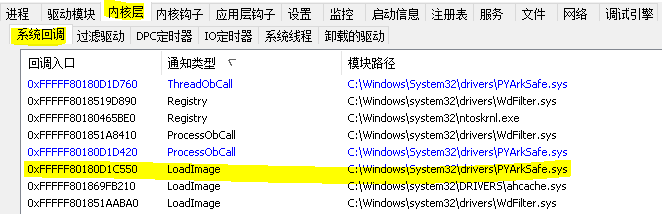
在笔者之前的文章 《驱动开发：内核特征码搜索函数封装》 中我们封装实现了特征码定位功能，本章将继续使用该功能，本次我们需要枚举内核 LoadImage 映像回调，在Win64环境下我们可以设置一个

# LoadImage 映像加载通告回调，当有新驱动或者DLL被加载时，回调函数就会被调用从而执行我们自己的回调例程，映像回调也存储在数组里，枚举时从数组中读取值之后，需要进行位运算解密得到地址。

我们来看一款闭源ARK工具是如何实现的：



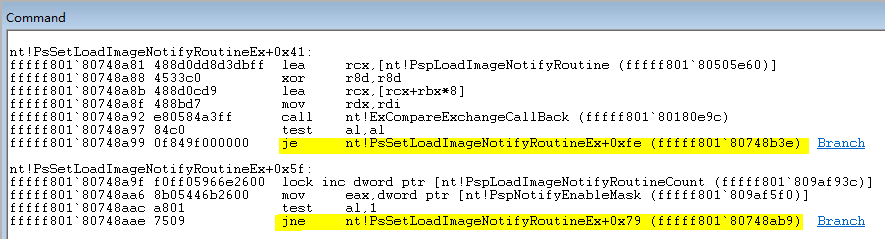
如上所述，如果我们需要拿到回调数组那么首先要得到该数组，数组的符号名是

PspLoadImageNotifyRoutine 我们可以在 PsSetLoadImageNotifyRoutineEx 中找到。

第一步使用WinDBG输入 uf PsSetLoadImageNotifyRoutineEx 首先定位到，能够找到

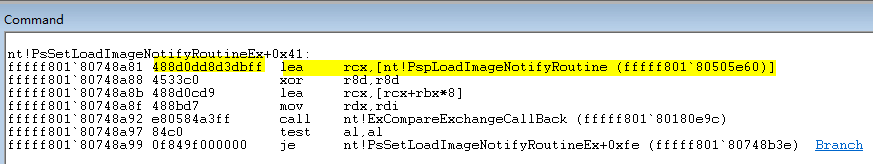
PsSetLoadImageNotifyRoutineEx 这里的两个位置都可以被引用，当然了这个函数可以直接通过

PsSetLoadImageNotifyRoutineEx 函数动态拿到此处不需要我们动态定位。



我们通过获取到 PsSetLoadImageNotifyRoutineEx 函数的内存首地址，然后向下匹配特征码搜索找到

488d0d88e8dbff 并取出 PspLoadImageNotifyRoutine 内存地址，该内存地址就是 LoadImage 映像模块的基址。



# 如果使用代码去定位这段空间，则你可以这样写，这样即可得到具体特征地址。

// 署名权

// right to sign one's name on a piece of work

// PowerBy: LyShark

// Email: [me@lyshark.com](mailto:me@lyshark.com)

#include <ntddk.h> #include <windef.h>



// 指定内存区域的特征码扫描

PVOID SearchMemory(PVOID pStartAddress, PVOID pEndAddress, PUCHAR pMemoryData, ULONG ulMemoryDataSize)

{

PVOID pAddress = NULL; PUCHAR i = NULL; ULONG m = 0;

// 扫描内存

for (i = (PUCHAR)pStartAddress; i < (PUCHAR)pEndAddress; i++)

{

// 判断特征码

for (m = 0; m < ulMemoryDataSize; m++)

{

if ( (PUCHAR)(i + m) != pMemoryData[m])

{

break;

}

}

// 判断是否找到符合特征码的地址

if (m >= ulMemoryDataSize)

