内核枚举进程使用 PspCidTable 这个未公开的函数，它能最大的好处是能得到进程的EPROCESS地址， 由于是未公开的函数，所以我们需要变相的调用这个函数，通过 PsLookupProcessByProcessId 函数查到进程的EPROCESS，如果 PsLookupProcessByProcessId 返回失败，则证明此进程不存在，如果返回成功则把EPROCESS、PID、PPID、进程名等通过DbgPrint打印到屏幕上。

# **内核枚举进程：** 进程就是活动起来的程序，每一个进程在内核里，都有一个名为 的结构记录

EPROCESS

它的详细信息，其中就包括进程名，PID，PPID，进程路径等，通常在应用层枚举进程只列出所有进程 的编号即可，不过在内核层需要把它的 EPROCESS 地址给列举出来。

内核枚举进程使用 PspCidTable 这个未公开的函数，它能最大的好处是能得到进程的EPROCESS地址， 由于是未公开的函数，所以我们需要变相的调用这个函数，通过 PsLookupProcessByProcessId 函数查到进程的EPROCESS，如果 PsLookupProcessByProcessId 返回失败，则证明此进程不存在，如果返回成功则把EPROCESS、PID、PPID、进程名等通过DbgPrint打印到屏幕上。



#include <ntifs.h>

NTKERNELAPI UCHAR PsGetProcessImageFileName(IN PEPROCESS Process); //未公开的进行导出即可

NTKERNELAPI HANDLE PsGetProcessInheritedFromUniqueProcessId(IN PEPROCESS

Process);//未公开进行导出

// 根据进程ID返回进程EPROCESS结构体,失败返回NULL PEPROCESS LookupProcess(HANDLE Pid)

{

PEPROCESS eprocess = NULL;

NTSTATUS Status = STATUS\_UNSUCCESSFUL;

Status = PsLookupProcessByProcessId(Pid, &eprocess); if (NT\_SUCCESS(Status))

return eprocess; return NULL;

}

VOID EnumProcess()

{

PEPROCESS eproc = NULL;

for (int temp = 0; temp < 100000; temp += 4)

{

eproc = LookupProcess((HANDLE)temp); if (eproc != NULL)

{

DbgPrint("进程名: %s --> 进程PID = %d --> 父进程PPID =

%d\r\n",PsGetProcessImageFileName(eproc),PsGetProcessId(eproc), PsGetProcessInheritedFromUniqueProcessId(eproc));

ObDereferenceObject(eproc);

}

}

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint(("Uninstall Driver Is OK \n"));

}

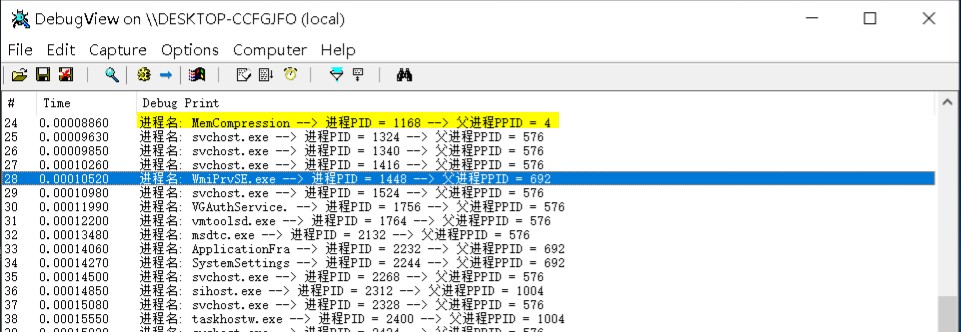
NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

EnumProcess();

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

}



# **内核枚举线程：** 内核线程的枚举与进程相似，线程中也存在一个ETHREAD结构，但在枚举线程之前需要先来枚举到指定进程的eprocess结构，然后在根据eprocess结构对指定线程进行枚举。



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

// 声明API

NTKERNELAPI UCHAR PsGetProcessImageFileName(IN PEPROCESS Process); NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE Id, PEPROCESS Process); NTKERNELAPI NTSTATUS PsLookupThreadByThreadId(HANDLE Id, PETHREAD Thread); NTKERNELAPI PEPROCESS IoThreadToProcess(PETHREAD Thread);

