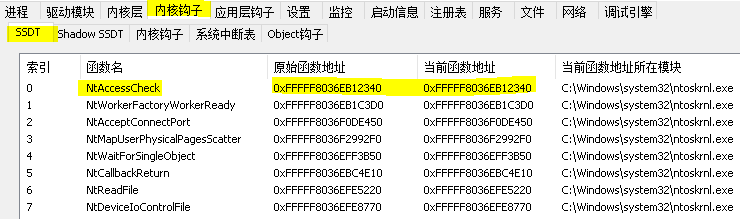
三年前面朝黄土背朝天的我，写了一篇如何在 Windows 7 系统下枚举内核 SSDT 表的文章 《驱动开发：内

核读取SSDT表基址》 三年过去了我还是个 单身狗 ，开个玩笑，微软的 Windows 10 系统已经覆盖了大多

数个人PC终端，以前的方法也该进行迭代更新了，或许在网上你能够找到类似的文章，但我可以百分百肯定都不能用，今天 LyShark 将带大家一起分析 Win10 x64 最新系统 SSDT 表的枚举实现。

# 看一款闭源ARK工具的枚举效果:



直接步入正题，首先 SSDT 表中文为系统服务描述符表，SSDT表的 作用 是把 应用 层与 内核 层 联系起来起到 桥梁 的作用，枚举 SSDT表 也是 反内核 工具最基本的功能，通常在 64位 系统中要想找到 SSDT 表， 需要先找到 KeServiceDescriptorTable 这个函数，由于该函数没有被导出，所以只能动态的查找它的地址，庆幸的是我们可以通过查找 msr(c0000082) 这个特殊的寄存器来替代查找

KeServiceDescriptorTable 这一步，在新版系统中查找SSDT可以归纳为如下这几个步骤。

# rdmsr c0000082 -> KiSystemCall64Shadow -> KiSystemServiceUser -> SSDT

首先第一步通过 rdmsr C0000082 MSR寄存器得到 KiSystemCall64Shadow 的函数地址，计算

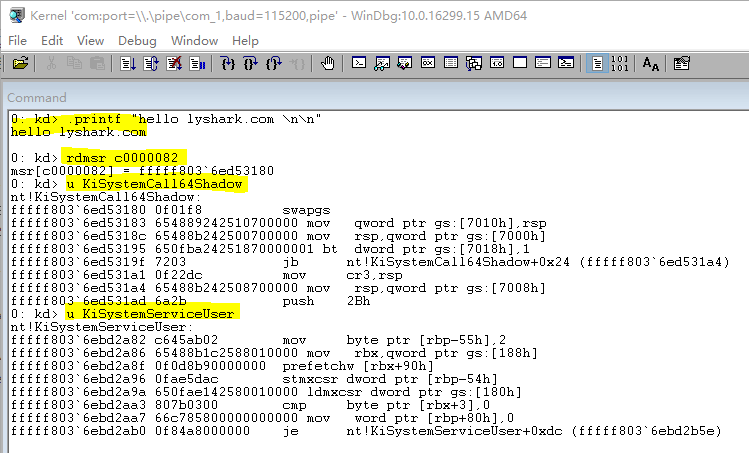
KiSystemCall64Shadow 与 KiSystemServiceUser 偏移量，如下图所示。

得到相对偏移 6ed53180(KiSystemCall64Shadow) - 6ebd2a82(KiSystemServiceUser) =

1806FE

6ed53180(rdmsr) - 1806FE = KiSystemServiceUser

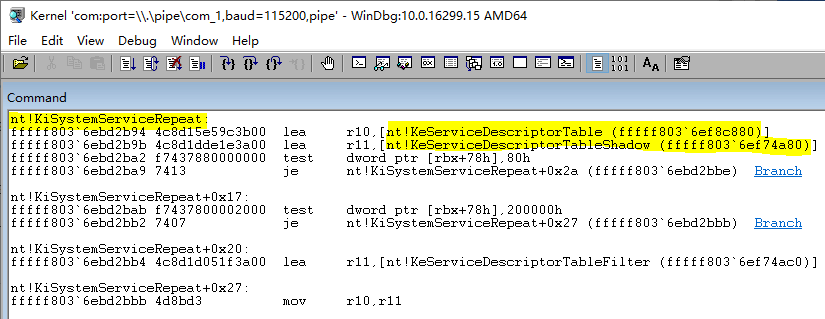
# 也就是说



如上当我们找到了 KiSystemServiceUser 的地址以后，在 KiSystemServiceUser 向下搜索可找到

KiSystemServiceRepeat 里面就是我们要找的 SSDT 表基址。

其中 fffff8036ef8c880 则是 SSDT表 的基地址，紧随其后的 fffff8036ef74a80 则是 SSSDT表 的基地址。



那么如果将这个过程通过代码的方式来实现，我们还需要使用 《驱动开发：内核枚举IoTimer定时器》 中所使用的特征码定位技术，如下我们查找这段特征。

// 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 <ntifs.h>

#pragma intrinsic( readmsr)

