MiniFilter 是目前杀毒软件用来实现“文件系统自我保护”和“文件实时监控”的方法。由于 MiniFilter 模型简单，开发快捷，通用性好，以前用 FSD HOOK 或者标准过滤驱动来实现相关功能的杀软纷纷改用 MiniFilter，比如卡巴斯基。不过，枚举 MiniFilter 就跟之前枚举回调的方法不太相同了，因为 MiniFilter 的框架不在 NTOSKRNL 里，自成一套系统，有专用的 API。不过“自成一套系统，有专用 API”的好处就是，无需我们自己通过特征码来定位数组或者链表，直接使用 MiniFilter 提供的 API 就行。

枚举 MiniFilter 主要使用 FltEnumerateFilters 这个 API，它会返回过滤器对象（FLT_FILTER）的地址，然后根据过滤器对象的地址，加上一个偏移，获得记录过滤器 PreCall、PostCall、IRP 等信息的结构体指针（PFLT_OPERATION_REGISTRATION）。上文之所以说要加上偏移，是因为 FLT_FILTER 的定义在每个系统都不同，比如 WIN7X64 中的定义为：

```
lkd> dt fltmgr!_FLT_FILTER
    +0x000 Base                       : _FLT_OBJECT
    +0x020 Frame                      : Ptr64 _FLTP_FRAME
    +0x028 Name                       : _UNICODE_STRING
    +0x038 DefaultAltitude    : _UNICODE_STRING
    +0x048 Flags                      : _FLT_FILTER_FLAGS
    +0x050 DriverObject         : Ptr64 _DRIVER_OBJECT
    +0x058 InstanceList         : _FLT_RESOURCE_LIST_HEAD
    +0x0d8 VerifierExtension : Ptr64 _FLT_VERIFIER_EXTENSION
    +0x0e0 VerifiedFiltersLink : _LIST_ENTRY
    +0x0f0 FilterUnload         : Ptr64          long
    +0x0f8 InstanceSetup         : Ptr64          long
    +0x100 InstanceQueryTeardown : Ptr64          long
    +0x108 InstanceTeardownStart : Ptr64          void
    +0x110 InstanceTeardownComplete : Ptr64          void
    +0x118 SupportedContextsListHead : Ptr64 _ALLOCATE_CONTEXT_HEADER
    +0x120 SupportedContexts : [6] Ptr64 _ALLOCATE_CONTEXT_HEADER
    +0x150 PreVolumeMount      : Ptr64          _FLT_PREOP_CALLBACK_STATUS
    +0x158 PostVolumeMount    : Ptr64          _FLT_POSTOP_CALLBACK_STATUS
    +0x160 GenerateFileName : Ptr64          long
    +0x168 NormalizeNameComponent : Ptr64          long
    +0x170 NormalizeNameComponentEx : Ptr64          long
    +0x178 NormalizeContextCleanup : Ptr64          void
    +0x180 KtmNotification    : Ptr64          long
    +0x188 Operations               : Ptr64 _FLT_OPERATION_REGISTRATION
    +0x190 OldDriverUnload    : Ptr64          void
    +0x198 ActiveOpens             : _FLT_MUTEX_LIST_HEAD
    +0x1e8 ConnectionList        : _FLT_MUTEX_LIST_HEAD
    +0x238 PortList                  : _FLT_MUTEX_LIST_HEAD
    +0x288 PortLock                   : _EX_PUSH_LOCK
```

不过幸好 FLT_OPERATION_REGISTRATION 的结构体定义是不变的：

```
lkd> dt_FLT_OPERATION_REGISTRATION
```

