Privilege Escalation via a Kernel Pointer Dereference (CVE-2017-18019)

mss**** / 2019-04-23 09:58:00 / 浏览数 4130 安全技术 漏洞分析 顶(0) 踩(0)

原文地址：https://labs.nettitude.com/blog/privilege-escalation-via-a-kernel-pointer-dereference-cve-2017-18019/

不久前，我发现了一个安全漏洞，即CVE-2017-18019，该漏洞影响到了K7 Computing公司以及Defenx公司的多款Windows平台安全产品的内核驱动程序。这两家公司之所以同时受到了影响，因为它们使用了相同的防病毒引擎，不过，现在该漏洞

对于该漏洞的POC来说，它们基于非法的内核指针解引用漏洞，最初只会导致系统蓝屏死机。当时，这项研究是由Securiteam提供赞助的，并且随后的漏洞披露协调工作也

Targeting

本文介绍的漏洞的攻击目标是64位Windows操作系统，即Windows 7 SP1 – Windows 10 v1809。

要想通过本文介绍的方法来利用该漏洞，需要具有中等完整性级别的访问权限。为了从低完整性级别来利用该漏洞，则必须设法泄漏某些内核指针。为此，可以通过目标驱动

Bug Analysis

这个安全漏洞根源在于，易受攻击的函数的作者信任从用户提供的输入缓冲区中获取的、用于读取数据的指针——只要该指针引用的是内核地址空间内的地址即可。

易受攻击的函数从IOCTL的输入缓冲区中获取指针，并检查该指针的值是否大于或等于nt!MmHighestUserAddress（在x64系统中，其值为0x00007fffffffffff）。如果上面

