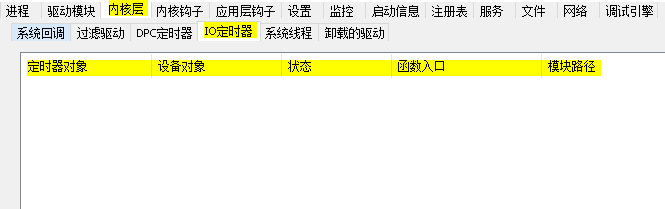
今天继续分享内核枚举系列知识，这次我们来学习如何通过代码的方式枚举内核 IoTimer 定时器，内核定时器其实就是在内核中实现的时钟，该定时器的枚举非常简单，因为在 IoInitializeTimer 初始化部分就可以找到 IopTimerQueueHead 地址，该变量内存储的就是定时器的链表头部。枚举IO定时器的案例并不多见，即便有也是无法使用过时的，此教程学到肯定就是赚到了。



# 枚举Io定时器过程是这样的：

1.找到 IoInitializeTimer 函数，该函数可以通过 MmGetSystemRoutineAddress 得到。

2.找到地址以后，我们向下增加 0xFF 偏移量，并搜索特征定位到 IopTimerQueueHead 链表头。

3.将链表头转换为 IO\_TIMER 结构体，并循环链表头输出。

# 这里解释一下为什么要找 IoInitializeTimer 这个函数他是一个初始化函数，既然是初始化里面一定会涉及到链表的存储问题，找到他就能找到定时器链表基址，该函数的定义如下。

IN PIO\_TIMER\_ROUTINE TimerRoutine, // 定时器例程

IN PVOID Context // 传给定时器例程的函数

);

