(Win7) PspCidTable遍历进程句柄表，枚举进程

转载

xlaomlng 于 2019-06-14 16:35:43 发布 790 收藏 1

先说一下句柄表是什么

在windows内核中定义了很多内核对象，像文件对象，线程对象，信号量对象啊等等；而Windows中并没有让我们直接去使用这些对象资源，而是通过句柄让我们去引用这些资源，句柄表就相当于资源的数组，而句柄就是对应资源的索引，每个进程内部都有一个自己的私有句柄表，这也就是说句柄是不能跨进程使用的。

PspCidTable句柄表

PspCidTable也是一个句柄表，但是这个句柄表不是私有句柄表，它里面存放的是系统中所有的进程和线程对象,其索引也就是进程ID(PID)或线程ID(TID).

在Win2000中，句柄表使用的是固定的3层索引模式，而在WinXp，2003以及之后的系统中使用的动态的可扩展的3层索引句柄表，也就是说句柄表的层数是不确定的，他会根据对象的数量进行动态调整.(说到多层的索引模式，这里我觉得还是要先举个形象的例子来说明比较好，比如图书馆的书，如果只有10本书，我们需要什么直接拿就是了，这就是1层索引； 后来书变得比较多了，这个时候就得分类了，查找一本书我们得知道这本书的书架号，找到对应的书架，查找数的号码才能找到这个本书，这便是2层索引； 再到后来由于书太多了 需要分好几间书库来存放，这个时候我们找一本书就得知道这个书在哪个书库放着，书架号是多少，书的编号是多少，这样我们才能找到对应的书，这也便是3层索引模式.)

好了，我们言归正传，开始通过PspCidTable句柄表来查找进程吧，如果你还不知道如何获取到PspCidTable的值，请参见这篇文章《Windows内核PspCidtable表地址的获取》 ,打开我们的WinDbg，执行命令：

kd> dd Pspcidtable

可以看到地址 0x83F59F34中存放的内容是 0x8d8010a8,这货就是是一个_HANDLE_TABLE的结构

我们在使用命令dt _HANDLE_TABLE 8d8010a8看看这货的内容

其中TableCode这个成员，我们可以认为它是一个指向句柄表的地址，其中这个数值的低2Bit表示的是句柄表的层数，所以我们实际得到的句柄表的地址是要掩掉低2位的，也就是 TableCode&~0x3 ，其中低两位 为0时表示 1层索引，为1时表示2层索引，为2时表示3层索引，最后我们索引到的是一个_HANDLE_TABLE_ENTRY的结构，这个结构里面有我们要的_EPROCESS地址。引用一张看雪的图，来描述3层索引的结构:

```
对于每一个索引表大小都为1页 4KB，其中一级表存放的是8Byte的_HANDLE_TABLE_ENTRY的结构，所以每一个1级表就只能存放512个项；2级表存放的是1级表的地址(4Byte)那么每一个2级表能够存放4KB/4B = 1024个1级表的地址，如果存在3级表的话，这个数目将会更大.



下面我们通过WinDbg手动找到一个_HANDLE_TABLE_ENTRY看看结果,上面我们知道TableCode为0x951a1001，低2Bite为01 ，可以知道当前是2层索引结构，其中句柄表的地址为0x951a1000，我们通过命令dd 0x951a1000可以看到对应的2级索引表：
```

可以看到就只有两项，也就是说有两个1级表，那么当前的句柄表能够容纳512*2=1024个句柄.

```
我们通过地址0x8d804000访问到第一个1级索引表来看看dd 8d804000
```

