在 NT5 平台下，要监控进线程句柄的操作，通常要挂钩 3 个 API：NtOpenProcess、NtOpenThread、NtDuplicateObject。但是在 VISTA SP1 以及之后的系统中，我们可以完全抛弃 HOOK 方案了，转而使用一个标准的 API：ObRegisterCallbacks。下面做一个监视进线程句柄操作的程序，并实现保护名为 CALC.EXE 的进程不被结束。

首先介绍一下 ObRegisterCallbacks 这个函数。此函数的前缀是 Ob，看得出它是属于对象管理器的函数，Register 是注册，Callbacks 是回调（复数）。因此从字面意思上看，它是注册一个对象回调的意思。现在它只能监控进程对象和线程对象。但微软承诺会给此函数增加功能，实现对其它内核对象的监控。这个函数在不能合法进行内核挂钩的 WIN64 上特别有用，但是微软做了一个很扯淡的限制：**驱动程序必须有数字签名才能使用此函数**。不过国外的黑客对此限制很不爽，他们通过逆向 ObRegisterCallbacks，找到了破解这个限制的方法。经研究，内核通过 MmVerifyCallbackFunction 验证此回调是否合法，但此函数只是简单的验证了一下 DriverObject->DriverSection->Flags 的值是不是为 0x20：

```
nt!MmVerifyCallbackFunction+0x75:
fffff800`01a66865 f6406820        test    byte ptr [rax+68h],20h
fffff800`01a66869 0f45fd          cmovne  edi,ebp
```

所以破解方法非常简单，只要把 DriverObject->DriverSection->Flags 的值“按位或”0x20 即可：

```
    ldr = (PLDR_DATA)DriverObject->DriverSection;
    ldr->Flags |= 0x20;
```

接下来看看此函数的原型：

```
NTSTATUS ObRegisterCallbacks(
  __in   POB_CALLBACK_REGISTRATION CallBackRegistration,
  __out  PVOID *RegistrationHandle
);
```

在注册回调时用到的两个结构体：

```
typedef struct _OB_CALLBACK_REGISTRATION {
  USHORT                    Version;
  USHORT                    OperationRegistrationCount;
  UNICODE_STRING            Altitude;
  PVOID                     RegistrationContext;
  OB_OPERATION_REGISTRATION *OperationRegistration;
} OB_CALLBACK_REGISTRATION, *POB_CALLBACK_REGISTRATION;
```

```
typedef struct _OB_OPERATION_REGISTRATION {
  POBJECT_TYPE                *ObjectType;
  OB_OPERATION                Operations;
  POB_PRE_OPERATION_CALLBACK  PreOperation;
  POB_POST_OPERATION_CALLBACK PostOperation;
} OB_OPERATION_REGISTRATION, *POB_OPERATION_REGISTRATION;
```

输入一个 OB_CALLBACK_REGISTRATION 结构体，输出一个句柄。在结构体 OB_CALLBACK_REGISTRATION 里面还有几个结构体指针，它们定义和每个成员的含义大家可以在这里查看。下面给出注册回调的源代码，重点就是加粗的三行代码：指出监视的对象类型、句柄产生的方式（直接创建句柄还是复制句柄）和回调函数的地址。设置完结构体后，调用 ObRegisterCallbacks 就能创建了。取消回调则使用 ObUnRegisterCallbacks。

```
NTSTATUS ObProtectProcess(BOOLEAN Enable)
{
    if(Enable==TRUE)
    {
        NTSTATUS obst1=0,obst2=0;
        OB_CALLBACK_REGISTRATION obReg,obReg2;
        OB_OPERATION_REGISTRATION opReg,opReg2;
        //reg ob callback 1
        memset(&obReg, 0, sizeof(obReg));
        obReg.Version = ObGetFilterVersion();
        obReg.OperationRegistrationCount = 1;
        obReg.RegistrationContext = NULL;
        RtlInitUnicodeString(&obReg.Altitude, L"321124");
        obReg.OperationRegistration = &opReg;
        memset(&opReg, 0, sizeof(opReg));
        opReg.ObjectType = PsProcessType;
        opReg.Operations = OB_OPERATION_HANDLE_CREATE | OB_OPERATION_HANDLE_DUPLICATE;
        opReg.PreOperation = (POB_PRE_OPERATION_CALLBACK)&preCall;
        obst1=ObRegisterCallbacks(&obReg, &obHandle);
        //reg ob callback 2
        memset(&obReg2, 0, sizeof(obReg2));
        obReg2.Version = ObGetFilterVersion();
        obReg2.OperationRegistrationCount = 1;
        obReg2.RegistrationContext = NULL;
        RtlInitUnicodeString(&obReg2.Altitude, L"321125");
        obReg2.OperationRegistration = &opReg2;
        memset(&opReg2, 0, sizeof(opReg2));
        opReg2.ObjectType = PsThreadType;
        opReg2.Operations = OB_OPERATION_HANDLE_CREATE | OB_OPERATION_HANDLE_DUPLICATE;
        opReg2.PreOperation = (POB_PRE_OPERATION_CALLBACK)&preCall2;
        obst1=ObRegisterCallbacks(&obReg2, &obHandle2);
        return NT_SUCCESS(obst1) & NT_SUCCESS(obst2);
    }
    else
    {
        if(obHandle!=NULL)
```

```
            ObUnRegisterCallbacks(obHandle);
        if(obHandle2!=NULL)
            ObUnRegisterCallbacks(obHandle2);
        return TRUE;
    }
}
```

