看雪CTF2019Q2-第8题 迷雾中的琴声 精

风间仁 专家 19

2019-6-19 00:51

sn长度为32

```
.text:00401250                call    ds:GetWindowTextLengthA
.text:00401256                cmp     eax, 20h
.text:00401259                jz      short loc_401296
```

hex2bin(大写的16进制)

sn = hex2bin(sn);

sn ^= key;

```
.text:00401299                call    x_init_sn
.data:0040401C g_xor_key      db 0E9h, 4Fh, 6Eh, 20h, 78h, 1Ah, 7, 0Fh, 0, 17h, 36h, 9, 0Ah, 7, 1Fh, 0Ch
```

创建5个线程分别计算

5个线程的一轮都计算结束后才会进行下一轮计算, 共300轮

每个线程更新sn中的两个字节(二元一次方程)

把计算中用到的sn的2个索引值和1个中间变量保存下来, 用来做逆运算(不用管随机数什么的)

```
.text:00401900 x_start_check
```

计算过程

```
DWORD xorshift32(DWORD x)
{
    x ^= x << 13;
    x ^= x >> 17;
    x ^= x << 5;
    return x;
}

// m2, m3
DWORD transform1(DWORD a, DWORD b)
{
    DWORD m = a ^ (a >> 7);
    DWORD n = b ^ (m << 7);
    return m ^ b ^ (n << 6);
}

void swap8(BYTE& a, BYTE& b)
{
    BYTE t = a;
    a = b;
    b = t;
}

string g_temp_data;

void one_round(DWORD id)
{
    WORD v_00;
    WORD v_01;
    WORD v_02;
    DWORD v_03;
    // wait for g_h_ary[id]
    if (g_thread_inited[id])
    {
        v_00 = (WORD)g_4045EC ^ g_rnd0[id];
        v_01 = (WORD)g_4045EC ^ g_rnd1[id];
        v_02 = (WORD)g_4045EC ^ g_rnd2[id];
        v_03 = g_4045EC ^ g_rnd_xorshift[id];
```

```
    } else
    {
        g_thread_inited[id] = 1;
        v_00 = 3923;
        v_01 = 28;
        v_02 = 1;
        v_03 = id + 0x1D4B42;
    }

    BYTE v36[16] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };

    // Wichmann-Hill
    g_rnd0[id] = 171 * v_00 % 30269;
    g_rnd1[id] = 172 * v_01 % 30307;
    g_rnd2[id] = 170 * v_02 % 30323;
    g_rnd_xorshift[id] = xorshift32(v_03);

    WORD rnd_sum = (WORD)(g_rnd0[id] + g_rnd2[id] + g_rnd1[id]);
    g_40402C[id] += rnd_sum;
    g_404040[id] -= rnd_sum;
    g_404068[id] *= rnd_sum;
    g_40407C[id] -= rnd_sum;
    g_404054[id] += rnd_sum;

    DWORD v20;
    DWORD v44;
    DWORD v38;

    DWORD v47 = g_404068[id];
    DWORD m0 = g_404040[id];
    DWORD m1 = g_40407C[id];
    DWORD m2 = g_40402C[id];
    DWORD m3 = g_404054[id];
    DWORD v15 = g_404068[id];

    for (int k = 0; k < 32; k++)
    {
        v20 = transform1(m2, m3);
        v44 = transform1(m0, v20);


        DWORD v22 = v47;
        v47 = m3;
        m2 = v22;

        BYTE v21 = (BYTE)(v20 * ((v15 >> 2) + m0 + 1)) & 0x0F;
        BYTE v26 = ((BYTE)v44 * ((BYTE)m2 + (BYTE)(m1 >> 2) + 1)) & 0xF;
        swap8(v36[v26], v36[v21]);

        m0 = m1;
        v38 = m1;
        m1 = v20;
        m3 = v44;
        v15 = v47;
    }

    g_40407C[id] = v20;
    g_404054[id] = v44;
    g_40402C[id] = m2;
    g_404040[id] = v38;
    g_404068[id] = v47;

    BYTE v28 = (BYTE)g_rnd_xorshift[id];
    WORD v28_2 = v28 * v28;
    BYTE v33 = g_buf256[(BYTE)(v28_2 >> 8) ^ g_buf256[(BYTE)v28_2]];
    BYTE sn1_idx = v36[2 * id];
    BYTE sn0_idx = v36[2 * id + 1];
    g_temp_data.append((char *)&v28, 1);
    g_temp_data.append((char *)&sn1_idx, 1);
    g_temp_data.append((char *)&sn0_idx, 1);

    BYTE sn0 = g_sn[sn0_idx];
    BYTE sn1 = g_sn[sn1_idx];
    g_sn[sn1_idx] = sn0 + 3 * (v33 + sn1);
    g_sn[sn0_idx] = (BYTE)(v33 + sn1 + g_buf256[v28] - 2 * sn0);

    g_4045F8[id] = v33 ^ g_buf256[v28];
    // release g_h
}

void one_round_rev(PBYTE temp_cur, int i, int k)
{
    BYTE v28 = temp_cur[0];
    BYTE sn1_idx = temp_cur[1];
    BYTE sn0_idx = temp_cur[2];
    WORD v28_2 = v28 * v28;
    BYTE v33 = g_buf256[(BYTE)(v28_2 >> 8) ^ g_buf256[(BYTE)v28_2]];

    BYTE new_sn0 = g_sn[sn0_idx];
    BYTE new_sn1 = g_sn[sn1_idx];
    g_sn[sn1_idx] = (2*new_sn1 + new_sn0 - 7*v33 - g_buf256[v28]) * 0xB7;
    g_sn[sn0_idx] = (new_sn1 - 3*new_sn0 + 3*g_buf256[v28]) * 0xB7;
}
```

```
void test2()
{
    string str = util::hex2bin("11121314212223243132333441424344");
    memcpy(g_sn, str.c_str(), str.size());
    xor_buf(g_sn, sizeof(g_sn), g_xor_key);
    for (int i = 0; i < 300; i++)
    {
        int k;
        for (k = 0; k < 5; k++)
        {
            one_round(k);
        }
        g_4045EC = 0;
        for (k = 0; k < 5; k++)
        {
            g_4045EC += g_4045F8[k];
        }
    }
    print_buf(g_sn, sizeof(g_sn));
    // 0x21, 0x19, 0x4A, 0xB9, 0x7E, 0x63, 0x04, 0x5F, 0xEA, 0xC3, 0xFC, 0x7C, 0x70, 0xC4, 0x80, 0xB2
    util::SaveFile(g_temp_data, "temp_data.dat");
}

void test2_rev()
{
    BYTE buf[16] = {
        0x21, 0x19, 0x4A, 0xB9, 0x7E, 0x63, 0x04, 0x5F, 0xEA, 0xC3, 0xFC, 0x7C, 0x70, 0xC4, 0x80, 0xB2
    };
    memcpy(g_sn, buf, sizeof(buf));
    util::LoadFile(g_temp_data, "temp_data.dat");
    PBYTE temp = (PBYTE)g_temp_data.c_str();

    for (int i = 300 - 1; i >= 0; i--)
    {
        for (int k = 5 - 1; k >= 0; k--)
        {
            PBYTE temp_cur = temp + i * 5 * 3 + k * 3;
            one_round_rev(temp_cur, i, k);
        }
    }
    xor_buf(g_sn, sizeof(g_sn), g_xor_key);
    printf("%s\n", util::bin2hex(g_sn, sizeof(g_sn)).c_str());
}
```