//根据进程ID返回进程EPROCESS，失败返回NULL PEPROCESS LookupProcess(HANDLE Pid)

{

PEPROCESS eprocess = NULL;

if (NT\_SUCCESS(PsLookupProcessByProcessId(Pid, &eprocess))) return eprocess;

else

return NULL;

}

//根据线程ID返回线程ETHREAD，失败返回NULL PETHREAD LookupThread(HANDLE Tid)

{

PETHREAD ethread;

if (NT\_SUCCESS(PsLookupThreadByThreadId(Tid, &ethread))) return ethread;

else

return NULL;

}

//枚举指定进程中的线程

VOID EnumThread(PEPROCESS Process)

{

ULONG i = 0, c = 0; PETHREAD ethrd = NULL; PEPROCESS eproc = NULL;



for (i = 4; i<262144; i = i + 4) // 一般来说没有超过100000的PID和TID

{

ethrd = LookupThread((HANDLE)i); if (ethrd != NULL)

{

//获得线程所属进程

eproc = IoThreadToProcess(ethrd); if (eproc == Process)

{

// 打 印 出 ETHREAD 和 TID DbgPrint("线程: ETHREAD=%p TID=%ld\n",ethrd,

(ULONG)PsGetThreadId(ethrd));

}

ObDereferenceObject(ethrd);

}

}

}

// 通过枚举的方式定位到指定的进程，这里传递一个进程名称

VOID MyEnumThread(char ProcessName)

{

ULONG i = 0;

PEPROCESS eproc = NULL;

for (i = 4; i<100000000; i = i + 4)

{

eproc = LookupProcess((HANDLE)i); if (eproc != NULL)

{

ObDereferenceObject(eproc);

if (strstr(PsGetProcessImageFileName(eproc), ProcessName) != NULL)

{

EnumThread(eproc); // 相等则说明是我们想要的进程，直接枚举其中的线程

}

}

}

}

VOID DriverUnload(IN PDRIVER\_OBJECT DriverObject){}

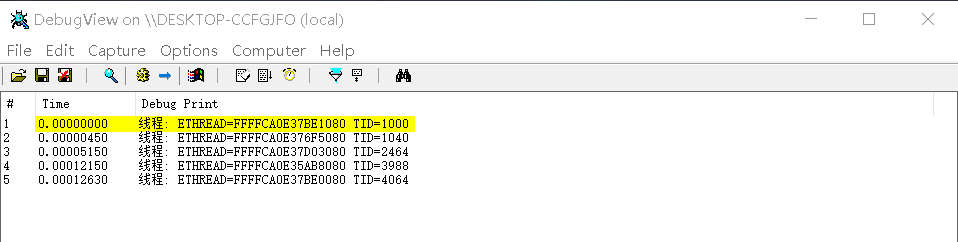
NTSTATUS DriverEntry(IN PDRIVER\_OBJECT DriverObject, IN PUNICODE\_STRING RegistryPath)

{

MyEnumThread("calc.exe");

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

}



**内核枚举进程模块：** 枚举进程中的所有模块信息，DLL模块记录在 PEB 的 LDR 链表里，LDR 是一个双向链表，枚举链表即可，相应的卸载可使用 MmUnmapViewOfSection 函数，分别传入进程的EPROCESS，DLL模块基址即可。

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



//声明结构体

typedef struct \_KAPC\_STATE

{

LIST\_ENTRY ApcListHead[2]; PKPROCESS Process;

UCHAR KernelApcInProgress; UCHAR KernelApcPending; UCHAR UserApcPending;

} KAPC\_STATE, PKAPC\_STATE;

typedef struct \_LDR\_DATA\_TABLE\_ENTRY

{

LIST\_ENTRY64 InLoadOrderLinks; LIST\_ENTRY64 InMemoryOrderLinks; LIST\_ENTRY64 InInitializationOrderLinks; PVOID DllBase;

PVOID EntryPoint;

ULONG SizeOfImage; UNICODE\_STRING FullDllName; UNICODE\_STRING BaseDllName; ULONG Flags;

USHORT LoadCount;

USHORT TlsIndex;

PVOID SectionPointer;

ULONG CheckSum;

PVOID LoadedImports;

