

在笔者前面有一篇文章《驱动开发：断链隐藏驱动程序自身》通过摘除驱动的链表实现了断链隐藏自身的目的，但此方法恢复时会触发PG会蓝屏，偶然间在网上找到了一个作者介绍的一种方法，觉得有必要详细分析一下他是如何实现的驱动隐藏的，总体来说作者的思路是最终寻找到 MiProcessLoaderEntry 的入口地址，该函数的作用是将驱动信息加入链表和移除链表，运用这个函数即可动态处理驱动的添加和移除问题。

- MiProcessLoaderEntry(pDriverObject->DriverSection, 1) 添加
- MiProcessLoaderEntry(pDriverObject->DriverSection, 0) 移除

那么如何找到 MiProcessLoaderEntry 函数入口地址就是下一步的目标，寻找入口可以总结为：

- 1.寻找 MmUnloadSystemImage 函数地址，可通过 MmGetSystemRoutineAddress 函数得到。
- 2.在 MmUnloadSystemImage 里面寻找 MiUnloadSystemImage 函数地址。
- 3.在 MiUnloadSystemImage 里面继续寻找 MiProcessLoaderEntry 即可。

搜索 MmUnloadSystemImage 可定位到 call nt!MiUnloadSystemImage 地址。

```
Kernel 'com:port=\\.\pipe\com_1,baud=115200,pipe' - WinDbg:10.0.16299.15 AMD64
File Edit View Debug Window Help
Command
0: kd> uf MmUnloadSystemImage
nt!MmUnloadSystemImage:
fffff801`37943500 48895c2408      mov     qword ptr [rsp+8],rbx
fffff801`37943505 57             push    rdi
fffff801`37943506 4883ec20       sub     rsp,20h
fffff801`3794350a 488bf9        mov     rdi,rcx
fffff801`3794350d e87eb3ebff    call    nt!MmAcquireLoadLock (fffff801`377fe890)
fffff801`37943512 83caff        or      edx,0FFFFFFFFh
fffff801`37943515 488bcf        mov     rcx,rdi
fffff801`37943518 488bd8        mov     rbx,rax
fffff801`3794351b e860b4ebff    call    nt!MiUnloadSystemImage (fffff801`377fe980)
fffff801`37943520 488bcb        mov     rcx,rbx
fffff801`37943523 e808b3ebff    call    nt!MmReleaseLoadLock (fffff801`377fe830)
fffff801`37943528 488b5c2430     mov     rbx,qword ptr [rsp+30h]
fffff801`3794352d 33c0          xor     eax,eax
fffff801`3794352f 4883c420       add     rsp,20h
fffff801`37943533 5f            pop     rdi
fffff801`37943534 c3            ret
```

搜索 MiUnloadSystemImage 定位到 call nt!MiProcessLoaderEntry 即得到了我们想要的。

```
Kernel 'com:port=\\.\pipe\com_1,baud=115200,pipe' - WinDbg:10.0.16299.15 AMD64
File Edit View Debug Window Help
Command
nt!MiUnloadSystemImage+0x399:
fffff801`377fed19 33d2          xor     edx,edx
fffff801`377fed1b 488bcb        mov     rcx,rbx
fffff801`377fed1e e84162b4ff    call    nt!MiProcessLoaderEntry (fffff801`37344f64)
fffff801`377fed23 8b05d756f7ff  mov     eax,dword ptr [nt!PerfGlobalGroupMask (fffff801`377fed29)
fffff801`377fed29 a804          test    al,4
fffff801`377fed2b 7440          je      nt!MiUnloadSystemImage+0x3ed (fffff801`377fed2b)
```

