170916 逆向-WHCTF（BabyRe/CrackMe）

原创

奈沙夜影 于 2017-09-16 18:48:27 发布 3220 收藏

分类专栏： CTF

本文链接：https://blog.csdn.net/whklhhhh/article/details/78005589

1625-5 王子昂 总结《2017年9月16日》【连续第349天总结】

A. XCTF（武汉站）-Reverse

B.

CRACKME

无壳，C++

打开是一个简单的输入框和注册按钮

点击注册按钮会弹框“哎，注册码错了，你得换个新的哟！”

IDA没有main函数，看起来是MFC写的

查找字符串、断GetDlgText等API都没有结果

说明字符串都被加密过了

IDA逐个函数查找，发现有两个函数调用了MessageBoxA

查看交叉引用，发现这两个函数的call都来自于同一个函数，并且还是分支关系

```
int __thiscall sub_4015E0(CWnd *this, int a2)
{
  CWnd *v2; // esi@1
  int result; // eax@3
  int v4; // [sp-Ch] [bp-Ch]@2
  int *v5; // [sp-4h] [bp-4h]@2

  v2 = this;
  CWnd::UpdateData(this, 1);
  if ( *(_DWORD *)(*((_DWORD *)v2 + 165) - 8) == 33
    && (v4 = *((_DWORD *)v2 + 165), v5 = &v4, CString::CString(&v4, (char *)v2 + 660), flag_check(v2, v4)) )
  {
    result = sub_4016E0(v2);
  }
  else
  {
    result = sub_401720(v2);
  }
  return result;
}
```


回到汇编状态找到sub_401720和sub_4016e0的地址，在OD中下断，再点击注册按钮果然被401720断下来了

说明这里就是关键跳，再回到sub_401720里看看：

```
int __thiscall sub_401720(CWnd *this)
{
  CWnd *v1; // ebx@1
  char v3; // [sp+Ch] [bp-24h]@1

  v1 = this;
  qmemcpy(&v3, &unk_403290, 0x22u);//字符串拷贝
  sub_4016A0(30, &v3);//处理
  return CWnd::MessageBoxA(v1, &v3, 0, 0);//弹窗
}
```

两个函数结构类似，都是调用sub_4016A0对v3进行处理，拷贝的字符串源当然不同
sub_4016A0对字符串进行了异或解密，分别是20和30

在OD里对sub_4015E0的判断处下断，爆破之
弹窗“看到我的注册码了么，那就是flag哦”

……果然没这么简单哦
那么关键问题就在于if里调用的函数了
因为是指针调用，所以只能在OD里看了

```
004015E6   .  E8 9F030000   call <jmp.&MFC42.#6334>                ;  get注册码
004015EB   .  8B8E 94020000 mov ecx,dword ptr ds:[esi+0x294]
004015F1   .  8D86 94020000 lea eax,dword ptr ds:[esi+0x294]
004015F7   .  8379 F8 21    cmp dword ptr ds:[ecx-0x8],0x21          ;  长度=0x21
004015FB   .  75 22         jnz short 0dc17c69.0040161F              ;  关键跳1
004015FD   .  51            push ecx
004015FE   .  8BCC          mov ecx,esp
00401600   .  896424 08     mov dword ptr ss:[esp+0x8],esp
00401604   .  50            push eax                                 ;  0dc17c69.004015EB
00401605   .  E8 7A030000   call <jmp.&MFC42.#535>
0040160A   .  8BCE          mov ecx,esi                              ;  0dc17c69.004022FB
0040160C   .  E8 1F000000   call 0dc17c69.00401630                   ;  Flag_Check
00401611   .  84C0          test al,al
00401613   .  74 0A         je short 0dc17c69.0040161F               ;  关键跳2
00401615   .  8BCE          mov ecx,esi                              ;  0dc17c69.004022FB
00401617   .  E8 C4000000   call 0dc17c69.004016E0
0040161C   .  5E            pop esi                                  ;  mfc42.0FBA50E5
0040161D   .  59            pop ecx                                  ;  mfc42.0FBA50E5
0040161E   .  C3            retn
```