很明显，该项检查的目的（即使实现是错误的）是验证将用于读取更多信息的指针地址位于内核内存中，从开发人员的角度来看，其虚拟地址和内容应该无法被用户看到和操

下面给出了上面描述的内容对应的代码。

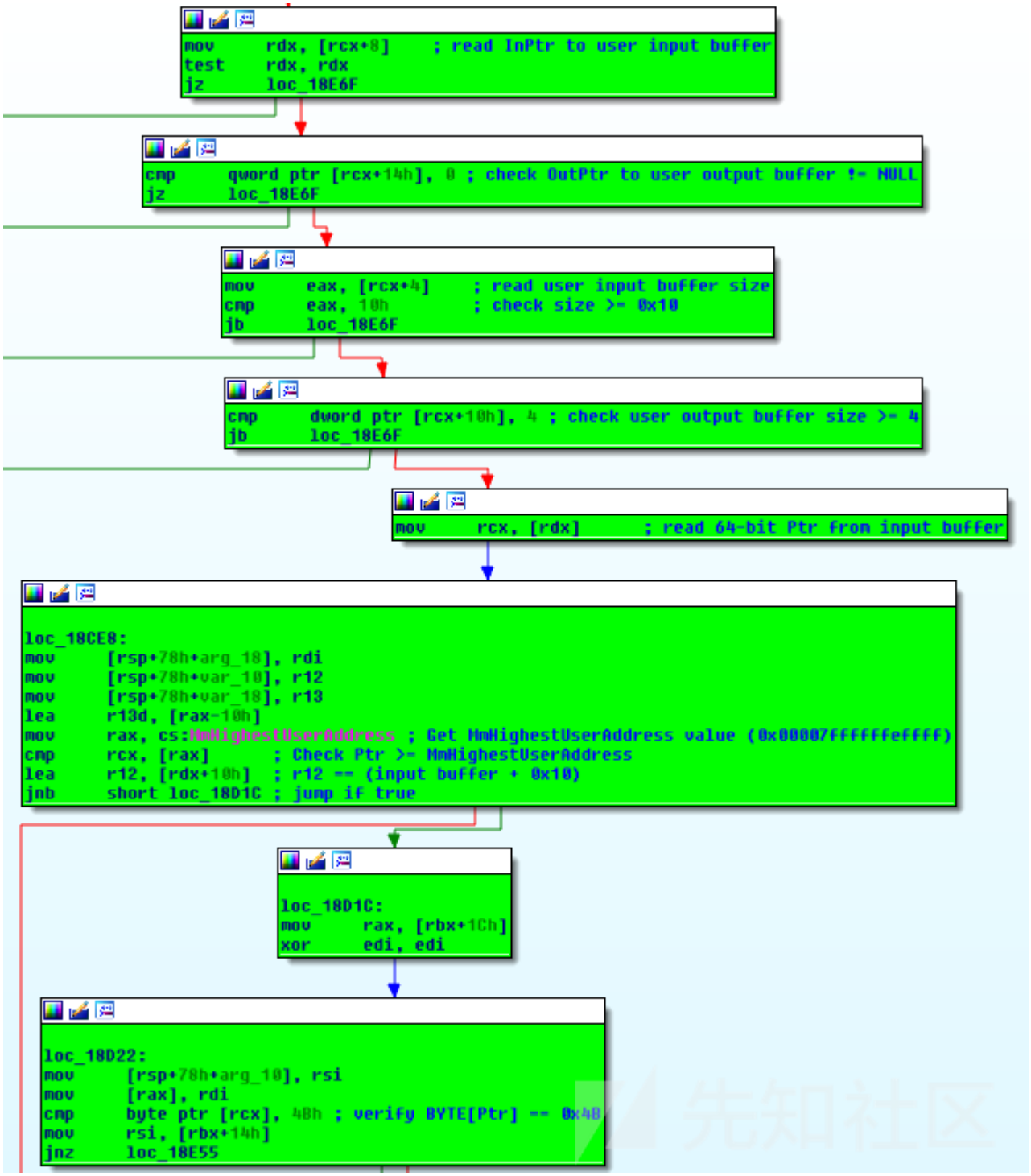

图1：验证是否为内核指针。

只要提供一个引用未分配的内存页的内核指针，就能轻松地让主机崩溃。

下图显示了发生内存非法访问时Windbg的输出内容。

```
FAULTING_IP:
K7Sentry+8d2d
fffff802`79b58d2d 80394b          cmp     byte ptr [rcx],4Bh

CONTEXT:  ffff8a0dc12c2ce0 -- (.cxr 0xffff8a0dc12c2ce0)
rax=ffff9683190fc848 rbx=ffff8a0dc12c3780 rcx=f8f8f8f8f8f8f8f8
rdx=0000009141f0f8c8 rsi=ffff9683190fc848 rdi=0000000000000000
rip=fffff80279b58d2d rsp=ffff8a0dc12c36d0 rbp=ffff96831596fe90
 r8=0000000000000000  r9=ffff968315790730 r10=fffff80279b5c770
r11=ffff8a0dc12c37e8 r12=0000009141f0f8d8 r13=0000000000000010
r14=0000000000000001 r15=0000000000000020
iopl=0         nv up ei pl zr na po nc
cs=0010  ss=0000  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
K7Sentry+0x8d2d:
fffff802`79b58d2d 80394b          cmp     byte ptr [rcx],4Bh ds:002b:f8f8f8f8`f8f8f8f8=??
Resetting default scope
```