根据前面 枚举篇 系列文章，定位这段特征很容易实现，如下是一段参考代码。

```
// PowerBy: LyShark
// Email: me@lyshark.com

#include <ntddk.h>
#include <ntstrsafe.h>
```

```
typedef NTSTATUS(__fastcall *MiProcessLoaderEntry)(PVOID pDriverSection, BOOLEAN
bLoad);
```

```
// 取出指定函数地址
```

```
PVOID GetProcAddress(WCHAR *FuncName)
{
    UNICODE_STRING u_FuncName = { 0 };
    PVOID ref = NULL;

    RtlInitUnicodeString(&u_FuncName, FuncName);
    ref = MmGetSystemRoutineAddress(&u_FuncName);

    if (ref != NULL)
    {
        return ref;
    }

    return ref;
}
```

```
// 特征定位 MiUnloadSystemImage
```

```
ULONG64 GetMiUnloadSystemImageAddress()
{
    // 在MmUnloadSystemImage函数中搜索的Code
    /*
    lyshark.com: kd> uf MmUnloadSystemImage
        fffff801`37943512 83caff          or          edx, 0FFFFFFFFh
        fffff801`37943515 488bcf          mov        rcx, rdi
        fffff801`37943518 488bd8          mov        rbx, rax
        fffff801`3794351b e860b4ebff      call       nt!MiUnloadSystemImage
(fffff801`377fe980)
    */
    CHAR MmUnloadSystemImage_Code[] = "\x83\xCA\xFF" // or          edx, 0FFFFFFFFh
        "\x48\x8B\xCF" // mov        rcx, rdi
        "\x48\x8B\xD8" // mov        rbx, rax
        "\xE8"; // call
    nt!MiUnloadSystemImage (fffff801`377fe980)

    ULONG_PTR MmUnloadSystemImageAddress = 0;
    ULONG_PTR MiUnloadSystemImageAddress = 0;
    ULONG_PTR StartAddress = 0;

    MmUnloadSystemImageAddress =
(ULONG_PTR)GetProcAddress(L"MmUnloadSystemImage");
    if (MmUnloadSystemImageAddress == 0)
    {
        return 0;
    }

    // 在MmUnloadSystemImage中搜索特征码寻找MiUnloadSystemImage
    StartAddress = MmUnloadSystemImageAddress;
    while (StartAddress < MmUnloadSystemImageAddress + 0x500)
    {
        if (memcmp((VOID*)StartAddress, MmUnloadSystemImage_Code,
strlen(MmUnloadSystemImage_Code)) == 0)
```

```

    {
        // 跳过call之前的指令
        StartAddress += strlen(MmUnloadSystemImage_Code);

        // 取出 MiUnloadSystemImage地址
        MiUnloadSystemImageAddress = *(LONG*)StartAddress + StartAddress +
4;
        break;
    }
    ++StartAddress;
}

if (MiUnloadSystemImageAddress != 0)
{
    return MiUnloadSystemImageAddress;
}
return 0;
}

// 特征定位 MiProcessLoaderEntry
MiProcessLoaderEntry GetMiProcessLoaderEntry(ULONG64 StartAddress)
{
    if (StartAddress == 0)
    {
        return NULL;
    }

    while (StartAddress < StartAddress + 0x600)
    {
        // 操作数MiProcessLoaderEntry内存地址是动态变化的
        /*
        lyshark.com: kd> uf MiUnloadSystemImage
        ffffffff801`377fed19 33d2          xor     edx,edx
        ffffffff801`377fed1b 488bcb       mov     rcx,rbx
        ffffffff801`377fed1e e84162b4ff   call    nt!MiProcessLoaderEntry
(ffffffff801`37344f64)
        ffffffff801`377fed23 8b05d756f7ff mov     eax,dword ptr
[nt!PerfGlobalGroupMask (ffffffff801`37774400)]
        ffffffff801`377fed29 a804         test    al,4
        ffffffff801`377fed2b 7440         je
nt!MiUnloadSystemImage+0x3ed (ffffffff801`377fed6d) Branch
E8 call | 8B 05 mov eax
*/

        // ffffffff801`377fed1e | ffffffff801`377fed23
        // 判断特征 0xE8(call) | 0x8B 0x05(mov eax)
        if (*(UCHAR*)StartAddress == 0xE8 && *(UCHAR *) (StartAddress + 5) ==
0x8B && *(UCHAR *) (StartAddress + 6) == 0x05)
        {
            // 跳过一个字节call的E8
            StartAddress++;

            // StartAddress + 1 + 4
            return (MiProcessLoaderEntry)(*(LONG*)StartAddress + StartAddress +
4);