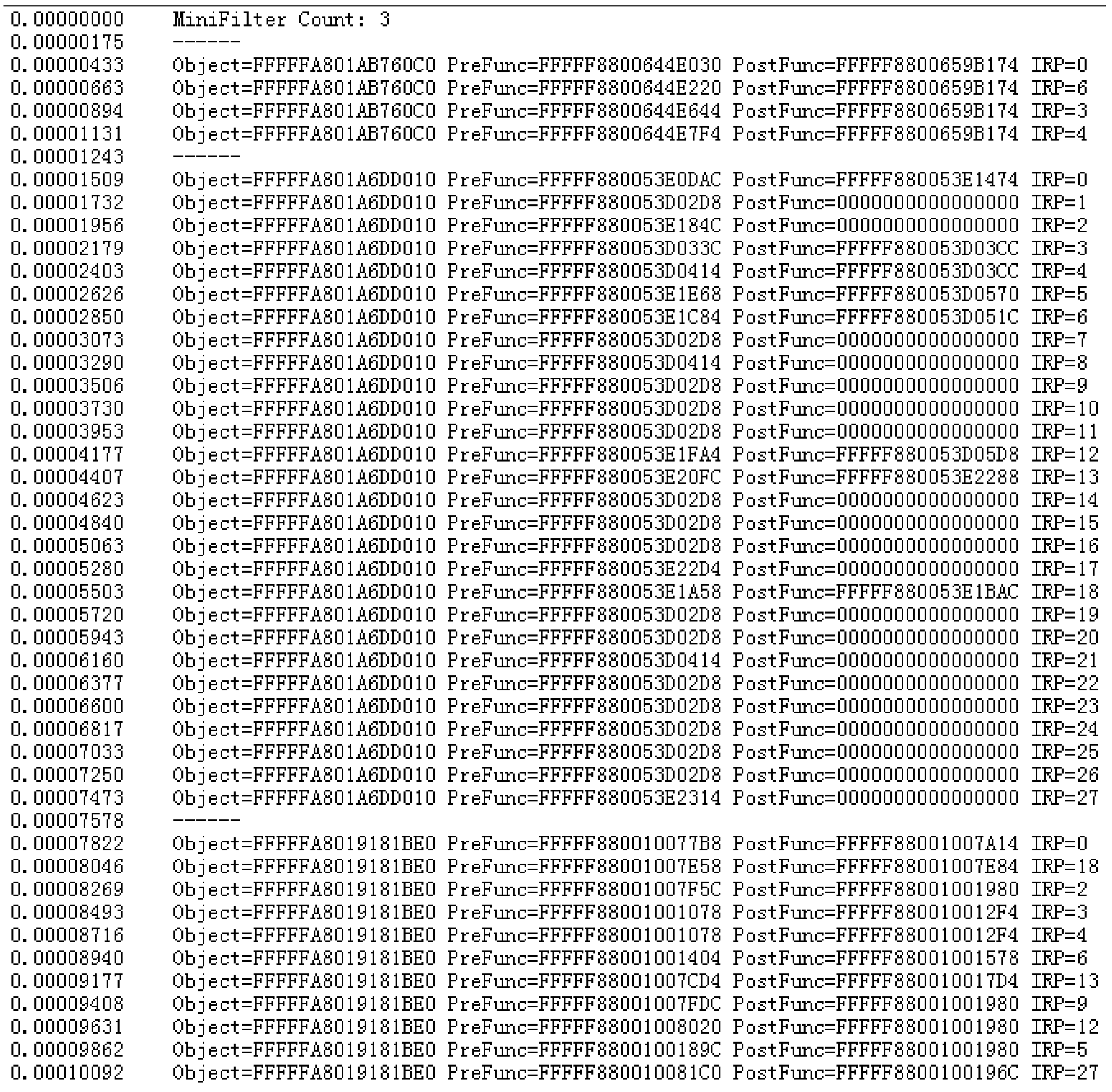
```
fltmgr!_FLT_OPERATION_REGISTRATION
    +0x000 MajorFunction        : UChar
    +0x004 Flags                : Uint4B
    +0x008 PreOperation         : Ptr64       _FLT_PREOP_CALLBACK_STATUS
    +0x010 PostOperation        : Ptr64       _FLT_POSTOP_CALLBACK_STATUS
    +0x018 Reserved1            : Ptr64 Void
```

