

Minifilter 是一种文件过滤驱动，该驱动简称为微过滤驱动，相对于传统的 sfilter 文件过滤驱动来说，微过滤驱动编写时更简单，其不需要考虑底层RIP如何派发且无需要考虑兼容性问题，微过滤驱动使用过滤管理器 FilterManager 提供接口，由于提供了管理结构以及一系列管理API函数，所以枚举过滤驱动将变得十分容易。

通常文件驱动过滤是ARK重要功能之一，如下是一款闭源ARK工具的输出效果图。

进程	驱动模块	内核	内核钩子	应用层钩子	网络	注册表	文件	启动信息	监控	系统杂项	关于
系统回调	过滤驱动	DPC定时器	工作队列线程	Hal	Wdf	文件系统	系统调试	对象劫持	直接IO	GDT	
文件系统项目											
微端口过滤器											
文件系统											
SFilter回调											
ClassInitData回调											
Npfs派发函数											
Msfs派发函数											
Usbport派发函数											
			函数名称	函数地址	函数所在模块	文件厂商	过滤器地址	高度			
			FilterUnload	0xFFFFF80209197640	C:\Windows\s...	Microsof...	0xFFFFF838E93D7E010	328010			
			Instance...	0xFFFFF80209197940	C:\Windows\s...	Microsof...	0xFFFFF838E93D7E010	328010			
			Instance...	0xFFFFF80209197B50	C:\Windows\s...	Microsof...	0xFFFFF838E93D7E010	328010			
			Instance...	0xFFFFF80209197B80	C:\Windows\s...	Microsof...	0xFFFFF838E93D7E010	328010			
			KtmNotifi...	0xFFFFF8020919C620	C:\Windows\s...	Microsof...	0xFFFFF838E93D7E010	328010			
			IRP_MJ_...	0xFFFFF8020918D920	C:\Windows\s...	Microsof...	0xFFFFF838E93D7E010	328010			
			IRP_MJ_...	0xFFFFF8020918F600	C:\Windows\s...	Microsof...	0xFFFFF838E93D7E010	328010			

由于 MiniFilter 提供了 FltEnumerateFilters 函数，所以只需要调用这些函数即可获取到所有的过滤器地址，我们看下微软公开的信息。

```
NTSTATUS FLTAPI FltEnumerateFilters(
    [out] PFLT_FILTER *FilterList,
    [in]  ULONG        FilterListSize,
    [out] PULONG       NumberFiltersReturned
);
```

此函数需要注意，如果用户将 FilterList 设置为 NULL 则默认是输出当前系统中存在的过滤器数量，而如果传入的是一个内存地址，则将会枚举系统中所有的过滤器信息。

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 PFLT_OPERATION_REGISTRATION 结构体指针。