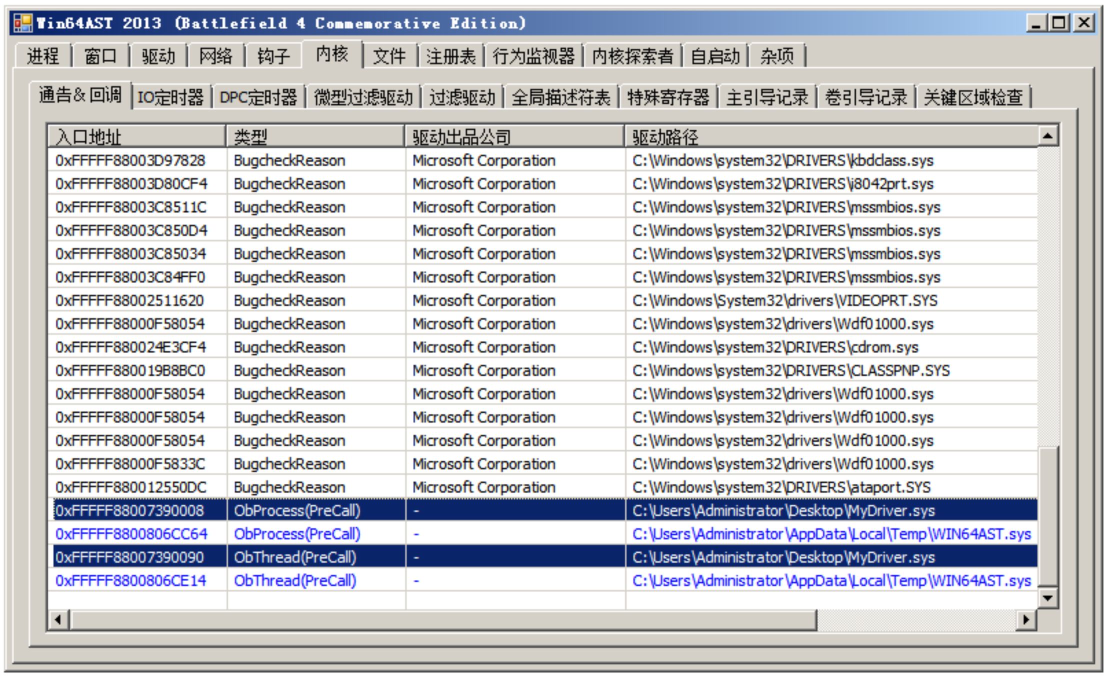
在回调函数里，从pOperationInformation->Object 获得进程或线程的对象，如果发现此对象是要保护的进程或者线程，就去掉 TERMINATE_PROCESS 或 TERMINATE_THREAD 权限：

```
OB_PREOP_CALLBACK_STATUS preCall(PVOID RegistrationContext, POB_PRE_OPERATION_INFORMATION
pOperationInformation)
{
    #define PROCESS_TERMINATE 0x1
    HANDLE pid;
    if(pOperationInformation->ObjectType!=*PsProcessType)
        goto exit_sub;
    pid = PsGetProcessId((PEPROCESS)pOperationInformation->Object);
    DbgPrint("[OBCALLBACK][Process]PID=%ld\n",pid);
    UNREFERENCED_PARAMETER(RegistrationContext);
    if( IsProtectedProcessName((PEPROCESS)pOperationInformation->Object) )
    {
        if (pOperationInformation->Operation == OB_OPERATION_HANDLE_CREATE)
        {
            if
((pOperationInformation->Parameters->CreateHandleInformation.OriginalDesiredAccess       &
PROCESS_TERMINATE) == PROCESS_TERMINATE)
            {
                pOperationInformation->Parameters->CreateHandleInformation.DesiredAccess
&= ~PROCESS_TERMINATE;
            }
        }
        if(pOperationInformation->Operation == OB_OPERATION_HANDLE_DUPLICATE)
        {
            if
((pOperationInformation->Parameters->DuplicateHandleInformation.OriginalDesiredAccess    &
PROCESS_TERMINATE) == PROCESS_TERMINATE)
            {

pOperationInformation->Parameters->DuplicateHandleInformation.DesiredAccess              &=
~PROCESS_TERMINATE;
            }
        }
    }
exit_sub:
    return OB_PREOP_SUCCESS;
}
```

```
OB_PREOP_CALLBACK_STATUS preCall2(PVOID RegistrationContext, POB_PRE_OPERATION_INFORMATION
pOperationInformation)
{
                                      #define THREAD_TERMINATE2 0x1
                                      PEPROCESS ep;
                                      PETHREAD et;
                                      HANDLE pid;
                                      if(pOperationInformation->ObjectType!=*PsThreadType
)
                                          goto exit_sub;
                                      et=(PETHREAD)pOperationInformation->Object;
                                      ep=IoThreadToProcess(et);
                                      pid = PsGetProcessId(ep);
                                      DbgPrint("[OBCALLBACK][Thread]PID=%ld;
TID=%ld\n",pid,PsGetThreadId(et));
                                      UNREFERENCED_PARAMETER(RegistrationContext);
                                      if( IsProtectedProcessName(ep) )
                                      {
                                          if      (pOperationInformation->Operation      ==
OB_OPERATION_HANDLE_CREATE)
                                          {
                                               if
((pOperationInformation->Parameters->CreateHandleInformation.OriginalDesiredAccess       &
THREAD_TERMINATE2) == THREAD_TERMINATE2)
            {
                pOperationInformation->Parameters->CreateHandleInformation.DesiredAccess
&= ~THREAD_TERMINATE2;
            }
                                          }
                                          if(pOperationInformation->Operation           ==
OB_OPERATION_HANDLE_DUPLICATE)
                                          {
                                               if
((pOperationInformation->Parameters->DuplicateHandleInformation.OriginalDesiredAccess    &
THREAD_TERMINATE2) == THREAD_TERMINATE2)
            {

pOperationInformation->Parameters->DuplicateHandleInformation.DesiredAccess           &=
~THREAD_TERMINATE2;
            }
                                          }
```

```
                                        }
exit_sub:
                                        return OB_PREOP_SUCCESS;
}
```