PVOID EntryPointActivationContext;

PVOID PatchInformation; LIST\_ENTRY64 ForwarderLinks; LIST\_ENTRY64 ServiceTagLinks; LIST\_ENTRY64 StaticLinks;

PVOID ContextInformation;

ULONG64 OriginalBase; LARGE\_INTEGER LoadTime;

} LDR\_DATA\_TABLE\_ENTRY, PLDR\_DATA\_TABLE\_ENTRY;

ULONG64 LdrInPebOffset = 0x018; //peb.ldr

ULONG64 ModListInPebOffset = 0x010; //peb.ldr.InLoadOrderModuleList



//声明API

NTKERNELAPI UCHAR PsGetProcessImageFileName(IN PEPROCESS Process); NTKERNELAPI PPEB PsGetProcessPeb(PEPROCESS Process);

NTKERNELAPI HANDLE PsGetProcessInheritedFromUniqueProcessId(IN PEPROCESS Process);

//根据进程ID返回进程EPROCESS，失败返回NULL PEPROCESS LookupProcess(HANDLE Pid)

{

PEPROCESS eprocess = NULL;

if (NT\_SUCCESS(PsLookupProcessByProcessId(Pid, &eprocess))) return eprocess;

else

return NULL;

}

//枚举指定进程的模块

VOID EnumModule(PEPROCESS Process)

{

SIZE\_T Peb = 0; SIZE\_T Ldr = 0;

PLIST\_ENTRY ModListHead = 0; PLIST\_ENTRY Module = 0; ANSI\_STRING AnsiString; KAPC\_STATE ks;

//EPROCESS地址无效则退出

if (!MmIsAddressValid(Process)) return;

//获取PEB地址

Peb = (SIZE\_T)PsGetProcessPeb(Process);

//PEB地址无效则退出if (!Peb)

return;

//依附进程KeStackAttachProcess(Process, &ks);

try

{

//获得LDR地址

Ldr = Peb + (SIZE\_T)LdrInPebOffset;

//测试是否可读，不可读则抛出异常退出ProbeForRead((CONST PVOID)Ldr, 8, 8);

//获得链表头

ModListHead = (PLIST\_ENTRY)( (PULONG64)Ldr + ModListInPebOffset);

//再次测试可读性

ProbeForRead((CONST PVOID)ModListHead, 8, 8);

//获得第一个模块的信息

Module = ModListHead->Flink; while (ModListHead != Module)

{

//打印信息：基址、大小、DLL路径

DbgPrint("模块基址=%p 大小=%ld 路径=%wZ\n",(PVOID) (((PLDR\_DATA\_TABLE\_ENTRY)Module)->DllBase),

(ULONG)(((PLDR\_DATA\_TABLE\_ENTRY)Module)->SizeOfImage),& (((PLDR\_DATA\_TABLE\_ENTRY)Module)->FullDllName));



Module = Module->Flink;

//测试下一个模块信息的可读性ProbeForRead((CONST PVOID)Module, 80, 8);

}

}

except (EXCEPTION\_EXECUTE\_HANDLER){;}

//取消依附进程KeUnstackDetachProcess(&ks);

}

// 通过枚举的方式定位到指定的进程，这里传递一个进程名称

VOID MyEnumModule(char ProcessName)

{

ULONG i = 0;

PEPROCESS eproc = NULL;

for (i = 4; i<100000000; i = i + 4)

{

eproc = LookupProcess((HANDLE)i); if (eproc != NULL)

{

ObDereferenceObject(eproc);

if (strstr(PsGetProcessImageFileName(eproc), ProcessName) != NULL)

{

EnumModule(eproc); // 相等则说明是我们想要的进程，直接枚举其中的线程

}

}

}

}

VOID DriverUnload(IN PDRIVER\_OBJECT DriverObject){}

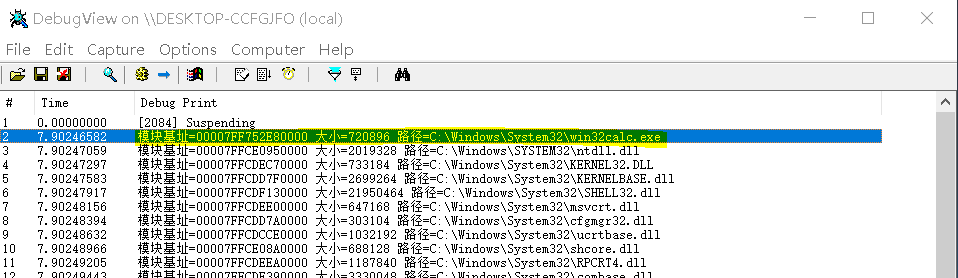
NTSTATUS DriverEntry(IN PDRIVER\_OBJECT DriverObject, IN PUNICODE\_STRING RegistryPath)

