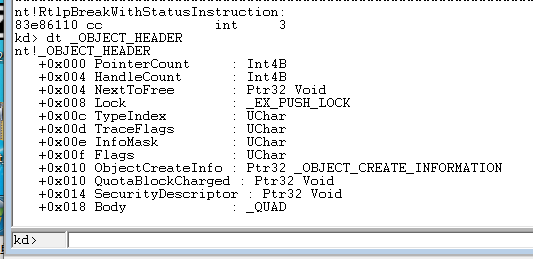
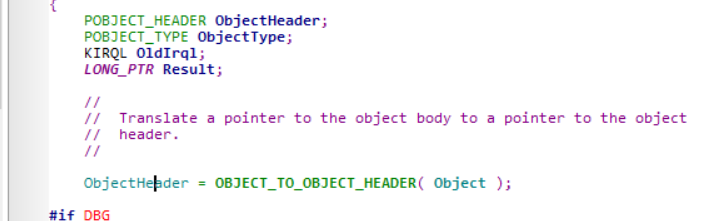
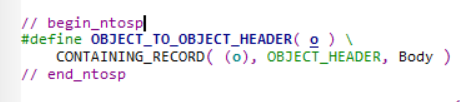
# 对象头OBJECT\_HEADER



今天讲的全局句柄表里有两种对象——进程和线程。内核对象有对象头OBJECT\_HEADER，这块内存在对象的上面，我们可以看看ObfDereferenceObject减少引用计数时是怎么做的：



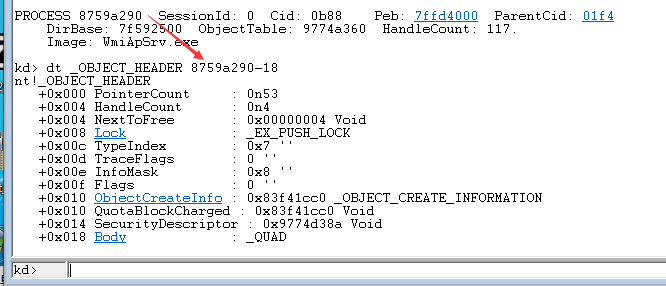


CONTAINING\_RECORD 这个宏经常在链表节点找对象时用到，作用是 o 这个内存是 OBJECT\_HEADER这个类里边的Body成员，来反查这个类的首地址。观察 \_OBJECT\_HEADER 发现 Body成员在 +18h的位置。换句话说，在x86环境中，要通过对象找对象头，只需首地址-18h即可。

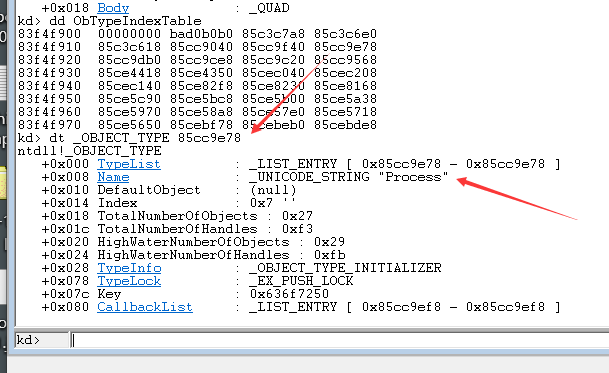
\_OBJECT\_HEADER的PointerCount就是引用计数，还有一个比较关注的字段是 TypeIndex，可以通过它获取这个对象的类型。

### ObGetObjectType 和 TypeIndex

随便找一个EPROCESS,看看他的对象头：

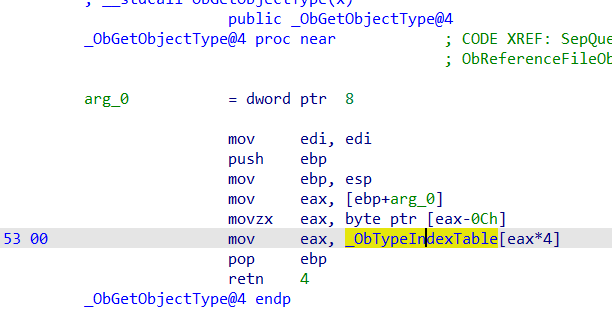


TypeIndex 是 0x7，从WIN7开始，这个字段表示一个索引号，有一个未导出数组叫 ObTypeIndexTable，里面每项都是一个对象类型结构OBJECT\_TYPE.我们去windbg看看7是什么类型：