加载驱动后，可以用 WIN64AST 看到回调：

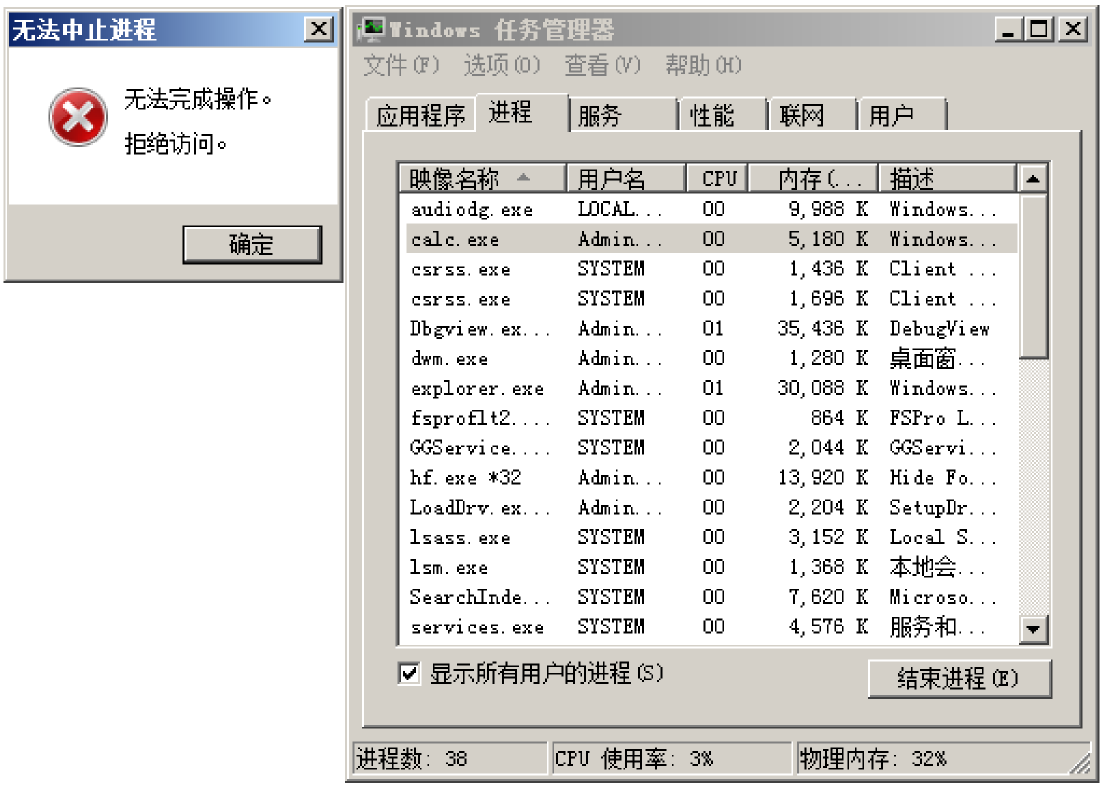

Win64AST 2013 (Battlefield 4 Commemorative Edition)

进程 | 窗口 | 驱动 | 网络 | 钩子 | 内核 | 文件 | 注册表 | 行为监视器 | 内核探索者 | 自启动 | 杂项

通告& 回调 | IO定时器 | DPC定时器 | 微型过滤驱动 | 过滤驱动 | 全局描述符表 | 特殊寄存器 | 主引导记录 | 卷引导记录 | 关键区域检查

入口地址	类型	驱动出品公司	驱动路径
0xFFFFF88003D97828	BugcheckReason	Microsoft Corporation	C:\Windows\system32\DRIVERS\kbdclass.sys
0xFFFFF88003D80CF4	BugcheckReason	Microsoft Corporation	C:\Windows\system32\DRIVERS\i8042prt.sys
0xFFFFF88003C8511C	BugcheckReason	Microsoft Corporation	C:\Windows\system32\DRIVERS\mssmbios.sys
0xFFFFF88003C850D4	BugcheckReason	Microsoft Corporation	C:\Windows\system32\DRIVERS\mssmbios.sys
0xFFFFF88003C85034	BugcheckReason	Microsoft Corporation	C:\Windows\system32\DRIVERS\mssmbios.sys
0xFFFFF88003C84FF0	BugcheckReason	Microsoft Corporation	C:\Windows\system32\DRIVERS\mssmbios.sys
0xFFFFF88002511620	BugcheckReason	Microsoft Corporation	C:\Windows\System32\drivers\VIDEOPRT.SYS
0xFFFFF88000F58054	BugcheckReason	Microsoft Corporation	C:\Windows\system32\drivers\Wdf01000.sys
