

在前面的系列教程如《内核枚举DpcTimer定时器》或者《内核枚举IoTimer定时器》里面 LyShark 大量使用了 特征码定位 这一方法来寻找符合条件的 汇编指令集，总体来说这种方式只能定位特征较小的指令如果特征值扩展到5位以上那么就需要写很多无用的代码，本章内容中将重点分析，并实现一个 通用 特征定位函数。

如下是一段特征码搜索片段，可以看到其实仅仅只是将上章中的搜索方式变成了一个 SearchSpecialCode 函数，如下函数，用户传入一个 扫描起始地址 以及搜索特征码的字节数组，即可完成搜索工作，具体的参数定义如下。

- pSearchBeginAddr 扫描的内存(内核)起始地址
- ulSearchLength 需要扫描的长度
- pSpecialCode 扫描特征码,传入一个UCHAR类型的字节数组
- ulSpecialCodeLength 特征码长度,传入字节数组长度

```
PVOID SearchSpecialCode(PVOID pSearchBeginAddr, ULONG ulSearchLength, PCHAR pSpecialCode,
ULONG ulSpecialCodeLength)
{
    PVOID pDestAddr = NULL;
    PCHAR pBeginAddr = (PCHAR)pSearchBeginAddr;
    PCHAR pEndAddr = pBeginAddr + ulSearchLength;
    PCHAR i = NULL;
    ULONG j = 0;

    for (i = pBeginAddr; i <= pEndAddr; i++)
    {
        // 遍历特征码
        for (j = 0; j < ulSpecialCodeLength; j++)
        {
            // 判断地址是否有效
            if (FALSE == MmIsAddressValid((PVOID)(i + j)))
            {
                break;
            }
            // 匹配特征码
            if (*(PCHAR)(i + j) != pSpecialCode[j])
            {
                break;
            }
        }

        // 匹配成功
        if (j >= ulSpecialCodeLength)
        {
            pDestAddr = (PVOID)i;
            break;
        }
    }
    return pDestAddr;
}
```

那么这个简单的特征码扫描函数该如何使用，这里我们就用《内核枚举IoTimer定时器》中枚举 IopTimerQueueHead 链表头部地址为案例进行讲解，如果你忘记了如何寻找链表头部可以去前面的文章中学习，这里只给出实现流程。

```

Command
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

nt!IoInitializeTimer+0x7e:
fffff805`74b85c0e 488b5c2430    mov     rbx,qword ptr [rsp+30h]
fffff805`74b85c13 488b6c2438    mov     rbp,qword ptr [rsp+38h]
fffff805`74b85c18 488b742440    mov     rsi,qword ptr [rsp+40h]
fffff805`74b85c1d 4883c420      add     rsp,20h
fffff805`74b85c21 5f           pop     rdi
fffff805`74b85c22 c3           ret

