2020安洵杯——EasyCM WriteUP

原创

Code Segment 于 2021-01-23 23:43:46 发布 220 收藏 1

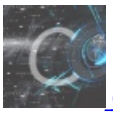
分类专栏： CTF CTF Reverse

本文链接：https://blog.csdn.net/qq_43547885/article/details/113064628

CTF 同时被 2 个专栏收录

5 篇文章 0 订阅

订阅专栏



CTF Reverse

6 篇文章 0 订阅

订阅专栏

文章目录

- 最近犯懒，没看新题，想起来之前安洵杯做过一道 SMC + 反调试的题，当时是动调 + 瞎蒙做出来的，今天来整理一下里面的知识点
- 题目链接：https://github.com/D0g3-Lab/i-SOON_CTF_2020/tree/main/re/EasyCM
- 需要用到的 IDA 插件（能够简化大量操作）：IDACode & IDAPython

概述

1. 入口点: 这题用到了 `TLSCallback` 在 `main` 函数前执行各种解密操作, 在 IDA 中按下 `Ctrl+E` 可以找到所有的入口点
2. 置换用的码表在 `TLSCallback` 中进行了解密, 调试环境下会置换为垃圾数据
3. 关键的加密函数是 SMC 技术保护的

详细

反调

- 使用了 `CheckRemoteDebuggerPresent` 这个 API, 处于调试环境下时返回值为 1
- 下图的函数在其中一个 `TLSCallback` 中被调用, 对置换用的码表进行解密. 可见当处于调试环境时不会进行解密的操作.

```
int sub_415A50()
{
  HANDLE v0; // eax
  int result; // eax
  int i; // [esp+D4h] [ebp-18h]
  BOOL pbDebuggerPresent; // [esp+E0h] [ebp-Ch] BYREF

  pbDebuggerPresent = 0;
  v0 = GetCurrentProcess();
  result = CheckRemoteDebuggerPresent(v0, &pbDebuggerPresent);
  if ( pbDebuggerPresent )
  {
    for ( i = 0; i < 64; ++i )
    {
      if ( i >= 26 )
      {
        if ( i >= 45 )
          byte_423010[i] = -122;
        else
          byte_423010[i] = -90;
      }
      else
      {
        byte_423010[i] = -57;
      }
      result = i + 1;
    }
  }
  return result;
}
```

花指令

- 花指令用于误导 IDA 的静态反汇编. 本题中的花指令都比较简单, 都是通过控制 `retn` 时栈顶的 `eip` 控制跳转. 当时做题的时候傻乎乎的动调, 现在反应过来了, 应该直接推算出跳转到哪, 然后用 IDA patch 上就行了
- 举例来说, 在 SMC 进行自揭密的函数中, 明显有花指令的干扰

```
78                mov     dword ptr [ebp-0Ch], 0
7F                push    eax
80                xor     eax, eax
82                test    eax, eax
84                jz      short near ptr locret_415A90+1
86                jnz     short $+2
88
88 loc_415A88:                            ; CODE XREF: .text:00415A86↑j
88                pop     esi
89                and     eax, ebx
8B                push    eax
8C                xor     eax, ebx
8E                jz      short loc_415A93
90
90 locret_415A90:                         ; CODE XREF: .text:00415A84↑j
90                retn    8B58h
93 ; ---------------------------------------------------------------------------
```

- 把没用的全 nop 掉, 条件跳转改成 `jmp` 即可

```
415A7F                push    eax
415A80                nop
415A81                nop
415A82                nop
415A83                nop
415A84                jmp     short loc_415A91
415A86 ; ---------------------------------------------------------------------------
415A86                nop
415A87                nop
415A88                nop
415A89                nop
415A8A                nop
415A8B                nop
415A8C                nop
415A8D                nop
415A8E                nop
415A8F                nop
415A90                nop
415A91
415A91 loc_415A91:                            ; CODE XREF: sub_415A50+34↑j
415A91                pop     eax
415A92                mov     esi, esp
415A94                lea     eax, [ebp+pbDebuggerPresent]
415A97                push    eax             ; pbDebuggerPresent
415A98                mov     edi, esp
415A9A                call    ds:GetCurrentProcess
415AA0                cmp     edi, esp
415AA2                call    j__RTC_CheckEsp
```

