SKREAM（一）:内核模式下的提权缓解措施

websni****@outlo / 2018-11-05 09:20:00 / 浏览数 2438 技术文章 技术文章 顶(0) 踩(0)

背景

攻击者们在利用内核漏洞时的主要目的是从较低的权限提升到较的高权限或是系统级别的权限。这种攻击类型的通常被称为LPE(本地权限升级)，无论是在NTOSKRNL本身还

虽然微软在防御这些漏洞方面做得很好，但仍有很大的改进空间。我们启动了一个新的开源项目，名为SKREAM
(SentinelOne's KeRnel Exploits Advanced
Mitigations)。这个项目将会有很多自己的特点，用于检测或缓解内核开发生命周期的不同类型/阶段的风险。目前它只有一种缓解措施，后续我们会添加更多的，敬请关注。

在这篇文章中，我们将探讨SKREAM首次引入的缓解措施。这种缓解措施有效的解决了一种特定的攻击技术，也就是pool overflow vulnerabilities
中的溢出漏洞，并使其在Windows 7和8系统上的利用失效。

内核溢出简介

内核溢出是众所周知的一类漏洞，过去几年LPE漏洞也被广泛使用。内核模式下驱动程序将用户输入的数据复制到内存池分配时，无需验证其大小就可以被利用。这允许攻击

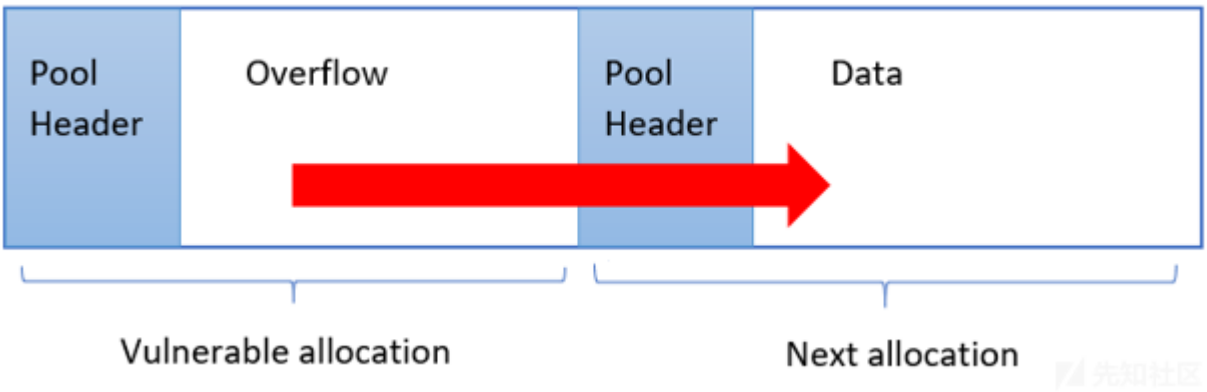

图1:漏洞示意

结合内核碰撞技术，可以预测以下池分配的内容，从而允许攻击者覆盖内存池。

TypeIndex覆盖

实际上有多种方法可以利用内存溢出漏洞。本文中主要研究通过覆盖每个`OBJECT_HEADER`结构中的类型索引成员来实现攻击。

正如numerous
sources（源记录）里的那样，Windows对象管理器分配的每个对象都有一个描述它的对象标头，它紧跟在内存中相应的池标头之后。这个对象头包含一个名为"TypeIndex

```
0: kd> dt nt!_OBJECT_HEADER
   +0x000 PointerCount     : Int8B
   +0x008 HandleCount      : Int8B
   +0x008 NextToFree       : Ptr64 Void
   +0x010 Lock             : _EX_PUSH_LOCK
   +0x018 TypeIndex        : UChar
   +0x019 TraceFlags       : UChar
   +0x01a InfoMask         : UChar
   +0x01b Flags            : UChar
   +0x020 ObjectCreateInfo : Ptr64 _OBJECT_CREATE_INFORMATION
   +0x020 QuotaBlockCharged : Ptr64 Void
   +0x028 SecurityDescriptor : Ptr64 Void
   +0x030 Body             : _QUAD
```

图2：`OBJECT_HEADER`结构的内存内布局。圈起来的是TypeIndex成员。

`nt!ObTypeIndexTable`是一个`OBJECT_TYPE`的结构数组，每个结构都描述了Windows上可用的众多对象类型之一(进程、事件、桌面等)。`OBJECT_TYPE`结构向Window

```
0: kd> dt nt!_OBJECT_TYPE TypeInfo.*
   +0x040 TypeInfo  :
      ...
      +0x030 DumpProcedure : Ptr64        void
      +0x038 OpenProcedure : Ptr64        long
      +0x040 CloseProcedure : Ptr64        void
      +0x048 DeleteProcedure : Ptr64        void
      ...
```