```

我们首先通过 `MmGetSystemRoutineAddress` 得到 `IoInitializeTimer` 首地址，然后在偏移长度为 `0x7e` 范围内搜索特征码 `48 8d 0d` 特征，其代码可以总结为如下样子。

```

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

    // 得到基址
    PCHAR IoInitializeTimer = GetIoInitializeTimerAddress();
    DbgPrint("IoInitializeTimer Address = %p \n", IoInitializeTimer);

    // -----
    // LyShark 开始定位特征

    // 设置起始位置
    PCHAR StartSearchAddress = (PCHAR)IoInitializeTimer;

    // 设置结束位置
    PCHAR EndSearchAddress = StartSearchAddress + 0x7e;
    DbgPrint("[LyShark 搜索区间] 起始地址: 0x%X --> 结束地址: 0x%X \n", StartSearchAddress,
    EndSearchAddress);

    // 设置搜索长度
    LONGLONG size = EndSearchAddress - StartSearchAddress;
    DbgPrint("[LyShark 搜索长度] 长度: %d \n", size);

    PVOID ptr;

    // 指定特征码
    UCHAR pSpecialCode[256] = { 0 };

    // 指定特征码长度
    ULONG ulSpecialCodeLength = 3;

    pSpecialCode[0] = 0x48;
    pSpecialCode[1] = 0x8d;
    pSpecialCode[2] = 0x0d;

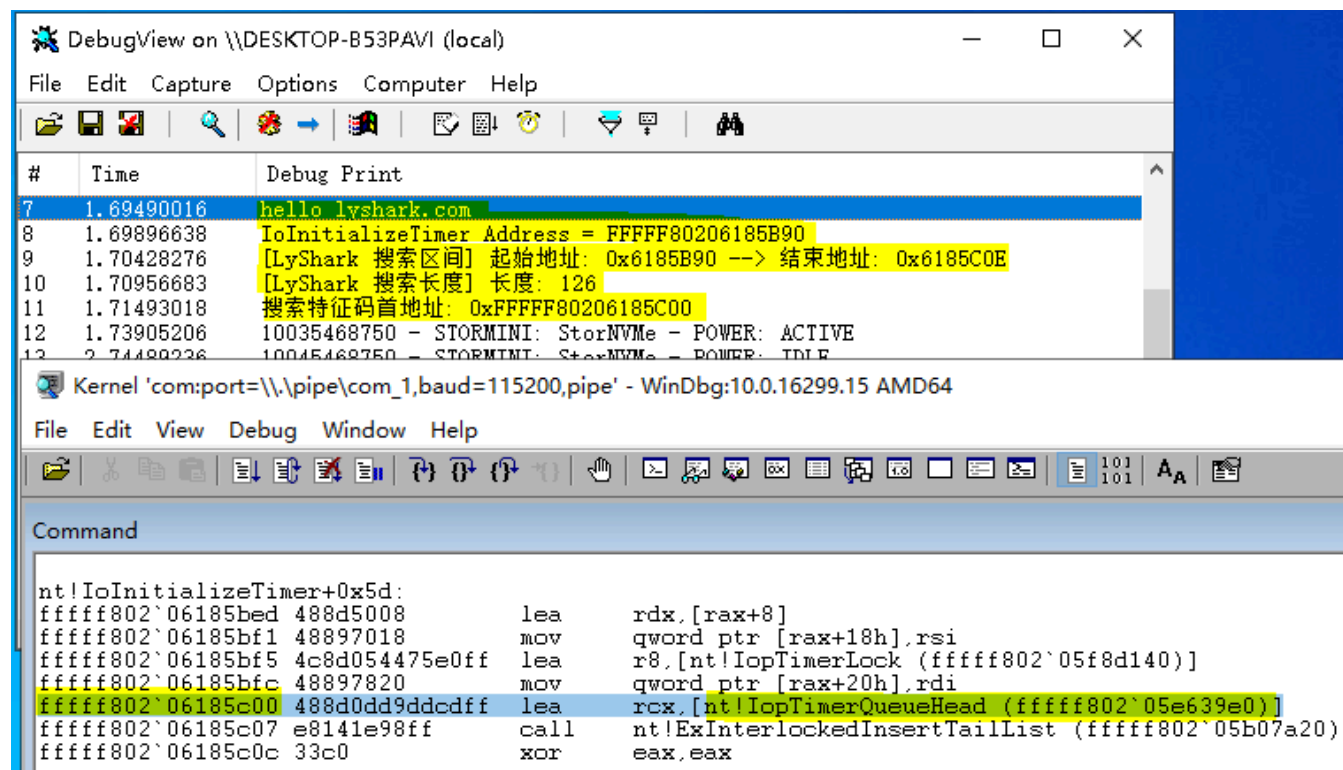
```

```
// 开始搜索,找到后返回首地址
ptr = SearchSpecialCode(StartSearchAddress, size, pSpecialCode, ulSpecialCodeLength);

DbgPrint("搜索特征码首地址: 0x%p \n", ptr);

Driver->DriverUnload = UnDriver;
return STATUS_SUCCESS;
}
```

代码运行后你会发现可以直接定位到我们所需要的位置上, 如下图所示:



如上图可以看到, 这个特征码定位函数返回的是内存地址, 而我们需要得到地址内的数据, 此时就需要提取以下。

例如当指令是:

```
fffff80206185c00 488d0dd9ddcdff lea rcx,[nt!IopTimerQueueHead (fffff80205e639e0)]
```

