PEB结构 (Process Envirorment Block Structure) 其中文名是进程环境块信息，进程环境块内部包含了进程运行的详细参数信息，每一个进程在运行后都会存在一个特有的PEB结构，通过附加进程并遍 历这段结构即可得到非常多的有用信息。

# 在应用层下，如果想要得到PEB的基地址只需要取 fs:[0x30] 即可，TEB线程环境块则是 fs:[0x18] ， 如果在内核层想要得到应用层进程的PEB信息我们需要调用特定的内核函数来获取，如下案例将教大家 如何在内核层取到应用层进程的PEB结构。

首先在开始写代码之前需要先定义好 PEB 进程环境快结构体，用于对内存指针解析，新建 peb.h 文件并保存如下代码，这些是微软的结构定义分为32位与64位，官方定义规范而已不需要费工夫。

#pragma once #include <ntifs.h>



typedef struct \_CURDIR // 2 elements, 0x18 bytes (sizeof)

{

/ 0x000 / struct \_UNICODE\_STRING DosPath; // 3 elements, 0x10 bytes (sizeof)

/ 0x010 / VOID Handle;

}CURDIR, PCURDIR;

typedef struct \_RTL\_DRIVE\_LETTER\_CURDIR // 4 elements, 0x18 bytes (sizeof)

{

/ 0x000 / UINT16 Flags;

/ 0x002 / UINT16 Length;

/ 0x004 / ULONG32 TimeStamp;

/ 0x008 / struct \_STRING DosPath; // 3 elements, 0x10 bytes (sizeof)

}RTL\_DRIVE\_LETTER\_CURDIR, PRTL\_DRIVE\_LETTER\_CURDIR;

typedef enum \_SYSTEM\_DLL\_TYPE // 7 elements, 0x4 bytes

{

PsNativeSystemDll = 0 / 0x0 /, PsWowX86SystemDll = 1 / 0x1 /, PsWowArm32SystemDll = 2 / 0x2 /, PsWowAmd64SystemDll = 3 / 0x3 /, PsWowChpeX86SystemDll = 4 / 0x4 /, PsVsmEnclaveRuntimeDll = 5 / 0x5 /, PsSystemDllTotalTypes = 6 / 0x6 /

}SYSTEM\_DLL\_TYPE, PSYSTEM\_DLL\_TYPE;

typedef struct \_EWOW64PROCESS // 3 elements, 0x10 bytes (sizeof)

{

/ 0x000 / VOID Peb;

/ 0x008 / UINT16 Machine;

/ 0x00A / UINT8 \_PADDING0\_[0x2];

/ 0x00C / enum \_SYSTEM\_DLL\_TYPE NtdllType;

}EWOW64PROCESS, PEWOW64PROCESS;

typedef struct \_RTL\_USER\_PROCESS\_PARAMETERS // 37 elements, 0x440 bytes (sizeof)

{

/ 0x000 / ULONG32 MaximumLength;

/ 0x004 / ULONG32 Length;



/ 0x008 / ULONG32 Flags;

/ 0x00C / ULONG32 DebugFlags;

/ 0x010 / VOID ConsoleHandle;

/ 0x018 / ULONG32 ConsoleFlags;

/ 0x01C / UINT8 \_PADDING0\_[0x4];

/ 0x020 / VOID StandardInput;

/ 0x028 / VOID StandardOutput;

/ 0x030 / VOID StandardError;

/ 0x038 / struct \_CURDIR CurrentDirectory; // 2 elements, 0x18 bytes (sizeof)

/ 0x050 / struct \_UNICODE\_STRING DllPath; // 3 elements, 0x10 bytes (sizeof)

/ 0x060 / struct \_UNICODE\_STRING ImagePathName; // 3 elements, 0x10 bytes (sizeof)

/ 0x070 / struct \_UNICODE\_STRING CommandLine; // 3

elements, 0x10 bytes (sizeof)

