

NCTF 2019 Re Writeup (四题)

原创

Siphre 于 2019-11-25 23:54:17 发布 637 收藏 3

分类专栏: [CTF 逆向 write up](#) 文章标签: [NCTF 逆向 writeup](#)

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_32465127/article/details/103248123

版权



[CTF 同时被 3 个专栏收录](#)

7 篇文章 0 订阅

订阅专栏



[逆向](#)

12 篇文章 0 订阅

订阅专栏



[write up](#)

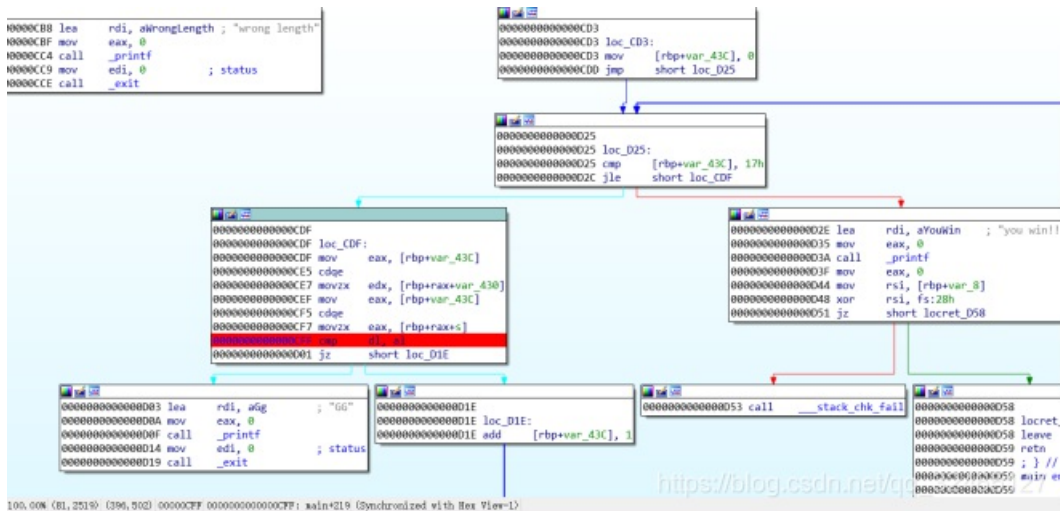
5 篇文章 0 订阅

订阅专栏

reverse

一、DEBUG

ELF动态调试, flag长度24字节, 用户输入的字符串跟经过处理后的字符串s做校验。因为变量s是经过处理的, 得在cmp dl,al下断看eax寄存器的值, 可以看到最终比较的时候s的值。



手抄s的值出来, 再python翻译成字符串即可。

```
flag=[0x4E,0x43,0x54,0x46,0x7b,0x6a,0x75,0x73,0x74,0x5f,0x64,0x65,0x62,0x75,0x67,0x5f,0x69,0x74,0x5f,0x32,0]
for i in range(len(flag)):
    print(chr(flag[i]),end="")
```

Flag:

NCTF{just_debug_it_2333}

二、签到题

手抄参数，丢到在线网站上解线性方程组。

主要是方程组格式太乱懒得写正则和Z3了，反正Z3也得抄参数，体力活。还好万幸的是参数每隔7行是一样的。方程组的常数项在dword_404000，用IDC扒下来，一起丢到求解网站。

IDC脚本：

```
auto from1 = 0x404000;

auto i, x;

Message("\n");

for(i = 0; i < 49*2; i++){

x = Word(from1);

if(x!=0)Message("0x%x,", x);

from1 = from1 + 2;

}
```

