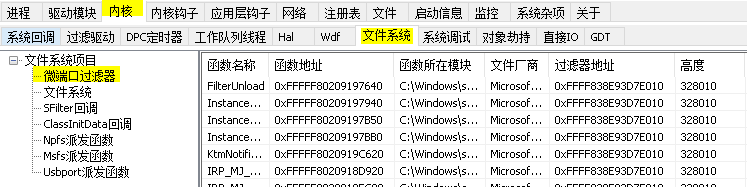
# Minifilter 是一种文件过滤驱动，该驱动简称为微过滤驱动，相对于传统的 sfilter 文件过滤驱动来

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。



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。

lyshark.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

+0x170 PostVolumeMount : Ptr64

+0x178 GenerateFileName : Ptr64

\_FLT\_PREOP\_CALLBACK\_STATUS

\_FLT\_POSTOP\_CALLBACK\_STATUS

long

+0x180 NormalizeNameComponent : Ptr64 long

+0x188 NormalizeNameComponentEx : Ptr64 long

+0x190 NormalizeContextCleanup : Ptr64 void

+0x198 KtmNotification : Ptr64 long

+0x1a0 SectionNotification : Ptr64 long

+0x1a8 Operations

+0x1b0 OldDriverUnload

+0x1b8 ActiveOpens

+0x208 ConnectionList

+0x258 PortList

+0x2a8 PortLock

: Ptr64 \_FLT\_OPERATION\_REGISTRATION

: Ptr64

void

: \_FLT\_MUTEX\_LIST\_HEAD

: \_FLT\_MUTEX\_LIST\_HEAD

: \_FLT\_MUTEX\_LIST\_HEAD

: \_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：

\_FLT\_PREOP\_CALLBACK\_STATUS

\_FLT\_POSTOP\_CALLBACK\_STATUS

: UChar

: Uint4B

: Ptr64

: Ptr64

: Ptr64 Void

+0x000 MajorFunction

+0x004 Flags

+0x008 PreOperation

+0x010 PostOperation

+0x018 Reserved1

lyshark.com: kd> dt fltmgr!\_FLT\_OPERATION\_REGISTRATION

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

{



>MajorFunction)

>MajorFunction)

// 同一过滤器下的回调信息

while (IRP\_MJ\_OPERATION\_END != pFltOperationRegistration-

{

if (IRP\_MJ\_MAXIMUM\_FUNCTION > pFltOperationRegistration-

{

// 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)

{

}

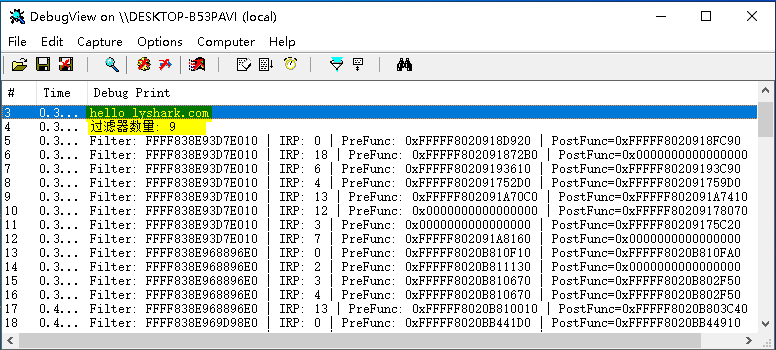
// 释放内存

ExFreePool(ppFilterList); ppFilterList = NULL;

return status;

}

# 运行代码输出枚举效果如下：



作者： 王瑞 (LyShark)

作者邮箱： m [e@lyshark.com](mailto:e@lyshark.com)

版权声明：本博客文章与代码均为学习时整理的笔记，文章 [均为原创] 作品，转载文章请遵守

《中华人民共和国著作权法》相关法律规定或遵守《署名CC BY-ND 4.0国际》规范，合理合规携带原创出处转载，如果不携带文章出处，并恶意转载多篇原创文章被本人发现，本人保留起诉权！