/ 0x080 / VOID Environment;

/ 0x088 / ULONG32 StartingX;

/ 0x08C / ULONG32 StartingY;

/ 0x090 / ULONG32 CountX;

/ 0x094 / ULONG32 CountY;

/ 0x098 / ULONG32 CountCharsX;

/ 0x09C / ULONG32 CountCharsY;

/ 0x0A0 / ULONG32 FillAttribute;

/ 0x0A4 / ULONG32 WindowFlags;

/ 0x0A8 / ULONG32 ShowWindowFlags;

/ 0x0AC / UINT8 \_PADDING1\_[0x4];

/ 0x0B0 / struct \_UNICODE\_STRING WindowTitle; // 3 elements, 0x10 bytes (sizeof)

/ 0x0C0 / struct \_UNICODE\_STRING DesktopInfo; // 3 elements, 0x10 bytes (sizeof)

/ 0x0D0 / struct \_UNICODE\_STRING ShellInfo; // 3 elements, 0x10 bytes (sizeof)

/ 0x0E0 / struct \_UNICODE\_STRING RuntimeData; // 3

elements, 0x10 bytes (sizeof)

/ 0x0F0 / struct \_RTL\_DRIVE\_LETTER\_CURDIR CurrentDirectores[32];

/ 0x3F0 / UINT64 EnvironmentSize;

/ 0x3F8 / UINT64 EnvironmentVersion;

/ 0x400 / VOID PackageDependencyData;

/ 0x408 / ULONG32 ProcessGroupId;

/ 0x40C / ULONG32 LoaderThreads;

/ 0x410 / struct \_UNICODE\_STRING RedirectionDllName; // 3 elements, 0x10 bytes (sizeof)

/ 0x420 / struct \_UNICODE\_STRING HeapPartitionName; // 3

elements, 0x10 bytes (sizeof)

/ 0x430 / UINT64 DefaultThreadpoolCpuSetMasks;

/ 0x438 / ULONG32 DefaultThreadpoolCpuSetMaskCount;

/ 0x43C / UINT8 \_PADDING2\_[0x4];

}RTL\_USER\_PROCESS\_PARAMETERS, PRTL\_USER\_PROCESS\_PARAMETERS;

typedef struct \_PEB\_LDR\_DATA // 9 elements, 0x58 bytes (sizeof)

{

/ 0x000 / ULONG32 Length;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| /  / | 0x004  0x005 | /  / | UINT8  UINT8 | Initialized;  \_PADDING0\_[0x3]; |
| / | 0x008 | / | VOID | SsHandle; |
| / 0x010 / struct \_LIST\_ENTRY InLoadOrderModuleList; // 2 elements, 0x10 bytes (sizeof)  / 0x020 / struct \_LIST\_ENTRY InMemoryOrderModuleList; // 2  elements, 0x10 bytes (sizeof)  / 0x030 / struct \_LIST\_ENTRY InInitializationOrderModuleList; // 2 elements, 0x10 bytes (sizeof)  / 0x040 / VOID EntryInProgress;  / 0x048 / UINT8 ShutdownInProgress;  / 0x049 / UINT8 \_PADDING1\_[0x7];  / 0x050 / VOID ShutdownThreadId;  }PEB\_LDR\_DATA, PPEB\_LDR\_DATA;  typedef struct \_PEB64  {  UCHAR InheritedAddressSpace; UCHAR ReadImageFileExecOptions; UCHAR BeingDebugged;  UCHAR BitField; ULONG64 Mutant;  ULONG64 ImageBaseAddress; PPEB\_LDR\_DATA Ldr;  PRTL\_USER\_PROCESS\_PARAMETERS ProcessParameters; ULONG64 SubSystemData;  ULONG64 ProcessHeap; ULONG64 FastPebLock; ULONG64 AtlThunkSListPtr; ULONG64 IFEOKey;  ULONG64 CrossProcessFlags; ULONG64 UserSharedInfoPtr; ULONG SystemReserved; ULONG AtlThunkSListPtr32; ULONG64 ApiSetMap;  } PEB64, PPEB64;  #pragma pack(4) typedef struct \_PEB32  {  UCHAR InheritedAddressSpace; UCHAR ReadImageFileExecOptions; UCHAR BeingDebugged;  UCHAR BitField; ULONG Mutant;  ULONG ImageBaseAddress; ULONG Ldr;  ULONG ProcessParameters;  ULONG SubSystemData; ULONG ProcessHeap; ULONG FastPebLock; ULONG AtlThunkSListPtr; ULONG IFEOKey;  ULONG CrossProcessFlags;  ULONG UserSharedInfoPtr; | | | | |



