2019 KCTF 晋级赛Q1 | 第七题点评及解题思路

小雪 看雪学院 1周前

圆圈舞

圆圈舞是祖传的用毕颂（直箫）伴奏，不分季节跳的集体舞蹈，舞蹈吹奏者在中间连吹带跳，其他舞者也在外围手拉手地呼喊随着音乐节奏起舞。

出题者就如同吹奏者，希望参赛者能够跟随着出题者的意图，在算法题+轻度ANTI中玩的开心。

攻破此题的战队

排名	战队名	破解时间	获取积分
	pizzatql	134694s	100
	金左手	214862s	100
	萌新队	285187s	100
4.	tekkens	440736s	100
5.	雨落星沉	572368s	100
6.	AceHub	783380s	100
7.	fade-vivi	795589s	100
8.	A2	1047735s	100

本道题目只有**8**支队伍解答出来，战队“**pizzatpl**”用时1天半左右的时间才破解成功，这道题目观战人数达到**1900**人。

出题战队

战队成员：lelfei

个人主页：https://bbs.pediy.com/user-30.htm

个人简介：lelfei，业余的crack爱好者。学生时代对电脑产生了浓厚的兴趣，经历了很长时间的游戏沉迷后，开始慢慢转向学习技术，工作后自学了ASM、VB、VC、HTML、ASP、Python等语言的入门。工作原因上网较少，对单机的逆向分析、算法比较感兴趣，但是由于缺少系统的学习，水平处于“入行较早层次较低知识较杂”的阶段。

看雪CTF crownless 评委 点评

这道题主要考察解数独的“舞蹈链”算法以及花指令、自校验等软件保护技术，总体来说难度中等偏上，这导致只有八支队伍成功破解此题。

题目设计思路

说明：

简单说一下程序流程，初始化一个数独游戏，使用DancingLinks算法计算出答案，与用户输入值比较，正确后输出“Well Done!”。

具体实现比较绕，希望这种隐藏思路的方式能带给大家一点惊喜。

第一步：对程序二处位置进行数据校验，生成大数Num1。

第二步：用Num1与一固定大数Num2相乘，得到数独初始化数据Data1。

第三步：对Data1进行解析并填入DancingLinks初始化数据DLX，同时用户输入16进制数据转换成10进制数后按位置填入一个九宫格UserData1中，一共80位。数独数据最中间位没有直接提供，根据9个CheckDebug()调试检查函数返回值计算得出，必须为7。

第四步：DLX开始Dancing，同时在DLX的cover()函数中对用户数据UserData1进行转置。转置方法为，把数据交换操作拆分为5个小步骤，每一次cover()执行一个小步骤，执行60次交换后，九宫格数据向右旋转90度。一共执行4次旋转，最后用户数据转了一圈Circle后回到原位置，只是每一位进行了6次数据变换操作，生成用户数据UserData2。一共需要执行5*60*4=1200次操作，也就是说DLX.cover()操作要多于1200次。

第五步：对UserData2查表与DLX运算结果按位比较，得到完全匹配的位数总和cnt，必须为79（比较到中间位时cnt--）。

第六步：用cnt对一个数据进行解码并计算校验值，当校验值正确时输出解码数据，正确时输出“Well Done!”。

最后修改程序，添加花指令和自校验，方法为：

1. 在代码中加了一些标志数据，用python把这些标志数据替换成花指令。

2. 在代码中加了一个大段数据的空函数，在python中把自校验代码添加花指令后填入这个空函数。

3. 在代码中3处调用空函数的位置添加标志位，通过调试得到代码自校验数据，然后在python中把自校验数据写到函数调用的标志位。

程序中使用的大数计算代码是前几年参加CTF时自己写的，源码中提供；花指令库来自于OD去花指令插件DeJunk的配置文件；加花指令和自校验代码是自己写的，详见byteenc.py；DancingLinks算法源码来自于网上，基本没做修改。