ULONGLONG ssdt\_address = 0;

// 获取 KeServiceDescriptorTable 首地址

ULONGLONG GetLySharkCOMKeServiceDescriptorTable()

{

// 设置起始位置

PUCHAR StartSearchAddress = (PUCHAR) readmsr(0xC0000082) - 0x1806FE;

// 设置结束位置

PUCHAR EndSearchAddress = StartSearchAddress + 0x100000; DbgPrint("[LyShark Search] 扫描起始地址: %p --> 扫描结束地址: %p \n",

StartSearchAddress, EndSearchAddress);

PUCHAR ByteCode = NULL;

UCHAR OpCodeA = 0, OpCodeB = 0, OpCodeC = 0; ULONGLONG addr = 0;

ULONG templong = 0;

for (ByteCode = StartSearchAddress; ByteCode < EndSearchAddress; ByteCode++)

{

// 使用MmIsAddressValid()函数检查地址是否有页面错误

if (MmIsAddressValid(ByteCode) && MmIsAddressValid(ByteCode + 1) && MmIsAddressValid(ByteCode + 2))

{

OpCodeA = ByteCode; OpCodeB = (ByteCode + 1); OpCodeC = (ByteCode + 2);



// 对比特征值 寻找 nt!KeServiceDescriptorTable 函数地址

/ nt!KiSystemServiceRepeat:

fffff803`6ebd2b94 4c8d15e59c3b00 lea r10, [nt!KeServiceDescriptorTable (fffff803`6ef8c880)]

fffff803`6ebd2b9b 4c8d1dde1e3a00 lea r11,

[nt!KeServiceDescriptorTableShadow (fffff803`6ef74a80)]

fffff803`6ebd2ba2 f7437880000000 test dword ptr [rbx+78h],80h fffff803`6ebd2ba9 7413 je

nt!KiSystemServiceRepeat+0x2a (fffff803`6ebd2bbe) Branch

/

if (OpCodeA == 0x4c && OpCodeB == 0x8d && OpCodeC == 0x15)

{

// 获 取 高 位 地 址 fffff802 memcpy(&templong, ByteCode + 3, 4);

// 与低位64da4880地址相加得到完整地址

addr = (ULONGLONG)templong + (ULONGLONG)ByteCode + 7; return addr;

}

}

}

return 0;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint(("驱动程序卸载成功! \n"));

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)

{

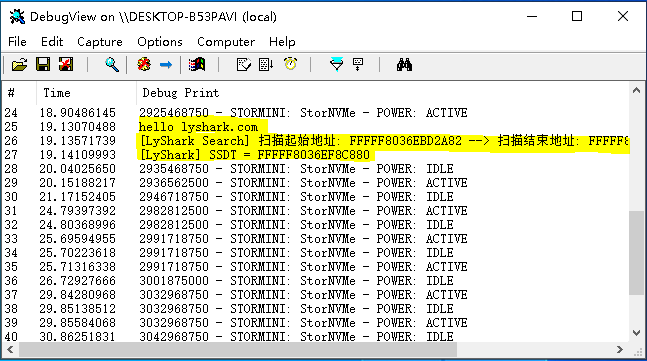
DbgPrint("hello lyshark.com");

ssdt\_address = GetLySharkCOMKeServiceDescriptorTable(); DbgPrint("[LyShark] SSDT = %p \n", ssdt\_address);

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

}

# 如上代码中所提及的步骤我想不需要再做解释了，这段代码运行后即可输出SSDT表的基址。



如上通过调用 GetLySharkCOMKeServiceDescriptorTable() 得到 SSDT 地址以后我们就需要对该地址进行解密操作。

得到 ServiceTableBase 的地址后，就能得到每个服务函数的地址。但这个表存放的并不是 SSDT 函数的完整地址，而是其相对于 ServiceTableBase[Index]>>4 的数据，每个数据占四个字节，所以计算指定 Index 函数完整地址的公式是；