图2：对任意内核指针的解除引用。

Further Analysis

到目前为止，我们知道该漏洞可以用来发动拒绝服务攻击：任何用户都可以触发该漏洞，从而使主机发生崩溃。因此，我们可以对这个函数做进一步的分析，看看能否挖掘更

下面展示的是图1后面的代码。

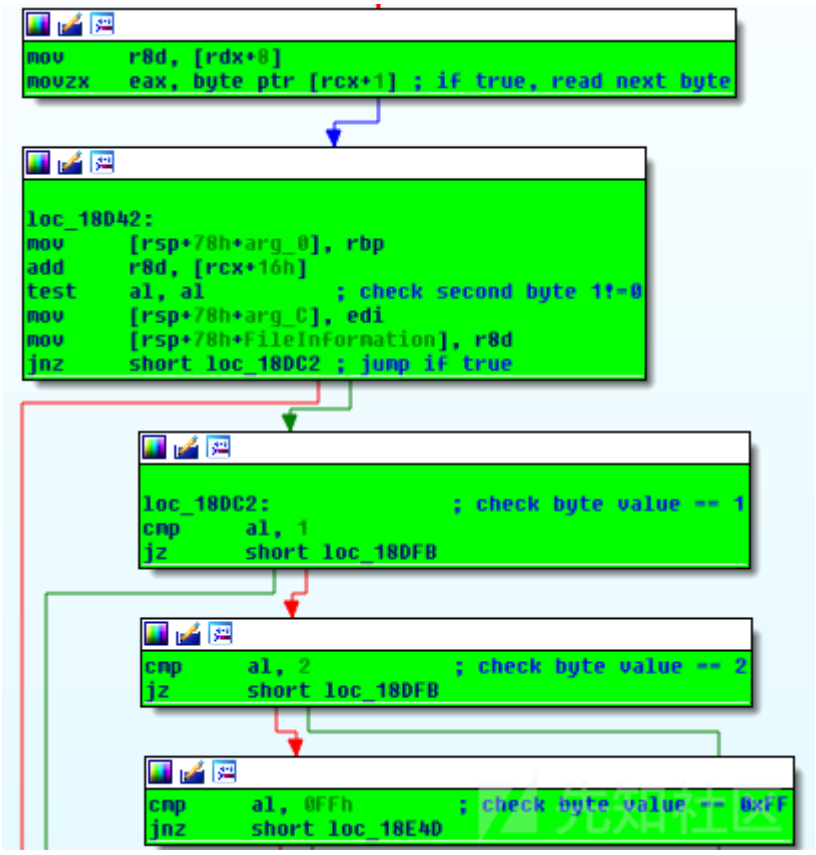

图3：内核内存缓冲区数据检查。

假设RCX指向第一个字节内容为0x4B的有效内核地址，这样的话，前面的检查便可以顺利通过（cmp byte ptr [rcx], 4Bh），这时，我们就能进入易受攻击函数的第二部分，具体如上图所示。

我们注意到，这里又对字节值进行了进一步的检查，因此，为了访问代码的后面部分，必须让RCX引用的缓冲区的第二个字节的值为0xFF。

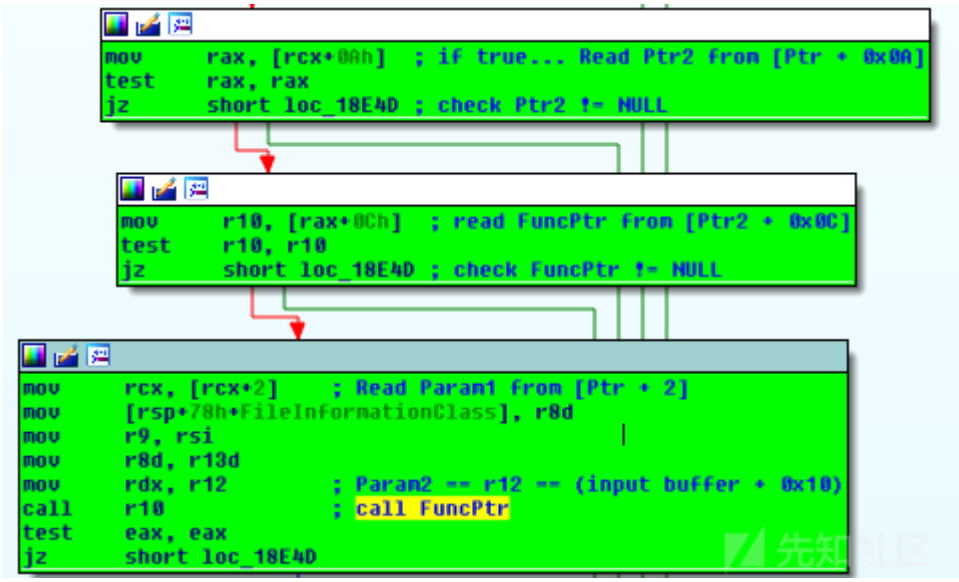

图4：任意函数指针调用。

我们发现，对指针进行相应的解除引用后，该函数会将最后一个指针作为函数指针处理。我们还注意到，传递给RCX和RDX的第一个和第二个参数也是可以控制的。

更具体地说，第一个参数取自处于我们控制之下的任意内核指针所引用的缓冲区，第二个参数指向通过调用DeviceIoControl函数定义的用户输入缓冲区。

Setting things up

为了进一步利用该漏洞，我们必须掌握所需的全部信息。为此，我们必须知道内核对象的地址，并能够在一定程度上控制其内容。如前所述，用于读取其余数据并导致一个函

在前一篇文章中，我们讨论了 Private Namespaces
以及在相关内核对象的主体中插入用户定义数据的方法。我们将使用这种类型的对象来利用这里的安全漏洞，因为就这里来说，这种利用方式也非常稳定。