// 设备对象指针

NTSTATUS

IoInitializeTimer(

IN PDEVICE\_OBJECT DeviceObject,

接着我们需要得到IO定时器的结构定义，在 DEVICE\_OBJECT 设备对象指针中存在一个 Timer 属性。

lyshark.com: kd> dt \_DEVICE\_OBJECT ntdll!\_DEVICE\_OBJECT

+0x000 Type : Int2B

+0x002 Size : Uint2B

+0x004 ReferenceCount : Int4B

+0x008 DriverObject : Ptr64 \_DRIVER\_OBJECT

+0x010 NextDevice : Ptr64 \_DEVICE\_OBJECT

+0x018 AttachedDevice : Ptr64 \_DEVICE\_OBJECT

+0x020 CurrentIrp : Ptr64 \_IRP

+0x028 Timer : Ptr64 \_IO\_TIMER

+0x030 Flags : Uint4B

+0x034 Characteristics : Uint4B

+0x038 Vpb : Ptr64 \_VPB

+0x040 DeviceExtension : Ptr64 Void

+0x048 DeviceType : Uint4B

+0x04c StackSize : Char

+0x050 Queue : <anonymous-tag>

+0x098 AlignmentRequirement : Uint4B

+0x0a0 DeviceQueue : \_KDEVICE\_QUEUE

+0x0c8 Dpc

: \_KDPC

+0x108 ActiveThreadCount : Uint4B

+0x110 SecurityDescriptor : Ptr64 Void

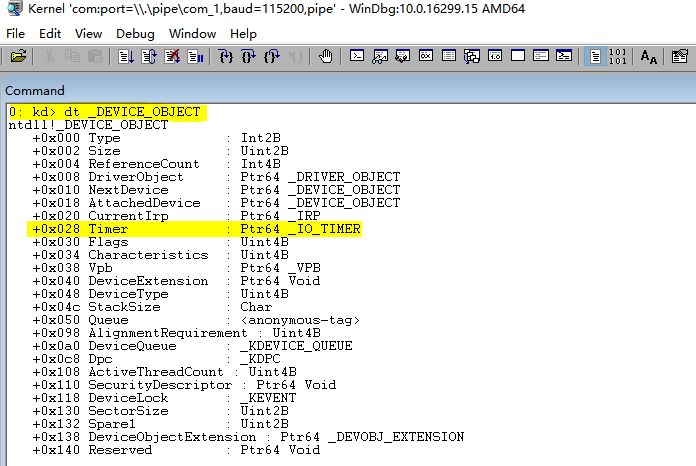
+0x118 DeviceLock : \_KEVENT

+0x130 SectorSize : Uint2B

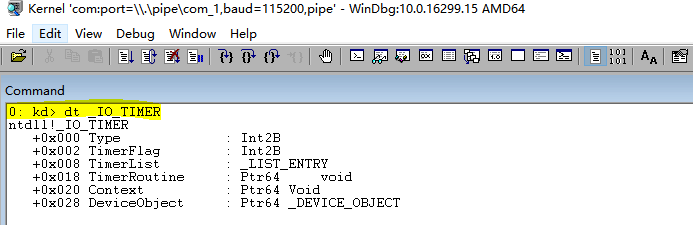
+0x132 Spare1 : Uint2B

+0x138 DeviceObjectExtension : Ptr64 \_DEVOBJ\_EXTENSION

+0x140 Reserved : Ptr64 Void



这里的这个 +0x028 Timer 定时器是一个结构体 \_IO\_TIMER 其就是IO定时器的所需结构体。



: Ptr64 Void

: Ptr64 \_DEVICE\_OBJECT

void

: Ptr64

: Int2B

: Int2B

: \_LIST\_ENTRY

+0x000 Type

+0x002 TimerFlag

+0x008 TimerList

+0x018 TimerRoutine

+0x020 Context

+0x028 DeviceObject

lyshark.com: kd> dt \_IO\_TIMER

ntdll!\_IO\_TIMER

# 如上方的基础知识有了也就够了，接着就是实际开发部分，首先我们需要编写一个

GetIoInitializeTimerAddress() 函数，让该函数可以定位到 IoInitializeTimer 所在内核中的基地址上面，具体实现调用代码如下所示。

#include <ntifs.h>

// 得到IoInitializeTimer基址

// By: LyShark 内核开发系列教程

PVOID GetIoInitializeTimerAddress()

{

PVOID VariableAddress = 0; UNICODE\_STRING uioiTime = { 0 };

RtlInitUnicodeString(&uioiTime, L"IoInitializeTimer"); VariableAddress = (PVOID)MmGetSystemRoutineAddress(&uioiTime); if (VariableAddress != 0)

{

return VariableAddress;

}

return 0;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint(("Uninstall Driver Is OK \n"));

}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint(("hello lyshark.com \n"));

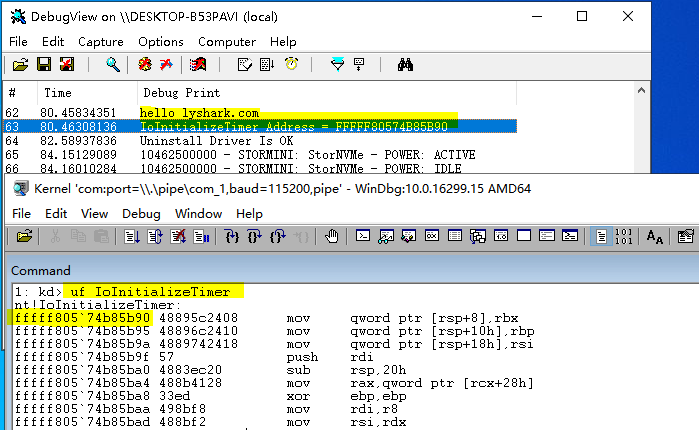
// 得到基址

PUCHAR IoInitializeTimer = GetIoInitializeTimerAddress(); DbgPrint("IoInitializeTimer Address = %p \n", IoInitializeTimer);

Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 运行这个驱动程序，然后对比下是否一致:



接着我们在反汇编代码中寻找 IoTimerQueueHead ，此处在LyShark系统内这个偏移位置是

# 具体输出位置如下。

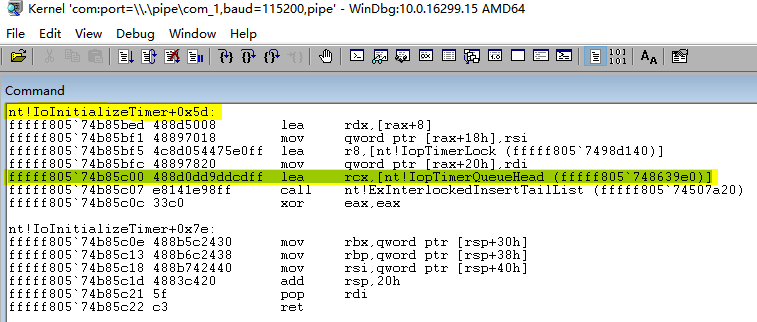
nt!IoInitializeTimer+0x5d

|  |  |  |
| --- | --- | --- |
| fffff805`74b85bed 488d5008 | lea | rdx,[rax+8] |
| fffff805`74b85bf1 48897018 | mov | qword ptr [rax+18h],rsi |
| fffff805`74b85bf5 4c8d054475e0ff | lea | r8,[nt!IopTimerLock |
| (fffff805`7498d140)]  fffff805`74b85bfc 48897820 | mov | qword ptr [rax+20h],rdi |
| fffff805`74b85c00 488d0dd9ddcdff | lea | rcx,[nt!IopTimerQueueHead |
| (fffff805`748639e0)]  fffff805`74b85c07 e8141e98ff | call | nt!ExInterlockedInsertTailList |
| (fffff805`74507a20) |  |  |
| fffff805`74b85c0c 33c0 | xor | eax,eax |