那么就需要 RtlCopyMemory 跳过前三个字节, 并在第四个字节开始取数据, 并将读入的数据放入到 IopTimerQueueHead\_LyShark\_Code 变量内。

```
// 开始搜索,找到后返回首地址
ptr = SearchSpecialCode(StartSearchAddress, size, pSpecialCode, ulSpecialCodeLength);

DbgPrint("搜索特征码首地址: 0x%p \n", ptr);

// 提取特征
// fffff802`06185c00 488d0dd9ddcdff lea rcx,[nt!IopTimerQueueHead (fffff802`05e639e0)]
ULONG64 ioffset = 0;
ULONG64 IopTimerQueueHead_LyShark_Code = 0;
```

```

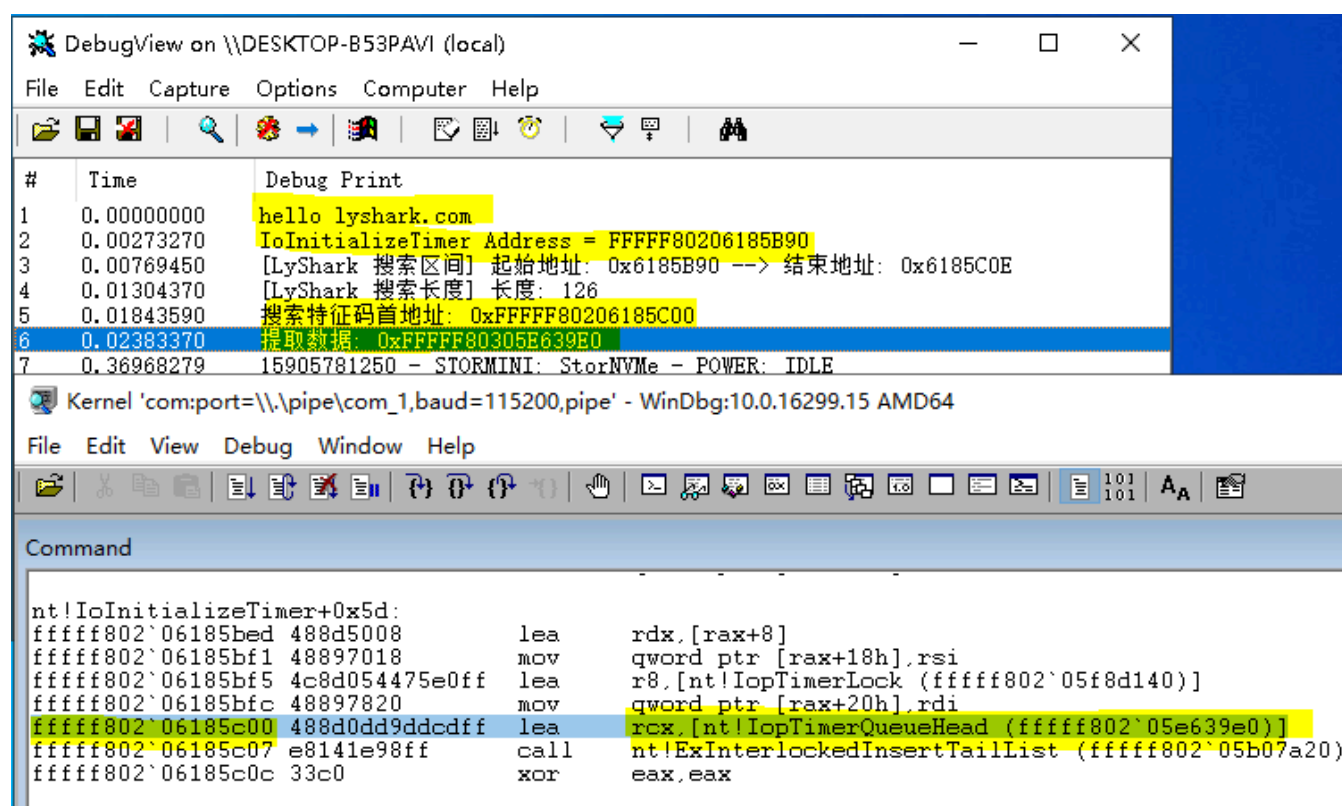
__try
{
    // 拷贝内存跳过lea, 向后四字节
    RtlCopyMemory(&ioffset, (ULONG64)ptr + 3, 4);

    // 取出后面的IopTimerQueueHead内存地址 LyShark.com
    IopTimerQueueHead_LyShark_Code = ioffset + (ULONG64)ptr + 7;

    DbgPrint("提取数据: 0x%p \n", IopTimerQueueHead_LyShark_Code);
}
__except (1)
{
    DbgPrint("[LyShark] 拷贝内存异常 \n");
}

```

这样即可得到我们所需要的地址，如下结果所示：



The top window, 'DebugView on \\DESKTOP-B53PAVI (local)', displays the following debug print messages:

#	Time	Debug Print
1	0.00000000	hello lyshark.com
2	0.00273270	IoInitializeTimer Address = FFFFF80206185B90
3	0.00769450	[LyShark 搜索区间] 起始地址: 0x6185B90 --> 结束地址: 0x6185C0E
4	0.01304370	[LyShark 搜索长度] 长度: 126
5	0.01843590	搜索特征码首地址: 0xFFFFF80206185C00
6	0.02383370	提取数据: 0xFFFFF80305E639E0
7	0.36968279	15905781250 - STORMINI: StorNVMe - POWER: IDLE

The bottom window, 'Kernel' (Kernel 'com:port=\\.\pipe\com\_1,baud=115200,pipe' - WinDbg:10.0.16299.15 AMD64), displays the following assembly code for 'nt!IoInitializeTimer+0x5d':

```

fffff802`06185bed 488d5008      lea     rdx,[rax+8]
fffff802`06185bf1 48897018      mov     qword ptr [rax+18h],rsi
fffff802`06185bf5 4c8d054475e0ff lea     r8,[nt!IopTimerLock (fffff802`05f8d140)]
fffff802`06185bfc 48897820      mov     qword ptr [rax+20h],rdi
fffff802`06185c00 488d0dd9ddcdff lea     rcx,[nt!IopTimerQueueHead (fffff802`05e639e0)]
fffff802`06185c07 e8141e98ff    call    nt!ExInterlockedInsertTailList (fffff802`05b07a20)
fffff802`06185c0c 33c0         xor     eax,eax

```