# 在x86平台上: FuncAddress = KeServiceDescriptorTable + 4 \* Index

在x64平台上：FuncAddress = [KeServiceDescriptorTable+4\*Index]>>4 +

KeServiceDescriptorTable

如下汇编代码就是一段解密代码，代码中 rcx 寄存器传入SSDT的下标，而 rdx 寄存器则是传入SSDT表基址。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 48:8BC1 | | | mov | rax,rcx |  |  | | | | |
| rcx=index  4C:8D12 | | | lea | r10,qword | ptr | ds:[rdx] | | | rdx=ssdt | |
| 8BF8 | | | mov | edi,eax |  |  | | | | |
| C1EF 07  83E7 20 | |  | | shr  and | edi,7  edi,20 |  |  | | |  | | |
| 4E:8B1417 | | | mov | r10,qword | ptr | ds:[rdi+r10] | | | | |
| 4D:631C82  49:8BC3 | |  | | movsxd r11,dword  mov rax,r11 | | | ptr | ds:[r10+rax | 4] | |  | |
| 49:C1FB 04 | | | sar r11,4 | | |  |  |  | | |
| 4D:03D3  49:8BC2 | |  | | add r10,r11  mov rax,r10 | | |  |  |  | |  | |
| C3 | | | ret | | |  |  |  | | |

有了解密公式以后代码的编写就变得很容易，如下是读取SSDT的完整代码。

// 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 <ntifs.h>

#pragma intrinsic( readmsr)



typedef struct \_SYSTEM\_SERVICE\_TABLE

{

PVOID ServiceTableBase;

PVOID ServiceCounterTableBase; ULONGLONG NumberOfServices; PVOID ParamTableBase;

} SYSTEM\_SERVICE\_TABLE, PSYSTEM\_SERVICE\_TABLE;

ULONGLONG ssdt\_base\_aadress; PSYSTEM\_SERVICE\_TABLE KeServiceDescriptorTable;

typedef UINT64( fastcall SCFN)(UINT64, UINT64); SCFN scfn;

// 解密算法

VOID DecodeSSDT()

{

UCHAR strShellCode[36] = "\x48\x8B\xC1\x4C\x8D\x12\x8B\xF8\xC1\xEF\x07\x83\xE7\x20\x4E\x8B\x14\x17\x4D\x6 3\x1C\x82\x49\x8B\xC3\x49\xC1\xFB\x04\x4D\x03\xD3\x49\x8B\xC2\xC3";

/

48:8BC1 | mov rax,rcx | rcx=index

4C:8D12 | lea r10,qword ptr ds:[rdx] |

rdx=ssdt

8BF8 | mov edi,eax |

C1EF 07 | shr edi,7 |

83E7 20 | and edi,20 |

4E:8B1417 | mov r10,qword ptr ds:[rdi+r10] |

4D:631C82 | movsxd r11,dword ptr ds:[r10+rax 4] |

49:8BC3 | mov rax,r11 |

49:C1FB 04 | sar r11,4 |

4D:03D3 | add r10,r11 |

49:8BC2 | mov rax,r10 |

C3 | ret |

/

scfn = ExAllocatePool(NonPagedPool, 36); memcpy(scfn, strShellCode, 36);

}

// 获取 KeServiceDescriptorTable 首地址

ULONGLONG GetKeServiceDescriptorTable()

{

// 设置起始位置

PUCHAR StartSearchAddress = (PUCHAR) readmsr(0xC0000082) - 0x1806FE;

// 设置结束位置

PUCHAR EndSearchAddress = StartSearchAddress + 0x8192;

DbgPrint("扫描起始地址: %p --> 扫描结束地址: %p \n", StartSearchAddress, EndSearchAddress);

PUCHAR ByteCode = NULL;

UCHAR OpCodeA = 0, OpCodeB = 0, OpCodeC = 0; ULONGLONG addr = 0;



ULONG templong = 0;

for (ByteCode = StartSearchAddress; ByteCode < EndSearchAddress; ByteCode++)

{

// 使用MmIsAddressValid()函数检查地址是否有页面错误

if (MmIsAddressValid(ByteCode) && MmIsAddressValid(ByteCode + 1) && MmIsAddressValid(ByteCode + 2))

{

OpCodeA = ByteCode; OpCodeB = (ByteCode + 1); OpCodeC = (ByteCode + 2);

// 对比特征值 寻找 nt!KeServiceDescriptorTable 函数地址

// LyShark.com

// 4c 8d 15 e5 9e 3b 00 lea r10,[nt!KeServiceDescriptorTable (fffff802`64da4880)]