0xFFFFF880024E3CF4	BugcheckReason	Microsoft Corporation	C:\Windows\system32\DRIVERS\cdrom.sys
0xFFFFF880019B8BC0	BugcheckReason	Microsoft Corporation	C:\Windows\system32\DRIVERS\CLASSPNP.SYS
0xFFFFF88000F58054	BugcheckReason	Microsoft Corporation	C:\Windows\system32\drivers\Wdf01000.sys
0xFFFFF88000F58054	BugcheckReason	Microsoft Corporation	C:\Windows\system32\drivers\Wdf01000.sys
0xFFFFF88000F58054	BugcheckReason	Microsoft Corporation	C:\Windows\system32\drivers\Wdf01000.sys
0xFFFFF88000F5833C	BugcheckReason	Microsoft Corporation	C:\Windows\system32\drivers\Wdf01000.sys
0xFFFFF880012550DC	BugcheckReason	Microsoft Corporation	C:\Windows\system32\DRIVERS\ataport.SYS
0xFFFFF88007390008	ObProcess(PreCall)	-	C:\Users\Administrator\Desktop\MyDriver.sys
0xFFFFF8800806CC64	ObProcess(PreCall)	-	C:\Users\Administrator\AppData\Local\Temp\WIN64AST.sys
0xFFFFF88007390090	ObThread(PreCall)	-	C:\Users\Administrator\Desktop\MyDriver.sys
0xFFFFF8800806CE14	ObThread(PreCall)	-	C:\Users\Administrator\AppData\Local\Temp\WIN64AST.sys

用任务管理器结束 CALC.EXE 时，会提示失败：


无法中止进程

无法完成操作。
拒绝访问。

确定

Windows 任务管理器

文件 (F) 选项 (O) 查看 (V) 帮助 (H)

