进程回调可以监视进程的创建和退出，这个在前面的章节已经讲过了。某些游戏保护的驱动喜欢用这个函数来监视有没有黑名单中的程序运行，如果运行则阻止运行或者把游戏退出。而线程回调则通常用来监控远程线程的建立，如果发现有远程线程注入到了游戏进程里，则马上把游戏退出。现在来详细讲解如何绕过这个两个监控。

我们注册的进程回调，会存储在一个名为 PspCreateProcessNotifyRoutine 的数组里。PspCreateProcessNotifyRoutine 可以理解成一个 PVOID 数组，它记录了系统里所有进程回调的地址。这个数组最大长度是 64*sizeof(PVOID)。所以枚举进程回调的思路如下：找到这个数组的地址，然后解密数组的数据，得到所有回调的地址（这个数组记录的数据并不是回调的地址，而是经过加密地址，需要解密才行）。枚举线程回调同理，要找到 PspCreateThreadNotifyRoutine 的地址（这个数组最大长度也是 64*sizeof(PVOID)），然后解密数据，并把解密后的地址打印出来。

至于怎么处理这些回调就简单了。可以使用标准函数（PsSetCreateProcessNotifyRoutine、PsRemoveCreateThreadNotifyRoutine）将其摘掉，也可以直接在回调函数首地址写入 RET 把回调函数废掉。WIN64AST 就提供了两种办法处理，以对付某些奸诈的游戏保护。

首先要获得 PspCreateProcessNotifyRoutine 的地址。PspCreateProcessNotifyRoutine 在 PspSetCreateProcessNotifyRoutine 函数里出现了。而 PspSetCreateProcessNotifyRoutine 则在 PsSetCreateProcessNotifyRoutine 中被调用（注意前一个是 PspXXX，后一个是 PsXXX）。找到 PspSetCreateProcessNotifyRoutine 之后，再匹配特征码：

```
lkd> U PsSetCreateProcessNotifyRoutine
nt!PsSetCreateProcessNotifyRoutine:
fffff800`042d83c0 4533c0          xor     r8d,r8d
fffff800`042d83c3  e9e8fdffff            jmp         nt!PspSetCreateProcessNotifyRoutine
(fffff800`042d81b0)
fffff800`042d83c8 90              nop
fffff800`042d83c9 90              nop
fffff800`042d83ca 90              nop
fffff800`042d83cb 90              nop
fffff800`042d83cc 90              nop
fffff800`042d83cd 90              nop
```

```
lkd> uf  PspSetCreateProcessNotifyRoutine
nt!PspSetCreateProcessNotifyRoutine:
fffff800`042d81b0 48895c2408      mov     qword ptr [rsp+8],rbx
fffff800`042d81b5 48896c2410      mov     qword ptr [rsp+10h],rbp
fffff800`042d81ba 4889742418      mov     qword ptr [rsp+18h],rsi
fffff800`042d81bf 57              push    rdi
fffff800`042d81c0 4154            push    r12
fffff800`042d81c2 4155            push    r13
fffff800`042d81c4 4156            push    r14
fffff800`042d81c6 4157            push    r15
fffff800`042d81c8 4883ec20        sub     rsp,20h
fffff800`042d81cc 4533e4          xor     r12d,r12d
```

```
fffff800`042d81cf 418ae8          mov     bpl,r8b
fffff800`042d81d2 4c8be9          mov     r13,rcx
fffff800`042d81d5 418d5c2401      lea     ebx,[r12+1]
fffff800`042d81da 413ad4          cmp     dl,r12b
fffff800`042d81dd  0f840e010000      je          nt!PspSetCreateProcessNotifyRoutine+0x141
(fffff800`042d82f1)

nt!PspSetCreateProcessNotifyRoutine+0x33:
fffff800`042d81e3 65488b3c2588010000 mov   rdi,qword ptr gs:[188h]
fffff800`042d81ec 83c8ff          or      eax,0FFFFFFFFh
fffff800`042d81ef 660187c4010000  add     word ptr [rdi+1C4h],ax
fffff800`042d81f6   4c8d358395d6ff     lea           r14,[nt!PspCreateProcessNotifyRoutine
(fffff800`04041780)]
//省略后续无关代码
```