图3：每个OBJECT_TYPE都实现了一些方法

结果是，nt!ObTypeIndexTable数组的前两条目实际上是伪条目，它似乎没有指向实际的OBJECT_TYPE结构。第一个条目包含一个空指针，第二个条目包含神奇的常量0

```
0: kd> dps nt!ObTypeIndexTable
fffff800`02a29d40  00000000`00000000
fffff800`02a29d48  00000000`bad0b0b0
fffff800`02a29d50  fffffa80`0186bf30
fffff800`02a29d58  fffffa80`0186bde0
fffff800`02a29d60  fffffa80`0186bc90
fffff800`02a29d68  fffffa80`0186b940
fffff800`02a29d70  fffffa80`01909f30
fffff800`02a29d78  fffffa80`01909de0
```

图4:nt!ObTypeIndexTable数组前的两个伪条目

在x64架构上这两个值都是由0扩展到普通用户模式地址的，这无疑为攻击者提供了一种使用内核级权限实现代码执行的可乘之机。即挖掘溢出漏洞，攻击者可以:

1. 分配0xbad0b0b0页面并为其伪造一个OBJECT_TYPE结构。这个伪对象类型将包含指向攻击者的用以提升权限的代码的函数指针(在Windows 7 x86上，我们也可以为此分配空页面，在新版本Windows上无效)。
2. 在内存池中碰撞已知类型和大小的对象。以确保攻击者知道溢出时所分配的内容。
3. 释放部分对象，在内存池中挖掘"漏洞"。对于攻击者来说，最理想的情况是，溢出分配将出现在这些"漏洞"中。
4. 触发此漏洞，并且溢出到下一个对象，销毁其OBJECT_HEADER并将其TypeIndex属性设置为1。
5. 对溢出对象触发一些操作(例如关闭句柄)。会导致系统从0xbad0b0b0获取OBJECT_TYPE并调用它的一个方法(示例中是CloseProcedure)。由于这个函数指针是由攻击