在WinDBG中标注出颜色 lea rcx,[nt!IopTimerQueueHead (fffff805748639e0)] 更容易看到。

lyshark.com: kd> uf IoInitializeTimer

nt!IoInitializeTimer+0x5d:



# 接着就是通过代码实现对此处的定位，定位我们就采用特征码搜索的方式，如下代码是特征搜索部分。

StartSearchAddress 代表开始位置

EndSearchAddress 代表结束位置，粗略计算0xff就可以定位到了。

#include <ntifs.h>

// 得到IoInitializeTimer基址

// By: LyShark 内核开发系列教程

PVOID GetIoInitializeTimerAddress()

{

PVOID VariableAddress = 0; UNICODE\_STRING uioiTime = { 0 };

RtlInitUnicodeString(&uioiTime, L"IoInitializeTimer"); VariableAddress = (PVOID)MmGetSystemRoutineAddress(&uioiTime); if (VariableAddress != 0)

{

return VariableAddress;

}

return 0;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint(("Uninstall Driver Is OK \n"));

}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint(("hello lyshark.com \n"));

// 得到基址

PUCHAR IoInitializeTimer = GetIoInitializeTimerAddress(); DbgPrint("IoInitializeTimer Address = %p \n", IoInitializeTimer);

INT32 iOffset = 0;

PLIST\_ENTRY IoTimerQueueHead = NULL;

PUCHAR StartSearchAddress = IoInitializeTimer; PUCHAR EndSearchAddress = IoInitializeTimer + 0xFF; UCHAR v1 = 0, v2 = 0, v3 = 0;

for (PUCHAR i = StartSearchAddress; i < EndSearchAddress; i++)

{

+ 2))

if (MmIsAddressValid(i) && MmIsAddressValid(i + 1) && MmIsAddressValid(i

{

v1 = i;

v2 = (i + 1);

v3 = (i + 2);

// 三个特征码

if (v1 == 0x48 && v2 == 0x8d && v3 == 0x0d)

{

memcpy(&iOffset, i + 3, 4);



IoTimerQueueHead = (PLIST\_ENTRY)(iOffset + (ULONG64)i + 7);

DbgPrint("IoTimerQueueHead = %p \n", IoTimerQueueHead); break;

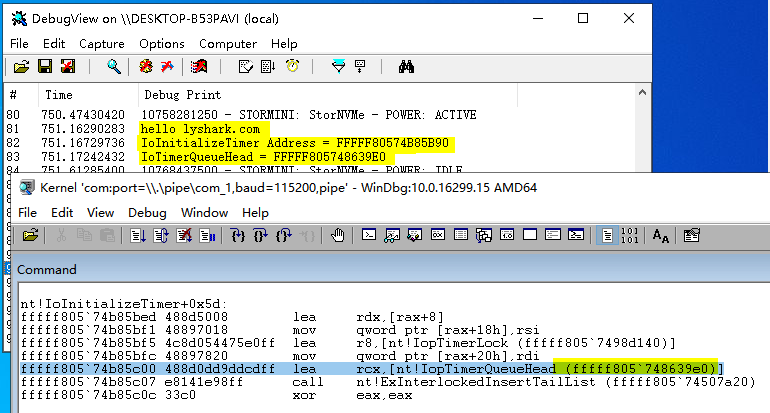
}

}

}

Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

搜索三个特征码 v1 == 0x48 && v2 == 0x8d && v3 == 0x0d 从而得到内存位置，运行驱动对比下。运行代码会取出 lea 指令后面的操作数，而不是取出lea指令的内存地址。

最后一步就是枚举部分，我们需要前面提到的 IO\_TIMER 结构体定义。

# PIO\_TIMER Timer = CONTAINING\_RECORD(NextEntry, IO\_TIMER, TimerList) 得到结构体，循环输出即可。



// By: LyShark 内核开发系列教程

// [https://www.cnblogs.com/LyShark/articles/16784393.html](http://www.cnblogs.com/LyShark/articles/16784393.html) #include <ntddk.h>

#include <ntstrsafe.h>

typedef struct \_IO\_TIMER

{

INT16 Type;

INT16 TimerFlag;

LONG32 Unknown; LIST\_ENTRY TimerList; PVOID TimerRoutine;

PVOID Context;