```

```

    }
    ++StartAddress;
}
return NULL;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("卸载完成... \n");
}

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

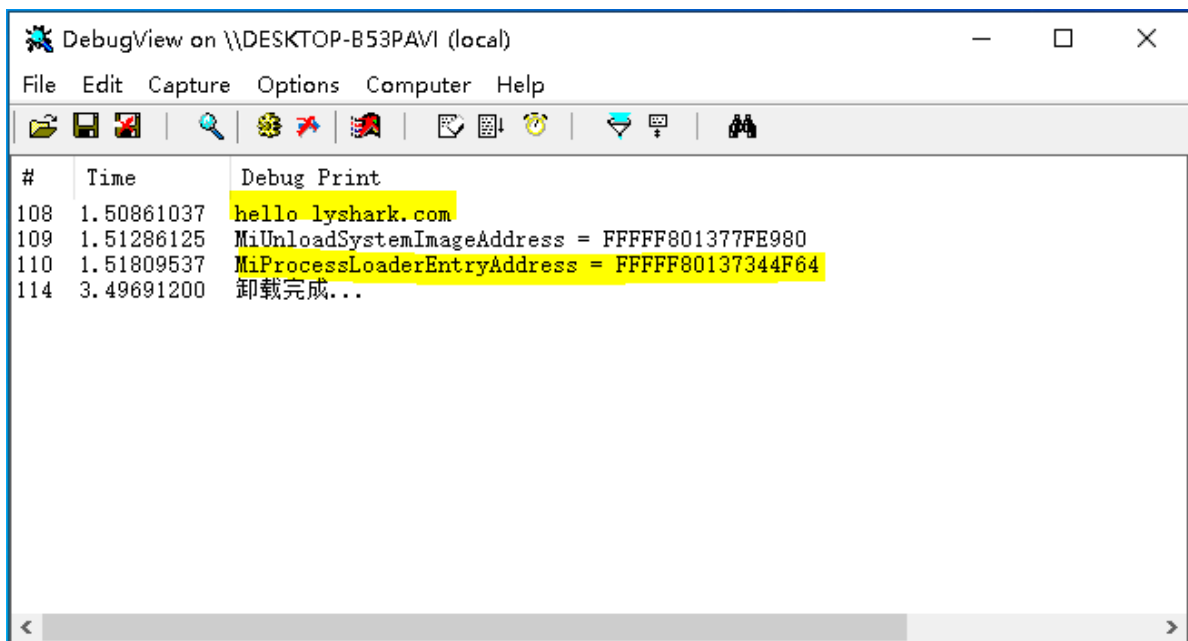
    ULONG64 MiUnloadSystemImageAddress = GetMiUnloadSystemImageAddress();
    DbgPrint("MiUnloadSystemImageAddress = %p \n", MiUnloadSystemImageAddress);

    MiProcessLoaderEntry MiProcessLoaderEntryAddress =
    GetMiProcessLoaderEntry(MiUnloadSystemImageAddress);
    DbgPrint("MiProcessLoaderEntryAddress = %p \n",
    (ULONG64)MiProcessLoaderEntryAddress);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

运行驱动程序，即可得到 MiProcessLoaderEntryAddress 的内存地址。



得到内存地址之后，直接破坏掉自身驱动的入口地址等，即可实现隐藏自身。

```

// PowerBy: LyShark
// Email: me@lyshark.com
#include <ntddk.h>
#include <ntstrsafe.h>

typedef NTSTATUS(*NTQUERYSYSTEMINFORMATION)(

```

```

    IN ULONG SystemInformationClass,
    OUT PVOID SystemInformation,
    IN ULONG_PTR SystemInformationLength,
    OUT PULONG_PTR ReturnLength OPTIONAL);

NTSYSAPI NTSTATUS NTAPI ObReferenceObjectByName(
    __in PUNICODE_STRING ObjectName,
    __in ULONG Attributes,
    __in_opt PACCESS_STATE AccessState,
    __in_opt ACCESS_MASK DesiredAccess,
    __in POBJECT_TYPE ObjectType,
    __in KPROCESSOR_MODE AccessMode,
    __inout_opt PVOID ParseContext,
    __out PVOID* Object
);

typedef struct _SYSTEM_MODULE_INFORMATION
{
    HANDLE Section;
    PVOID MappedBase;
    PVOID Base;
    ULONG Size;
    ULONG Flags;
    USHORT LoadOrderIndex;
    USHORT InitOrderIndex;
    USHORT LoadCount;
    USHORT PathLength;
    CHAR ImageName[256];
} SYSTEM_MODULE_INFORMATION, *PSYSTEM_MODULE_INFORMATION;

typedef struct _LDR_DATA_TABLE_ENTRY
{
    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID DllBase;
    PVOID EntryPoint;
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;

extern POBJECT_TYPE *IoDriverObjectType;
typedef NTSTATUS(__fastcall *MiProcessLoaderEntry)(PVOID pDriverSection, BOOLEAN
bLoad);
ULONG64 MiUnloadSystemImageAddress = 0;