python脚本把求解结果（ascii码）转成字符串：

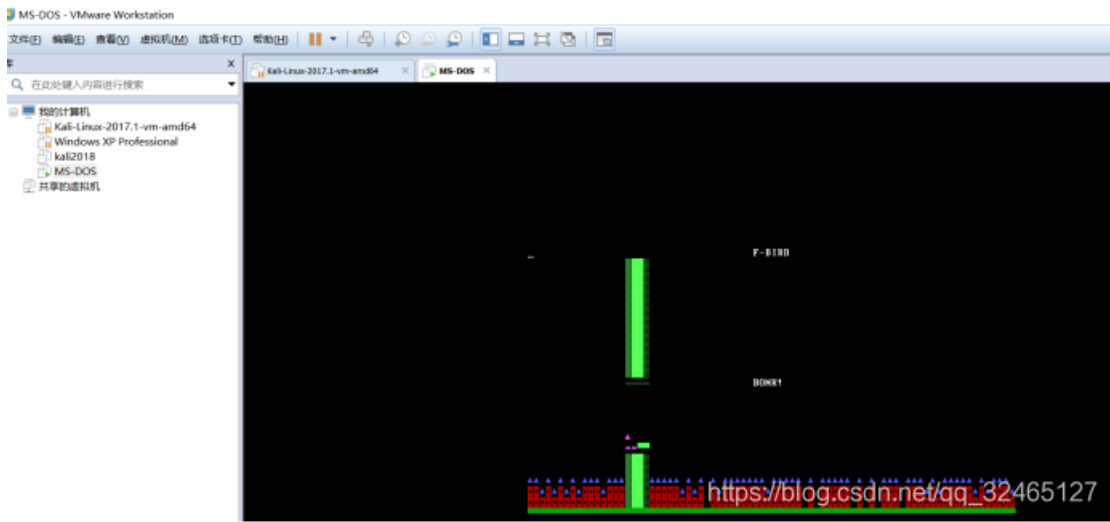
```
flag=[78,67,84,70,123,110,99,116,102,50,48,49,57,95,108,105,110,101,97,114,95,97,108,103,101,98,114,97,95,1
for i in range(len(flag)):
    print(chr(flag[i]),end="")
```

Flag:

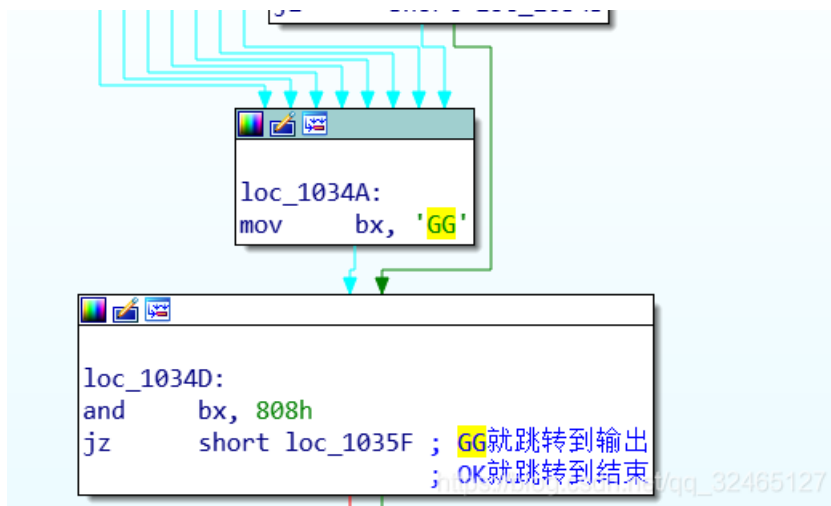
NCTF{nctf2019_linear_algebra_is_very_interesting}

三、Our 16bit Games

MSDOX逆向，装个虚拟机，是F-Bird。



IDA看源码，看不太懂，patch了几处都没啥效果，代码最后，看到用GG、OK字符串异或0x808来控制跳转。



如果传入的字符是GG，就跳转到胜利并打印flag的流程，打印flag时先把栈顶pop给bx寄存器，然后用xchg指令交换bx的高低位，再让高低位间隔着、轮流跟30字节异或，输出异或结果作为flag。

```
for i in range(256):
    if chr(i^0x9d)=='C':
        print(i)
for i in range(256):
    if chr(i^0x8e)=='N':
        print(i)
```

