

```
#!/usr/bin/python3
#coding=utf8
if __name__ == '__main__':
    f=open("that_girl")
    l=[]
    for line in f:
        for i in line:
            if ord(i)>=ord('A') and ord(i)<=ord('Z'):
                l.extend(chr(ord(i)+32))
            else:
                l.extend(i)
    summ={}
    while '\n' in l:
        l.remove('\n')
    for ch in l:
        if(ch not in summ):
            summ[ch]=0
        summ[ch]+=1
    print(summ)
    data={}
    data=sorted(summ.items(),key=lambda x:x[0])
    print(data)

    f.close()
```

然后用 dbg 调试了一下 发现这两个相同 那么就很好解释了

```

pwndbg> x/30gx 0x603038
0x603038: 0x0000001e00000068 0x00000001d000000f
0x603048: 0x00000013000000a9 0x0000000430000026
0x603058: 0x000000000000003c 0x0000000270000014
0x603068: 0x000000076000001c 0x00000001a00000a5
0x603078: 0x00000003d0000000 0x0000000850000033
0x603088: 0x000000007000002d 0x0000000000000022
0x603098: 0x000000000000003e 0x0000000000000000
0x6030a8: 0x0000000000000000 0x0000000280000000
0x6030b8: 0x0000000000000047 0x0000000420000000
0x6030c8: 0x00000000000000f5 0x0000000000000000
0x6030d8: 0x0000000000000061 0x0000000000445341
0x6030e8: 0x0000000000000000 0x0000000000000000
0x6030f8: 0x0000000000000000 0x0000000000000000

```

其实这里就是 得出的 输入字符 对应 那个文件的 频率 假如 你输入的 T 而 文件出现了133 次 那么就代表了 v5[0]是133

OK 那么我们往下看

```

2 {
3  __int64 result; // rax
4  char v2[5]; // [rsp+13h] [rbp-5h]
5
6  v2[4] = 0;
7  *v2 = *a1;
8  while ( s[*&v2[1]] )
9  {
10     a1[*&v2[1]] = a1[s[*&v2[1]]];
11     *&v2[1] = s[*&v2[1]];
12 }
13
14 // v2[0]=a[0]
15 // a[index]=a[[s[index]]]
16 // index=s[index]
17 // while(s[index]==0)
18 // a[index]=a[0]
19
20 result = v2[0];
21 a1[*&v2[1]] = v2[0];
22 return result;
23 }

```

https://blog.csdn.net/qq_41071646

```

IDA Vie... Pseudocod... Stack of sub_400... Pseudocod... Hex Vie... Structu...
1 __BYTE * __fastcall sub_400DB4(__BYTE *a1, int a2)
2 {
3  __BYTE *result; // rax
4  char v3; // [rsp+17h] [rbp-5h]
5  int i; // [rsp+18h] [rbp-4h]
6
7  v3 = *a1 >> 5;
8  for ( i = 0; a2 - 1 > i; ++i )
9     a1[i] = 8 * a1[i] | (a1[i + 1] >> 5);
10
11 // for(int i=0;i<len-1;i++)
12 // {
13 //   a[i] << 3 | a[i+1] >> 5
14 // }
15
16 result = &a1[i];
17 *result = 8 * *result | v3;
18 return result;
19 // a[len-1]*8|a[0]>>5
20 }

```

https://blog.csdn.net/qq_41071646

emmmm 两个加密 我们解密一下就好了

然后这里是我们的C语言实现的代码

```

#include <stdio.h>
#include<iostream>
#include<iomanip>
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<vector>
#include<iostream>
#include<map>
#include<time.h>
#include<queue>
#include <Windows.h>
#include "windows.h"
using namespace std;
map<int,int>mapps;
map<int,char>ma;
int vable[25]={' ', '\\', '_', 'a', 'c', 'b', 'e', 'd', 'g', 'f', 'i', 'h', 'k', 'm', 'l', 'o', 'n', 'p', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'};
int key[25]={71, 40, 245, 104, 15, 30, 169, 29, 38, 19, 60, 67, 20, 28, 39, 165, 118, 26, 51, 61, 45, 133, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255};
int mapp[38]={22, 0, 6, 2, 30, 24, 9, 1, 21, 7, 18, 10, 8, 12, 17, 23, 13, 4, 3, 14, 19, 11, 20, 16, 15, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255};
int l[38]={0xec,0x29,0xe3,0x41,0xe1,0xf7,0xaa,0x1d,0x29,0xed,0x29,0x99,0x39,0xf3,0xb7,0xa9,0xe7,0xac,0x2b,0x2c,0x2d,0x2e,0x2f,0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e,0x7f,0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0xa0,0xa1,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9,0xaa,0xab,0xac,0xad,0xae,0xaf,0xb0,0xb1,0xb2,0xb3,0xb4,0xb5,0xb6,0xb7,0xb8,0xb9,0xba,0xbb,0xbc,0xbd,0xbe,0xbf,0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xca,0xcb,0xcc,0xcd,0xce,0xcf,0xd0,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,0xd7,0xd8,0xd9,0xda,0xdb,0xdc,0xdd,0xde,0xdf,0xe0,0xe1,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,0xe8,0xe9,0xea,0xeb,0xec,0xed,0xee,0xef,0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfd,0xfe,0xff};
int main()
{
    for(int i=0;i<25;i++)
    {
        ma[key[i]]=vable[i];
    }
    for(int i=0;i<38;i++)
    {
        mapps[mapp[i]]=i;
    }
    int s[38];
    int temp=s[37]&0x7;
    for(int i=0;i<38;i++)
    {
        s[i]=(temp<<5)|(l[i]>>3);
        temp=l[i]&0x7;
    }
    int j=37;
    temp=s[j];
    while(mapps[j]!=37)
    {
        s[j]=s[mapps[j]];
        j=mapps[j];
    }
    s[j] = temp;
    for(int i=0;i<38;i++)
    {
        printf("%c",ma[s[i]]);
    }
    return 0;
}

```

然后 官方给的是 python代码 很简洁。

。 <https://www.xctf.org.cn/library/details/8723e039db0164e2f7345a12d2edd2a5e800adf7/>

```
#!/usr/bin/python3
# -*- coding:utf-8 -*-
s = [22, 0, 6, 2, 30, 24, 9, 1, 21, 7, 18, 10, 8, 12, 17, 23, 13, 4, 3, 14, 19, 11, 20, 16, 15, 5, 25, 36,
mapp={' ': 71, '"': 40, '_': 245, 'a': 104, 'c': 15, 'b': 30, 'e': 169, 'd': 29, 'g': 38, 'f': 19, 'i': 60,
def decrypt():
    enc = open("out", "rb").read()
    d0 = []
    temp = ord(enc[len(enc)-1]) & 0x7
    for i in range(len(enc)):
        d0.append((temp << 5) | (ord(enc[i]) >> 3))
        temp = ord(enc[i]) & 0x7

    i = 37
    temp = d0[37]
    while s.index(i) != 37:
        d0[i] = d0[s.index(i)]
        i = s.index(i)
    d0[i] = temp
    flag = []
    for i in d0:
        flag.append(list(mapp.keys())[list(mapp.values()).index(i)])
    return "QCTF{%s}" % ''.join(flag)
print(decrypt())
```