{

// 找到特征码位置, 获取紧接着特征码的下一地址pAddress = (PVOID)(i + ulMemoryDataSize); break;

}

}

return pAddress;

}

// 根据特征码获取 PspLoadImageNotifyRoutine 数组地址

PVOID SearchPspLoadImageNotifyRoutine(PUCHAR pSpecialData, ULONG ulSpecialDataSize)

{

UNICODE\_STRING ustrFuncName; PVOID pAddress = NULL;

LONG lOffset = 0;

PVOID pPsSetLoadImageNotifyRoutine = NULL; PVOID pPspLoadImageNotifyRoutine = NULL;

// 先获取 PsSetLoadImageNotifyRoutineEx 函数地址

RtlInitUnicodeString(&ustrFuncName, L"PsSetLoadImageNotifyRoutineEx"); pPsSetLoadImageNotifyRoutine = MmGetSystemRoutineAddress(&ustrFuncName); if (NULL == pPsSetLoadImageNotifyRoutine)

{

return pPspLoadImageNotifyRoutine;

}

// 查找 PspLoadImageNotifyRoutine 函数地址

pAddress = SearchMemory(pPsSetLoadImageNotifyRoutine, (PVOID) ((PUCHAR)pPsSetLoadImageNotifyRoutine + 0xFF), pSpecialData, ulSpecialDataSize);

if (NULL == pAddress)

{

return pPspLoadImageNotifyRoutine;



}

// 先获取偏移, 再计算地址

lOffset = (PLONG)pAddress;

pPspLoadImageNotifyRoutine = (PVOID)((PUCHAR)pAddress + sizeof(LONG) + lOffset);

return pPspLoadImageNotifyRoutine;

}

VOID UnDriver(PDRIVER\_OBJECT Driver)

{

}

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

{

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

PVOID pPspLoadImageNotifyRoutineAddress = NULL; RTL\_OSVERSIONINFOW osInfo = { 0 };

UCHAR pSpecialData[50] = { 0 };

ULONG ulSpecialDataSize = 0;

// 获取系统版本信息, 判断系统版本

RtlGetVersion(&osInfo);

if (10 == osInfo.dwMajorVersion)

{

// 48 8d 0d 88 e8 db ff

// 查找指令 lea rcx,[nt!PspLoadImageNotifyRoutine (fffff804`44313ce0)]

/ nt!PsSetLoadImageNotifyRoutineEx+0x41:

fffff801`80748a81 488d0dd8d3dbff lea rcx, [nt!PspLoadImageNotifyRoutine (fffff801`80505e60)]

fffff801`80748a88 4533c0 xor r8d,r8d

fffff801`80748a8b 488d0cd9 lea rcx,[rcx+rbx 8] fffff801`80748a8f 488bd7 mov rdx,rdi

fffff801`80748a92 e80584a3ff call nt!ExCompareExchangeCallBack

(fffff801`80180e9c)

fffff801`80748a97 84c0 test al,al fffff801`80748a99 0f849f000000 je

nt!PsSetLoadImageNotifyRoutineEx+0xfe (fffff801`80748b3e) Branch

/

pSpecialData[0] = 0x48; pSpecialData[1] = 0x8D; pSpecialData[2] = 0x0D; ulSpecialDataSize = 3;

}

// 根据特征码获取地址 获取 PspLoadImageNotifyRoutine 数组地址

pPspLoadImageNotifyRoutineAddress = SearchPspLoadImageNotifyRoutine(pSpecialData, ulSpecialDataSize);

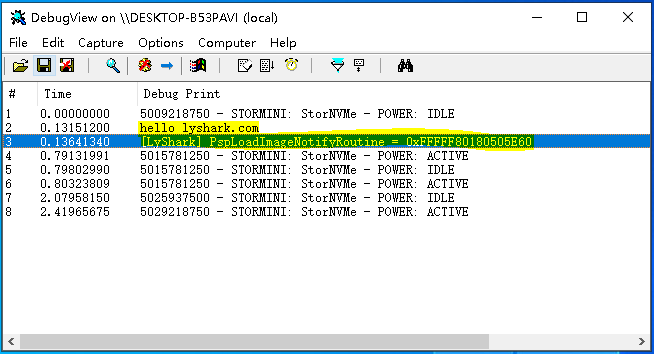
DbgPrint("[LyShark] PspLoadImageNotifyRoutine = 0x%p \n", pPspLoadImageNotifyRoutineAddress);