为了利用该漏洞，我们将使用上述类型的两个内核对象。第一个对象用于控制后续指针解引用，以便让我们可以调用任意函数指针；而第二个对象将用于控制初始内核指针检

Exploitation in Windows 7 SP1 x64

在没有诸如SMEP（管理员模式执行保护）之类的安全防御机制的情况下，这个漏洞利用起来非常简单。我们可以在用户空间中执行我们的payload代码，而无需采取任何额

首先，我们可以使用随机的边界名称来创建一个Private Namespace对象，然后使用NtQuerySystemInformation函数泄漏其地址。

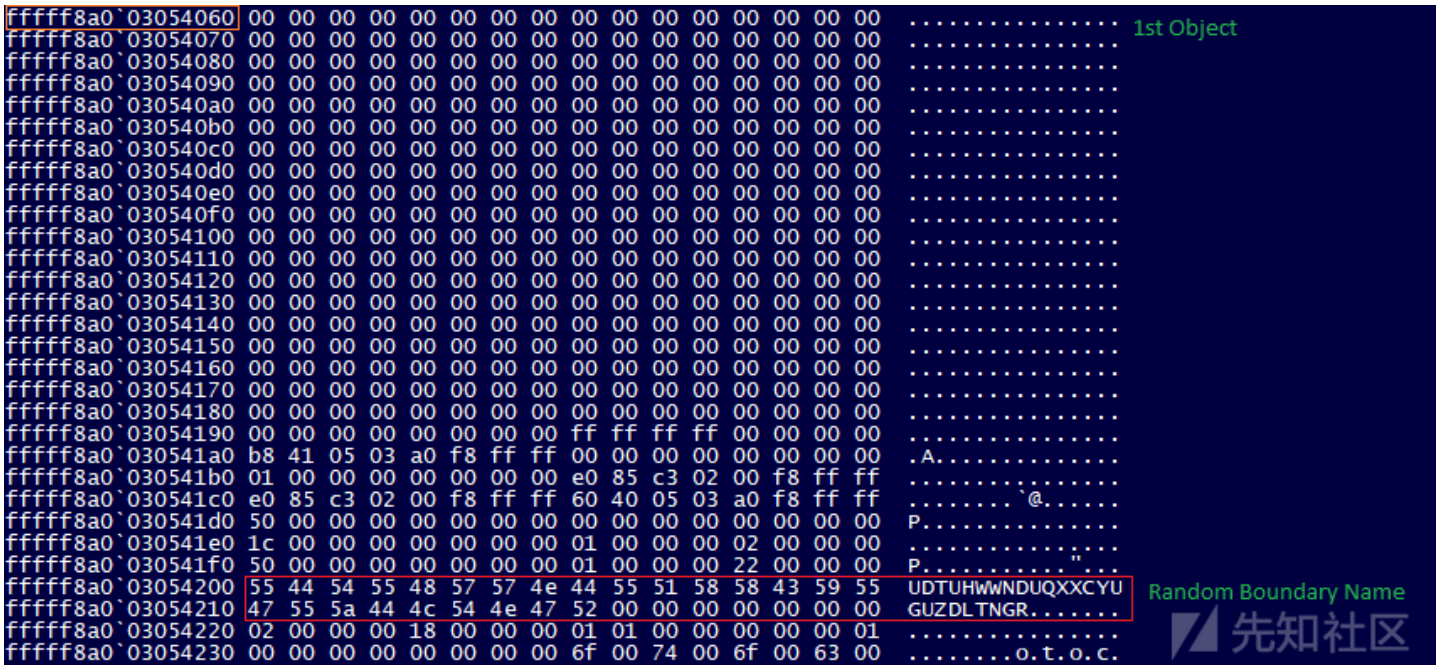

图5：第一个对象（Win7 SP1 x64）。

然后，我们将使用精心设计的边界名称来创建另一个相同类型的对象。

需要注意的是，第一个字节和第二个字节的值必须分别为0x4B和0xFF（见图1和3），以满足字节值检查的要求。此外，在精心设计的边界名称的偏移0x0A（图4：第一个指

