Win10遍历句柄表+修改权限过Callback保护

转载

xlaomlng 于 2019-05-26 16:33:25 发布 1362 收藏 1
本帖转载于http://www.m5home.com/bbs/thread-8847-1-1.html
本想发到看雪，但自己太菜，看雪“牛人”又太多，想想还是发到紫水晶吧。
感谢 TA 的 WIN64 教程带我走上驱动之路，想想除了个 VIP 账号就没教过学费，以后多在论坛发帖来回报一下吧。

正篇：
XP 和 Win7 上关于句柄表的文章不少，一点不懂的朋友请自行百度谷歌搜索，方法没变，依旧使用 ExpLookupHandleTableEntry 。

主要说说 Win10 的变化，相较于 Win7 这两个结构体有所改变： HANDLE_TABLE 和 HANDLE_TABLE_ENTRY，另外多了一个 HANDLE_TABLE_FREE_LIST 。

根据 WinDBG 逆向出的结构定义如下：
1: kd> dt nt!_HANDLE_TABLE

- 0x000 NextHandleNeedingPool : Uint4B
- 0x004 ExtraInfoPages : Int4B
- 0x008 TableCode : Uint8B
- 0x010 QuotaProcess : Ptr64 _EPROCESS
- 0x018 HandleTableList : _LIST_ENTRY
- 0x028 UniqueProcessId : Uint4B
- 0x02c Flags : Uint4B
- 0x02c StrictFIFO : Pos 0, 1 Bit
- 0x02c EnableHandleExceptions : Pos 1, 1 Bit
- 0x02c Rundown : Pos 2, 1 Bit
- 0x02c Duplicated : Pos 3, 1 Bit
- 0x02c RaiseUMExceptionOnInvalidHandleClose : Pos 4, 1 Bit
- 0x030 HandleContentionEvent : _EX_PUSH_LOCK
- 0x038 HandleTableLock : _EX_PUSH_LOCK
- 0x040 FreeLists : [1] _HANDLE_TABLE_FREE_LIST
- 0x040 ActualEntry : [32] UChar
- 0x060 DebugInfo : Ptr64 _HANDLE_TRACE_DEBUG_INFO

```
typedef struct _HANDLE_TABLE
{
ULONG32 NextHandleNeedingPool;
LONG32 ExtraInfoPages;
ULONG_PTR TableCode;
PEPROCESS QuotaProcess;
LIST_ENTRY HandleTableList;
ULONG32 UniqueProcessId;
union
{
ULONG32 Flags;
struct
{
BOOLEAN StrictFIFO : 1;
BOOLEAN EnableHandleExceptions : 1;
BOOLEAN Rundown : 1;
BOOLEAN Duplicated : 1;
BOOLEAN RaiseUMExceptionOnInvalidHandleClose : 1;
};
};
ULONG_PTR HandleContentionEvent;
ULONG_PTR HandleTableLock;
union
{
HANDLE_TABLE_FREE_LIST FreeLists[1];
BOOLEAN ActualEntry[32];
};
PVOID DebugInfo;
} HANDLE_TABLE, *PHANDLE_TABLE;
```

复制代码

```
1: kd> dt nt!_HANDLE_TABLE_ENTRY
```

- 0x000 VolatileLowValue : Int8B
- 0x000 LowValue : Int8B
- 0x000 InfoTable : Ptr64 _HANDLE_TABLE_ENTRY_INFO
- 0x008 HighValue : Int8B
- 0x008 NextFreeHandleEntry : Ptr64 _HANDLE_TABLE_ENTRY
- 0x008 LeafHandleValue : _EXHANDLE
- 0x000 RefCountField : Int8B
- 0x000 Unlocked : Pos 0, 1 Bit
- 0x000 RefCnt : Pos 1, 16 Bits
- 0x000 Attributes : Pos 17, 3 Bits
- 0x000 ObjectPointerBits : Pos 20, 44 Bits
- 0x008 GrantedAccessBits : Pos 0, 25 Bits
- 0x008 NoRightsUpgrade : Pos 25, 1 Bit
- 0x008 Spare1 : Pos 26, 6 Bits
- 0x00c Spare2 : Uint4B

```
typedef struct _HANDLE_TABLE_ENTRY
{
union
{
LONG_PTR VolatileLowValue;
LONG_PTR LowValue;
PVOID InfoTable;
LONG_PTR RefCountField;
struct
{
ULONG_PTR Unlocked : 1;
ULONG_PTR RefCnt : 16;
ULONG_PTR Attributes : 3;
ULONG_PTR ObjectPointerBits : 44;
};
};
union
{
LONG_PTR HighValue;
struct _HANDLE_TABLE_ENTRY *NextFreeHandleEntry;
EXHANDLE LeafHandleValue;
struct
{
ULONG32 GrantedAccessBits : 25;
ULONG32 NoRightsUpgrade : 1;
ULONG32 Spare1 : 6;
};
ULONG32 Spare2;
};
} HANDLE_TABLE_ENTRY, *PHANDLE_TABLE_ENTRY;
```