校验sn的hash

```
.text:00401A39                lea     edi, [esp+60h+var_31]
...
.text:00401B1B                xor     eax, 18000h
.text:00401B20                xor     ecx, 0C00h
.text:00401B26                shld    ecx, eax, 1
.text:00401B2A                add     eax, eax
.text:00401B2C                sub     edx, 1
.text:00401B2F                jnz     short loc_401B11
.text:00401B31                cmp     eax, 38B0000h
.text:00401B36                jnz     short loc_401B55
.text:00401B38                cmp     ecx, 0B2A289CFh
.text:00401B3E                jnz     short loc_401B55

// multi -> one
__int64 hash_v(__int64 v)
{
    for (int i = 0; i < 64; i++)
    {
        if (v < 0)
        {
            v ^= 0x00000C0000018000;
        }
        v <<= 1;
    }
    return v;
}

__int64 hash_sn(PBYTE sn)
{
    __int64 v1 = util::byteswap64(*(__int64*)(sn));
    __int64 v2 = util::byteswap64(*(__int64*)(sn+8));
    __int64 v;
    v = hash_v(v1);
    v ^= v2;
    v = hash_v(v);
    return v;
}
```

校验通过后, 跳向sn执行弹框(界面上提示弹出Win表示正确)

```
.text:00401321                mov     ecx, [ebp+hDlg]
.text:00401324                mov     g_hDlg, ecx
.text:0040132A                mov     eax, offset g_sn
.text:0040132F                jmp     eax
.text:00401331                mov     edx, [ebp+hDlg]
.text:00401334                push    edx             ; hWnd
.text:00401335                call    ds:DestroyWindow
.text:0040133B                jmp     short loc_401380
```

尝试构造shellcode(未通过hash验证)

```
004044C8    83C0 0C         add     eax, 0C
004044CB    6A 00           push    0
004044CD    50              push    eax
004044CE    50              push    eax
004044CF  - E9 1ECEFFFF     jmp     004012F2
004044D4    Win

83 C0 0C 6A 00 50 50 E9 1E CE FF FF 57 69 6E 00
```