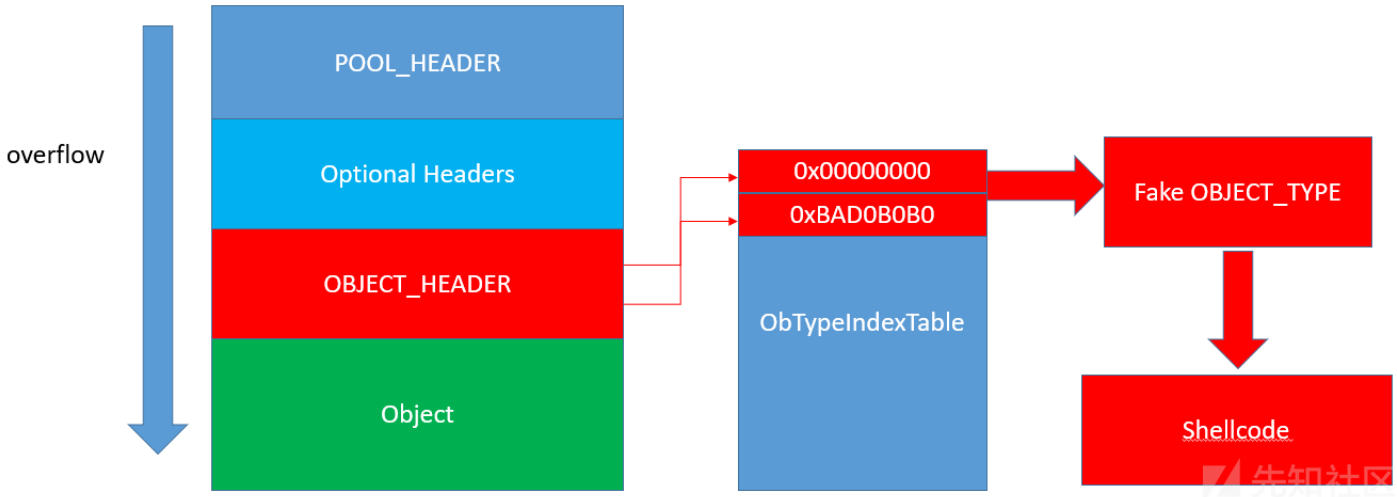

图5 内存池溢出可视化，来源Nikita Tarakanov

这项技术是由Nikita Tarakanov 首发于这儿。

缓解措施

这里我们提出的缓解措施主要是通过预先分配包含0xbad0b0b0的内存区域，以便在任何攻击代码想利用它之前在每个进程的基础上阻止这种攻击。为了让措施尽可能的有效

早在2011年，Tarjei Mandt就进行了一个类似的项目，他演示了在Windows
7系统上保护空页不受空页反引用攻击的手法。为此他还编写了一个内核驱动程序，该驱动程序使用了一组VAD(Virtual Address Descriptors
虚拟地址描述符)操作技术来手动为空页面构建VAD条目，然后将其作为“合成”叶状条目插入到VAD树中。

我们要做的事情同理，但希望以一种更“有机”的方式来做，即尽可能的将卸载工资交给Windows虚拟内存管理器来做，从而避免问题复杂化。通过分配包含0xbad0b0b0的

```
ULONG MM_ALLOCATION_GRANULARITY = 0x10000;
ULONG_PTR OBJECT_TYPE_BAD0B0B0 = 0x00000000bad0b0b0;

baseAddress = ALIGN_DOWN_POINTER_BY(OBJECT_TYPE_BAD0B0B0, MM_ALLOCATION_GRANULARITY);
regionSize = MM_ALLOCATION_GRANULARITY;

ZwAllocateVirtualMemory(hProcess, &baseAddress, 0, &regionSize, MEM_RESERVE, PAGE_NOACCESS);
```

紧接着我们继续检索刚才创建的VAD条目，以便进一步编辑它来满足保护内存的范围。我们借鉴了Blackbone
library库中的BBFindVad函数，在不同的Windows版本中都可以轻松实现了这个功能。

不幸的是从VAD树中检索到的VAD条目是MMVAD_SHORT类型的，我们需要的VAD是MMVAD_LONG类型。我们感兴趣的内容似乎在MMVAD_LONG结构中，而不是在MMVAD_SH

```
0: kd> !vad fffffa8002ff0e40 1

VAD @ fffffa8002ff0e40
  Start VPN               bad00  End VPN           bad0f  Control Area  0000000000000000
  FirstProtoPte 0000000000000000  LastPte 0000000000000000  Commit Charge                0 (0n0)
  Secured.Flink                0  Blink                 0  Banked/Extend                0
  File Offset                  0
      ViewUnmap PrivateMemory NO_ACCESS
```

图6 VAD描述了在被保护之前包含0cbad0b0b0的内存区域

为了解决这个问题，我们分配了自己的MMVAD_LONG结构并初始化。事实证明，每个MMVAD_LONG结构中都有一个MMVAD_SHORT子结构嵌在其中，因此我们可以复制前面检

```
PMMVAD_SHORT shortVad = nullptr;
BBFindVAD(processObject, OBJECT_TYPE_BAD0B0B0, &shortVad);

PMMVAD_LONG longVad = nullptr;
longVad = ExAllocatePoolWithTag(NonPagedPool, sizeof(*longVad), LONG_VAD_POOL_TAG);
RtlZeroMemory(longVad, sizeof(*longVad));
longVad->vad.vadShort = *shortVad;
```

下一步就是编辑VAD标志使其安全一些。根据Tarjei的论文所叙述，我们所做的修改最终归结为:

```
longVad->vad.vadShort.u.VadFlags.CommitCharge = MM_MAX_COMMIT;
longVad->vad.vadShort.u.VadFlags.NoChange = TRUE;

longVad->vad.u2.VadFlags2.OneSecured = TRUE;
longVad->vad.u2.VadFlags2.LongVad = TRUE;

longVad->u3.Secured.u1.StartVa = longVad->vad.vadShort.StartingVpn << PAGE_SHIFT;
longVad->u3.Secured.EndVa = ((longVad->vad.vadShort.EndingVpn + 1) << PAGE_SHIFT) - 1;
```

这些变化使VAD变成这样:

```
0: kd> !vad fffffa80035fd010 1

VAD @ fffffa80035fd010
  Start VPN                bad00  End VPN            bad0f  Control Area  0000000000000000
  FirstProtoPte 0000000000000000  LastPte 0000000000000000  Commit Charge fffffffffffffff (0n-1)
  Secured.Flink         bad00000  Blink           bad0efff  Banked/Extend                 0
  File Offset                  0
      ViewUnmap NoChange PrivateMemory NO_ACCESS
      OneSecured
```


