【GACTF】EasyRe WriteUp

原创

古月浪子 于 2020-08-31 14:29:55 发布 480 收藏

文章标签： CTF

本文链接：https://blog.csdn.net/tqydyqt/article/details/108312602

IDA打开直接能看到main函数

```
int __cdecl main()
{
  _DWORD *v0; // eax
  _DWORD *v1; // ST28_4

  sub_804873B();
  __isoc99_scanf("%lld", &input);
  v0 = sub_8048DE2();
  v1 = v0;
  v0[8] = &unk_804B080;
  sub_8049065();
  sub_8048838(v1);
  return 0;
}
```

```
unsigned int sub_8049065()
{
  int v1; // [esp+Ch] [ebp-1Ch]
  int v2; // [esp+10h] [ebp-18h]
  int v3; // [esp+14h] [ebp-14h]
  int v4; // [esp+18h] [ebp-10h]
  unsigned int v5; // [esp+1Ch] [ebp-Ch]

  v5 = __readgsdword(0x14u);
  v1 = 24;
  v2 = 34;
  v3 = 48;
  v4 = 17;
  mprotect(&dword_8048000, 0x2000u, 7);
  sub_8048EA5(sub_8048838, 362, &v1);
  return __readgsdword(0x14u) ^ v5;
}
```

```
unsigned int __cdecl sub_8048EA5(unsigned int *a1, int a2, int a3)
{
  unsigned int v4; // [esp+14h] [ebp-34h]
  unsigned int v5; // [esp+18h] [ebp-30h]
  int i; // [esp+1Ch] [ebp-2Ch]
  int v7; // [esp+20h] [ebp-28h]
  int v8; // [esp+24h] [ebp-24h]
  unsigned int v9; // [esp+2Ch] [ebp-1Ch]

  v9 = __readgsdword(0x14u);
  v7 = 34 / a2 + 9;
  v5 = 1316499440 * v7;
  v4 = *a1;
  do
  {
    v8 = (v5 >> 2) & 3;
    for ( i = a2 - 1; i; --i )
    {
      a1[i] -= ((2 * v4 ^ (a1[i + 0x3FFFFFFF] >> 4)) + ((v4 >> 3) ^ 32 * a1[i + 0x3FFFFFFF]) + 40) ^ ((v4 ^ v5 ^ 0x77) + (a1[i + 0x3FFFFFFF] ^ *(4 * (v8 ^ i & 3) + a3)) - 15);
      v4 = a1[i];
    }
    *a1 -= ((2 * v4 ^ (a1[a2 + 0x3FFFFFFF] >> 4)) + ((v4 >> 3) ^ 32 * a1[a2 + 0x3FFFFFFF]) + 40) ^ ((v4 ^ v5 ^ 0x77)
                                                                                                    + (a1[a2 + 0x3FFFFFFF] ^ *(4 * v8 + a3))
                                                                                                    - 15);
    v4 = *a1;
    v5 -= 1316499440;
    --v7;
  }
  while ( v7 );
  return __readgsdword(0x14u) ^ v9;
}
```

可以看到，在执行sub_8048838前，调用了前一个函数，对它进行动态patch（直接点进去sub_8048838会发现是无意义代码）这个动态patch代码的算法有点复杂，直接gdb下断点然后dump memory ./dmp 0x8048838 0x8048de0，然后用WinHex把dump出来的替换掉原本的，IDA重新打开patch后的，就可以了

```
unsigned int __cdecl sub_8048838(_DWORD *a1)
{
  _BYTE *v1; // ST28_4
  unsigned int v3; // [esp+2Ch] [ebp-Ch]

  v3 = __readgsdword(0x14u);
  while ( 1 )
  {
    if ( *a1[8] == 113 )
    {
      a1[6] -= 4;
      *a1[6] = *(a1[8] + 1);
      a1[8] += 5;
    }
    if ( *a1[8] == 65 )
    {
      a1[1] += a1[2];
      ++a1[8];
    }
    if ( *a1[8] == 66 )
    {
      a1[1] -= a1[4];
      ++a1[8];
    }
    if ( *a1[8] == 67 )
    {
      a1[1] *= a1[3];
      ++a1[8];
    }
    if ( *a1[8] == 68 )
    {
      a1[1] /= a1[5];
      ++a1[8];
    }
    if ( *a1[8] == -128 )
    {
      a1[sub_80487EF(a1, 1u)] = *(a1[8] + 2);
      a1[8] += 6;
```

可以看出来，这是一个虚拟机，main函数里有对其进行初始化的函数

```
_DWORD *sub_8048DE2()
{
  _DWORD *v0; // eax
  _DWORD *v1; // ST18_4

  v0 = malloc(0x3Cu);
  v1 = v0;
  *v0 = 0;
  v0[1] = 0;
  v0[2] = 0;
  v0[3] = 0;
  v0[4] = 0;
  v0[5] = 0;
  v0[9] = 0;
  v0[10] = calloc(4u, 0x50u);
  v1[6] = v1[10] + 316;
  v1[7] = v1[10] + 316;
  v1[8] = 0;
  return v1;
}
```

同样在main函数中我们可以找到opcode的位置

```
.data:0804B080 unk_804B080     db    9
.data:0804B081                 db  10h
.data:0804B082                 db  80h
.data:0804B083                 db    2
.data:0804B084                 db  0Dh
.data:0804B085                 db    0
.data:0804B086                 db    0
.data:0804B087                 db    0
.data:0804B088                 db  22h ; "
.data:0804B089                 db  77h ; w
```

```
.data:0804B08A                  db  10h
.data:0804B08B                  db  80h
.data:0804B08C                  db    2
.data:0804B08D                  db    9
.data:0804B08E                  db    0
.data:0804B08F                  db    0
.data:0804B090                  db    0
.data:0804B091                  db  23h ; #
.data:0804B092                  db  80h
.data:0804B093                  db    2
.data:0804B094                  db    0
.data:0804B095                  db  96h
.data:0804B096                  db 0F3h
.data:0804B097                  db  78h ; x
.data:0804B098                  db  31h ; 1
.data:0804B099                  db  77h ; w
.data:0804B09A                  db  10h
.data:0804B09B                  db  80h
```