枚举的代码如下：

```
ULONG EnumMiniFilter()
{
    long ntStatus;
    ULONG   uNumber;
    PVOID   pBuffer = NULL;
    ULONG   uIndex = 0, DrvCount = 0;
    PVOID   pCallBacks, pFilter;
    PFLT_OPERATION_REGISTRATION pNode;
    do
    {
        if(pBuffer != NULL)
        {
            ExFreePool(pBuffer);
            pBuffer = NULL;
        }
        ntStatus = FltEnumerateFilters(NULL,   0, &uNumber);
        if(ntStatus != STATUS_BUFFER_TOO_SMALL)
            break;
        pBuffer = ExAllocatePoolWithTag(NonPagedPool, sizeof(PFLT_FILTER) * uNumber, 
'mnft');
        if(pBuffer == NULL)
        {
            ntStatus = STATUS_INSUFFICIENT_RESOURCES;
            break;
        }
        ntStatus = FltEnumerateFilters(pBuffer, uNumber, &uNumber);
    }
    while (ntStatus == STATUS_BUFFER_TOO_SMALL);
    if(! NT_SUCCESS(ntStatus))
    {
        if(pBuffer != NULL)
            ExFreePool(pBuffer);
        return 0;
    }
    DbgPrint("MiniFilter Count: %ld\n",uNumber);
    DbgPrint("------\n");
```

```
	__try
	{
		while(DrvCount<uNumber)
		{
			pFilter = (PVOID)(*(PULONG64)((PUCHAR)pBuffer + DrvCount * 8));
			pCallBacks = (PVOID)((PUCHAR)pFilter + FltFilterOperationsOffset);
			pNode = (PFLT_OPERATION_REGISTRATION)(*(PULONG64)pCallBacks);
			__try
			{
				while(pNode->MajorFunction != 0x80)  //IRP_MJ_OPERATION_END
				{
					if(pNode->MajorFunction<28)      //MajorFunction id is 0~27
					{
						DbgPrint("Object=%p\tPreFunc=%p\tPostFunc=%p\tIRP=%d\n",
								pFilter,
								pNode->PreOperation,
								pNode->PostOperation,
								pNode->MajorFunction);
					}
					pNode++;
				}
			}
			__except(EXCEPTION_EXECUTE_HANDLER)
			{
				FltObjectDereference(pFilter);
				DbgPrint("[EnumMiniFilter]EXCEPTION_EXECUTE_HANDLER:
pNode->MajorFunction\n");
				ntStatus = GetExceptionCode();
				ExFreePool(pBuffer);
				return uIndex;
			}
			DrvCount++;
			FltObjectDereference(pFilter);
			DbgPrint("------\n");
		}
	}
	__except(EXCEPTION_EXECUTE_HANDLER)
	{
		FltObjectDereference(pFilter);
		DbgPrint("[EnumMiniFilter]EXCEPTION_EXECUTE_HANDLER\n");
		ntStatus = GetExceptionCode();
		ExFreePool(pBuffer);
		return uIndex;
	}
	if(pBuffer != NULL)
	{
		ExFreePool(pBuffer);
		ntStatus=STATUS_SUCCESS;
	}
	return uIndex;
}
```

