# PsSetCreateProcessNotifyRoutine监控进程创建

原创

FFE4 于 2019-09-25 11:22:16 发布 3061 收藏 4

分类专栏： 内核开发 文章标签： PsSetCreateProcessNotifyRoutine



本文链接：https://blog.csdn.net/cssxn/article/details/101352152





PsSetCreateProcessNotifyRoutine函数用来注册一个进程创建的回调函数，当有新从进程被创建时，就把父进程的ID，和子进程（被创建的进程）ID传给回调函数，通过回调函数，可以监控新创建的进程

```
NTSTATUS PsSetCreateProcessNotifyRoutine(
  _In_  PCREATE_PROCESS_NOTIFY_ROUTINE NotifyRoutine,
  _In_  BOOLEAN Remove
);
```

想要实现的功能：新进程被系统创建时，打印出父进程和子进程的名字

在回调函数中可以直接获取两个进程的PID，但是要想通过PID得到进程名字，也有很多种方式。
但最简单的是还是通过`PsLookupProcessByProcessId`函数来根据PID获取`EPROCESS`结构体，然后可以直接通过结构体+偏移的方式读取到该进程的名字。

需要注意的是，在使用`PsLookupProcessByProcessId`得到一个`EPROCESS`结构以后，系统会对该进程的引用计数累加1，需要在不使用该结构时，及时调用`ObDereferenceObject`解除引用计数

也可以使用`PsGetProcessImageFileName`函数来获取，其内部实现原理也是通过`EPROCESS+偏移`的方式得到的进程名字

进程名字在`EPROCESS`结构体偏移`0x16C`的位置，可能不同的系统版本，偏移会有区别。

```
1: kd> dt _EPROCESS 0x880559f0
nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x098 ProcessLock      : _EX_PUSH_LOCK
   +0x0a0 CreateTime       : _LARGE_INTEGER 0x01d5734d`32be6c3e
   +0x0a8 ExitTime         : _LARGE_INTEGER 0x0
   +0x0b0 RundownProtect   : _EX_RUNDOWN_REF
   +0x0b4 UniqueProcessId  : 0x00000e30 Void
   +0x0b8 ActiveProcessLinks : _LIST_ENTRY [ 0x83f4ff18 - 0x880133f8 ]
   +0x0c0 ProcessQuotaUsage : [2] 0x1e0
   +0x0c8 ProcessQuotaPeak : [2] 0x1e0
   +0x0d0 CommitCharge     : 0x3a
   +0x0d4 QuotaBlock       : 0x87972640 _EPROCESS_QUOTA_BLOCK
   +0x0d8 CpuQuotaBlock    : (null)
   +0x0dc PeakVirtualSize  : 0x290000
   +0x0e0 VirtualSize      : 0x290000
   +0x0e4 SessionProcessLinks : _LIST_ENTRY [ 0x94c6e010 - 0x88013424 ]
   +0x0ec DebugPort        : (null)
   +0x0f0 ExceptionPortData : (null)
   +0x0f0 ExceptionPortValue : 0
   +0x0f0 ExceptionPortState : 0y000
   +0x0f4 ObjectTable      : 0x97727780 _HANDLE_TABLE
   +0x0f8 Token            : _EX_FAST_REF
   +0x0fc WorkingSetPage   : 0x73ce0
   +0x100 AddressCreationLock : _EX_PUSH_LOCK
   +0x104 RotateInProgress : (null)
   +0x108 ForkInProgress   : (null)
   +0x10c HardwareTrigger  : 0
   +0x110 PhysicalVadRoot  : (null)
   +0x114 CloneRoot        : (null)
   +0x118 NumberOfPrivatePages : 9
   +0x11c NumberOfLockedPages : 0
   +0x120 Win32Process     : (null)
   +0x124 Job              : (null)
   +0x128 SectionObject    : 0xa393d418 Void
   +0x12c SectionBaseAddress : 0x004e0000 Void
   +0x130 Cookie           : 0
   +0x134 Spare8           : 0
   +0x138 WorkingSetWatch  : (null)
   +0x13c Win32WindowStation : (null)
   +0x140 InheritedFromUniqueProcessId : 0x000006dc Void
   +0x144 LdtInformation   : (null)
   +0x148 VdmObjects       : (null)
   +0x14c ConsoleHostProcess : 0
   +0x150 DeviceMap        : (null)
   +0x154 EtwDataSource    : (null)
   +0x158 FreeTebHint      : 0x7ffdf000 Void
   +0x160 PageDirectoryPte : _HARDWARE_PTE
   +0x160 Filler           : 0
   +0x168 Session          : 0x94c6e000 Void
   +0x16c ImageFileName    : [15]  "calc.exe"
   +0x17b PriorityClass    : 0x2 ''
   +0x17c JobLinks         : _LIST_ENTRY [ 0x0 - 0x0 ]
1: kd>
```



再来看`PsGetProcessImageFileName`函数的反汇编，是不是也贼简单：

```
1: kd> uf PsGetProcessImageFileName
nt!PsGetProcessImageFileName:
83e97ea7 8bff            mov     edi,edi
83e97ea9 55              push    ebp
83e97eaa 8bec            mov     ebp,esp
83e97eac 8b4508          mov     eax,dword ptr [ebp+8]
83e97eaf 056c010000      add     eax,16Ch
83e97eb4 5d              pop     ebp
83e97eb5 c20400          ret     4
```

效果：

```
1: kd> g
ParentName:explorer.exe---> ChildName: notepad.exe
ParentName:explorer.exe---> ChildName: calc.exe

*BUSY* Debuggee is running...
```

完整代码：

```
extern NTSYSAPI PUCHAR NTAPI PsGetProcessImageFileName(PEPROCESS Process);


// 监控进程创建回调函数
VOID CreateProcessNotify(IN HANDLE  ParentId, IN HANDLE  ChildId, IN BOOLEAN  Create)
{
    PEPROCESS ParentEprocess = NULL;
    PEPROCESS ChildEprocess = NULL;
    NTSTATUS status;
    if (Create)
    {

        // 获取EPROCESS结构体
        status = PsLookupProcessByProcessId(ParentId, &ParentEprocess);
        if (!NT_SUCCESS(status))
        {
            KdPrint(("Get Parent Eprocess Failed\n"));
            return;
        }
        status = PsLookupProcessByProcessId(ChildId, &ChildEprocess);
        if (!NT_SUCCESS(status))
        {
            KdPrint(("Get Child Eprocess Failed\n"));
            return;
        }

        // 通过EPROCESS获取进程名
        KdPrint((
            "ParentName:%s---> ChildName: %s\n",
            PsGetProcessImageFileName(ParentEprocess),
            PsGetProcessImageFileName(ChildEprocess)
            ));

        ObDereferenceObject(ParentEprocess);
        ObDereferenceObject(ChildEprocess);
    }
}


VOID DriverUnload(PDRIVER_OBJECT driver)
{
    // 卸载时，移除回掉
    PsSetCreateProcessNotifyRoutine(CreateProcessNotify, TRUE);
    DbgPrint("first: Our driver is unloading...\r\n");
}


NTSTATUS DriverEntry(PDRIVER_OBJECT pDriver, PUNICODE_STRING reg_path)
{
    DbgBreakPoint();

    PsSetCreateProcessNotifyRoutine(CreateProcessNotify, FALSE);

    pDriver->DriverUnload = DriverUnload;

    return STATUS_SUCCESS;
}
```