Name字段表明这是一个进程对象。

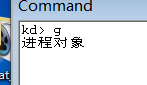
未文档化函数 ObGetObjectType 使用了这个未导出数组：



参数是对象，存到eax后-0ch就是TypeIndex，然后拿着TypeIndex去ObTypeIndexTable里找POBJECT\_TYPE，然后返回。

注意ObGetObjectType函数是导出的，只是未文档化，我们可以获取到这个函数的地址。

|  |
| --- |
| #include <ntifs.h>  // 未文档化函数  typedef POBJECT\_TYPE(NTAPI\* \_ObGetObjectType)(\_\_in PVOID Object);  VOID Unload(PDRIVER\_OBJECT pDriver)  {  }  NTSTATUS DriverEntry(PDRIVER\_OBJECT pDriver, PUNICODE\_STRING pReg)  {  UNICODE\_STRING FuncName = RTL\_CONSTANT\_STRING(L"ObGetObjectType");  \_ObGetObjectType ObGetObjectType\_I = MmGetSystemRoutineAddress(&FuncName);  POBJECT\_TYPE ObjTypetype = ObGetObjectType\_I(IoGetCurrentProcess());  if (ObjTypetype == \*PsProcessType)  {  DbgPrintEx(77, 0, "进程对象\r\n");  }  else if (ObjTypetype == \*PsThreadType)  {  DbgPrintEx(77, 0, "线程对象\r\n");  }  else  {  DbgPrintEx(77, 0, "其他类型对象\r\n");  }  pDriver->DriverUnload = Unload;  return STATUS\_SUCCESS;  } |



因为PsProcessType 是导出可用的，直接拿来对比就好了，不然还得判断一下Name字段的偏移，很麻烦。

# 全局句柄表结构

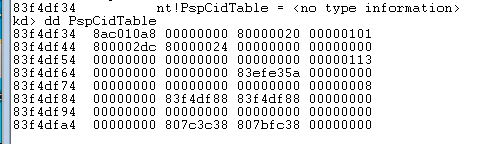
句柄表分全局和私有两种，结构是一样的，可扩展，最多三层。x86和x64只是地址长度的区别，结构是一样的。

全局句柄表存储了所有线程，进程对象。全局句柄表叫PspCidTable:

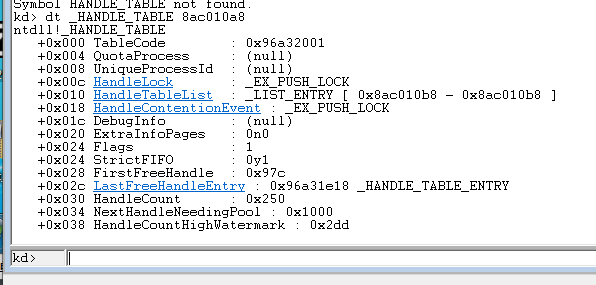


如果忘记叫啥可以模糊搜索。

这是一个二级指针，dd一下拿到一级指针：

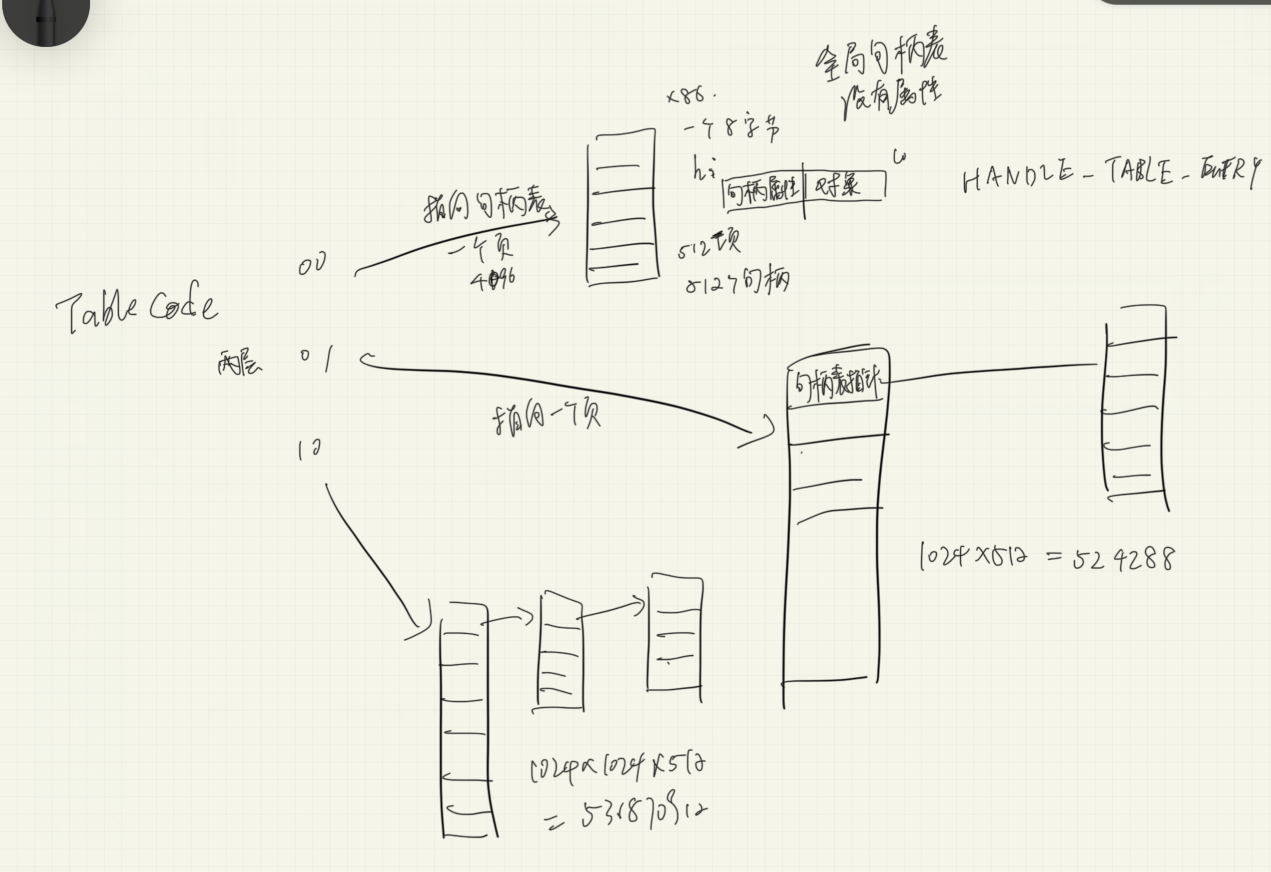


类型是 HANDLE\_TABLE，可以dt看看：



TableCode字段就是句柄表，低2位表示层级结构，00表示一层，01表示二层，10表示三层，没有四层。

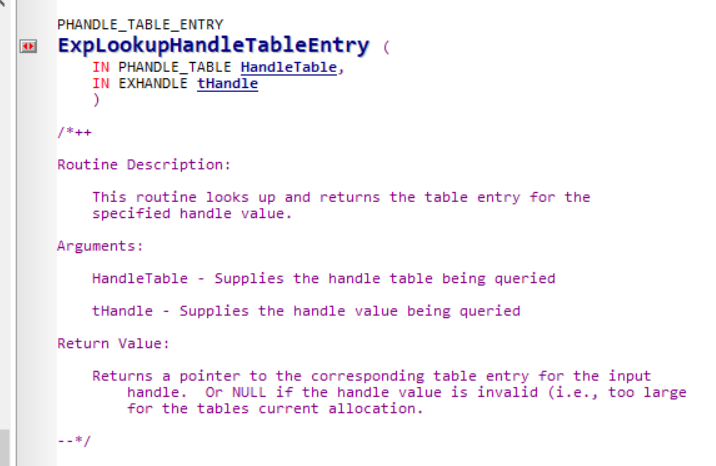
我这虚拟机的全局句柄表显然是二层结构。



层级结构是以页为单位的，例如一层结构就是4KB页里存了512个句柄表项，每个表项是8字节，高4字节是句柄属性，低四字节是对象指针的加密值。如果是二层，就是4KB的目录，共1024个指针，分别指向句柄表，那就是1024x512个句柄。