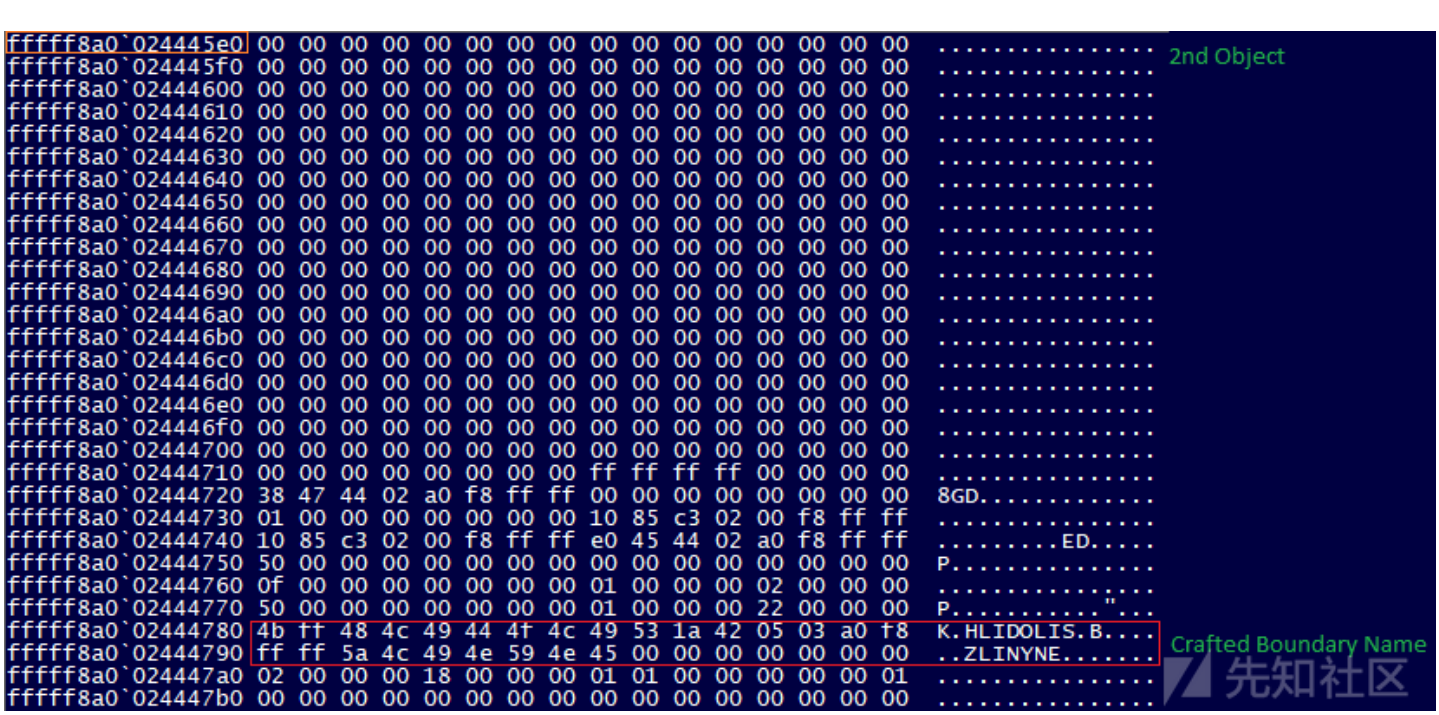

图6：第二个对象（Win7 SP1 x64）。

让我们仔细看看这两个对象是如何"相互连接"的。

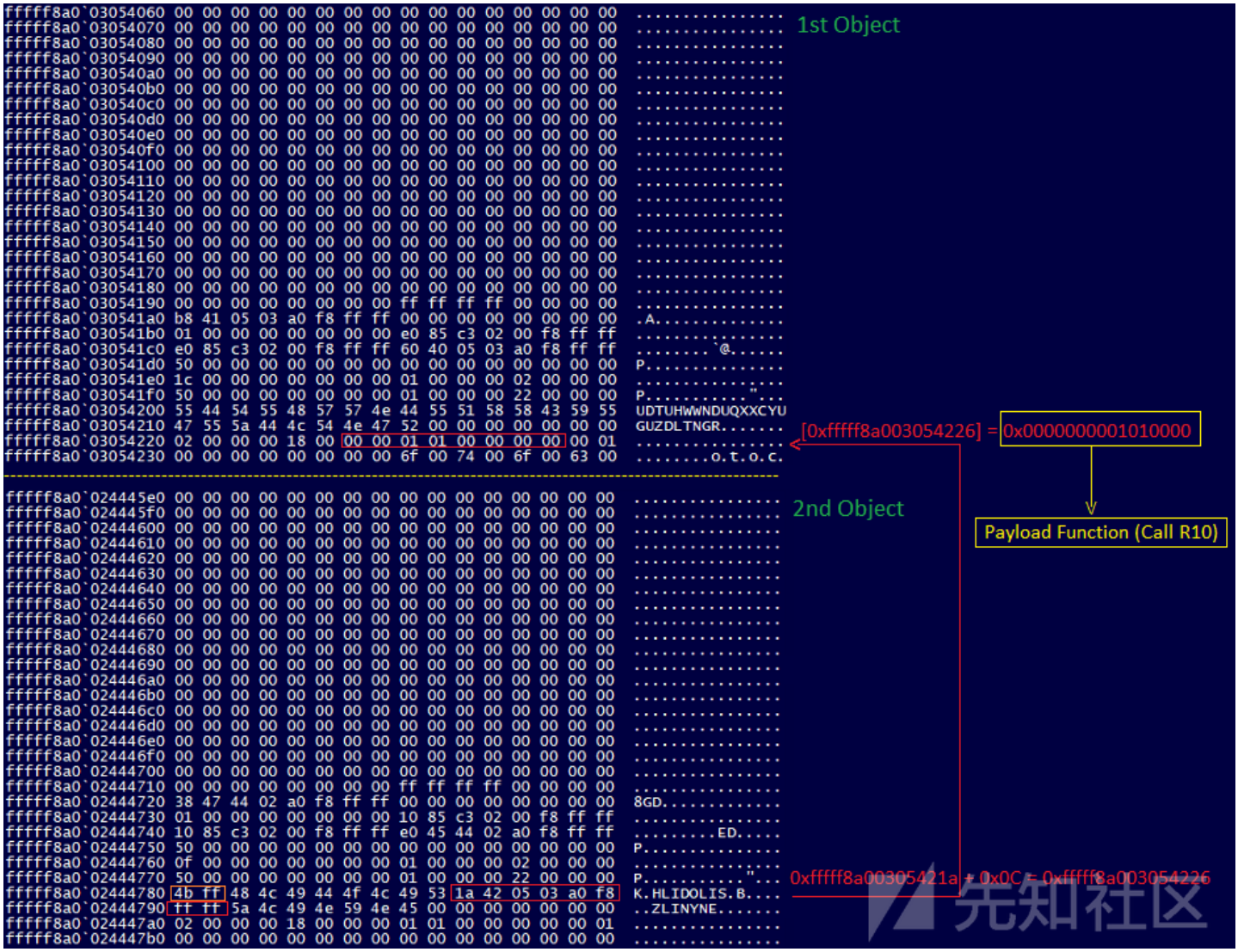

图7.对象互连（Win7 SP1 x64）。

最后，我们可以看到这个函数指针将被调用，从而执行我们在地址0x1010000处的payload代码。

```
fffff880`011aadce 488b410a        mov     rax,qword ptr [rcx+0Ah]
fffff880`011aadd2 4885c0          test    rax,rax
fffff880`011aadd5 7476            je      K7Sentry+0x8e4d (fffff880`011aae4d)
fffff880`011aadd7 4c8b500c        mov     r10,qword ptr [rax+0Ch]
fffff880`011aaddb 4d85d2          test    r10,r10
fffff880`011aadde 746d            je      K7Sentry+0x8e4d (fffff880`011aae4d)
fffff880`011aade0 488b4902        mov     rcx,qword ptr [rcx+2]
fffff880`011aade4 4489442420      mov     dword ptr [rsp+20h],r8d
fffff880`011aade9 4c8bce          mov     r9,rsi
fffff880`011aadec 458bc5          mov     r8d,r13d
fffff880`011aadef 498bd4          mov     rdx,r12
fffff880`011aadf2 41ffd2          call    r10 {00000000`01010000}
fffff880`011aadf5 85c0            test    eax,eax
fffff880`011aadf7 7454            je      K7Sentry+0x8e4d (fffff880`011aae4d)
```