程序用CodeBlocks开发，gcc编译。程序在Win7 64位下运行正常，其他系统未做测试。

破解思路：

1. 首先要分析或穷举出对输出字符进行解码的校验值，得知用户数据与DLX结果必须完全匹配。

2. 分析出在DLX.cover()函数中，每5次调用为一次交换并移位操作。

3. 进一步分析出1200次交换移位最终只是转了一圈回到原位，每一位进行了6次移位操作。

4. 穷举移位操作，还原用户数据，得到答案。

5. 按惯例留了一个后门：如果能直接识别出核心算法为DancingLinks，结合数独初始化数据，能直接得到答案。

需要穷举的操作都是按字节进行查表，对字节穷举的时间基本可以忽略不计。

参考答案：

CBC25EF8D9F482BC1F3DA3CA1F154EC89FC3F1414EDD93A3

src.rar：程序源码

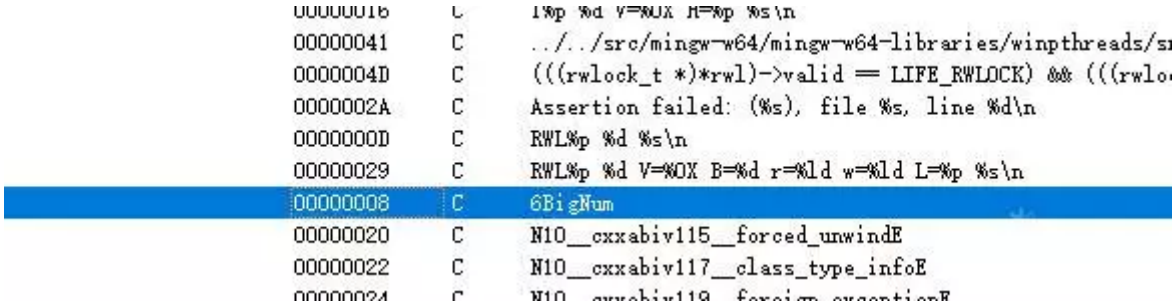
DancingCircle.rar：参赛程序

解题思路

本题解题思路由看雪论坛 **新手慢慢来** 提供

观察

题目描述为“算法题+轻度ANTI”，说明程序中有反调试。将程序用ida打开，在字符串中发现了“6BigNum”，说明程序中可能有大数结构和运算。



程序分析

找到main函数，发现程序中有很多花指令，并且.text:004B8E55 call sub_401F58 后的数据无法转换为代码，猜测程序中可能有SMC，动态调试程序，看执行完 call sub_401F58 后程序会在哪执行，单步跟踪函数sub_401F58，发现函数sub_401F58在执行到 .text:00402129 retn 后会跳转到004B8E5F处执行，且004B8E5A处的一大段数据并没有被修改，说明只是作者在这里插入了一些数据来扰乱ida的分析。

```
.text:004B8E2E                mov     [ebp-2B84h], esp
.text:004B8E34                mov     [esp], eax
.text:004B8E37                call    sub_40D9D0
.text:004B8E3C                call    sub_40C510
.text:004B8E41                jp      short loc_4B8E46
.text:004B8E43                jnp     short loc_4B8E46
.text:004B8E45                dec     esp
.text:004B8E46
.text:004B8E46 loc_4B8E46:                            ; CODE XREF: .text:004B8E41↑j
.text:004B8E46                                        ; .text:004B8E43↑j
.text:004B8E46                mov     dword ptr [esp+4], 25Ah
.text:004B8E4E                mov     dword ptr [esp], 0
.text:004B8E55                call    sub_401F58
.text:004B8E55 ; ---------------------------------------------------------------------------
.text:004B8E5A                dw 55C6h
.text:004B8E5C                dd 0E800F35Fh, 2, 0C4830CA2h, 2404C704h, 1000h, 0D45885C7h
.text:004B8E5C                dd 0FFFFFFFFh, 1DE8FFFFh, 8DFFF49Bh, 0FFD7C88Dh, 8D3CE8FFh
.text:004B8E5C                dd 8D8DFFF4h, 0FFFFEBD8h, 0D45885C7h, 1FFFFh, 27E80000h
.text:004B8E5C                dd 7AFFF48Dh, 6C017B03h, 0D4C8858Dh, 4C7FFFFh, 4BD7C024h
```

将004B8E5F处的数据转化为代码，并将上面的垃圾数据nop掉

```
.text:004B8E41                jp      short loc_4B8E46
.text:004B8E43                jnp     short loc_4B8E46
.text:004B8E45                dec     esp
.text:004B8E46
.text:004B8E46 loc_4B8E46:                            ; CODE XREF: .text:004B8E41↑j
.text:004B8E46                                        ; .text:004B8E43↑j
.text:004B8E46                mov     dword ptr [esp+4], 25Ah
.text:004B8E4E                mov     dword ptr [esp], 0
.text:004B8E55                call    sub_401F58
.text:004B8E5A                nop
.text:004B8E5B                nop
.text:004B8E5C                nop
.text:004B8E5D                nop
.text:004B8E5E                nop
.text:004B8E5F                call    loc_4B8E66
.text:004B8E5F ; ---------------------------------------------------------------------------
.text:004B8E64                db 0A2h
.text:004B8E65                db  0Ch
.text:004B8E66 ; ---------------------------------------------------------------------------
.text:004B8E66
```

去除花指令

通过观察程序中代码，发现花指令主要有以下4种模式：

```
clc
jnb

stc
jb

call
add esp, 4
```

```
j
j
```

可以通过如下的idapython脚本去除花指令，有部分花指令没有识别到，任然需要自己手动nop掉。

```
bg = 0x00401000
end = 0x004BBE00
addr = bg

def patch_nop(begin,end):
while(end>begin):
PatchByte(begin,0x90)
begin=begin+1
def next_instr(addr):
return addr+ItemSize(addr)
while(addr<end):
next =next_instr(addr)
MakeCode(next)
if 'j' in GetMnem(addr) and 'j' in GetMnem(next) :
if GetOperandValue(addr,0) == GetOperandValue(next,0):
print 'jmp %08x'%addr
dest_addr = GetOperandValue(addr,0)
patch_nop(addr,dest_addr)
addr=dest_addr
MakeCode(addr)
if 'clc' == GetMnem(addr) and 'jnb' in GetMnem(next) :
print 'clc %08x'%addr
dest_addr = GetOperandValue(next,0)
patch_nop(addr,dest_addr)
addr=dest_addr
MakeCode(addr)
if 'stc' == GetMnem(addr) and 'jb' in GetMnem(next) :
print 'clc %08x'%addr
dest_addr = GetOperandValue(next,0)
patch_nop(addr,dest_addr)
addr=dest_addr
MakeCode(addr)
if 'call' in GetMnem(addr):
dest_addr = GetOperandValue(addr,0)
idc.del_items(next_instr(addr))
MakeCode(dest_addr)
if "add esp, 4" == GetDisasm(dest_addr):
print 'call %08x'%addr
```

```
dest_addr=next_instr(dest_addr)
patch_nop(addr,dest_addr)
addr=dest_addr
MakeCode(addr)
addr = next_instr(addr)
MakeCode(addr)
```