# 通过PID在全局句柄表里找EPROCESS

有个内核函数叫ExpLookupHandleTableEntry



他从句柄表里找一个句柄值，返回句柄表项。我们可以直接调用他。

37课讲了用PID暴力枚举进程，用了 PsLookupProcessByProcessId 函数，我们可以看看他是怎么做的。

手动找的话，如PID=3152,因为PID以4为单位递增，我们要除以4看看它是第几个句柄：

3152/4=788，是第788个句柄

然后看看他在第几个表，因为每个表存了512个句柄，所以 788/512=1，所以是第二个句柄表

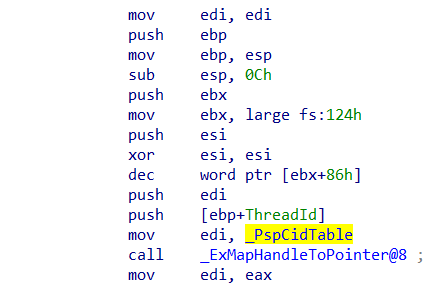
然后看看是在表中第几项，788 % 512

这样就找到了句柄表项，高4字节属性，低4字节是对象，注意要清除低3位属性，后面遍历的时候可以看到这个操作。

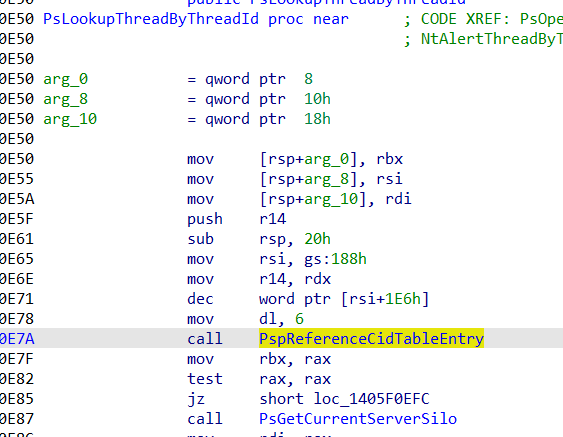
# 遍历全局句柄表，打印进程名

### 如何找到PspCidTable

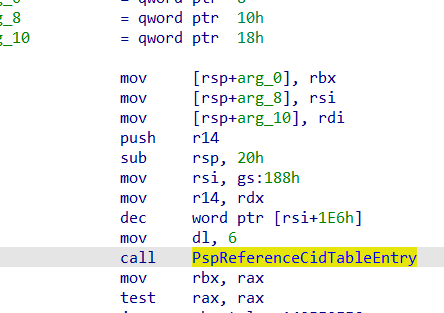
一个比较流行的做法是找一个使用了PspCidTable的函数，通过特征码找。来看看PsLookupThreadByThreadId.



上图是win7x86的代码，是OK的，我们再来看看win7x64和win10x64这个函数长啥样呢？



Win7x64要通过一个函数调用来获取



Win10x64也是一样。

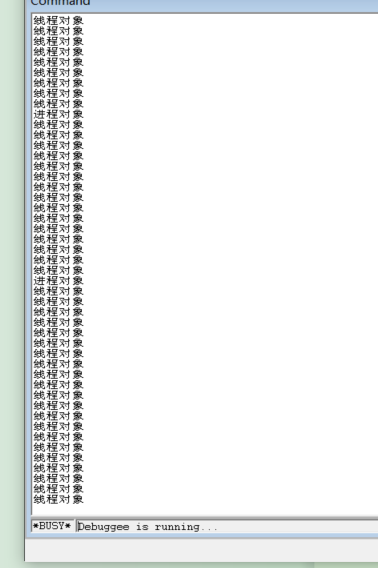
所以我们可以在代码里区分一下x86和x64，如果是x86，就找 mov edi,xxx + call xxx的特征；x64就找mov dl,6 + call xxx的特征。

时间关系就没写X64版本：

|  |
| --- |
| // 不支持x64  PHANDLE\_TABLE GetPspCidTable()  {  PUCHAR data = (PUCHAR)PsLookupThreadByThreadId;  #ifdef \_X86\_  for (int i = 0;i<100; i++)  {  // mov edi, [\_PspCidTable]  // call xxx  if (memcmp(&data[i], "\x8b\x3d", 2) == 0 && data[i+6]== 0xe8)  {  return \*(PHANDLE\_TABLE\*)(\*(ULONG\_PTR\*)&data[i + 2]);  }  }  #else  for (int i = 0; i < 100; i++)  {  // call PspReferenceCidTableEntry  // mov rbx, rax  if (data[i] == 0xe8 && memcmp(&data[i+5],"\x48\x8b\xd8",3) == 0)  {  ULONG32 E8Code = \*(ULONG32\*)&data[i + 1];  ULONG\_PTR \_PspReferenceCidTableEntry = E8Code + (ULONG\_PTR)&data[i + 5];  for (int j = 0; j < 100; j++)  {  if ()  }  }  }  #endif  return 0;  } |