进行了两次检查，第一次是长度必须为0x21，第二次是sub_00401630进行检查
回到IDA查看flag_check：

```
char __thiscall flag_check(_BYTE *this, int a2)
{
  int v2; // edi@1
  _BYTE *v3; // ebp@1
  int v4; // edx@1
  signed int v5; // esi@1
  char v7; // [sp+13h] [bp-1h]@1

  v2 = 0;
  v3 = this;
  v7 = 1;
  v4 = 10;
  v5 = 0;
  do
  {
    srand(v4);                                  // 以1作为种子，rand[0]恒为41，模10后仍为1
                                                // 因此v4恒=1
    v4 = rand() % 10;
    if ( *(_BYTE *)(v2 + a2) != *(&v3[v4 + 96] + v5) )// v3[96]为字符串基址，a2为输入字符串的基址
      v7 = 0;
    v5 += 10;
    ++v2;
  }
  while ( v5 < 330 );
  CString::~CString((CString *)&a2);
  return v7;
}
```

函数中虽然有随机数，但是注意srand(v4)，把v4作为初始化种子
由于rand()是伪随机数，实际上依赖种子；当种子相同的时候，rand()序列是相同的
在OD中debug可知，初始以10作为种子，rand[0]为0x47（71）；以1作为种子，rand[0]为0x29（41）
模10后都令v4=1
因此，只需要提取字符串第(10n+1)个字符即可
字符串可以直接shift+f12复制出来，提取得到：

```
s = ";f1K3{c5:efl21t4;1t1zaxpim9}5+?gtux;=vc9v{v7+buhU{bT=-am2q}=fh[xk{y?xrqe{?}l5-sd2-Mo+:j{9=sY[dalvp
for i in range(len(s)//10):
    print("%s"%s[i*10+1],end='')
```

> flag{The-Y3ll0w-turb4ns-Upri$ing}Gizuw1dz4;:6j=9j]:a5+]n

提取{}内字符串提交即可

BABYRE

IDA反编译失败，显示堆栈不平衡
读汇编发现main结构很简单，input flag后调用judge函数，然后puts right or wrong
关键在于judge，汇编显示test r14啥的，只有两三句就ret了
于是IDA远程调试，在judge下断发现直接报错
在main头下断，一步一步跟下去后发现judge内的语句变成一堆赋值和循环判断了

刚开始还以为是IDA日常分析错误，但是想想没道理动态调试中就分析正确了
只可能是因为当走到那里的时候语句改变了
即本程序对关键函数judge进行了加密，main中对其进行解密后再执行judge

观察汇编

```
.text:0000000000400617     loc_400617:                             ; CODE XREF: main+38j
.text:0000000000400617 028                 mov     eax, [rbp+var_4]
.text:000000000040061A 028                 cdqe
.text:000000000040061C 028                 movzx   eax, byte ptr judge[rax]
; 取出judge的第n个字节
.text:0000000000400623 028                 xor     eax, 0Ch
; 与0xc异或
.text:0000000000400626 028                 mov     edx, eax
.text:0000000000400628 028                 mov     eax, [rbp+var_4]
.text:000000000040062B 028                 cdqe
.text:000000000040062D 028                 mov     byte ptr judge[rax], dl
; 送回judge
.text:0000000000400633 028                 add     [rbp+var_4], 1
.text:0000000000400637
.text:0000000000400637     loc_400637:                             ; CODE XREF: main+Fj
.text:0000000000400637 028                 cmp     [rbp+var_4], 0B5h
.text:000000000040063E 028                 jle     short loc_400617
; 循环0xb5次
```

写一个IDC脚本对judge进行解密即可：

```
IDC>auto i; for(i=0;i<0xb5;i++) PatchByte(0x600b00 + i, Byte(0x600b00 + i)^ 0xc);
```

再进行反编译就已经可以了：

```
  v1 = 102;
  v2 = 109;
  v3 = 99;
  v4 = 100;
  v5 = 127;
  v6 = 107;
  v7 = 55;
  v8 = 100;
  v9 = 59;
  v10 = 86;
  v11 = 96;
  v12 = 59;
  v13 = 110;
  v14 = 112;
  for ( i = 0; i <= 13; ++i )
    *(_BYTE *)(i + a1) ^= i;
  for ( i = 0; i <= 13 && *(_BYTE *)(i + a1) == *(&v1 + i); ++i )
    ;
  __asm { iret }
}
```

很常见的与序号异或解密，脚本跑一下即可：

> flag{n1c3_j0b}

C. 明日计划
国赛WriteUp