在.text:004B8DF3处创建函数并修改函数结尾为.text:004B9827，F5后得到该函数的代码如下：

```
int __usercall sub_4B8DF3@<eax>(char a1@<cl>, int a2@<ebx>, signed int a3@<edi>)
{
void *v3; // esp
int v4; // ecx
int v5; // eax
int v6; // edx
int v7; // eax
char *v8; // ebx
char *v9; // esi
char *v10; // edx
char v11; // ah
char v12; // dl
int v13; // eax
int v14; // eax
int v15; // edx
int v16; // ebx
int v17; // eax
int v18; // ebx
int v19; // eax
int v20; // edx
int v21; // edx
int v22; // ecx
signed int v23; // eax
int (__cdecl *v24)(char *); // eax
int v25; // ebx
__int64 v26; // rax
__int64 v27; // rax
char v28; // al
int v29; // edx
int v30; // eax
int v31; // ST20_4
unsigned __int8 *v32; // ST1C_4
signed int v33; // eax
_BYTE *v34; // ecx
```

```
int v35; // eax
int v36; // ebx
int v37; // edx
int v38; // eax
int v39; // edx
int v41; // [esp+4h] [ebp-2BDCh]
int v42; // [esp+8h] [ebp-2BD8h]
int v43; // [esp+Ch] [ebp-2BD4h]
int v44; // [esp+10h] [ebp-2BD0h]
int v45; // [esp+14h] [ebp-2BCCh]
int v46; // [esp+18h] [ebp-2BC8h]
signed int v47; // [esp+18h] [ebp-2BC8h]
int v48; // [esp+1Ch] [ebp-2BC4h]
int i; // [esp+20h] [ebp-2BC0h]
int v50; // [esp+24h] [ebp-2BBCh]
int v51; // [esp+24h] [ebp-2BBCh]
int v52; // [esp+24h] [ebp-2BBCh]
int v53; // [esp+28h] [ebp-2BB8h]
char *v54; // [esp+28h] [ebp-2BB8h]
int v55; // [esp+28h] [ebp-2BB8h]
int v56; // [esp+2Ch] [ebp-2BB4h]
int v57; // [esp+2Ch] [ebp-2BB4h]
signed int v58; // [esp+2Ch] [ebp-2BB4h]
int v59; // [esp+2Ch] [ebp-2BB4h]
signed int length; // [esp+30h] [ebp-2BB0h]
char v61; // [esp+34h] [ebp-2BACh]
int v62; // [esp+38h] [ebp-2BA8h]
int (__cdecl *v63)(int, int, int, int, int, int); // [esp+4Ch] [ebp-2B94h]
int *v64; // [esp+50h] [ebp-2B90h]
char *v65; // [esp+54h] [ebp-2B8Ch]
int (*v66)(); // [esp+58h] [ebp-2B88h]
int *v67; // [esp+5Ch] [ebp-2B84h]
int v68; // [esp+68h] [ebp-2B78h]
int v69; // [esp+6Ch] [ebp-2B74h]
int v70; // [esp+70h] [ebp-2B70h]
int v71; // [esp+74h] [ebp-2B6Ch]
int v72; // [esp+80h] [ebp-2B60h]
int v73; // [esp+84h] [ebp-2B5Ch]
int v74; // [esp+88h] [ebp-2B58h]
int v75; // [esp+8Ch] [ebp-2B54h]
int v76; // [esp+90h] [ebp-2B50h]
int v77; // [esp+94h] [ebp-2B4Ch]
char input[256]; // [esp+A8h] [ebp-2B38h]
char v79[256]; // [esp+1A8h] [ebp-2A38h]
char keydata[256]; // [esp+2A8h] [ebp-2938h]
```

```
char v81; // [esp+3A8h] [ebp-2838h]
char v82; // [esp+17B8h] [ebp-1428h]
char v83; // [esp+2BC8h] [ebp-18h]
char *v84; // [esp+2BD0h] [ebp-10h]
int v85; // [esp+2BD4h] [ebp-Ch]

v3 = alloca(sub_40D480(a1));
v63 = sub_4B85B0;
v64 = dword_4B9D1C;
v66 = sub_4B9827;
v65 = &v83;
v67 = (int *)&v84;
sub_40D9D0(&v61);
sub_40C510();
sub_401F58(0, 0x25A);
sub_40299C(0x1000);
sub_401BC6(&v81);
sub_401BC6(&v82);
v62 = 2;
getinput(&dword_4BD7C0, input);
for ( length = 0; ; ++length )
{
v5 = length;
if ( !input[length] )
break;
}
v6 = 0;
LOBYTE(a2) = '7';
while ( v6 != length )
{
LOBYTE(v5) = input[v6];
LOBYTE(v4) = '0';
if ( (char)v5 > '9' )
{
LOBYTE(v4) = '=';
if ( (char)v5 < 'a' )
v4 = a2;
}
v7 = v5 - v4;
v79[v6] = v7;
v5 = v7 - 1;
if ( (unsigned __int8)v5 > 0xEu ) // 输入为16进制数据，且不能为0，即1-F
goto LABEL_61;
++v6;
}
```

```
v8 = keydata;
v9 = v79;
v79[length] = 0;
while ( length )
{
v10 = &v79[length];
do
{
v11 = (unsigned __int8)*v10 % 10u;
*v10 = (unsigned __int8)*v10 / 10u;
*(v10-- - 1) += 16 * v11;
}
while ( v79 != v10 );
v12 = v79[0] % 10u; // 16进制转10进制，从右往左，依次为高位到低位
v79[0] /= 10u;
*v8 = v12;
do
{
a3 = length;
v13 = length - 1;
if ( v79[length - 1] )
break;
--length;
}
while ( v13 );
++v8;
}
v62 = 2;
init_bignum(&unk_4BC080, 16); // 0xc23f6401c93adb
v85 = init_bignum(&unk_4BC088, 16); // 0xeedcea3743d03a263af94f386de1
sub_401E4C(&v82);