观察所有MessageBoxA的调用,
其中第1个参数都是mov ecx, hDlg; push ecx的形式,
而00401324处却把hDlg额外保存到了全局变量g_hDlg中

```
.text:004012E8                push    offset Caption  ; lpCaption
.text:004012ED                push    offset Text     ; lpText
.text:004012F2                mov     ecx, [ebp+hDlg]
.text:004012F5                push    ecx             ; hWnd
.text:004012F6                call    ds:MessageBoxA

.text:00401321                mov     ecx, [ebp+hDlg]
.text:00401324                mov     g_hDlg, ecx
.text:0040132A                mov     eax, offset g_sn
.text:0040132F                jmp     eax

.data:004043FC g_hDlg
```

搜索g_hDlg的引用(搜索16进制FC434000)
另外找到1处引用(位于资源6.ico中, 所有区段都是RWE的)
根据提示要弹出Win, 所以跳到这里时eax必须为0x125A3F2F ^ 0x006E6957(Win) = 0x12345678

```
.rsrc:0041C398                xor     ds:dword_41C3BB, eax
.rsrc:0041C39E                push    0
.rsrc:0041C3A0                push    offset dword_41C3BB
.rsrc:0041C3A5                push    offset dword_41C3BB
.rsrc:0041C3AA                push    g_hDlg
.rsrc:0041C3B0                call    ds:MessageBoxA
.rsrc:0041C3B6                jmp     loc_401331
.rsrc:0041C3BB dword_41C3BB   dd 125A3F2Fh
```

再次构造shellcode(未通过hash验证)

```
004044C8    B8 78563412     mov     eax, 12345678
004044CD    3105 BBC34100   xor     dword ptr [41C3BB], eax
004044D3  - E9 C67E0100     jmp     0041C39E

B8 78 56 34 12 31 05 BB C3 41 00 E9 C6 7E 01 00
```

看来得用到hash验证了, 固定前10个字节, 后面6个字节用z3约束求解

```
004044C8    B8 78563412              mov eax,0x12345678
004044CD  - E9 C67E0100              jmp 0041C398

B8 78 56 34 12 E9 C6 7E 01 00 00 00 00 00 00 00
```

运行脚本得到: b878563412e9c67e0100a6e11213382f
逆运算得到sn(正解): 8CF4BD334ACF8F1222B70EA1FF8085D6

```python
from z3 import *


def fn(n):
    v = n
    for i in range(64):
        bit = (v >> 63) & 1
        v ^= bit * 0xC0000018000
        v <<= 1
        v &= 0xFFFFFFFFFFFFFFFF
    return v


def big_int64_to_str(v):
    s = ''
    for i in range(8):
        s += chr((v >> ((7 - i) * 8)) & 0xFF)
    return s


def test1():
    a = BitVec('a', 64)
    b = BitVec('b', 64)
    v = fn(a)
    v ^= b
    v = fn(v)
    solver = Solver()
    solver.add(v == 0xB2A289CF038B0000)
    solver.add(a == 0xB878563412E9C67E)
    solver.add(((b >> 56) & 0xFF) == 1)
    solver.add(((b >> 48) & 0xFF) == 0)
    print solver.check()
    m = solver.model()
    print (big_int64_to_str(m[a].as_long()) + big_int64_to_str(m[b].as_long())).encode('hex')
    return


def test():
    test1()
    return


test()
```

这里的mov与jmp并不是一定得紧挨着的,
另外0041C396那一句也可以利用,
丢两个多解
sn: 6A0E470F088F93B27E3366C375FCE132
sn: 8AE0942B5CE0778A643DDBAAE3E46CC3

```
mov eax,0x12345678
 jmp 0041C398

.rsrc:0041C396                and     al, 0FFh
.rsrc:0041C398                xor     ds:dword_41C3BB, eax
```

[公告]看雪.纽盾 KCTF 2019晋级赛Q3攻击方规则，9月10日开赛，华为P30 Pro、iPad、kindle等你来拿！

最后于 2019-6-19 02:12 被风间仁编辑，原因:

最新回复 (4)

最新回复 (4)

KevinsBobo 4 2019-6-24 14:06 2 楼 0

版主

tql，不仅算出正解，还搞出2个多解！

黄瓜香蕉 2019-6-24 14:20 3 楼 0

初级

tql不仅正解，还多解。。。。

readyu 8 2019-6-24 23:22 4 楼 0

高级

风神太强大了

看场雪 3 2019-6-25 13:36 5 楼 0

版主

风神强悍

勇士小蓝

内容

回帖 表情 高级回复

返回