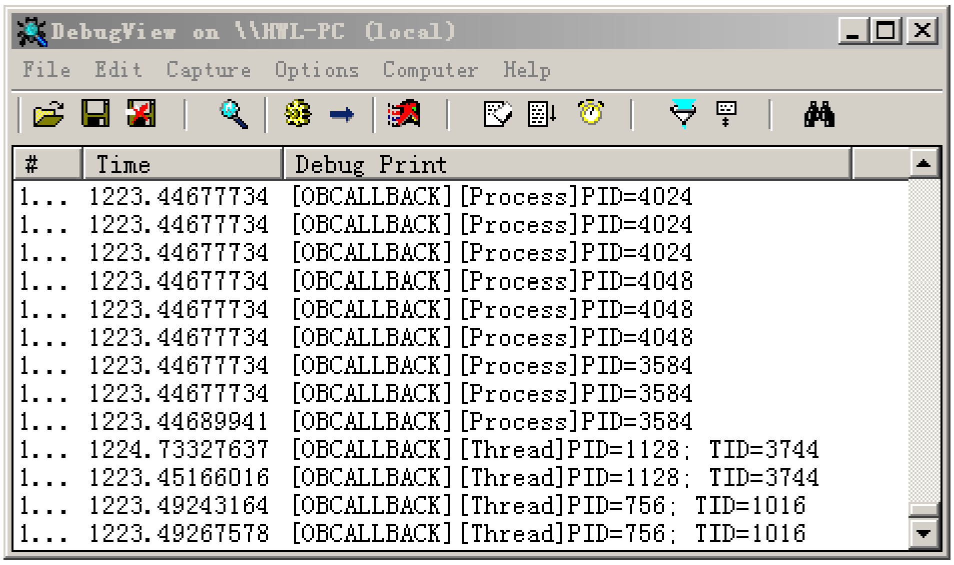
应用程序 | 进程 | 服务 | 性能 | 联网 | 用户

映像名称	用户名	CPU	内存(...	描述
audiodg.exe	LOCAL...	00	9,988 K	Windows...
calc.exe	Admin...	00	5,180 K	Windows...
csrss.exe	SYSTEM	00	1,436 K	Client ...
csrss.exe	SYSTEM	00	1,696 K	Client ...
Dbgview.ex...	Admin...	01	35,436 K	DebugView
dwm.exe	Admin...	00	1,280 K	桌面窗...
explorer.exe	Admin...	01	30,088 K	Windows...
fsproflt2....	SYSTEM	00	864 K	FSPro L...
GGService....	SYSTEM	00	2,044 K	GGServi...
hf.exe *32	Admin...	00	13,920 K	Hide Fo...
LoadDrv.ex...	Admin...	00	2,204 K	SetupDr...
lsass.exe	SYSTEM	00	3,152 K	Local S...
lsm.exe	SYSTEM	00	1,368 K	本地会...
SearchInde...	SYSTEM	00	7,620 K	Microso...
services.exe	SYSTEM	00	4,576 K	服务和...

☑ 显示所有用户的进程 (S)　　结束进程 (E)

进程数: 38　CPU 使用率: 3%　物理内存: 32%

打开 DBGVIEW，会看到一堆类似的输出：


DebugView on \\HWL-PC (local)
File Edit Capture Options Computer Help

```
#     Time           Debug Print
1...  1223.44677734  [OBCALLBACK][Process]PID=4024
1...  1223.44677734  [OBCALLBACK][Process]PID=4024
1...  1223.44677734  [OBCALLBACK][Process]PID=4024
1...  1223.44677734  [OBCALLBACK][Process]PID=4048
1...  1223.44677734  [OBCALLBACK][Process]PID=4048
1...  1223.44677734  [OBCALLBACK][Process]PID=4048
1...  1223.44677734  [OBCALLBACK][Process]PID=3584
1...  1223.44677734  [OBCALLBACK][Process]PID=3584
1...  1223.44689941  [OBCALLBACK][Process]PID=3584
1...  1224.73327637  [OBCALLBACK][Thread]PID=1128; TID=3744
1...  1223.45166016  [OBCALLBACK][Thread]PID=1128; TID=3744
1...  1223.49243164  [OBCALLBACK][Thread]PID=756; TID=1016
1...  1223.49267578  [OBCALLBACK][Thread]PID=756; TID=1016
```

课后作业：在回调里把进程的详细信息打印出来（比如进程名字等）。