sub_401684(&v81, a3); // 10转16进制
v68 = 0;
v69 = 0;
v70 = 0;
v44 = 0;
v48 = 0;
v45 = 0;
v53 = 0;
while ( 1 )
{
v62 = 3;
if ( v45 >= sub_4016AE(&v81) )
break;
v50 = get_index_data(&v81, v45);
```

```
v73 = 0;
v14 = sub_4B77F0(1296);
v15 = v14 + 1296;
v16 = v14;
v72 = v14;
v17 = 0;
v74 = v15;
do
{
*(_DWORD *)(v16 + v17) = 0;
v17 += 4;
}
while ( v17 != 1296 );
v73 = v15;
if ( v50 )
{
if ( v50 <= 9 )
{
v27 = v53;
*(_DWORD *)(v16 + 4 * v27) = 1;
*(_DWORD *)(v16 + 4 * (v50 + 9 * (unsigned __int64)(v27 / 9) + 80)) = 1;
a3 = 3;
*(_DWORD *)(v16 + 4 * (v50 + 9 * (v53 % 9) + 161)) = 1;
v9 = (char *)v50;
*(_DWORD *)(v16 + 4 * (v50 + 9 * (3 * (v53 / 27) + v53 % 9 / 3) + 242)) = 1;
v62 = 4;
sub_49CF5C(&v68, (int)&v72);
v46 = v48 + 1;
v85 = set_index_data(&unk_4C8020, v48, v50);
v84 = (char *)v85;
++v53;
}
else
{
for ( i = 0; ; ++i )
{
v46 = i + v48;
v56 = i + v53;
if ( i >= v50 - 9 )
break;
v85 = (unsigned __int8)keydata[v44 + i];
v62 = 4;
set_index_data(&unk_4C8020, v46, v85);
v85 = i;
v84 = (char *)i;
```

```
v43 = 4 * v56;
v18 = 9 * (v56 / 9);
v47 = 36 * (v56 / 9) + 324;
v41 = 36 * (v56 / 9) + 360;
v42 = 4 * (9 * (v56 % 9) - v18);
v57 = 4 * (9 * (v56 % 9 / 3 + 3 * (v56 / 27)) - v18);
do
{
v76 = 0;
v62 = 4;
v19 = sub_4B77F0(1296);
v75 = v19;
v20 = 0;
v77 = v19 + 1296;
do
{
*(_DWORD *)(v19 + v20) = 0;
v20 += 4;
}
while ( v20 != 1296 );
a3 = v47;
v76 = v19 + 1296;
*(_DWORD *)(v19 + v43) = 1;
*(_DWORD *)(v19 + v47) = 1;
*(_DWORD *)(v47 + v19 + v42 + 324) = 1;
*(_DWORD *)(v47 + v57 + v19 + 648) = 1;
v62 = 5;
sub_49CF5C(&v68, (int)&v75);
sub_402950(&v75, v21, v22);
v47 += 4;
v9 = (char *)v47;
}
while ( v41 != v47 );
}
v62 = 4;
v23 = get_index_data(&v81, v45 + 1);
v84 = (char *)v15;
if ( v23 <= 9 )
{
v44 += i;
v53 += i;
}
else
{
v54 = 0;
```

```
v51 = 0;
do
{
v24 = (int (__cdecl *)(char *))dword_4BC020[v51];
v62 = 4;
v54 += (unsigned int)v24(v84) < 1;
++v51;
}
while ( v51 != 9 );
v25 = v72;
v26 = v56;
*(_DWORD *)(v72 + 4 * v26) = 1;
*(_DWORD *)(v25 + 4 * (_DWORD)&v54[9 * (unsigned __int64)(v26 / 9) + 80]) = 1;
a3 = 3;
*(_DWORD *)(v25 + 4 * (_DWORD)&v54[9 * (v56 % 9) + 161]) = 1;
v9 = v54;
*(_DWORD *)(v25 + 4 * (_DWORD)&v54[9 * (3 * (v56 / 27) + v56 % 9 / 3) + 242]) = 1;
v62 = 4;
v84 = (char *)sub_49CF5C(&v68, (int)&v72);
v53 = v56 + 1;
v44 += i;
}
}
}
else
{
v46 = v48;
}
sub_402950(&v72, v15, v84);
++v45;
v48 = v46;
}
sub_402BEE(&v68, -1431655765 * ((v69 - v68) >> 2), 324);
v62 = 6;
v28 = sub_402DB6(0);
v84 = (char *)a3;
if ( v28 )
{
v62 = 6;
sub_49D054(&v76);
v84 = v9;
v55 = 0;
v52 = 0;
do
{
```

```
v84 = (char *)(v68 + 12 * (*(_DWORD *)(v71 + v52) - 1));
v62 = 8;
sub_49D054(v84);
v84 = (char *)v52;
v29 = v72;
v58 = 0;
do
{
if ( *(_DWORD *)(v72 + 4 * v58) == 1 )
break;
++v58;
}
while ( v58 != 81 );
if ( v58 == 40 )
{
--v55;
}
else
{
v59 = v58 - (v58 >= 41);
v30 = 0;
do
{
if ( *(_DWORD *)(v72 + 4 * v30 + 324) == 1 )
break;
++v30;
}
while ( v30 != 81 );
v31 = v30 % 9 + 1;
v62 = 7;
get_index_data(&unk_4C8020, v59);
v32 = (unsigned __int8 *)off_4BC060;
v33 = get_index_data(&unk_4C8020, v59);
v84 = (char *)v29;
v55 += v31 == v32[v33];
}
sub_402950(&v72, v29, v84);
v52 += 4;
}
while ( v52 != 324 );
v34 = off_4BC05C;
v35 = 0;
do
{
v36 = (unsigned __int8)v34[v35];
```

```
length += 9 * (v36 ^ v55) ^ 0x37;
v37 = v36 ^ v55;
v34[v35] = v36 ^ v55;
if ( v55 == v36 )
break;
++v35;
}
while ( v35 != 513 );
if ( length == 0x1F1A )
{
v62 = 8;
v38 = sub_4B3F00(&dword_4BD9A0, (char *)off_4BC05C);
sub_4B0DB0(v38);
}
sub_402950(&v71, v37, v84);
}
j_free(v75);
sub_402950(&v76, v39, v84);
sub_402966(&v68);
LABEL_61:
sub_401570(&v82);
sub_401570(&v81);
sub_40DA40(&v61);
return 0;
}
```