PVOID DeviceObject;

}IO\_TIMER, PIO\_TIMER;

// 得到IoInitializeTimer基址

PVOID GetIoInitializeTimerAddress()

{

PVOID VariableAddress = 0; UNICODE\_STRING uioiTime = { 0 };

RtlInitUnicodeString(&uioiTime, L"IoInitializeTimer"); VariableAddress = (PVOID)MmGetSystemRoutineAddress(&uioiTime); if (VariableAddress != 0)

{

return VariableAddress;

}

return 0;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("卸载完成... \n");

}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint(("hello lyshark.com \n"));

// 得到基址

PUCHAR IoInitializeTimer = GetIoInitializeTimerAddress(); DbgPrint("IoInitializeTimer Address = %p \n", IoInitializeTimer);

// 搜索IoTimerQueueHead地址

/

nt!IoInitializeTimer+0x5d:

fffff806`349963cd 488d5008 lea rdx,[rax+8] fffff806`349963d1 48897018 mov qword ptr [rax+18h],rsi fffff806`349963d5 4c8d05648de0ff lea r8,[nt!IopTimerLock

(fffff806`3479f140)]

fffff806`349963dc 48897820 mov qword ptr [rax+20h],rdi fffff806`349963e0 488d0d99f6cdff lea rcx,[nt!IopTimerQueueHead

(fffff806`34675a80)]

fffff806`349963e7 e8c43598ff call nt!ExInterlockedInsertTailList (fffff806`343199b0)

fffff806`349963ec 33c0 xor eax,eax

/

INT32 iOffset = 0;

PLIST\_ENTRY IoTimerQueueHead = NULL;

PUCHAR StartSearchAddress = IoInitializeTimer; PUCHAR EndSearchAddress = IoInitializeTimer + 0xFF; UCHAR v1 = 0, v2 = 0, v3 = 0;

for (PUCHAR i = StartSearchAddress; i < EndSearchAddress; i++)

{



2))

if (MmIsAddressValid(i) && MmIsAddressValid(i + 1) && MmIsAddressValid(i +

{

v1 = i;

v2 = (i + 1);

v3 = (i + 2);

// fffff806`349963e0 48 8d 0d 99 f6 cd ff lea rcx,[nt!IopTimerQueueHead (fffff806`34675a80)]

if (v1 == 0x48 && v2 == 0x8d && v3 == 0x0d)

{

memcpy(&iOffset, i + 3, 4);

IoTimerQueueHead = (PLIST\_ENTRY)(iOffset + (ULONG64)i + 7); DbgPrint("IoTimerQueueHead = %p \n", IoTimerQueueHead); break;

}

}

}

// 枚举列表

KIRQL OldIrql;

// 获得特权级

OldIrql = KeRaiseIrqlToDpcLevel();

if (IoTimerQueueHead && MmIsAddressValid((PVOID)IoTimerQueueHead))

{

PLIST\_ENTRY NextEntry = IoTimerQueueHead->Flink; while (MmIsAddressValid(NextEntry) && NextEntry !=

(PLIST\_ENTRY)IoTimerQueueHead)

{

PIO\_TIMER Timer = CONTAINING\_RECORD(NextEntry, IO\_TIMER, TimerList);

if (Timer && MmIsAddressValid(Timer))

{

DbgPrint("IO对象地址: %p \n", Timer);

}

NextEntry = NextEntry->Flink;

}

}

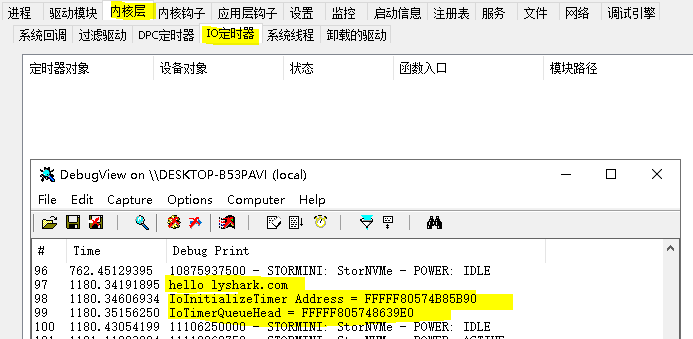
// 恢复特权级

KeLowerIrql(OldIrql);

Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 运行这段源代码，并可得到以下输出，由于没有IO定时器所以输出结果是空的：



至此IO定时器的枚举就介绍完了，在教程中你已经学会了使用特征码定位这门技术，相信你完全可以输 出内核中想要得到的任何结构体。

作者： 王瑞 (LyShark)

作者邮箱： [me@lyshark.com](mailto:me@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！