// 取出指定函数地址
PVOID GetProcAddress(WCHAR *FuncName)
{
    UNICODE_STRING u_FuncName = { 0 };
    PVOID ref = NULL;

    RtlInitUnicodeString(&u_FuncName, FuncName);
    ref = MmGetSystemRoutineAddress(&u_FuncName);

    if (ref != NULL)
    {

```

```

        return ref;
    }

    return ref;
}

// 特征定位 MiUnloadSystemImage
ULONG64 GetMiUnloadSystemImageAddress()
{
    CHAR MmUnloadSystemImage_Code[] =
        "\x83\xCA\xFF\x48\x8B\xCF\x48\x8B\xD8\xE8";

    ULONG_PTR MmUnloadSystemImageAddress = 0;
    ULONG_PTR MiUnloadSystemImageAddress = 0;
    ULONG_PTR StartAddress = 0;

    MmUnloadSystemImageAddress =
        (ULONG_PTR)GetProcAddress(L"MmUnloadSystemImage");
    if (MmUnloadSystemImageAddress == 0)
    {
        return 0;
    }

    // 在MmUnloadSystemImage中搜索特征码寻找MiUnloadSystemImage
    StartAddress = MmUnloadSystemImageAddress;
    while (StartAddress < MmUnloadSystemImageAddress + 0x500)
    {
        if (memcmp((VOID*)StartAddress, MmUnloadSystemImage_Code,
            strlen(MmUnloadSystemImage_Code)) == 0)
        {
            StartAddress += strlen(MmUnloadSystemImage_Code);
            MiUnloadSystemImageAddress = *(LONG*)StartAddress + StartAddress + 4;
            break;
        }
        ++StartAddress;
    }

    if (MiUnloadSystemImageAddress != 0)
    {
        return MiUnloadSystemImageAddress;
    }
    return 0;
}

// 特征定位 MiProcessLoaderEntry
MiProcessLoaderEntry GetMiProcessLoaderEntry(ULONG64 StartAddress)
{
    if (StartAddress == 0)
    {
        return NULL;
    }

    while (StartAddress < StartAddress + 0x600)
    {

```

```

        if (*(UCHAR*)StartAddress == 0xE8 && *(UCHAR *)(StartAddress + 5) == 0x8B &&
            *(UCHAR *)(StartAddress + 6) == 0x05)
        {
            StartAddress++;
            return (MiProcessLoaderEntry)(*(LONG*)StartAddress + StartAddress + 4);
        }
        ++StartAddress;
    }
    return NULL;
}

// 根据驱动名获取驱动对象
BOOLEAN GetDriverObjectByName(PDRIVER_OBJECT *DriverObject, WCHAR *DriverName)
{
    PDRIVER_OBJECT TempObject = NULL;
    UNICODE_STRING u_DriverName = { 0 };
    NTSTATUS Status = STATUS_UNSUCCESSFUL;

    RtlInitUnicodeString(&u_DriverName, DriverName);
    Status = ObReferenceObjectByName(&u_DriverName, OBJ_CASE_INSENSITIVE, NULL,
0, *IoDriverObjectType, KernelMode, NULL, &TempObject);
    if (!NT_SUCCESS(Status))
    {
        *DriverObject = NULL;
        return FALSE;
    }

    *DriverObject = TempObject;
    return TRUE;
}

BOOLEAN SupportSEH(PDRIVER_OBJECT DriverObject)
{
    PDRIVER_OBJECT Object = NULL;;
    PLDR_DATA_TABLE_ENTRY LdrEntry = NULL;

    GetDriverObjectByName(&Object, L"\\Driver\\tdx");
    if (Object == NULL)
    {
        return FALSE;
    }

    // 将获取到的驱动对象节点赋值给自身LDR
    LdrEntry = (PLDR_DATA_TABLE_ENTRY)DriverObject->DriverSection;
    LdrEntry->DllBase = Object->DriverStart;
    ObDereferenceObject(Object);
    return TRUE;
}

VOID InitInLoadOrderLinks(PLDR_DATA_TABLE_ENTRY LdrEntry)
{
    InitializeListHead(&LdrEntry->InLoadOrderLinks);
    InitializeListHead(&LdrEntry->InMemoryOrderLinks);
}

```