函数sub_401F58的花指令要更复杂一些，去除花指令后执行的汇编代码如下：

```
.text:00401F58 push ebp
.text:00401F59 mov ebp, esp
.text:00401F5B pusha
.text:00401F5C pushf
.text:00401F5D mov edx, offset unk_4BC080
.text:00401F76 mov ebx, [esp+28h]
.text:00401F7F mov esi, [esp+2Ch]
.text:00401F91 mov ecx, [esp+30h]
.text:00401FAE test esi, esi
.text:00401FB0 jnz short loc_401FBB
.text:00401FB9 mov esi, ebx
.text:00401FC0 add esi, 5
.text:00401FCE xor eax, eax
.text:00401FDF xor eax, [esi]
.text:00401FF0 add esi, 4
.text:00401FFD xor eax, 78563412h
```

```
.text:0040201D ror eax, 9
.text:0040202E dec ecx
.text:0040202F jnz short loc_401FD0
.text:0040203B xor eax, [ebx]
.text:00402049 add ebx, 4
.text:00402060 xor ecx, ecx
.text:00402070 mov cl, [ebx]
.text:0040207D inc ebx
.text:00402097 lea edx, [ecx+edx]
.text:004020AB mov [edx], eax
.text:004020BE popf
.text:004020CB popa
.text:004020E1 pop ebp
.text:00402115 add dword ptr [esp], 5
.text:00402129 retn
```