Driver->DriverUnload = UnDriver;

return STATUS\_SUCCESS;

}

# 将这个驱动拖入到虚拟机中并运行，输出结果如下：



有了数组地址接下来就是要对数组进行解密，如何解密？

1.首先拿到数组指针 pPspLoadImageNotifyRoutineAddress + sizeof(PVOID) i 此处的i也就是下标。



2.得到的新地址在与 pNotifyRoutineAddress & 0xfffffffffffffff8 进行与运算。

3.最后 (PVOID )pNotifyRoutineAddress 取出里面的参数。



# 增加解密代码以后，这段程序的完整代码也就可以被写出来了，如下所示。



// 署名权

// right to sign one's name on a piece of work

// PowerBy: LyShark

// Email: [me@lyshark.com](mailto:me@lyshark.com)

#include <ntddk.h> #include <windef.h>

// 指定内存区域的特征码扫描

PVOID SearchMemory(PVOID pStartAddress, PVOID pEndAddress, PUCHAR pMemoryData, ULONG ulMemoryDataSize)

{

PVOID pAddress = NULL; PUCHAR i = NULL; ULONG m = 0;

// 扫描内存

for (i = (PUCHAR)pStartAddress; i < (PUCHAR)pEndAddress; i++)

{

// 判断特征码

for (m = 0; m < ulMemoryDataSize; m++)

{

if ( (PUCHAR)(i + m) != pMemoryData[m])

{



break;

}

}

// 判断是否找到符合特征码的地址

if (m >= ulMemoryDataSize)

{

// 找到特征码位置, 获取紧接着特征码的下一地址pAddress = (PVOID)(i + ulMemoryDataSize); break;

}

}

return pAddress;

}

// 根据特征码获取 PspLoadImageNotifyRoutine 数组地址

PVOID SearchPspLoadImageNotifyRoutine(PUCHAR pSpecialData, ULONG ulSpecialDataSize)

{

UNICODE\_STRING ustrFuncName; PVOID pAddress = NULL;

LONG lOffset = 0;

PVOID pPsSetLoadImageNotifyRoutine = NULL; PVOID pPspLoadImageNotifyRoutine = NULL;

// 先获取 PsSetLoadImageNotifyRoutineEx 函数地址RtlInitUnicodeString(&ustrFuncName, L"PsSetLoadImageNotifyRoutineEx"); pPsSetLoadImageNotifyRoutine = MmGetSystemRoutineAddress(&ustrFuncName);

if (NULL == pPsSetLoadImageNotifyRoutine)

{

return pPspLoadImageNotifyRoutine;

}

// 查找 PspLoadImageNotifyRoutine 函数地址

pAddress = SearchMemory(pPsSetLoadImageNotifyRoutine, (PVOID) ((PUCHAR)pPsSetLoadImageNotifyRoutine + 0xFF), pSpecialData, ulSpecialDataSize);

if (NULL == pAddress)

{

return pPspLoadImageNotifyRoutine;

}

// 先获取偏移, 再计算地址

lOffset = (PLONG)pAddress;

pPspLoadImageNotifyRoutine = (PVOID)((PUCHAR)pAddress + sizeof(LONG) + lOffset);

return pPspLoadImageNotifyRoutine;

}

VOID UnDriver(PDRIVER\_OBJECT Driver)

{

}

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

{



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

PVOID pPspLoadImageNotifyRoutineAddress = NULL; RTL\_OSVERSIONINFOW osInfo = { 0 };

UCHAR pSpecialData[50] = { 0 };

ULONG ulSpecialDataSize = 0;

