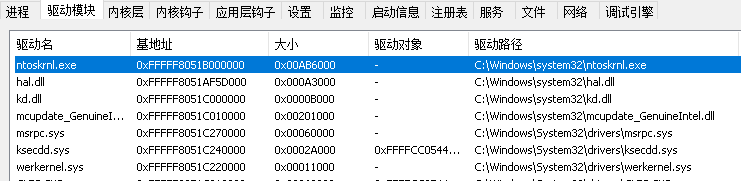
# 模块是程序加载时被动态装载的，模块在装载后其存在于内存中同样存在一个内存基址，当我们需要操 作这个模块时，通常第一步就是要得到该模块的内存基址，模块分为用户模块和内核模块，这里的用户 模块指的是应用层进程运行后加载的模块，内核模块指的是内核中特定模块地址，本篇文章将实现一个 获取驱动 ntoskrnl.exe 的基地址以及长度，此功能是驱动开发中尤其是安全软件开发中必不可少的一个功能。

关于该程序的解释，官方的解析是这样的 ntoskrnl.exe 是 Windows 操作系统的一个重要内核程序，里面存储了大量的二进制内核代码，用于调度系统时使用，也是操作系统启动后第一个被加载的程序，通 常该进程在任务管理器中显示为 System 。

# 使用ARK工具也可看出其代表的是第一个驱动模块。



那么如何使用代码得到如上图中所展示的 基地址 以及 大小 呢，实现此功能我们需要调用ZwQuerySystemInformation 这个API函数，这与上一篇文章 《驱动开发：判断自身是否加载成功》 所使用的 NtQuerySystemInformation 只是开头部分不同，但其本质上是不同的，如下是一些参考资料；

从内核模式调用 Nt 和 Zw 系列API，其最终都会连接到 nooskrnl.lib 导出库:

Nt系列API将直接调用对应的函数代码，而Zw系列API则通过调用 KiSystemService 最终跳转到对应的函数代码。

重要的是两种不同的调用对内核中 previous mode 的改变，如果是从用户模式调用 Native API 则 previous mode 是用户态，如果从内核模式调用 Native API 则 previous mode 是内核态。

如果 previous 为用户态时 Native API 将对传递的参数进行严格的检查，而为内核态时则不会检查。

调用 Nt API 时不会改变 previous mode 的状态，调用 Zw API 时会将 previous mode 改为内核态，因此在进行 Kernel Mode Driver 开发时可以使用 Zw 系列API可以避免额外的参数列表检查，提高效率。Zw 会设置 KernelMode 已避免检查， Nt 不会自动设置，如果是 KernelMode 当然没问题，如果就



UserMode 就挂了。

回到代码上来，下方代码就是获取 ntoskrnl.exe 基地址以及长度的具体实现，核心代码就是调用

ZwQuerySystemInformation 得到 SystemModuleInformation ，里面的对比部分是在比较当前获取的地址是否超出了 ntoskrnl 的最大和最小范围。

#include <ntifs.h>

static PVOID g\_KernelBase = 0; static ULONG g\_KernelSize = 0;

#pragma pack(4) typedef struct \_PEB32

{

UCHAR InheritedAddressSpace;

UCHAR ReadImageFileExecOptions; UCHAR BeingDebugged;



UCHAR BitField;

ULONG Mutant;

ULONG ImageBaseAddress; ULONG Ldr;

ULONG ProcessParameters; ULONG SubSystemData; ULONG ProcessHeap;

ULONG FastPebLock;

ULONG AtlThunkSListPtr; ULONG IFEOKey;

ULONG CrossProcessFlags;

ULONG UserSharedInfoPtr; ULONG SystemReserved; ULONG AtlThunkSListPtr32; ULONG ApiSetMap;

} PEB32, PPEB32;

typedef struct \_PEB\_LDR\_DATA32

{

ULONG Length; UCHAR Initialized; ULONG SsHandle;

LIST\_ENTRY32 InLoadOrderModuleList; LIST\_ENTRY32 InMemoryOrderModuleList; LIST\_ENTRY32 InInitializationOrderModuleList;

} PEB\_LDR\_DATA32, PPEB\_LDR\_DATA32;

typedef struct \_LDR\_DATA\_TABLE\_ENTRY32

{

LIST\_ENTRY32 InLoadOrderLinks; LIST\_ENTRY32 InMemoryOrderLinks; LIST\_ENTRY32 InInitializationOrderLinks; ULONG DllBase;

ULONG EntryPoint; ULONG SizeOfImage;