图8：调用Payload函数指针（Win7 SP1 x64）。

Exploitation in Windows 8.1 – 10 v1809 x64

在最新版本的Windows系统中，由于已添加了相应的漏洞利用缓解措施，所以，利用内核驱动程序的漏洞会非常棘手。在这种情况下，我们可以通过调用任意函数来控制执

一个常见的解决方案是，尝试通过清除特定处理器的CR4寄存器中的第20位来暂时禁用SMEP，并将我们的线程执行锁定为仅在该处理器上运行，以便我们可以像以前一样在Patch Protection/PatchGuard）杀死主机，我们必须恢复CR4。

接下来，我们要使用的另一种方法是利用执行流控制将其转换为“write-what-where”原语，一旦实现了这一点，我们就能够修改内核内存中的任意数据，从而可以继续借助

第一种方式是使用NULL值覆盖作为SYSTEM运行的提权进程的对象头部中的SD（安全描述符）指针。这将允许非特权进程在同一安全上下文中注入并执行恶意代码。但是，10 v1511（Build 10586）及更早的版本，详情请参考这篇文章。

利用“write-what-where”原语的另一种方法，是在非特权进程的主令牌中启用特权，以使其能够在作为SYSTEM运行的进程的安全上下文中注入和执行代码。目前，该方法仅
10 v1709（Build 15063）之后的版本来说，需要稍作修改，详情请参考这篇文章。我们在这里要描述的内容同样适用于Windows 7。

此外，一旦我们的任意函数被调用，我们就能够控制在RCX和RDX中传递的前两个参数（参见图4）。这一点，马上就会在后面用到。

在这种情况下，我们首先需要泄漏进程的主令牌的地址，以启用其他权限。我们将使用该地址作为漏洞利用原语的目标。与Windows
7中一样，NtQuerySystemInformation可以利用用户进程的标准“中等完整性”权限完成同样的事情，以泄漏所需的内核对象和函数地址。

然后，我们将创建具有自定义边界名称的第一个Private
Namespace对象，其中前8个字节将设置为用于修改任意内核数据的“gadget”的内核地址。因此，我们不是在用户地址空间中执行payload，而是将执行流程重定向到内核

```
nt!RtlCopyLuid:
fffff805`035956e0 488b02          mov     rax,qword ptr [rdx]
fffff805`035956e3 488901          mov     qword ptr [rcx],rax
fffff805`035956e6 c3              ret
```

图9：内核函数nt!RtlCopyLuid。

由于我们能够控制RCX和RDX寄存器，因此，我们可以使用这个函数来实现“write-what-where”原语。

同样，我们还需要第二个Private
Namespace对象，其自定义边界名称在名称数据的偏移量0x0A处（图8：第一个指针取消引用）必须包含第一个对象的地址+该地址与边界名称在该对象中的位置之间的、
Namespace对象的边界名称的前8个字节中插入了nt!RtlCopyLuid函数的地址。请注意，和以前一样，我们必须考虑上一次计算的结果，所以要加上值0x0C，使其指向任意

所以，它看起来应该是这样的：

```
*(ULONG_PTR*)(boundaryName + 0x0A) = customPrivNameSpaceAddress + boundaryNameOffsetInDireObject - 0x0C;
```

此外，我们还需要控制前两个参数。

加载到RCX中第一个参数是从我们的自定义边界名称中读取的，偏移量为2（自定义边界名称的前两个字节必须为0x4B和0xFF）。因此，我们需要将其设置为进程令牌对象的

```
kd> dt nt!_SEP_TOKEN_PRIVILEGES
+0x000 Present           : Uint8B
+0x008 Enabled           : Uint8B
+0x010 EnabledByDefault  : Uint8B
```

图10：nt!_SEP_TOKEN_PRIVILEGES结构成员。

它应该如下所示：

```
*(ULONG_PTR*)(boundaryName + 0x02) = tokenAddress + 0x40;
```

最后，我们还可以控制RDX，因为R12的值被copied
over，它指向我们的用户地址空间的输入缓冲区地址+0x10处（参见图1中的第6个节点）。这是我们从中读取数据，写入任意内核地址的地方。就这里来说，我们将覆盖上