// 4c 8d 1d de 20 3a 00 lea r11,[nt!KeServiceDescriptorTableShadow

(fffff802`64d8ca80)]

if (OpCodeA == 0x4c && OpCodeB == 0x8d && OpCodeC == 0x15)

{

// 获 取 高 位 地 址 fffff802 memcpy(&templong, ByteCode + 3, 4);

// 与低位64da4880地址相加得到完整地址

addr = (ULONGLONG)templong + (ULONGLONG)ByteCode + 7; return addr;

}

}

}

return 0;

}

// 得到函数相对偏移地址

ULONG GetOffsetAddress(ULONGLONG FuncAddr)

{

ULONG dwtmp = 0;

PULONG ServiceTableBase = NULL;

if (KeServiceDescriptorTable == NULL)

{

KeServiceDescriptorTable = (PSYSTEM\_SERVICE\_TABLE)GetKeServiceDescriptorTable();

}

ServiceTableBase = (PULONG)KeServiceDescriptorTable->ServiceTableBase; dwtmp = (ULONG)(FuncAddr - (ULONGLONG)ServiceTableBase);

return dwtmp << 4;

}

// 根据序号得到函数地址

ULONGLONG GetSSDTFunctionAddress(ULONGLONG NtApiIndex)

{

ULONGLONG ret = 0;

if (ssdt\_base\_aadress == 0)

{

// 得到ssdt基地址

ssdt\_base\_aadress = GetKeServiceDescriptorTable();

}

if (scfn == NULL)

{

DecodeSSDT();

}

ret = scfn(NtApiIndex, ssdt\_base\_aadress); return ret;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint(("驱动程序卸载成功! \n"));

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)

{

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

ULONGLONG ssdt\_address = GetKeServiceDescriptorTable();

DbgPrint("SSDT基地址 = %p \n", ssdt\_address);

// 根据序号得到函数地址

ULONGLONG address = GetSSDTFunctionAddress(51); DbgPrint("[LyShark] NtOpenFile地址 = %p \n", address);

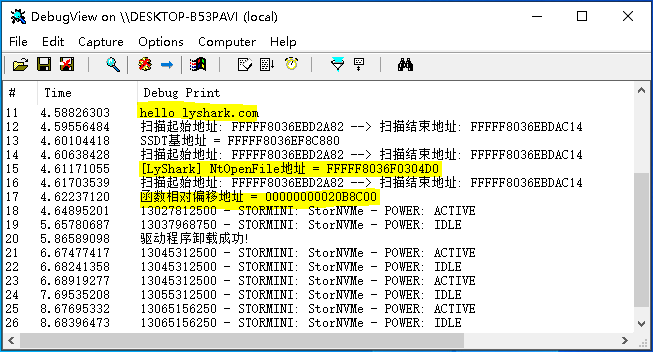
// 得到相对SSDT的偏移量

DbgPrint("函数相对偏移地址 = %p \n", GetOffsetAddress(address));

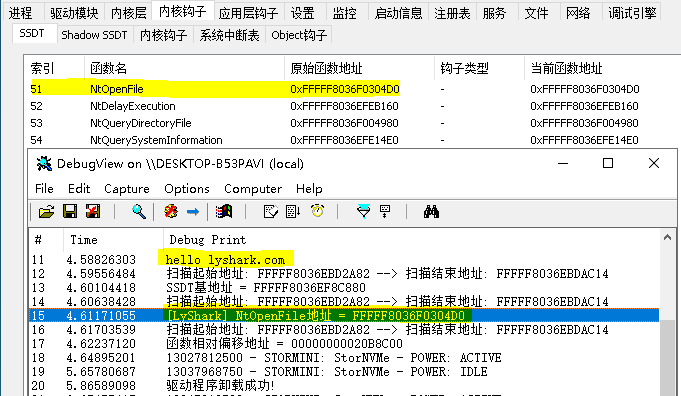
DriverObject->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

运行后即可得到 SSDT 下标为 51 的函数也就是得到 NtOpenFile 的绝对地址和相对地址。



# 你也可以打开ARK工具，对比一下是否一致，如下图所示， LyShark 的代码是没有任何问题的。



本书作者： 王瑞 (LyShark)

作者邮箱： [me@lyshark.com](mailto:me@lyshark.com)

作者博客： https://lyshark.cnblogs.com

团队首页： [www.lyshark.com](http://www.lyshark.com/)