取个巧，用题目给出的格式NCTF{.*}的前两字节爆出高低位分别是222和192，跑脚本即可输出flag。

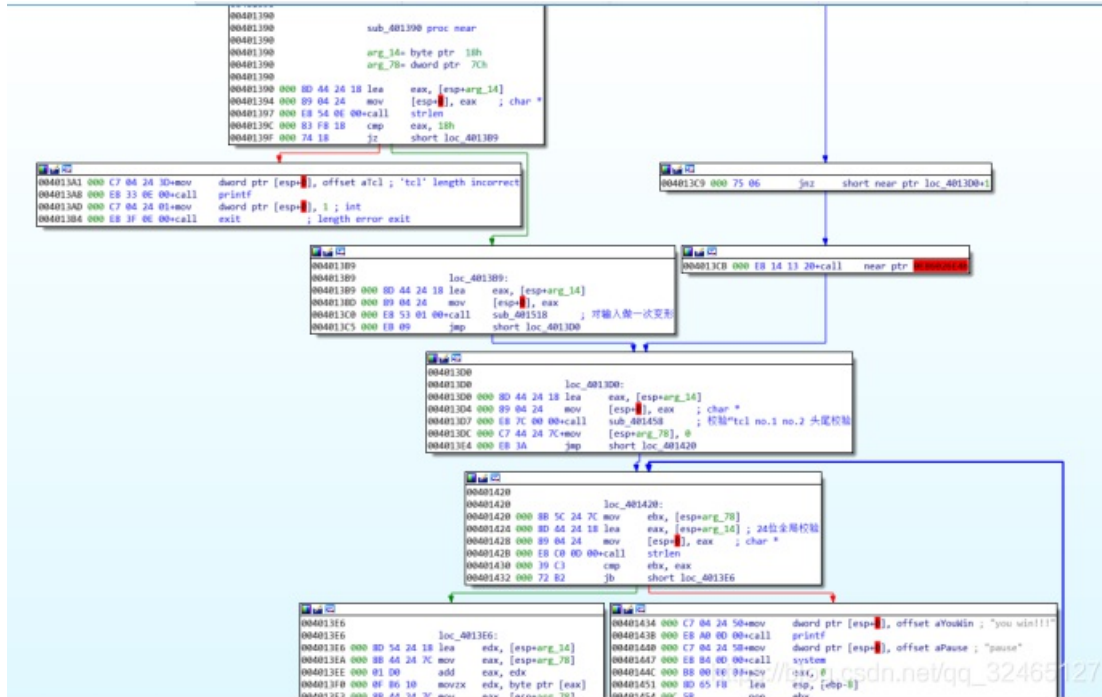
```
a=[0x8e,0x9d,0x94,0x98,0x0bb,0x89,0xf3,0xef,0x83,0xee,0xad,0x9b,0x9f,0xec,0x9f,0x9a,0xf0,0xeb,0x9f,0x97,0xf
for i in range(len(a)):
    if i%2==0:
        xor=192
    else:
        xor=222
    print(chr(a[i]^xor),end="")
```

Flag:

NCTF{W31C0mE_2_D05_l6b17_9am3}

四、难看的代码

主流程:



sub_401390: 校验长度，要求flag长度24字节。

└ sub_401518: 对用户输入做第一次变形。

└ sub_401458: 校验用户输入的头尾，要求格式是NCTF{(18字节)}，对输入做第二次变形。

└ loc_4013FF: 变形结果跟unk_403005比较

第一次变形:

```

00401518
00401518
00401518 ; Attributes: bp-based frame
00401518 sub_401518 proc near
00401518
00401518 var_C= dword ptr -0Ch
00401518
00401518 000 55 push ebp |
00401519 004 89 E5 mov ebp, esp
0040151B 004 53 push ebx
0040151C 008 83 EC 24 sub esp, 24h
0040151F 02C EB 09 jmp short loc_40152A