图7 VAD描述了被保护后包含`0cbad0b0b0`的内存区域

最后，我们需要用新的`MMVAD_LONG`替换VAD树中的`MMVAD_SHORT`。简而言之，这涉及到三个阶段的操作:

1. 设置短VAD子节点的父节点指向我们的`MMVAD_LONG`条目。
2. 设置短VAD的父节点中适当是我子节点(左或右)以指向我们的`MMVAD_LONG`条目。
3. 释放不再被VAD树引用的短的VAD条目。

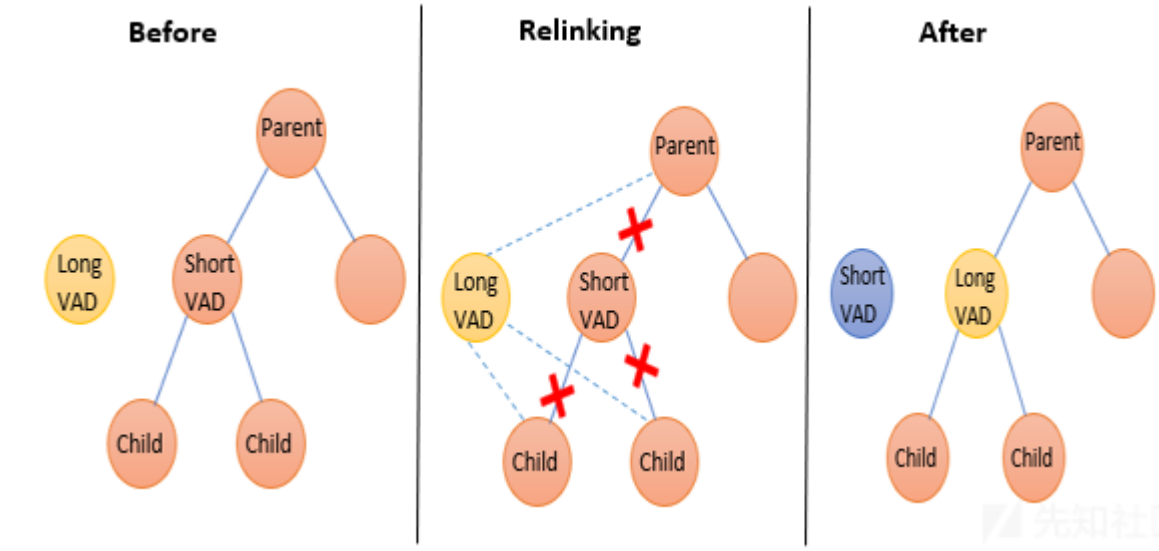

图8 用`MMVAD_LONG`替换VAD树中的`MMVAD_SHORT`

效果展示

KdExploitMe中SKREAM缓解池溢出漏洞利用
https://www.youtube.com/embed/NHG4v_g3Fi8

Windows 8

在Windows 8上，微软对各种VAD结构做了一些细微的修改，我们也需要对代码基进行了一些调整:

1. 修改中包括不再有`MMVAD_LONG`结构，只有`MMVAD_SHORT`和`MMVAD`。

   我们在Windows 7上设置的标志要么改变了它们在`MMVAD`结构中的位置(`NoChange`, `StartVA`, `EndVA`)，要么根本不存在(`OneSecured`)。

   在考虑了这些因素后，针对Windows 8的代码调整如下:

```
PMMVAD longerVad = nullptr;
longerVad = ExAllocatePoolWithTag(NonPagedPool, sizeof(MMVAD), LONG_VAD_POOL_TAG);
RtlZeroMemory(longerVad, sizeof(*longerVad));
longerVad->Core = *shortVad;

longerVad->Core.u.VadFlags.NoChange = TRUE;
longerVad->Core.u1.VadFlags1.CommitCharge = MM_MAX_COMMIT;

pVadEventBlock->SecureInfo.u1.StartVa = longerVad->Core.StartingVpn << PAGE_SHIFT;
pVadEventBlock->SecureInfo.EndVa = ((longerVad->Core.EndingVpn + 1) << PAGE_SHIFT) - 1;
longerVad->Core.EventList = pVadEventBlock;
```

Windows 8.1或者其他版本

本文中提到的技术仅适用于Windows 7和Windows
8系统。从win8.1开始`nt!ObTypeIndexTable[1]`不再指向`0xbad0b0b0`，而是nt!MmBadPointer的值，释放引用时会冲突。另外，在Win10中，存储在`OBJECT_HEADE`

原文：https://www.sentinelone.com/blog/skream-kernel-mode-exploits-mitigations-rest-us/