伪C代码如下：

```
int __cdecl sub_401F58(_DWORD *a1, int a2)
{
_DWORD *v2; // esi
int v3; // ecx
_DWORD *v4; // esi
int v5; // eax
int v6; // eax
int v7; // ecx
_DWORD *retaddr; // [esp+24h] [ebp+4h]

v2 = a1;
v3 = a2;
if ( !a1 )
v2 = retaddr;
v4 = (_DWORD *)((char *)v2 + 5);
v5 = 0;
do
{
v6 = *v4 ^ v5;
++v4;
v5 = __ROR4__(v6 ^ 0x78563412, 9);
--v3;
}
while ( v3 );
v7 = *((unsigned __int8 *)retaddr + 4);
```

```
*(_DWORD *)((char *)&unk_4BC080 + v7) = *retaddr ^ v5;
retaddr = (_DWORD *)((char *)retaddr + 5);
```

该函数一共被调用了三次，函数有三个参数，第一个参数是返回地址，后面两个参数是依次传入的，例如第一次调用时参数为（0x4b8e5a,0,0x25a），第二次调用的参数为（0x4029C6,0x4025D3,0xF0），第三次调用的参数为（0x4025F2,0x401F58,0x68）。

该函数的作用是进行代码检验，防止patch代码和下int 3断点，然后跳到返回地址加5处执行。

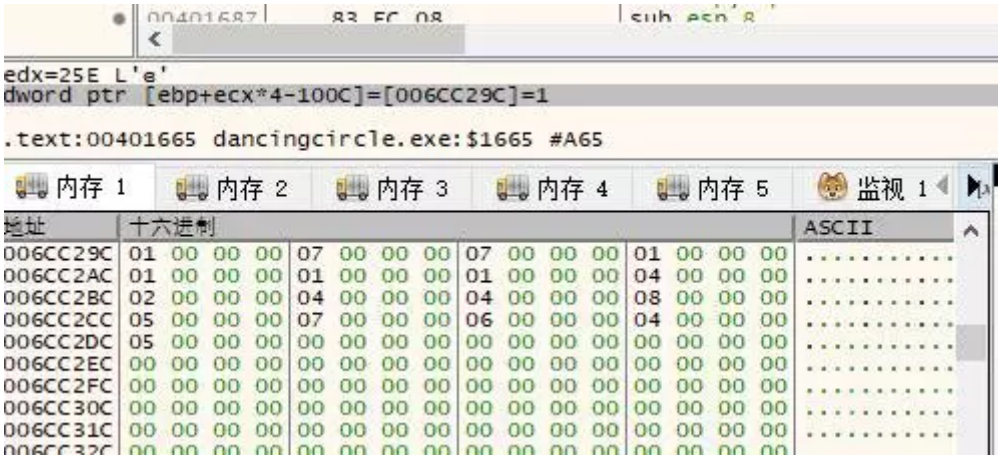
例如第一次执行时就是检验0x4b8e5f到0x4b97c7处的代码，然后将结果填入到（0x4BC080+0）处，第二次检验0x4025D8到0x402998处的代码，然后将结果填入到（0x4BC080+4）处，第三次检验0x401F5d到0x4020fd处的代码，然后将结果填入到（0x4BC080+0x9d）处。