复制代码

```
1: kd> dt nt!_HANDLE_TABLE_FREE_LIST
```

- 0x000 FreeListLock : _EX_PUSH_LOCK
- 0x008 FirstFreeHandleEntry : Ptr64 _HANDLE_TABLE_ENTRY
- 0x010 LastFreeHandleEntry : Ptr64 _HANDLE_TABLE_ENTRY
- 0x018 HandleCount : Int4B
- 0x01c HighWaterMark : Uint4B
- 0x020 Reserved : [8] Uint4B

```
typedef struct _HANDLE_TABLE_FREE_LIST
{
ULONG_PTR FreeListLock;
PHANDLE_TABLE_ENTRY FirstFreeHandleEntry;
PHANDLE_TABLE_ENTRY lastFreeHandleEntry;
LONG32 HandleCount;
ULONG32 HighWaterMark;
ULONG32 Reserved[8];
} HANDLE_TABLE_FREE_LIST, *PHANDLE_TABLE_FREE_LIST;
```
复制代码

只是一部分成员的偏移变化了，要注意的是表示对象地址的成员移到了 ObjectPointerBits 上，长度为 44 位的数据。
Win10上内核对象的地址算法：
```
Object = Entry->ObjectPointerBits;
Object <<= 4;
Object |= 0xFFFF000000000000;
Object += 0x30;
```
复制代码

另外对象权限也变到了 GrantedAccessBits 成员中，长度为 25 位的数据。

关于 ExpLookupHandleTableEntry 的实现上，相较 Win7 的版本仅仅优化了下代码逻辑，可以说没有变化。
直接抄 WRK 的代码，建议多看几遍代码，配合网上的其他帖子理解一下句柄表结构，我就不废话了（懒）。

本来想写一大堆，但感觉都是废话删掉了，结果就变成这么一篇偷懒贴，干脆直接上核心代码吧：
```
NTSTATUS ViewHandle(ULONG32 ProcessId, POBJECT_INFO Buffer)
{
PEPROCESS EProcess = NULL;
ULONG_PTR Handle = 0;
PHANDLE_TABLE_ENTRY Entry = NULL;
PVOID Object = NULL;
POBJECT_TYPE ObjectType = NULL;
```

```
    if (!NT_SUCCESS(PsLookupProcessByProcessId((HANDLE)ProcessId, &EProcess)))
    {
            return STATUS_UNSUCCESSFUL;
    }

    for (Handle = 0;; Handle += HANDLE_VALUE_INC)
    {
            Entry = ExpLookupHandleTableEntry(*(PHANDLE_TABLE*)((PUCHAR)EProcess + g_HandleTableOffset), *(PEXHA
NDLE)&Handle);
            if (Entry == NULL)
            {
                    break;
            }

            *(ULONG_PTR*)&Object = Entry->ObjectPointerBits;
            *(ULONG_PTR*)&Object <<= 4;
            if (Object == NULL)
            {
                    continue;
            }

            *(ULONG_PTR*)&Object |= 0xFFFF000000000000;
            *(ULONG_PTR*)&Object += 0x30;
            ObjectType = ObGetObjectType(Object);
            if (ObjectType == NULL)
            {
                    continue;
            }

            wcscpy(Buffer->szTypeName, *(PCWSTR*)((PUCHAR)ObjectType + 0x18));
            Buffer->Handle = (HANDLE)Handle;
            Buffer->AccessMask = Entry->GrantedAccessBits;
            Buffer->Address = Object;

            Buffer++;
    }

    ObDereferenceObject(EProcess);

    return STATUS_SUCCESS;
```

}
复制代码

效果是这样的，对照WIN64AST：

这点破东西貌似不够看，再加个然并卵的玩意。

修改进程句柄权限过 Callback 保护：

```
if (wcscmp((PCWSTR)((PUCHAR)ObjectType + 0x18), L"Process") == 0)
{
if (*(PULONG32)((PUCHAR)Object + 0x2E8) == PassiveId)
{
Entry->GrantedAccessBits = 0x1FFFFF;
Status = STATUS_SUCCESS;
}
}
```

复制代码

拿 TP 举例，通常情况下，在 Ring3 调用 OpenProcess 传递 PROCESS_ALL_ACCESS 参数，该句柄对应的进程对象权限是 0x1FFFFF ，但对 TP 保护的游戏进行同样做法后会发现权限变成了 0x1FFD85，它的 Callback 中抹掉了进程对象的这些权限：

```
#define PROCESS_CREATE_THREAD (0x0002)
#define PROCESS_VM_OPERATION (0x0008)
#define PROCESS_VM_READ (0x0010)
#define PROCESS_VM_WRITE (0x0020)
#define PROCESS_DUP_HANDLE (0x0040)
#define PROCESS_SET_INFORMATION (0x0200)
```

复制代码

直接在 TP 的 Callback 中写 ret 游戏会封号，而且 TP 卸载时会蓝屏，内部通信肯定是有的，尝试过逆向，一堆 jmp 把我弄晕了，干脆就换种办法。

如上代码，我的做法是，遍历句柄表，判断对象类型为 “PROCESS” ，然后根据 EPROCESS 对象获得对应进程进程的 PID ，如果是被保护的进程就把权限写成 0x1FFFFF ，测试过 DXF 有效。

效果是这样的，恢复权限之前和之后分别为：

听说这是一种比较二逼的解决方法，没办法谁让我菜呢。如果有更好的办法还望不吝赐教。