### 遍历句柄表

三级和二级原理一样我就没写了。打印进程名我也没整了，可以去之前的代码里找找。



|  |
| --- |
| #include <ntifs.h>  typedef struct \_HANDLE\_TABLE {  ULONG\_PTR TableCode;  // ...  }HANDLE\_TABLE,\*PHANDLE\_TABLE;  typedef struct \_HANDLE\_TABLE\_ENTRY {  ULONG Object;  ULONG Attr;  }HANDLE\_TABLE\_ENTRY,\* PHANDLE\_TABLE\_ENTRY;  PHANDLE\_TABLE GetPspCidTable();  VOID LiteralHandleTable(PHANDLE\_TABLE HandleTable);  // 未文档化函数  typedef POBJECT\_TYPE(NTAPI\* \_ObGetObjectType)(\_\_in PVOID Object);  VOID Unload(PDRIVER\_OBJECT pDriver)  {  }  NTSTATUS DriverEntry(PDRIVER\_OBJECT pDriver, PUNICODE\_STRING pReg)  {  PHANDLE\_TABLE PspCidTable = GetPspCidTable();  DbgPrintEx(77, 0, "%x\r\n", PspCidTable->TableCode);  LiteralHandleTable(PspCidTable);    pDriver->DriverUnload = Unload;  return STATUS\_SUCCESS;  }  // 不支持x64  PHANDLE\_TABLE GetPspCidTable()  {  PUCHAR data = (PUCHAR)PsLookupThreadByThreadId;  #ifdef \_X86\_  for (int i = 0;i<100; i++)  {  // mov edi, [\_PspCidTable]  // call xxx  if (memcmp(&data[i], "\x8b\x3d", 2) == 0 && data[i+6]== 0xe8)  {  return \*(PHANDLE\_TABLE\*)(\*(ULONG\_PTR\*)&data[i + 2]);  }  }  #else  for (int i = 0; i < 100; i++)  {  // call PspReferenceCidTableEntry  // mov rbx, rax  if (data[i] == 0xe8 && memcmp(&data[i+5],"\x48\x8b\xd8",3) == 0)  {  ULONG32 E8Code = \*(ULONG32\*)&data[i + 1];  ULONG\_PTR \_PspReferenceCidTableEntry = E8Code + (ULONG\_PTR)&data[i + 5];  for (int j = 0; j < 100; j++)  {  if ()  }  }  }  #endif  return 0;  }  VOID LiteralHandleTable(PHANDLE\_TABLE HandleTable)  {  UNICODE\_STRING FuncName = RTL\_CONSTANT\_STRING(L"ObGetObjectType");  \_ObGetObjectType ObGetObjectType\_I = MmGetSystemRoutineAddress(&FuncName);  ULONG TableLevel = HandleTable->TableCode & 3;  PUCHAR TableCode = HandleTable->TableCode & ~3;  switch (TableLevel)  {  case 0:  {  DbgPrintEx(77,0,"一级句柄表\r\n");  for (int i = 0; i < PAGE\_SIZE; i+=8)  {  PHANDLE\_TABLE\_ENTRY entry = (PHANDLE\_TABLE\_ENTRY)&TableCode[i];  if (MmIsAddressValid(entry->Object & ~7))  {  POBJECT\_TYPE ObjTypetype = ObGetObjectType\_I(entry->Object & ~7);  if (ObjTypetype == \*PsProcessType)  {  DbgPrintEx(77, 0, "进程对象\r\n");  }  else if (ObjTypetype == \*PsThreadType)  {  DbgPrintEx(77, 0, "线程对象\r\n");  }  else  {  DbgPrintEx(77, 0, "其他类型对象\r\n");  }  }  }  break;  }  case 1:  {  DbgPrintEx(77, 0, "二级句柄表\r\n");  for (int j = 0;j < PAGE\_SIZE; j+=sizeof(ULONG\_PTR))  {  PUCHAR Level2HandleTable = (PUCHAR)(\*(PULONG\_PTR)&TableCode[j]);  if (MmIsAddressValid(Level2HandleTable))  {  for (int i = 0; i < PAGE\_SIZE; i += 8)  {  PHANDLE\_TABLE\_ENTRY entry = (PHANDLE\_TABLE\_ENTRY)&Level2HandleTable[i];  if (MmIsAddressValid(entry->Object & ~7))  {  POBJECT\_TYPE ObjTypetype = ObGetObjectType\_I(entry->Object & ~7);  if (ObjTypetype == \*PsProcessType)  {  DbgPrintEx(77, 0, "进程对象\r\n");  }  else if (ObjTypetype == \*PsThreadType)  {  DbgPrintEx(77, 0, "线程对象\r\n");  }  else  {  DbgPrintEx(77, 0, "其他类型对象\r\n");  }  }  }  }  }  break;  }  case 2:  {  DbgPrintEx(77, 0, "三级句柄表\r\n");    break;  }  }  } |