通过动态调试得知函数4B5240是获取用户输入的，输入只能为16进制数且只能为1-F，然后会转化为10进制数据，转化规则如下：

例如输入为123AF，表示数据为0xfa321，转化为10进制为1024801，在程序里表示为1084201（从右到左依次为高位到低位）。

```
}
v66 = 2;
init_bignum(&v85, &unk_4BC080, 16);          // 0xc23f6401c93adb
v89 = init_bignum(&v86, &unk_4BC088, 16);    // 0xeedcea3743d03a263af94f386de1
mul(&v85, (int)&v86);
v88 = (char *)v6;
sub_401684(&v85);                            // 10转16进制
v72 = 0;
v73 = 0;
```

函数004017A0是初始化大数的，大数的数据可以在.text:00401665 1014 mov edx, [ebp+ecx4+var_100C]处下断点，然后查看ebp+ecx4+var_100C处的数据，就是大数的值，即0xc23f6401c93adb，即54675844241111771。



注意004BC080处的数据是根据代码校验后的值填入的，可以附加程序，得到此处真正的值。然后计算了两个大数的乘积，即
0xc23f6401c93adb*0xeedcea3743d03a263af94f386de1=0xb53e8f2b7b1a5b4d53c1bc6b32c8b6a5d9b4d3f97b

函数4016B6和4018DE十分重要，分别是根据下标取大数的数据和根据下标写大数中写入数据。

通过在.text:004B9533 call set_index_data处和.text:004B91BE call set_index_data处下断点，可以得到输入的数据和乘积的数据是怎么插入大数中的。

[0x2,0x3,0xA,0xF,0x12,0x17,0x19,0x1c,0x20,0x21,0x24,0x2a,0x2e,0x2f,0x34,0x37,0x39,0x3c,0x3f,0x46,0x4c,0x4d]是乘积依次插入大数的下标

[0x0,0x1,0x4,0x5,0x6,0x7,0x8,0x9,0xb,0xc,0xd,0xe,0x10,0x11,0x13,0x14,0x15,0x16,0x18,0x1a,0x1b,0x1d,0x1e,0x1f,0x22,0x23,0x25,0x26,0x27,0x28,0x29,0x2b,0x2c,0x2d,0x30,0x31,0x32,0x33,0x35,0x36,0x38,0x3a,0x3b,0x3d,0x3e,0x40,0x41,0x42,0x43,0x44,0x45,0x47,0x48,0x49,0x4a,0x4b,0x4e,0x4f]是输入的数据依次插入大数的下标

查看set_index_data函数和get_index_data函数的调用可以得到变换的算法就是查找表，根据表中的值作为大数的下标，取大数数据后，乘9后异或0x37，交换大数中的数据。

```
    }
    while ( v56 != 324 );
    v38 = off_4BC05C;
    v39 = 0;
    do
    {
      v40 = (unsigned __int8)v38[v39];
      length += 9 * (v40 ^ v59) ^ 0x37;
      v41 = v40 ^ v59;
      v38[v39] = v40 ^ v59;
      if ( v59 == v40 )
        break;
      ++v39;
    }
    while ( v39 != 513 );
    if ( length == 0x1F1A )
    {
      v66 = 8;
      v42 = sub_4B3F00(&dword_4BD9A0, (char *)off_4BC05C);
      sub_4B0DB0(v42);
    }
    sub_402950(&v75, v41, v88);
  }
000B8B7D main:351 (4B977D)
```

根据length为0x1f1a，可以推导出v59为79。

然后在.text:004B975C call sub_402950和.text:004B9768 cmp [ebp+var_2BBC]，144h处下断点，查看比较的下标跟数据。

得到依次比较的下标为
r1=
[0x2,0x3,0xA,0xF,0x12,0x17,0x19,0x1c,0x20,0x21,0x24,0x2a,0x2e,0x2f,0x34,0x37,0x39,0x3c,0x3f,0x46,0x4c,0x4d]

和r2=
[0x27,0x42,0x13,0x22,0x25,0x1,0xb,0x2c,0x31,0xc,0x5,0xd,0xe,0x29,0x33,0x10,0x9,0x0,0x11,0x14,0x2b,0x2d,0x1b,0x30,0x32,0x23,0x1a,0x7,0x4,0x6,0x8,0x18,0x3d,0x44,0x3b,0x4e,0x45,0x3e,0x47,0x35,0x48,0x36,0x38,0x3a,0x28,0x1e,0x1f,0x1d,0x26,0x41,0x15,0x16,0x43,0x40,0x49,0x4a,0x4b,0x4f]

比较的值依次为
result1=[7,9,3,4,9,5,6,8,2,3,6,1,3,5,4,5,1,7,2,8,3,5]
和result2=
[0x3,0x5,0x1,0x5,0x5,0x6,0x2,0x2,0x1,0x1,0x8,0x6,0x7,0x8,0x9,0x8,0x5,0x4,0x9,0x8,0x2,0x7,0x1,0x8,0x6,0x7,0x3,0x2,0x3,0x1,0x5,0x7,0x6,0x9,0x2,0x4,0x3,0x7,0x8,0x3,0x9,0x4,0x8,0x9,0x4,0x6,0x9,0x4,0x9,0x4,0x2,0x4,0x6,0x1,0x6,0x7,0x2,0x1]