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 WIN10 x64 中的定义以下样子，这里我们需要记下 +0x1a8 operations 因为他指向的就是 _FLT_OPERATION_REGISTRATION 结构的偏移地址。

```
tyshark.com: kd> dt fltmgr!_FLT_FILTER
+0x000 Base           : _FLT_OBJECT
+0x030 Frame          : Ptr64 _FLTP_FRAME
+0x038 Name           : _UNICODE_STRING
+0x048 DefaultAltitude : _UNICODE_STRING
+0x058 Flags          : _FLT_FILTER_FLAGS
+0x060 DriverObject   : Ptr64 _DRIVER_OBJECT
+0x068 InstanceList   : _FLT_RESOURCE_LIST_HEAD
+0x0e8 VerifierExtension : Ptr64 _FLT_VERIFIER_EXTENSION
+0x0f0 VerifiedFiltersLink : _LIST_ENTRY
+0x100 FilterUnload    : Ptr64 long
+0x108 InstanceSetup   : Ptr64 long
+0x110 InstanceQueryTeardown : Ptr64 long
+0x118 InstanceTeardownStart : Ptr64 void
+0x120 InstanceTeardownComplete : Ptr64 void
+0x128 SupportedContextsListHead : Ptr64 _ALLOCATE_CONTEXT_HEADER
+0x130 SupportedContexts : [7] Ptr64 _ALLOCATE_CONTEXT_HEADER
+0x168 PreVolumeMount : Ptr64 _FLT_PREOP_CALLBACK_STATUS
+0x170 PostVolumeMount : Ptr64 _FLT_POSTOP_CALLBACK_STATUS
```

```

+0x178 GenerateFileName : Ptr64      long
+0x180 NormalizeNameComponent : Ptr64      long
+0x188 NormalizeNameComponentEx : Ptr64      long
+0x190 NormalizeContextCleanup : Ptr64      void
+0x198 KtmNotification : Ptr64      long
+0x1a0 SectionNotification : Ptr64      long
+0x1a8 Operations : Ptr64 _FLT_OPERATION_REGISTRATION
+0x1b0 OldDriverUnload : Ptr64      void
+0x1b8 ActiveOpens : _FLT_Mutex_LIST_HEAD
+0x208 ConnectionList : _FLT_Mutex_LIST_HEAD
+0x258 PortList : _FLT_Mutex_LIST_HEAD
+0x2a8 PortLock : _EX_PUSH_LOCK

```

解析 `_FLT_OPERATION_REGISTRATION` 结构体，可以看到这就是我们需要枚举的过滤器，只要拿到它输出即可：

```

lyshark.com: kd> dt 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

```

枚举过滤器代码如下所示，需要配置连接器增加 `fltMgr.lib` 头文件。

- 配置属性 > 连接器 > 输入 > 附加依赖 -> `fltMgr.lib`
- 配置属性 > C/C++ > 常规 > 设置 警告等级2级 (警告视为错误关闭)

```

#include <fltkernel.h>
#include <dontuse.h>
#include <suppress.h>

// 设置默认回调
NTSTATUS DriverDefaultHandle(PDEVICE_OBJECT pDevObj, PIRP pIrp)
{
    NTSTATUS status = STATUS_SUCCESS;
    pIrp->IoStatus.Status = status;
    pIrp->IoStatus.Information = 0;
    IoCompleteRequest(pIrp, IO_NO_INCREMENT);

    return status;
}

VOID DriverUnload(PDRIVER_OBJECT pDriverObject)
{
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriverObject, PUNICODE_STRING pRegPath)
{
    DbgPrint("hello lyshark.com \n");

    NTSTATUS status = STATUS_SUCCESS;
    pDriverObject->DriverUnload = DriverUnload;
}

```

```

for (ULONG i = 0; i < IRP_MJ_MAXIMUM_FUNCTION; i++)
{
    pDriverObject->MajorFunction[i] = DriverDefaultHandle;
}

ULONG ulFilterListSize = 0;
PFLT_FILTER *ppFilterList = NULL;
ULONG i = 0;
LONG lOperationsOffset = 0;
PFLT_OPERATION_REGISTRATION pFltOperationRegistration = NULL;

// 获取 Minifilter 过滤器Filter 的数量
FltEnumerateFilters(NULL, 0, &ulFilterListSize);

// 申请内存
ppFilterList = (PFLT_FILTER *)ExAllocatePool(NonPagedPool, ulFilterListSize
*sizeof(PFLT_FILTER));
if (NULL == ppFilterList)
{
    return FALSE;
}

// 获取 Minifilter 中所有过滤器Filter 的信息
status = FltEnumerateFilters(ppFilterList, ulFilterListSize, &ulFilterListSize);
if (!NT_SUCCESS(status))
{
    return FALSE;
}

DbgPrint("过滤器数量: %d \n", ulFilterListSize);

// 获取 PFLT_FILTER 中 Operations 偏移
lOperationsOffset = 0x1A8;

// 开始遍历 Minifilter
__try
{
    for (i = 0; i < ulFilterListSize; i++)
    {
        // 获取 PFLT_FILTER 中 Operations 成员地址
        pFltOperationRegistration = (PFLT_OPERATION_REGISTRATION)(*(PVOID *)
((PUCHAR)ppFilterList[i] + lOperationsOffset));

        __try
        {
            // 同一过滤器下的回调信息
            while (IRP_MJ_OPERATION_END != pFltOperationRegistration->MajorFunction)
            {
                if (IRP_MJ_MAXIMUM_FUNCTION > pFltOperationRegistration->MajorFunction)
                {
                    // 显示
                    DbgPrint("Filter: %p | IRP: %d | PreFunc: 0x%p | PostFunc=0x%p \n",
ppFilterList[i], pFltOperationRegistration->MajorFunction,