于是我们根据特征码写出了以下代码（仅在 WIN7X64 上有效，WIN8、8.1 需要自己重新定义特征码）：

```
ULONG64 FindPspCreateProcessNotifyRoutine()
{
	LONG			OffsetAddr=0;
	ULONG64		i=0,pCheckArea=0;
	UNICODE_STRING	unstrFunc;
	//获得 PsSetCreateProcessNotifyRoutine 的地址
	RtlInitUnicodeString(&unstrFunc, L"PsSetCreateProcessNotifyRoutine");
	pCheckArea = (ULONG64)MmGetSystemRoutineAddress (&unstrFunc);
	//获得 PspSetCreateProcessNotifyRoutine 的地址
	memcpy(&OffsetAddr,(PUCHAR)pCheckArea+4,4);
	pCheckArea=(pCheckArea+3)+5+OffsetAddr;
	DbgPrint("PspSetCreateProcessNotifyRoutine: %llx",pCheckArea);
	//获得 PspCreateProcessNotifyRoutine 的地址
	for(i=pCheckArea;i<pCheckArea+0xff;i++)
	{
		if(*(PUCHAR)i==0x4c && *(PUCHAR)(i+1)==0x8d && *(PUCHAR)(i+2)==0x35)
		{
			LONG OffsetAddr=0;
			memcpy(&OffsetAddr,(PUCHAR)(i+3),4);
			return OffsetAddr+7+i;
		}
	}
	return 0;
}
```

找到了 PspCreateProcessNotifyRoutine，枚举操作就好办了。需要说明的是，在 PspCreateProcessNotifyRoutine 里的数据竟然被加密了，需要把数组的值和 0xfffffffffffffff8 进

行“与”位运算才行：

```
void EnumCreateProcessNotify()
{
	int i=0;
	BOOLEAN b;
	ULONG64NotifyAddr=0,MagicPtr=0;
	ULONG64PspCreateProcessNotifyRoutine=FindPspCreateProcessNotifyRoutine();
	DbgPrint("PspCreateProcessNotifyRoutine: %llx",PspCreateProcessNotifyRoutine);
	if(!PspCreateProcessNotifyRoutine)
		return;
	for(i=0;i<64;i++)
	{
		MagicPtr=PspCreateProcessNotifyRoutine+i*8;
		NotifyAddr=*(PULONG64)(MagicPtr);
		if(MmIsAddressValid((PVOID)NotifyAddr) && NotifyAddr!=0)
		{
			NotifyAddr=*(PULONG64)(NotifyAddr & 0xfffffffffffffff8);
			DbgPrint("[CreateProcess]%llx",NotifyAddr);
		}
	}
}
```

枚举线程回调同理，先找到 PspCreateThreadNotifyRoutine 的地址。此符号存在于 PsSetCreateThreadNotifyRoutine 里：