SMC

- 这里进行了 SMC 的自解密, 当然也有几处花指令, 全部改了之后非常清晰

```
char *sub_419BD0()
{
  char *result; // eax
  unsigned int i; // [esp+D0h] [ebp-5Ch]
  char *v2; // [esp+DCh] [ebp-50h]
  char *Str1; // [esp+E8h] [ebp-44h]
  HMODULE v4; // [esp+124h] [ebp-8h]

  v4 = GetModuleHandleA(0);
  for ( Str1 = (char *)v4 + *((_DWORD *)v4 + 15) + *(unsigned __int16 *)((char *)v4 + *((_DWORD *)v4 + 15) + 20) + 24;
        j_strcmp(Str1, ".cyzcc");
        Str1 += 40 )
  {
    ;
  }
  v2 = (char *)v4 + *((_DWORD *)Str1 + 3);
  for ( i = 0; ; ++i )
  {
    result = Str1;
    if ( i >= *((_DWORD *)Str1 + 4) )
      break;
    v2[i] ^= aD0g3[i % 4];
  }
  return result;
}
```

- 可以写个 IDAPython 脚本修改 idb

```
def unpack_cyzcc():
    f = open("backup", "wb")
    key = b'D0g3'
    start = 0x41e000
    length = 0x1200
    for i in range(length):
        ea = start+i
        old = idc.Byte(ea)
        f.write(bytes(old))
        k = key[i % 4]
        new = old ^ k
        idaapi.patch_byte(ea, new)
```

写脚本解密

- 按照上面说的把所有该 patch 的都 patch 了其实这道题就非常简单了
- 解密脚本:

```
import idc
import idaapi


def unpack_cyzcc():
    f = open("backup", "wb")
    key = b'D0g3'
    start = 0x41e000
    length = 0x1200
    for i in range(length):
        ea = start+i
        old = idc.Byte(ea)
        f.write(bytes(old))
```

```
        k = key[i % 4]
        new = old ^ k
        idaapi.patch_byte(ea, new)


def unpack_charset():
    start = 0x133010
    f = open("charset.bak", "wb")
    for i in range(64):
        ea = i+start
        c = idc.get_wide_byte(ea)
        f.write(bytes(c))
        new_b = ''
        if i >= 26:
            if i >= 45:
                new_b = c+122
            else:
                new_b = c+90
        else:
            new_b = c+57
        idaapi.patch_byte(ea, new_b)
    f.close()


def get_data(start, end, data_type="byte"):
    data = []
    while start < end:
        if data_type == "byte":
            d = idc.Byte(start)
            start += 1
        data.append(d)
    return data


def solve():
    charset = get_data(0x133010, 0x133010+64, "byte")
    fake_flag = get_data(0x1330CC, 0x01330EC, "byte")
    cipher = get_data(0x01330A8, 0x01330c8, "byte")
    plain = []
    for i in fake_flag:
        print(chr(i), end='')
    for i in cipher:
        print(hex(i), end=',')
    for i in charset:
        print(chr(i), end='')

    def get_index(d):
        for i in range(64):
            if charset[i] == d:
                return i
    breakpoint()
    for i in range(len(fake_flag)):
        cipher[i] ^= fake_flag[i]
    breakpoint()
    for i in range(len(cipher)//4):
        charset = charset[1:]+charset[0:1]
        tmp_cipher = cipher[4*i:4*i+4]
        a, b, c, d = [get_index(n)for n in tmp_cipher]
        plain.append(a << 2 & 0xC0 | c << 2 & 0x30 | d >> 2 & 0xC | b & 0x3)
```

```
        plain.append(b << 2 & 0xC0 | a << 2 & 0x30 | d >> 0 & 0xC | c & 0x3)
        plain.append(c << 2 & 0xC0 | b << 2 & 0x30 | d << 2 & 0xC | a & 0x3)
    for i in plain:
        print(chr(i), end='')


if __name__ == "__main__":
    unpack_cyzcc()
    # unpack_charset()
    # solve()
```

- 这题的关键就在于改那几个花指令, 改完之后分析出伪代码, 就非常简单了