```

VOID Reinitialize(PDRIVER_OBJECT DriverObject, PVOID Context, ULONG Count)
{
    MiProcessLoaderEntry m_MiProcessLoaderEntry = NULL;
    ULONG *p = NULL;

    m_MiProcessLoaderEntry =
GetMiProcessLoaderEntry(MiUnloadSystemImageAddress);
    if (m_MiProcessLoaderEntry == NULL)
    {
        return;
    }

    SupportSEH(DriverObject);

    m_MiProcessLoaderEntry(DriverObject->DriverSection, 0);
    InitInLoadOrderLinks((PLDR_DATA_TABLE_ENTRY)DriverObject->DriverSection);

    // 破坏驱动对象特征
    DriverObject->DriverSection = NULL;
    DriverObject->DriverStart = NULL;
    DriverObject->DriverSize = 0;
    DriverObject->DriverUnload = NULL;
    DriverObject->DriverInit = NULL;
    DriverObject->DeviceObject = NULL;

    DbgPrint("驱动隐藏 \n");
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("卸载完成... \n");
}

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

    MiUnloadSystemImageAddress = GetMiUnloadSystemImageAddress();
    MiProcessLoaderEntry MiProcessLoaderEntryAddress =
GetMiProcessLoaderEntry(MiUnloadSystemImageAddress);

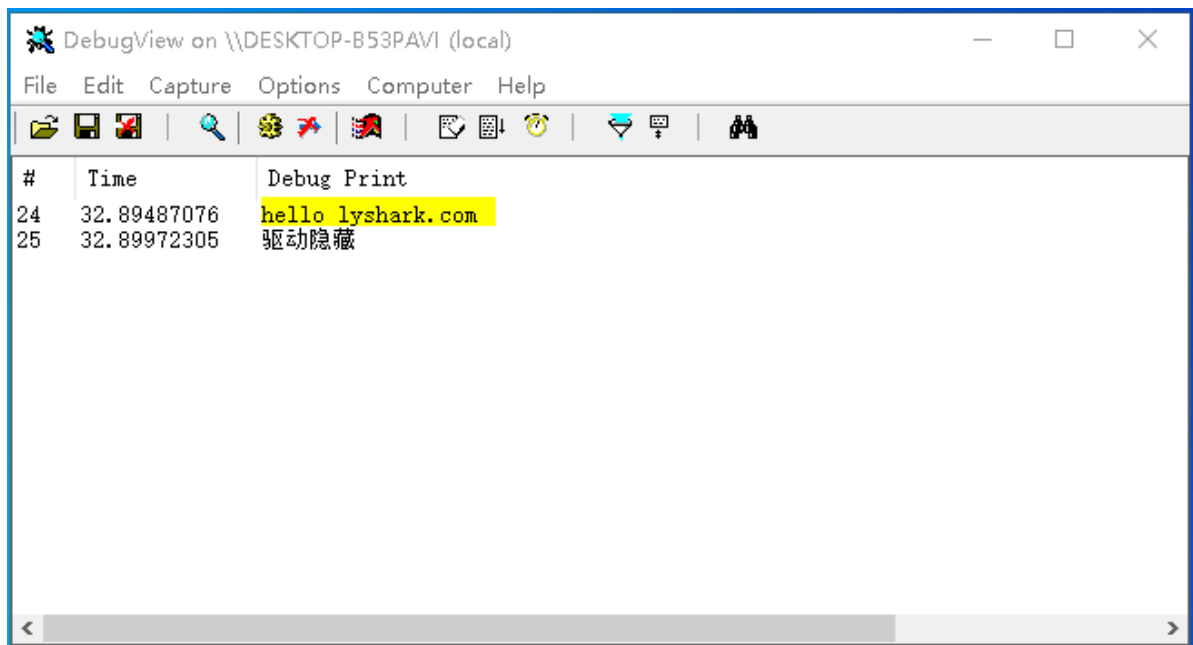
    // 无痕隐藏
    IoRegisterDriverReinitialization(Driver, Reinitialize, NULL);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

运行驱动程序，让后看到如下输出信息；





## 参考文献

<https://blog.csdn.net/zhuhuibeishadiao/article/details/75658816>

<https://github.com/ZhuHuiBeiShaDiao/NewHideDriverEx>

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)