{

MyEnumModule("calc.exe");

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

}



**内核枚举加载SYS文件：** 内核中的SYS文件也是通过双向链表的方式相连接的，我们可以通过遍历

LDR\_DATA\_TABLE\_ENTRY 结构(遍历自身DriverSection成员)，就能够得到全部的模块信息。

#include <ntddk.h> #include <wdm.h>

typedef struct \_LDR\_DATA\_TABLE\_ENTRY { LIST\_ENTRY InLoadOrderLinks; LIST\_ENTRY InMemoryOrderLinks; LIST\_ENTRY InInitializationOrderLinks; PVOID DllBase;



PVOID EntryPoint;

ULONG SizeOfImages; UNICODE\_STRING FullDllName; UNICODE\_STRING BaseDllName; ULONG Flags;

USHORT LoadCount; USHORT TlsIndex; union {

LIST\_ENTRY HashLinks; struct {

PVOID SectionPointer;

ULONG CheckSum;

};

};

union {

struct {

ULONG TimeDateStamp;

};

struct {

PVOID LoadedImports;

};

};

}LDR\_DATA\_TABLE\_ENTRY, PLDR\_DATA\_TABLE\_ENTRY;

VOID DriverUnload(IN PDRIVER\_OBJECT DriverObject){}

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

{

ULONG count = 0; NTSTATUS Status;

DriverObject->DriverUnload = DriverUnload;

PLDR\_DATA\_TABLE\_ENTRY pLdr = NULL;

PLIST\_ENTRY pListEntry = NULL; PLDR\_DATA\_TABLE\_ENTRY pModule = NULL;

PLIST\_ENTRY pCurrentListEntry = NULL;

pLdr = (PLDR\_DATA\_TABLE\_ENTRY)DriverObject->DriverSection; pListEntry = pLdr->InLoadOrderLinks.Flink; pCurrentListEntry = pListEntry->Flink;

while (pCurrentListEntry != pListEntry)

{

pModule = CONTAINING\_RECORD(pCurrentListEntry, LDR\_DATA\_TABLE\_ENTRY, InLoadOrderLinks);

if (pModule->BaseDllName.Buffer != 0)

{

DbgPrint("基址：%p ---> 偏移：%p ---> 结束地址：%p---> 模块名：%wZ \r\n", pModule->DllBase, pModule->SizeOfImages - (LONGLONG)pModule->DllBase,

(LONGLONG)pModule->DllBase + pModule->SizeOfImages,pModule-

>BaseDllName);

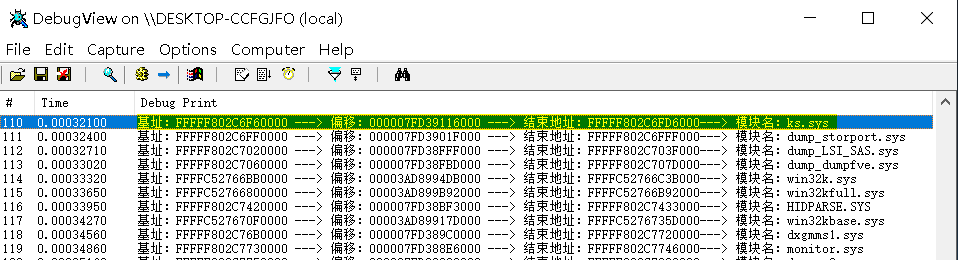
}

pCurrentListEntry = pCurrentListEntry->Flink;

}

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

}



# 本书作者： 王瑞 (LyShark)

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

作者博客： h ttps://lyshark.cnblogs.com

团队首页： w ww.lyshark.com