我们在1级表中看到了好多8Byte的_HANDLE_TABLE_ENTRY,他的结构是这样子的：

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
typedef struct _HANDLE_TABLE_ENTRY
{
<span id="11_nwp"style=“width: auto; height: auto; float: none;”><a id="11_nwl"style="text-decoration:
none;"href=“http://cpro.baidu.com/cpro/ui/uijs.php?
app_id=0&c=news&cf=1001&ch=0&di=128&fv=0&is_app=0&jk=d58860036fb270fd&k=union&k0=union&kdi0=0&luki=9&n=10&
p=baidu&q=80007180_cpr&rb=0&rs=1&seller_id=1&sid=fd70b26f36088d5&ssp2=1&stid=0&t=tpclicked3_hc&tu=u1925163&u=
http%3A%2F%2Fwww%2Eyouranshare%2Ecom%2Fblog%2Fsid%2F102%2Ehtml&urlid=0"target=”_blank"mpid=“11”>union
{
PVOID Object;
UINT32 ObAttributes;
struct _HANDLE_TABLE_ENTRY *InfoTable;
UINT32 Value;
};
union
{
UINT32 GrantedAccess;
struct
{
USHORT GrantedAccessIndex;
USHORT CreatorBackTraceIndex;
};
INT32 NextFreeTableEntry;
};
} HANDLE_TABLE_ENTRY, *PHANDLE_TABLE_ENTRY;
```

也就是说8Byte的前4Byte是一个Object对象，这也就是我们要找的_EPROCESS指针，但是这里值得一说的是，句柄表中的Object指针的低3Bit 是有另外意义的:

①第0位OBJ_PROTECT_CLOSE，表示调用者是否允许关闭该句柄;

②第1位OBJ_ INHERIT，指示该进程所创建的子进程是否可以继承该句柄，即是否将该句柄项拷贝到子进程的句柄表中;

③第2位OBJ_ AUDIT_OBJECT_ CLOSE。指示关闭该对象时是否产生一个审计事件。

所以我们在使用该指针的时候要掩掉低3Bit

也就是说对于Object=86ae88a9应该变为Object=0x86ae88a9&~0x07 = 0x86ae88a8,这才是我们要的_EPROCESS 的地址；

在开头我们说过句柄是在句柄表中的索引，也就是说我们PID或者CID也就是对应索引的值，但是在Windows中Handle值的低2Bit是来表示这个句柄位于句柄表中的层次的，所以我们实际的句柄值是不能占用低2位的，所以我们的实际的索引值0，1，2，3 都应至少向左移位2Bit 把低2位给空闲下来，Windows定义了一个最小的移位HANLDE_VALUE_INC 为4，也就是把实际的索引号都乘以4，这样我们实际索引号为0，1，2，3，4，5…对应的句柄索引就变味了0，4，8，12，16，20…

所以在上面途中第2个红色方框的_TABLE_HANDLE_ENTRY的索引应当为1x4 = 4,也就是这个进程(或线程)的ID为 4(自己看看任务管理器PID为0的IDEL不是真正存在的，PID为4的进程是不是System)；我们来验证一下看看：

将0x86ae88a9 低3位掩掉变为0x86ae88a8，来看看这个对象是什么类型:

kd> !object 86ae88a8

可以知道这个对象是一个Process，那么我们用命令

kd> dt _EPROCESS 86ae88a8

验证一下这个EPROCESS是不是PID为4的System进程:

如图可以看到，这确实是进程PID为4的System进程.

回过头来，我们再看看_HANDLE_TABLE中的一个叫做NextHandleNeedingPool的成员，如图所示:

这个成员描述了下次句柄表增长的时，起始的句柄值（别忘了句柄是以4为步长增长的）,上面的分析我们知道我的系统有2个2级索引那么最多能描述5122=*1024个_HANDLE_ENTRY,也就是说最大能表示的句柄值为10244*=4096=0x1000,因为是从0x00开始的，所以当前的索引表状态能够描述的最大句柄上限为0x1000，这个值也就是下次句柄表扩展的起始句柄值.

自己动手通过PspCidTable和进程的PID找到进程对应的_EPROCESS进行验证
说了这么多，我们来动手亲自通过一个进程的PID找到他的_EPROCESS进行验证一下吧。我以一个notepad.exe记事本程序为例子.

```
    如图，记事本的PID为3708(十进制)，应当位于第3708/4=927 个表项，我们每一个1级索引表能容纳512个表项，毋庸置疑，PID3708 应该在第2个二级索引指向的1级索引的第927-512=415=0x19F个表项(每个表项8Byte)，所以我们找到第二个2级索引指向的1级索引的地址：




    地址为0x95193000，那么我们要找的notepad.exe的表项地址为: 0x95193000+0x19F*0x8,所以：
```

我们找到的_EPROCESS地址为0x88d2a031,掩掉低3位变为0x88d2a030:

可以看到PID为0x00000e7c,也正是3078，并且ImageFileName也是notepad.exe没错吧.

如何判断一个表项是否为一个进程呢？
我们已经知道，在句柄表中存放的是一个个的_HANDLE_TABLE_ENTRY表项，句柄表的索引是PID或者TID；也就是说我们查找到的_HANDLE_TABLE_ENTRY中的Object对象有可能是一个_EPROCESS(进程)，也有可能是一个_ETHREAD(线程)对象，对于每一个Object对象都会有一个_OBJECT_HEADER，在这个结构中有一个TypeIndex的属性，其描述了该对象Object的类型，所以我们只需要查看每一个Object的TypeIndex属性即可知道对于一个Object是进程还是线程了，在Win7中_OBJECT_HEADER位于Object上0x18字节；

我们用WinDbg去看一下，已知System进程的_EPROCESS地址为0x86ae88a8,我们偏移0x18字节查看一下：

kd> dt _OBJECT_HEADER 0x86ae88a8-0x18

可以看到TypeIndex为0x7，这个0x7就是进程对象的意思，对于更多的TypeIndex类型，你可以参见这篇文章
http://www.cnblogs.com/unixstudio/archive/2012/11/09/2762906.html

总结
OK说了这么多我们来总结一下流程：

①获取到PspCidTable的地址，根据Tablecode低2位判断句柄表的层数。

②遍历句柄表：只有一级句柄表才是_HANDLE_TABLE_ENTRY(8字节),二级和三级都是指针(4字节），每一个表都是1页(4KB)大小，遍历要注意范围。

③获取到Object之后，要通过ObjectHeader的TypeIndex看看是不是Process.