// 获取系统版本信息, 判断系统版本

RtlGetVersion(&osInfo);

if (10 == osInfo.dwMajorVersion)

{

// 48 8d 0d 88 e8 db ff

// 查找指令 lea rcx,[nt!PspLoadImageNotifyRoutine (fffff804`44313ce0)]

/ nt!PsSetLoadImageNotifyRoutineEx+0x41:

fffff801`80748a81 488d0dd8d3dbff lea rcx, [nt!PspLoadImageNotifyRoutine (fffff801`80505e60)]

fffff801`80748a88 4533c0 xor r8d,r8d

fffff801`80748a8b 488d0cd9 lea rcx,[rcx+rbx 8] fffff801`80748a8f 488bd7 mov rdx,rdi

fffff801`80748a92 e80584a3ff call nt!ExCompareExchangeCallBack

(fffff801`80180e9c)

fffff801`80748a97 84c0 test al,al fffff801`80748a99 0f849f000000 je

nt!PsSetLoadImageNotifyRoutineEx+0xfe (fffff801`80748b3e) Branch

/

pSpecialData[0] = 0x48; pSpecialData[1] = 0x8D; pSpecialData[2] = 0x0D; ulSpecialDataSize = 3;

}

// 根据特征码获取地址 获取 PspLoadImageNotifyRoutine 数组地址

pPspLoadImageNotifyRoutineAddress = SearchPspLoadImageNotifyRoutine(pSpecialData, ulSpecialDataSize);

DbgPrint("[LyShark] PspLoadImageNotifyRoutine = 0x%p \n", pPspLoadImageNotifyRoutineAddress);

// 遍历回调

ULONG i = 0;

PVOID pNotifyRoutineAddress = NULL;

// 获取 PspLoadImageNotifyRoutine 数组地址

if (NULL == pPspLoadImageNotifyRoutineAddress)

{

return FALSE;

}

// 获取回调地址并解密

for (i = 0; i < 64; i++)

{

pNotifyRoutineAddress = (PVOID ) ((PUCHAR)pPspLoadImageNotifyRoutineAddress + sizeof(PVOID) i);



pNotifyRoutineAddress = (PVOID)((ULONG64)pNotifyRoutineAddress & 0xfffffffffffffff8);

if (MmIsAddressValid(pNotifyRoutineAddress))

{

pNotifyRoutineAddress = (PVOID )pNotifyRoutineAddress;

DbgPrint("[LyShark] 序号: %d | 回调地址: 0x%p \n", i, pNotifyRoutineAddress);

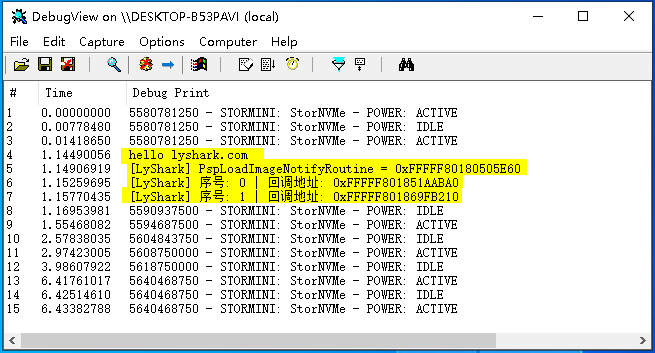
}

}

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

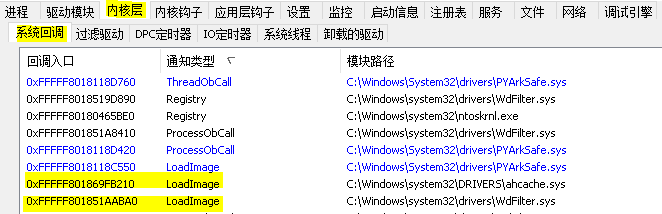
}

# 运行这段完整的程序代码，输出如下效果：



目前系统中只有两个回调，所以枚举出来的只有两条，打开ARK验证一下会发现完全正确，忽略 pyark

这是后期打开的。



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！