它应该如下所示：

```
*(unsigned __int64*)(inputBuf + 0x10) = _ULLONG_MAX;
```

因此，我们的漏洞利用代码必须两次reach易受攻击的函数才能完成攻击。

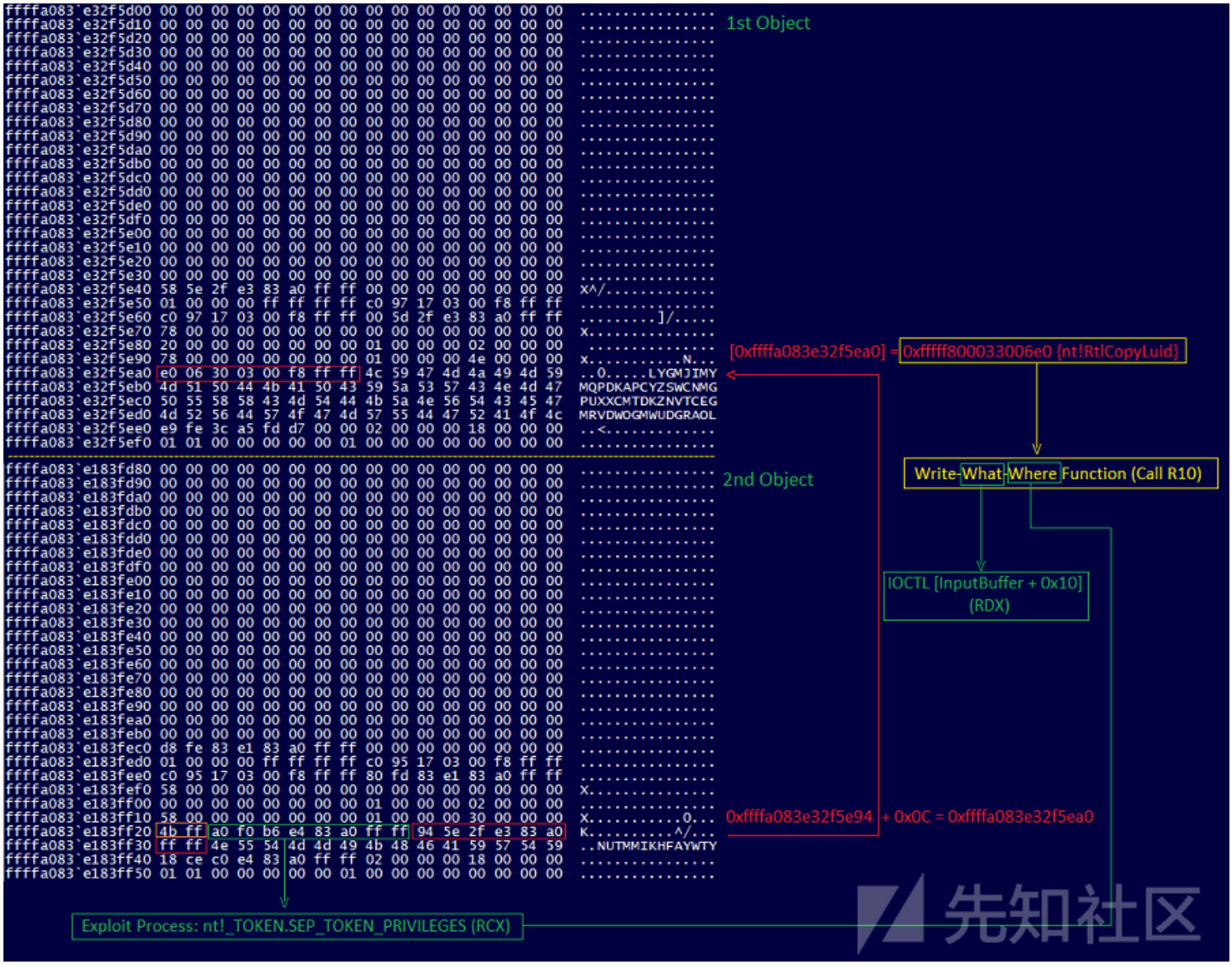

图11：对象互连——Write-What-Where。

上图显示了两个对象如何"互连"以实现"write-what-where"原语，进而实现漏洞利用。

Conclusion

本文为读者介绍了一个非常有趣和非常值得研究的安全漏洞，它再次表明，任何输入数据都不应该被盲目地信任。从开发人员的角度来看，信任用于从用户无法控制的地方读