用IDAPython导出一下，得到

```
9 16 128 2 13 0 0 0 34 119 16 128 2 9 0 0 0 35 128 2 0 150 243 120 49 119 16 128 2 17 0 0 0 35 128 2 0 0 212 133 49 119
16 128 2 19 0 0 0 34 119 160 9 128 2 255 0 0 0 49 128 3 2 0 0 0 67 128 2 24 0 0 0 65 164 0 0 0 9 128 2 8 0 0 0 34 128 2
255 0 0 0 49 128 5 7 0 0 0 68 128 2 33 0 0 0 65 164 1 0 0 9 128 2 16 0 0 0 34 128 2 255 0 0 0 49 128 9 187 0 0 0 119 128
2 255 0 0 0 65 164 2 0 0 9 128 2 24 0 0 0 34 128 2 255 0 0 0 49 128 4 160 0 0 0 66 128 2 119 0 0 0 65 164 3 0 0
161
193 0 177 119 194 11 1 0 0
193 1 178 119 194 122 0 0 0
193 2 180 119 194 149 0 0 0
193 3 179 119 194 6 1 0 0
193 4 178 119 194 125 0 0 0
193 5 180 119 194 173 0 0 0
193 6 177 119 194 47 1 0 0
193 7 179 119 194 101 1 0 0
193 8 177 119 194 45 1 0 0
193 9 177 119 194 47 1 0 0
193 10 179 119 194 57 1 0 0
193 11 179 119 194 13 1 0 0
193 12 180 119 194 187 0 0 0
193 13 178 119 194 8 0 0 0
193 14 179 119 194 13 1 0 0
193 15 177 119 194 63 1 0 0
193 16 179 119 194 58 1 0 0
193 17 179 119 194 97 1 0 0
193 18 178 119 194 87 0 0 0
193 19 177 119 194 32 1 0 0
193 20 179 119 194 13 1 0 0
193 21 177 119 194 63 1 0 0
193 22 179 119 194 63 1 0 0
193 23 180 119 194 181 0 0 0
193 24 177 119 194 19 1 0 0
193 25 180 119 194 160 0 0 0
193 26 177 119 194 33 1 0 0
193 27 179 119 194 13 1 0 0
193 28 178 119 194 11 0 0 0
193 29 179 119 194 57 1 0 0
193 30 177 119 194 115 1 0 0
193 31 178 119 194 70 0 0 0
153
```

分析opcode真的难受。。。我简单分析了一下（你看不懂没关系，我自己写的通常只有我看得懂QAQ）

```
9: [1]=input; +1
16: [9]=[1]; +1
34: [1]>>=[2]; +1
35: [1]<<=[2]; +1
49: [1]&=[2]; +1
```

```
119: [1]^=[9]; +1
153: break
161: read flag; if(flag.len!=33)exit; +1
177: [9]=dA0; +1
178: [9]=dA4; +1
179: [9]=dA8; +1
180: [9]=dAC; +1
193: [1]=flag[next]; +2
194: if([1]!=next)exit; +5
```

我们可以看到，以opcode-161作为分界线，前面的是对我们一开始输入的%lld进行各种变换和各种比较，不对就退出，对的话，又会对dA0、dA4、dA8、dAC赋值
后面开始，就是逐位对我们输入的flag进行异或，然后和一个数比较，异或的数是前面提到的被赋值的4个地址的数之一
根据flag的格式是GACTF{xxx}，而前4次比较直接把4个数都用过一遍了，于是可以直接逆推这4个数，所以opcode-161之前的可以完全不用管了
写个脚本，提取opcode里面的数据，生成了下面的代码

```
int main()
{
 cout << char(11 + 256 ^ 332);
 cout << char(122 ^ 59);
 cout << char(149 ^ 214);
 cout << char(6 + 256 ^ 338);
 cout << char(125 ^ 59);
 cout << char(173 ^ 214);
 cout << char(47 + 256 ^ 332);
 cout << char(101 + 256 ^ 338);
 cout << char(45 + 256 ^ 332);
 cout << char(47 + 256 ^ 332);
 cout << char(57 + 256 ^ 338);
 cout << char(13 + 256 ^ 338);
 cout << char(187 ^ 214);
 cout << char(8 ^ 59);
 cout << char(13 + 256 ^ 338);
 cout << char(63 + 256 ^ 332);
 cout << char(58 + 256 ^ 338);
 cout << char(97 + 256 ^ 338);
 cout << char(87 ^ 59);
 cout << char(32 + 256 ^ 332);
 cout << char(13 + 256 ^ 338);
 cout << char(63 + 256 ^ 332);
 cout << char(63 + 256 ^ 338);
 cout << char(181 ^ 214);
 cout << char(19 + 256 ^ 332);
 cout << char(160 ^ 214);
 cout << char(33 + 256 ^ 332);
 cout << char(13 + 256 ^ 338);
 cout << char(11 ^ 59);
 cout << char(57 + 256 ^ 338);
 cout << char(115 + 256 ^ 332);
 cout << char(70 ^ 59);
}
```

跑出来即可

> GACTF{c7ack_m3_sh3ll_smc_vm_0k?}