

在上一篇文章《驱动开发：内核取ntoskrnl模块基地址》中我们通过调用内核API函数获取到了内核进程ntoskrnl.exe的基址，当在某些场景中，我们不仅需要得到内核的基址，也需要得到特定进程内某个模块的基址，显然上篇文章中的方法是做不到的，本篇文章将实现内核层读取32位应用层中特定进程模块基址功能。

上一篇文章中的PPEB32, PLIST\_ENTRY32等结构体定义依然需要保留，此处只保留核心代码，定义部分请看前一篇文章，自定义读取模块基址核心代码如下，调用GetModuleBaseWow64() 用户需传入进程的PROCESS结构该结构可通过内核函数PsLookupProcessByProcessId获取到。

对于函数内部执行过程如下：

- 1.根据传入的EProcess结构调用KeStackAttachProcess附加到该进程内。
- 2.调用内核函数PsGetProcessWow64Process此函数可得到该进程空间内PEB结构数据。
- 3.通过for循环遍历整个pPeb->Ldr链表，并在遍历过程中通过RtlEqualUnicodeString判断是否是我们需要的模块。
- 4.如果判断是我们需要取出的模块名，则将LdrEntry->DllBase取出，此处取出的基址也即是我们所需要的。
- 5.比较结束后，通过调用KeUnstackDetachProcess这个内核模块脱离进程空间。

```
// By: LyShark
ULONGLONG GetModuleBaseWow64(_In_ PEPROCESS pEProcess, _In_ UNICODE_STRING
usModuleName)
{
    ULONGLONG BaseAddr = 0;
    KAPC_STATE KAPC = { 0 };
    KeStackAttachProcess(pEProcess, &KAPC);
    PPEB32 pPeb = (PPEB32)PsGetProcessWow64Process(pEProcess);
    if (pPeb == NULL || pPeb->Ldr == 0)
    {
        KeUnstackDetachProcess(&KAPC);
        return 0;
    }

    for (PLIST_ENTRY32 pListEntry = (PLIST_ENTRY32)((PPEB_LDR_DATA32)pPeb->Ldr)-
>InLoadOrderModuleList.Flink;
        pListEntry != &((PPEB_LDR_DATA32)pPeb->Ldr)->InLoadOrderModuleList;
        pListEntry = (PLIST_ENTRY32)pListEntry->Flink)
    {
        PLDR_DATA_TABLE_ENTRY32 LdrEntry = CONTAINING_RECORD(pListEntry,
LDR_DATA_TABLE_ENTRY32, InLoadOrderLinks);

        if (LdrEntry->BaseDllName.Buffer == NULL)
        {
            continue;
        }

        // 当前模块名链表
        UNICODE_STRING usCurrentName = { 0 };
        RtlInitUnicodeString(&usCurrentName, (PWCHAR)LdrEntry-
>BaseDllName.Buffer);

        // 比较模块名是否一致
```

```

        if (RtlEqualUnicodeString(&usModuleName, &usCurrentName, TRUE))
        {
            BaseAddr = (ULONGLONG)LdrEntry->DllBase;
            KeUnstackDetachProcess(&KAPC);
            return BaseAddr;
        }
    }
    KeUnstackDetachProcess(&KAPC);
    return 0;
}

```

如上就是如何得到特定模块基址的方法，如下是入口函数的调用方法，首先通过传入 6164 这个PID号，得到进程 EProcess 结构，其次使用 RtlInitUnicodeString(&unicode, wchar\_string) 初始化得到 kernel32.dll 字符串，最终调用 GetModuleBaseWow64 函数获取到进程 6164 中 kernel32.dll 的模块基址信息。

```

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("驱动卸载成功 \n");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint(("hello lyshark \n"));

    PEPROCESS pEProcess;
    HANDLE PID = (HANDLE)6164;

    // 初始化字符串
    UNICODE_STRING unicode;
    wchar_t *wchar_string = L"kernel32.dll";
    RtlInitUnicodeString(&unicode, wchar_string);

    // 取模块句柄
    PsLookupProcessByProcessId((HANDLE)PID, &pEProcess);
    ULONGLONG base32 = GetModuleBaseWow64(pEProcess, unicode);

    DbgPrint("ModuleBaseAddress: 0x%X \n", base32);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

这段代码输出效果如下所示:

DebugView on \\DESKTOP-B53PAVI (local)		
File Edit Capture Options Computer Help		
#	Time	Debug Print
118	85.74868774	hello lyshark
119	85.74869537	ModuleBaseAddress: 0x76AD0000
120	86.74810028	4154375000 - STORMINI: StorNVMe - POWER: IDLE
121	86.92612457	驱动卸载成功
122	87.08043671	4157656250 - STORMINI: StorNVMe - POWER: ACTIVE
123	88.08354187	4167812500 - STORMINI: StorNVMe - POWER: IDLE
124	91.38561249	4200781250 - STORMINI: StorNVMe - POWER: ACTIVE
125	91.38615417	4200781250 - STORMINI: StorNVMe - POWER: IDLE
126	92.41188049	4211093750 - STORMINI: StorNVMe - POWER: ACTIVE
127	92.41244507	4211093750 - STORMINI: StorNVMe - POWER: IDLE
128	92.41275024	4211093750 - STORMINI: StorNVMe - POWER: ACTIVE
129	93.41336823	4221093750 - STORMINI: StorNVMe - POWER: IDLE
130	96.39055634	4250781250 - STORMINI: StorNVMe - POWER: ACTIVE
131	96.39131165	4250781250 - STORMINI: StorNVMe - POWER: IDLE
132	96.39143372	4250781250 - STORMINI: StorNVMe - POWER: ACTIVE

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！