代码执行的效果如下图（可以对比一下运行 WIN64AST 前后枚举的结果有什么不同）：



```
0.00000000	MiniFilter Count: 3
0.00000175	------
0.00000433	Object=FFFFFA801AB760C0 PreFunc=FFFFF8800644E030 PostFunc=FFFFF8800659B174 IRP=0
0.00000663	Object=FFFFFA801AB760C0 PreFunc=FFFFF8800644E220 PostFunc=FFFFF8800659B174 IRP=6
0.00000894	Object=FFFFFA801AB760C0 PreFunc=FFFFF8800644E644 PostFunc=FFFFF8800659B174 IRP=3
0.00001131	Object=FFFFFA801AB760C0 PreFunc=FFFFF8800644E7F4 PostFunc=FFFFF8800659B174 IRP=4
0.00001243	------
0.00001509	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053E0DAC PostFunc=FFFFF880053E1474 IRP=0
0.00001732	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=1
0.00001956	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053E184C PostFunc=0000000000000000 IRP=2
0.00002179	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D033C PostFunc=FFFFF880053D03CC IRP=3
0.00002403	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D0414 PostFunc=FFFFF880053D03CC IRP=4
0.00002626	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053E1E68 PostFunc=FFFFF880053D0570 IRP=5
0.00002850	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053E1C84 PostFunc=FFFFF880053D051C IRP=6
0.00003073	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=7
0.00003290	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D0414 PostFunc=0000000000000000 IRP=8
0.00003506	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=9
0.00003730	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=10
0.00003953	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=11
0.00004177	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053E1FA4 PostFunc=FFFFF880053D05D8 IRP=12
0.00004407	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053E20FC PostFunc=FFFFF880053E2288 IRP=13
0.00004623	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=14
0.00004840	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=15
0.00005063	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=16
0.00005280	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053E22D4 PostFunc=0000000000000000 IRP=17
0.00005503	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053E1A58 PostFunc=FFFFF880053E1BAC IRP=18
0.00005720	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=19
0.00005943	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=20
0.00006160	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D0414 PostFunc=0000000000000000 IRP=21
0.00006377	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=22
0.00006600	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=23
0.00006817	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=24
0.00007033	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=25
0.00007250	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053D02D8 PostFunc=0000000000000000 IRP=26
0.00007473	Object=FFFFFA801A6DD010 PreFunc=FFFFF880053E2314 PostFunc=0000000000000000 IRP=27
0.00007578	------
0.00007822	Object=FFFFFA8019181BE0 PreFunc=FFFFF880010077B8 PostFunc=FFFFF88001007A14 IRP=0
0.00008046	Object=FFFFFA8019181BE0 PreFunc=FFFFF88001007E58 PostFunc=FFFFF88001007E84 IRP=18
0.00008269	Object=FFFFFA8019181BE0 PreFunc=FFFFF88001007F5C PostFunc=FFFFF88001001980 IRP=2
0.00008493	Object=FFFFFA8019181BE0 PreFunc=FFFFF88001001078 PostFunc=FFFFF880010012F4 IRP=3
0.00008716	Object=FFFFFA8019181BE0 PreFunc=FFFFF88001001078 PostFunc=FFFFF880010012F4 IRP=4
0.00008940	Object=FFFFFA8019181BE0 PreFunc=FFFFF88001001404 PostFunc=FFFFF88001001578 IRP=6
0.00009177	Object=FFFFFA8019181BE0 PreFunc=FFFFF88001007CD4 PostFunc=FFFFF880010017D4 IRP=13
0.00009408	Object=FFFFFA8019181BE0 PreFunc=FFFFF88001007FDC PostFunc=FFFFF88001001980 IRP=9
0.00009631	Object=FFFFFA8019181BE0 PreFunc=FFFFF88001008020 PostFunc=FFFFF88001001980 IRP=12
0.00009862	Object=FFFFFA8019181BE0 PreFunc=FFFFF8800100189C PostFunc=FFFFF88001001980 IRP=5
0.00010092	Object=FFFFFA8019181BE0 PreFunc=FFFFF880010081C0 PostFunc=FFFFF8800100196C IRP=27
```

不过对抗 MiniFilter 似乎就只有两种方法了：1.把记录的函数地址改为自己设置的空函数；2.把处理函数头改为 RET 直接返回。**为什么不能直接把 MiniFilter 对象反注册呢？因为** **MSDN 对 FltUnregisterFilter 的用途给出了这样的解释**：**A minifilter driver can only call FltUnregisterFilter to unregister itself, not another minifilter driver**。据我测试，如果第三方驱动强制使用此函数注销一个 MiniFilter，轻则无效，重则蓝屏。

把 MINIFILTER 的处理函数禁用掉之后，卡巴斯基 2013 在 WIN64 系统上的文件保护就彻

底失效了，可以直接使用最简单的方法来删除卡巴斯基文件夹内的文件，国内那些采用同样方法实现文件自我保护的杀毒软件（360、金山毒霸等）同理。