UNICODE\_STRING32 FullDllName;

UNICODE\_STRING32 BaseDllName; ULONG Flags;

USHORT LoadCount; USHORT TlsIndex; LIST\_ENTRY32 HashLinks; ULONG TimeDateStamp;

} LDR\_DATA\_TABLE\_ENTRY32, PLDR\_DATA\_TABLE\_ENTRY32;

#pragma pack()

typedef struct \_RTL\_PROCESS\_MODULE\_INFORMATION

{

HANDLE Section; PVOID MappedBase; PVOID ImageBase; ULONG ImageSize; ULONG Flags;

USHORT LoadOrderIndex;

USHORT InitOrderIndex; USHORT LoadCount; USHORT OffsetToFileName;



UCHAR FullPathName[256];

} RTL\_PROCESS\_MODULE\_INFORMATION, PRTL\_PROCESS\_MODULE\_INFORMATION;

typedef struct \_RTL\_PROCESS\_MODULES

{

ULONG NumberOfModules; RTL\_PROCESS\_MODULE\_INFORMATION Modules[1];

} RTL\_PROCESS\_MODULES, PRTL\_PROCESS\_MODULES;

typedef enum \_SYSTEM\_INFORMATION\_CLASS

{

SystemModuleInformation = 0xb,

} SYSTEM\_INFORMATION\_CLASS;

// 取出KernelBase基地址

// By: lyshark.com

PVOID UtilKernelBase(OUT PULONG pSize)

{

NTSTATUS status = STATUS\_SUCCESS;

ULONG bytes = 0; PRTL\_PROCESS\_MODULES pMods = 0;

PVOID checkPtr = 0; UNICODE\_STRING routineName;

if (g\_KernelBase != 0)

{

if (pSize)

pSize = g\_KernelSize; return g\_KernelBase;

}

RtlInitUnicodeString(&routineName, L"NtOpenFile");

checkPtr = MmGetSystemRoutineAddress(&routineName); if (checkPtr == 0)

return 0;

try

{

status = ZwQuerySystemInformation(SystemModuleInformation, 0, bytes, &bytes);

if (bytes == 0)

{

DbgPrint("Invalid SystemModuleInformation size\n"); return 0;

}

pMods = (PRTL\_PROCESS\_MODULES)ExAllocatePoolWithTag(NonPagedPoolNx, bytes, "lyshark");

RtlZeroMemory(pMods, bytes);

status = ZwQuerySystemInformation(SystemModuleInformation, pMods, bytes, &bytes);



if (NT\_SUCCESS(status))

{

PRTL\_PROCESS\_MODULE\_INFORMATION pMod = pMods->Modules;

for (ULONG i = 0; i < pMods->NumberOfModules; i++)

{

if (checkPtr >= pMod[i].ImageBase &&

checkPtr < (PVOID)((PUCHAR)pMod[i].ImageBase +

pMod[i].ImageSize))

{

}

g\_KernelBase = pMod[i].ImageBase; g\_KernelSize = pMod[i].ImageSize; if (pSize)

pSize = g\_KernelSize; break;

}

}

}

except (EXCEPTION\_EXECUTE\_HANDLER)

{

return 0;

}

if (pMods)

ExFreePoolWithTag(pMods, "lyshark"); return g\_KernelBase;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

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

}

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

{

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

PULONG ulong = 0; UtilKernelBase(ulong);

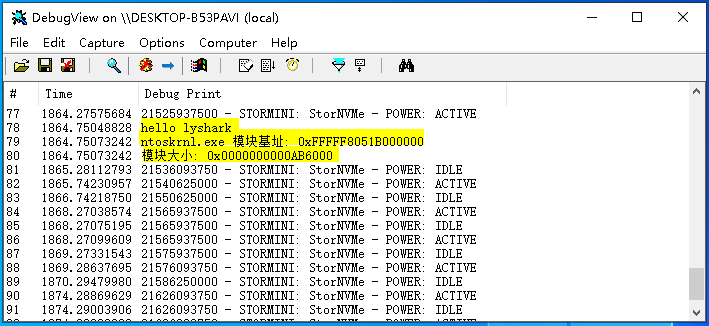
DbgPrint("ntoskrnl.exe 模块基址: 0x%p \n", g\_KernelBase);

DbgPrint("模块大小: 0x%p \n", g\_KernelSize);

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

}

# 我们编译并运行上方代码，效果如下：



**参考文献**

# https://blog.csdn.net/u012410612/article/details/17096597

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！