```

```

        pFltOperationRegistration->PreOperation,
pFltOperationRegistration->PostOperation);
    }

    // 获取下一个消息回调信息
    pFltOperationRegistration = (PFLT_OPERATION_REGISTRATION)
((PUCHAR)pFltOperationRegistration + sizeof(FLT_OPERATION_REGISTRATION));
    }
}
__except (EXCEPTION_EXECUTE_HANDLER)
{
}
}
__except (EXCEPTION_EXECUTE_HANDLER)
{
}

// 释放内存
ExFreePool(ppFilterList);
ppFilterList = NULL;

return status;
}

```

运行代码输出枚举效果如下：

#	Time	Debug Print
3	0.3...	hello lyshark.com
4	0.3...	过滤器数量: 9
5	0.3...	Filter: FFFF838E93D7E010 IRP: 0 PreFunc: 0xFFFFF8020918D920 PostFunc=0xFFFFF8020918FC90
6	0.3...	Filter: FFFF838E93D7E010 IRP: 18 PreFunc: 0xFFFFF802091872B0 PostFunc=0x0000000000000000
7	0.3...	Filter: FFFF838E93D7E010 IRP: 6 PreFunc: 0xFFFFF80209193610 PostFunc=0xFFFFF80209193C90
8	0.3...	Filter: FFFF838E93D7E010 IRP: 4 PreFunc: 0xFFFFF802091752D0 PostFunc=0xFFFFF802091759D0
9	0.3...	Filter: FFFF838E93D7E010 IRP: 13 PreFunc: 0xFFFFF802091A70C0 PostFunc=0xFFFFF802091A7410
10	0.3...	Filter: FFFF838E93D7E010 IRP: 12 PreFunc: 0x0000000000000000 PostFunc=0xFFFFF80209178070
11	0.3...	Filter: FFFF838E93D7E010 IRP: 3 PreFunc: 0x0000000000000000 PostFunc=0xFFFFF80209175C20
12	0.3...	Filter: FFFF838E93D7E010 IRP: 7 PreFunc: 0xFFFFF802091A8160 PostFunc=0x0000000000000000
13	0.3...	Filter: FFFF838E968896E0 IRP: 0 PreFunc: 0xFFFFF8020B810F10 PostFunc=0xFFFFF8020B810FA0
14	0.3...	Filter: FFFF838E968896E0 IRP: 2 PreFunc: 0xFFFFF8020B811130 PostFunc=0x0000000000000000
15	0.3...	Filter: FFFF838E968896E0 IRP: 3 PreFunc: 0xFFFFF8020B810670 PostFunc=0xFFFFF8020B802F50
16	0.3...	Filter: FFFF838E968896E0 IRP: 4 PreFunc: 0xFFFFF8020B810670 PostFunc=0xFFFFF8020B802F50
17	0.4...	Filter: FFFF838E968896E0 IRP: 13 PreFunc: 0xFFFFF8020B810010 PostFunc=0xFFFFF8020B803C40
18	0.4...	Filter: FFFF838E969D98E0 IRP: 0 PreFunc: 0xFFFFF8020BB441D0 PostFunc=0xFFFFF8020BB44910