陷阱

三次调用sub_401F58，修改了后面计算用到的数据，在特定位置下断点和修改代码，都会导致值变化，从而得不到正确的结果，特别是第三次调用是在反调试函数数组的第一个，如果直接patch掉.text:004B9398 call eax，会导致0x4BC11D处的数据错误。

这三处数据的值可通过附加程序和下内存断点得到，三处数据的正确数值为如下所示：

```
004BC080 BD A3 9C 10
004BC084 46 F3 2C 00
004BC11D 01 02 23 24
```

.text:004B9398 call eax处的返回值并不固定，patch时，会出现问题，导致后面比较的数据和下标不对，可以通过修改00402BEE处的代码为00402BEE | EB FE | jmp dancing1.402BEE，即死循环，然后附加程序，修改回原来的代码，然后在比较的地方下断点，得到正确的结果。

完整的代码

知道算法后，就可以利用z3约束求解，z3脚本如下：

```
#coding=utf-8
from z3 import *

m = [BitVec("x%d"%(i), 16) for i in xrange(58)]
cs=[]
for i in m:
cs.append(And(i >= 1, i <= 9))
s = Solver()
table1=[0x86, 0x89, 0x84, 0x8D, 0xA7, 0x83, 0xA6, 0x25, 0x47, 0x14, 0x35, 0x1D, 0x0F, 0
0x95, 0x01, 0xAE, 0x3F, 0x37, 0xB6, 0x02, 0x14, 0x0B, 0x17, 0x8A, 0x1F, 0x93, 0xAC, 0x9
0x2D, 0x0E, 0x07, 0x9F, 0x92, 0x8C, 0x15, 0xBE, 0x0A, 0x86, 0x07, 0xA4, 0x03, 0x85, 0x2
0x8E, 0x0B, 0x9B, 0x05, 0x0F, 0x84, 0x80, 0x9E, 0x8D, 0x00, 0x16, 0x01]

table2=[
0xBD, 0x48, 0x2B, 0xAA, 0xB0, 0xA3, 0xB9, 0x42, 0xCF, 0x98,
0x4D, 0xB8, 0x3C, 0xA0, 0x32, 0x41, 0x21, 0x91, 0x3A, 0x45,
0x3B, 0x44, 0x9A, 0xBB, 0x19, 0x38, 0x10, 0x28, 0x40, 0x4C,
0xA9, 0xCD, 0x43, 0x33, 0xC6, 0x30, 0x49, 0xA2, 0xBA, 0x4E,
0xC5, 0xC9, 0xC8, 0xCB, 0xCC, 0x34, 0xB1, 0xC3, 0x41, 0xC4,
0xCA, 0x4A, 0x40, 0xCB, 0x08, 0x31, 0xC2, 0xCF, 0x39, 0x4E]
```

```
S=[0x38, 0x39, 0x07, 0xE0, 0x5A, 0x18, 0x1E, 0xE7, 0x5E, 0x3B, 0xEB, 0xA5, 0xAE, 0x07,
0x0D, 0x89, 0xF1, 0x6E, 0xD8, 0x0D, 0x3A, 0xE0, 0x1E, 0x96, 0xA7, 0xB2, 0x7D, 0x15, 0x1
0x01, 0x41, 0x4B, 0x08, 0x1C, 0x1D, 0x33, 0x2B, 0x40, 0x49, 0x79, 0xE3, 0x04, 0xFA, 0x0
0x66, 0x1B, 0xDB, 0x9E, 0x2C, 0x2D, 0x4E, 0x20, 0x87, 0x9E, 0xBB, 0xE9, 0x14, 0x8A, 0x3
0xD5, 0x63, 0x51, 0x71, 0x3C, 0x41, 0x37, 0xA8, 0x2E, 0x01, 0x02, 0x23, 0x24, 0x63, 0x4
0x08, 0x09, 0x19, 0x2B, 0x48, 0x4D, 0xBD, 0xBC, 0x22, 0x11, 0x6C, 0x33, 0x34, 0x77, 0xE
0x18, 0x6C, 0x35, 0x3B, 0xB6, 0x8F, 0x46, 0x10, 0xC3, 0x43, 0x22, 0x5E, 0x44, 0xB3, 0x8
0x28, 0xA4, 0xCB, 0xF0, 0x81, 0xD6, 0x96, 0x0F, 0x5C, 0x31, 0x32, 0x19, 0xF0, 0xDA, 0x5
0x94, 0xF7, 0x27, 0x4E, 0x40, 0x7A, 0x33, 0x1F, 0x20, 0x41, 0x42, 0x6C, 0x66, 0x05, 0x4
0x48, 0x49, 0x30, 0xAC, 0x0C, 0xBE, 0x2E, 0x2F, 0x30, 0x9D, 0x18, 0x2A, 0x23, 0x23, 0x6
0x20, 0x21, 0x2A, 0x6E, 0x4F, 0x83, 0x3E, 0x3F, 0x51, 0x63, 0x9A, 0x03, 0x53, 0x25, 0x2
0xCF, 0x4C, 0x0A, 0x0B, 0x98, 0xB7, 0x88, 0x4F, 0xE9, 0xD3, 0x1E, 0x13, 0x57, 0x35, 0xD
0x08, 0x9A, 0x1A, 0x1B, 0x84, 0x3D, 0xC9, 0x66, 0x0E, 0x8B, 0xB4, 0x58, 0xD0, 0x45, 0xD
0xA5, 0xB9, 0x2A, 0x1C, 0x4C, 0x2B, 0xDB, 0x07, 0xFF, 0x19, 0x12, 0x31, 0x1E, 0x96, 0xE
0x4B, 0x19, 0x3A, 0x19, 0x1A, 0x19, 0xE6, 0xDE, 0x00, 0x21, 0xF7, 0x43, 0x4B, 0x32, 0x3
0x46, 0x29, 0x4A, 0x4B, 0x29, 0x97, 0x0E, 0x19, 0x10, 0xE0, 0x90, 0x73, 0x65, 0xE4, 0x4

r1=[0x2,0x3,0xA,0xF,0x12,0x17,0x19,0x1c,0x20,0x21,0x24,0x2a,0x2e,0x2f,0x34,0x37,0x39,0x
result1=[7,9,3,4,9,5,6,8,2,3,6,1,3,5,4,5,1,7,2,8,3,5]
r2=[0x27,0x42,0x13,0x22,0x25,0x1,0xb,0x2c,0x31,0xc,0x5,0xd,0xe,0x29,0x33,0x10,0x9,0x0,0
result2=[0x3,0x5,0x1,0x5,0x5,0x6,0x2,0x2,0x1,0x1,0x8,0x6,0x7,0x8,0x9,0x8,0x5,0x4,0x9,0x

in1=[0x2,0x3,0xA,0xF,0x12,0x17,0x19,0x1c,0x20,0x21,0x24,0x2a,0x2e,0x2f,0x34,0x37,0x39,0
in2=[0x0,0x1,0x4,0x5,0x6,0x7,0x8,0x9,0xb,0xc,0xd,0xe,0x10,0x11,0x13,0x14,0x15,0x16,0x18
input=[0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0]
for i in range(22):
input[in1[i]]=result1[i]
for i in range(58):
input[in2[i]]=m[i]
for i in range(4):
for j in range(60):
tmp=input[table1[j]&0x7f]
input[table1[j]&0x7f]=((input[table2[j]&0x7f]*9)%256)^0x37
input[table2[j]&0x7f]=((tmp*9)%256)^0x37

for i in range(22):
if(result1[i]==1):
```

```
    cs.append(Or(input[r1[i]]== 32, input[r1[i]]== 73))
    if(result1[i]==2):
    cs.append(input[r1[i]]==74)
    if(result1[i]==3):
    cs.append(input[r1[i]]==171)
    if(result1[i]==4):
    cs.append(input[r1[i]]==44)
    if(result1[i]==5):
    cs.append(Or(input[r1[i]]== 46, input[r1[i]]== 141))
    if(result1[i]==6):
    cs.append(input[r1[i]]==14)
    if(result1[i]==7):
    cs.append(Or(input[r1[i]]== 2, input[r1[i]]== 13,input[r1[i]]== 239,input[r1[i]]== 215)
    if(result1[i]==8):
    cs.append(Or(input[r1[i]]== 35, input[r1[i]]== 80,input[r1[i]]== 192))
    if(result1[i]==9):
    cs.append(input[r1[i]]== 81)
    for i in range(58):
    if(result2[i]==1):
    cs.append(Or(input[r2[i]]== 32, input[r2[i]]== 73))
    if(result2[i]==2):
    cs.append(input[r2[i]]==74)
    if(result2[i]==3):
    cs.append(input[r2[i]]==171)
    if(result2[i]==4):
    cs.append(input[r2[i]]==44)
    if(result2[i]==5):
    cs.append(Or(input[r2[i]]== 46, input[r2[i]]== 141))
    if(result2[i]==6):
    cs.append(input[r2[i]]==14)
    if(result2[i]==7):
    cs.append(Or(input[r2[i]]== 2, input[r2[i]]== 13,input[r2[i]]== 239,input[r2[i]]== 215)
    if(result2[i]==8):
    cs.append(Or(input[r2[i]]== 35, input[r2[i]]== 80,input[r2[i]]== 192))
    if(result2[i]==9):
    cs.append(input[r2[i]]== 81)


    """
    for i in range(58):
    cs.append(S[input[r2[i]]]==result2[i])
    for i in range(22):
    cs.append(S[input[r1[i]]]==result1[i])
    #"""
    #cs.append(Or(input[r1[i]]== 0, input[r1[i]]== 9))
```

```
s.add(cs)

## 去重并求出所有解
while(s.check()==sat):
t = s.model()
flag = ""
for i in m:
flag=flag+str(t[i].as_long())
print flag
flag=hex(int(flag[::-1],10))
flag=(flag.upper())[2:]
print flag[::-1][1:]
exp = []
for val in m:
exp.append(val!=t[val])
s.add(Or(exp))
```

最后求出结果为CBC25EF8D9F482BC1F3DA3CA1F154EC89FC3F1414EDD93A3，可以发现中间那个查表交换位置，其实就是在转圈圈，并没有修改数据，这就是题目名的含义。

看雪CTF晋级赛Q1 题解列表

1、2019KCTF 晋级赛Q1 | 第一题点评及解题思路

2、2019KCTF 晋级赛Q1 | 第二题点评及解题思路

3、2019 KCTF 晋级赛Q1 | 第三题点评及解题思路

4、2019KCTF 晋级赛Q1 | 第四题点评及解题思路

5、2019 KCTF 晋级赛Q1 | 第五题点评及解题思路

6、2019 KCTF 晋级赛Q1 | 第六题点评及解题思路

- End -

公众号ID：ikanxue

官方微博：看雪安全

商务合作：wsc@kanxue.com

戳原文，查看更多精彩writeup！