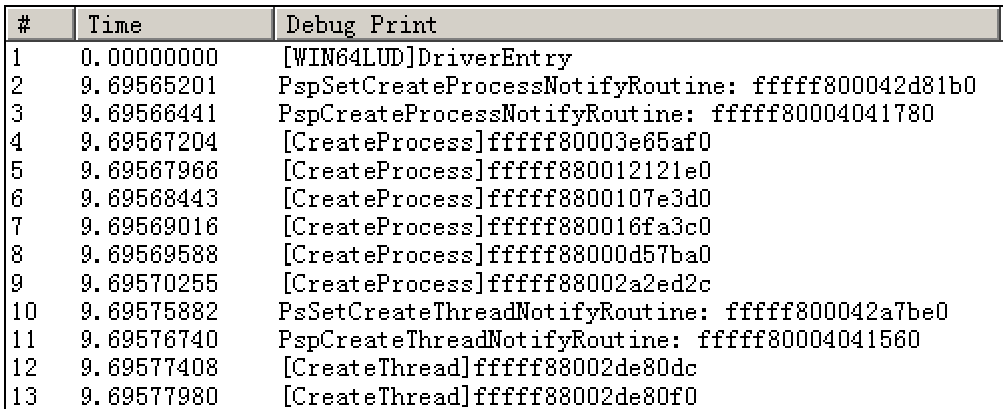
```
lkd> uf PsSetCreateThreadNotifyRoutine
nt!PsSetCreateThreadNotifyRoutine:
fffff800`042a7be0 48895c2408       mov     qword ptr [rsp+8],rbx
fffff800`042a7be5 57               push    rdi
fffff800`042a7be6 4883ec20         sub     rsp,20h
fffff800`042a7bea 33d2             xor     edx,edx
fffff800`042a7bec e86faffeff       call    nt!ExAllocateCallBack (fffff800`04292b60)
fffff800`042a7bf1 488bf8           mov     rdi,rax
fffff800`042a7bf4 4885c0           test    rax,rax
fffff800`042a7bf7 7507             jne     nt!PsSetCreateThreadNotifyRoutine+0x20
(fffff800`042a7c00)

nt!PsSetCreateThreadNotifyRoutine+0x19:
fffff800`042a7bf9 b89a0000c0       mov     eax,0C000009Ah
fffff800`042a7bfe eb4a             jmp     nt!PsSetCreateThreadNotifyRoutine+0x6a
(fffff800`042a7c4a)

nt!PsSetCreateThreadNotifyRoutine+0x20:
fffff800`042a7c00 33db             xor     ebx,ebx
```

```
nt!PsSetCreateThreadNotifyRoutine+0x22:
fffff800`042a7c02  488d0d5799d9ff    lea          rcx,[nt!PspCreateThreadNotifyRoutine
(fffff800`04041560)]
fffff800`042a7c09 4533c0          xor     r8d,r8d
fffff800`042a7c0c 488bd7          mov     rdx,rdi
fffff800`042a7c0f 488d0cd9        lea     rcx,[rcx+rbx*8]
fffff800`042a7c13 e83814f8ff      call    nt!ExCompareExchangeCallBack (fffff800`04229050)
fffff800`042a7c18 84c0            test    al,al
fffff800`042a7c1a  7511                jne        nt!PsSetCreateThreadNotifyRoutine+0x4d
(fffff800`042a7c2d)
//省略后续无关内容
```

枚举的代码也类似：

```
void EnumCreateThreadNotify()
{
	int i=0;
	BOOLEAN b;
	ULONG64NotifyAddr=0,MagicPtr=0;
	ULONG64PspCreateThreadNotifyRoutine=FindPspCreateThreadNotifyRoutine();
	DbgPrint("PspCreateThreadNotifyRoutine: %llx",PspCreateThreadNotifyRoutine);
	if(!PspCreateThreadNotifyRoutine)
		return;
	for(i=0;i<64;i++)
	{
		MagicPtr=PspCreateThreadNotifyRoutine+i*8;
		NotifyAddr=*(PULONG64)(MagicPtr);
		if(MmIsAddressValid((PVOID)NotifyAddr) && NotifyAddr!=0)
		{
			NotifyAddr=*(PULONG64)(NotifyAddr & 0xfffffffffffffff8);
			DbgPrint("[CreateThread]%llx",NotifyAddr);
		}
	}
}
```

最后执行的效果如下：



```
#    Time          Debug Print
1    0.00000000    [WIN64LUD]DriverEntry
2    9.69565201    PspSetCreateProcessNotifyRoutine: fffff800042d81b0
3    9.69566441    PspCreateProcessNotifyRoutine: fffff80004041780
4    9.69567204    [CreateProcess]fffff80003e65af0
5    9.69567966    [CreateProcess]fffff880012121e0
6    9.69568443    [CreateProcess]fffff8800107e3d0
7    9.69569016    [CreateProcess]fffff880016fa3c0
8    9.69569588    [CreateProcess]fffff88000d57ba0
9    9.69570255    [CreateProcess]fffff88002a2ed2c
10   9.69575882    PsSetCreateThreadNotifyRoutine: fffff800042a7be0
11   9.69576740    PspCreateThreadNotifyRoutine: fffff80004041560
12   9.69577408    [CreateThread]fffff88002de80dc
13   9.69577980    [CreateThread]fffff88002de80f0
```

备注：干净的 WIN7X64 系统是没有 CreateThread 回调的。为了体现枚举效果，特地在示例代码里增加了创建线程回调的代码。