```

```

0040152A
0040152A loc_40152A: ; int
0040152A 02C C7 44 24 08+mov dword ptr [esp+8], 12h
00401532 02C C7 44 24 04+mov dword ptr [esp+4], 0DEh ; dwSize
0040153A 02C C7 04 24 E1+mov dword ptr [esp], offset loc_4018E1 ; lpAddress
00401541 02C E8 F2 01 00+call sub_401738 ; virtual protect
00401546 02C C7 45 F4 00+mov [ebp+var_C], 0
0040154D 02C E9 84 00 00+jmp loc_4015D6
0040154D 02C 00 sub_401518 endp
0040154D

```

https://blog.csdn.net/qq_32465127

sub_401518: 调用sub_401738对loc_4018E1处若干字节做SMC，进入loc_4015D6。

└ sub_401738: 执行loc_4018E1对loc_40166A处若干字节做SMC。

└ sub_401596: OD看汇编代码，先从程序中部(loc_4015D6)开始执行，再call sub_401596，使得用户输入的1-8字节分别add 0xC 0x22 0x38 0x4E 0xC 0x22 0x38 0x4E，紧接着24字节逐位进行左移、右移、或、异或操作： $byte = ((byte << 3) | (byte >> 5)) \wedge 0x5A$ 。

第二次变形:

sub_401458: 读取并存储unk_403020处24字节的值，执行loc_40166A。

└ loc_40166A: OD看汇编代码，主要操作是取出unk_403020的值，8字节一组，与用户输入每8字节做复杂运算。涉及到常量0x9E3779B9，百度分析得知是TEA加密，unk_403020是key。

00401675	unqly	Disabled	push	edi	edi byte ptr ds:[esi]	将4字节寄存器输入 esp-0x4
0040168E	unqly	Disabled	and	al, 0x00	and al, 0x00	当轮开始前0xC的(值+0E3779B9)
004016C4	unqly	Disabled	sub	byte ptr ds:[edi], bh	sub byte ptr ds:[edi], bh	后四字节<4
004016D5	unqly	Disabled	and	al, 0x39	and al, 0x39	0xC+0E3779B9+后四字节
004016D7	unqly	Disabled	das		das	(0xC+0E3779B9+后四字节)^(12345678+后四字节)<<4
004016E0	unqly	Disabled	sub	byte ptr ds:[edi+0x0C28AC64], bh	sub byte ptr ds:[edi+0x0C28AC64], bh	后四字节>8
004016E6	unqly	Disabled	and	al, 0x80	and al, 0x80	0BADF00D+(后四字节)>8
004016E9	unqly	Disabled	das		das	(0xC+0E3779B9+后四字节)^(12345678+后四字节)<<4)^(0BADF00D+(后四字节)>8)
004016EA	unqly	Disabled	and	al, 0x00	and al, 0x00	前四字节+上面结果
004016F0	unqly	Disabled	sub	byte ptr ds:[edi], bh	sub byte ptr ds:[edi], bh	(前四字节+上面结果)<<4
00401701	unqly	Disabled	and	al, 0x39	and al, 0x39	(每轮开始前0xC的(值+0E3779B9)+(前四字节+上面结果)
00401703	unqly	Disabled	das		das	((前四字节+上面结果)<<4)+5201314)+((每轮开始前0xC的(值+0E3779B9)+(前四字节+上面结果))
0040170A	unqly	Disabled	sub	byte ptr ds:[edi+0x0C28AC64], bh	sub byte ptr ds:[edi+0x0C28AC64], bh	(前四字节+上面结果)>8
00401712	unqly	Disabled	and	al, 0x80	and al, 0x80	07054321+(前四字节+上面结果)>8
00401714	unqly	Disabled	das		das	(07054321+(前四字节+上面结果)>8)^[((前四字节+上面结果)<<4)+5201314)+((每轮开始前0xC的(值+0E3779B9)+(前四字节+上面结果))]

计算flag思路:

程序数据处理流程是——原始输入-->第一次变形-->TEA加密-->与unk_403005做校验。所以思路就是，先用TEA算法解密unk_403005，得到第一次变形后的字节数组，再用第一次变形的算法爆破出合法的用户原始输入。下面开始具体操作:

首先，上网扒一段TEA解密代码，分析程序运算细节，把key和v对号入座，输入是每4字节逆序，输出也是每4字节逆序。

```
key=[0x12345678, 0x0BADF00D,0x5201314,0x87654321]
```

```
v1=[0x61869f5e,0x0a9cf08d]
```

```
v2=[0xad74c0ca,0xa57f16b8]
```

```
v3=[0xb559626d,0xd17b68e0]
```

TEA解密 python代码:

```

a=[0x5e,0x9f,0x86,0x61,0x8d,0xf0,0x9c,0xa,#0x61869f5e,0x0a9cf08d
    0xca,0xc0,0x74,0xad,0xb8,0x16,0x7f,0xa5,#0xad74c0ca,0xa57f16b8
    0x6d,0x62,0x59,0xb5,0xe0,0x68,0x7b,0xd1]#0xb559626d,0xd17b68e0
from ctypes import *

def encipher(v, k):
    y = c_uint32(v[0])
    z = c_uint32(v[1])
    sum = c_uint32(0)
    delta = 0x9e3779b9
    n = 32
    w = [0,0]

    while(n>0):
        sum.value += delta
        y.value += ( z.value << 4 ) + k[0] ^ z.value + sum.value ^ ( z.value >> 5 ) + k[1]
        z.value += ( y.value << 4 ) + k[2] ^ y.value + sum.value ^ ( y.value >> 5 ) + k[3]
        n -= 1

    w[0] = y.value
    w[1] = z.value
    return w

def decipher(v, k):
    y = c_uint32(v[0])
    z = c_uint32(v[1])
    sum = c_uint32(0xc6ef3720)
    delta = 0x9e3779b9
    n = 32
    w = [0,0]

    while(n>0):
        z.value -= ( y.value << 4 ) + k[2] ^ y.value + sum.value ^ ( y.value >> 5 ) + k[3]
        y.value -= ( z.value << 4 ) + k[0] ^ z.value + sum.value ^ ( z.value >> 5 ) + k[1]
        sum.value -= delta
        n -= 1

    w[0] = y.value
    w[1] = z.value
    return w

if __name__ == "__main__":
    key = [0x12345678, 0x0BADF00D,0x5201314,0x87654321]
    cipher=[0x61869f5e,0x0a9cf08d]
    dec=decipher(cipher, key)
    for i in range(len(dec)):
        print(hex(dec[i]),end=",")

```

接着，使用脚本解密unk_403005的数据，手工做逆序恢复处理，得到flag第一次变形后的字节数组。

```

a=
[0x88,0x71,0x3e,0xfe,0x66,0xf6,0x77,0xd7,0xa0,0x51,0x29,0xf9,0x11,0x79,0x71,0x49,0xf1,0x61,0xa0,0x9,0xf1,C

```

最后，根据第一次变形的算法写python脚本爆破：

```
a=[0x88,0x71,0x3e,0xfe,0x66,0xf6,0x77,0xd7,0xa0,0x51,0x29,0xf9,0x11,0x79,0x71,0x49,0xf1,0x61,0xa0,0x9,0xf1,
index=0
while(index<24):
    for i in range(32,128):
        byte=i
        if (index==0)|(index==4):
            byte+=0xC
        if (index==1)|(index==5):
            byte+=0x22
        if (index==2)|(index==6):
            byte+=0x38
        if (index==3)|(index==7):
            byte+=0x4e
        byte = ((byte << 3)&0xFF | (byte >> 5)) ^ 0x5A

        if byte==a[index]:
            print(chr(i),end="")
            index+=1
            i=32
            break
```

Flag:

NCTF{smc_antidebug_junk}

补充说明：

分析过程中若干jz+jnz垃圾指令，手工D键、C键去除。

过程中若干反调试，用吾爱的某版本强力OD，OD自动绕过了。

•