ULONG SystemReserved; ULONG AtlThunkSListPtr32; ULONG ApiSetMap;



} PEB32, PPEB32;

typedef struct \_PEB\_LDR\_DATA32

{

ULONG Length; BOOLEAN Initialized; ULONG SsHandle;

LIST\_ENTRY32 InLoadOrderModuleList; LIST\_ENTRY32 InMemoryOrderModuleList; LIST\_ENTRY32 InInitializationOrderModuleList; ULONG EntryInProgress;

} PEB\_LDR\_DATA32, PPEB\_LDR\_DATA32;

typedef struct \_LDR\_DATA\_TABLE\_ENTRY32

{

LIST\_ENTRY32 InLoadOrderLinks; LIST\_ENTRY32 InMemoryOrderModuleList;

LIST\_ENTRY32 InInitializationOrderModuleList; ULONG DllBase;

ULONG EntryPoint; ULONG SizeOfImage;

UNICODE\_STRING32 FullDllName; UNICODE\_STRING32 BaseDllName; ULONG Flags;

USHORT LoadCount; USHORT TlsIndex; union

{

LIST\_ENTRY32 HashLinks; ULONG SectionPointer;

}u1;

ULONG CheckSum; union

{

ULONG TimeDateStamp; ULONG LoadedImports;

}u2;

ULONG EntryPointActivationContext; ULONG PatchInformation;

} LDR\_DATA\_TABLE\_ENTRY32, PLDR\_DATA\_TABLE\_ENTRY32;

#pragma pack()

接着就来实现对PEB的获取操作，以 64位 为例，我们需要调用 PsGetProcessPeb() 这个内核函数，因为该内核函数没有被公开所以调用之前需要头部导出，该函数需要传入用户进程的 EProcess 结构，该结构可用 PsLookupProcessByProcessId 函数动态获取到，获取到以后直接

KeStackAttachProcess() 附加到应用层进程上，即可直接输出进程的PEB结构信息，如下代码。

#include "peb.h" #include <ntifs.h>

// 定义导出

NTKERNELAPI PVOID NTAPI PsGetProcessPeb(\_In\_ PEPROCESS Process);

VOID UnDriver(PDRIVER\_OBJECT driver)

{

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

}

// LyShark

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

{

DbgPrint("hello lyshark \n");

NTSTATUS status = STATUS\_UNSUCCESSFUL; PEPROCESS eproc = NULL;

KAPC\_STATE kpc = { 0 };

PPEB64 pPeb64 = NULL;

try

{

// HANDLE)4656 进程PID

status = PsLookupProcessByProcessId((HANDLE)4656, &eproc);

// 得到64位PEB

pPeb64 = (PPEB64)PsGetProcessPeb(eproc);

DbgPrint("PEB64 = %p \n", pPeb64);

if (pPeb64 != 0)

{

// 验证可读性

ProbeForRead(pPeb64, sizeof(PEB32), 1);

// 附加进程

KeStackAttachProcess(eproc, &kpc);

DbgPrint("进程基地址: 0x%p \n", pPeb64->ImageBaseAddress); DbgPrint("ProcessHeap = 0x%p \n", pPeb64->ProcessHeap);

DbgPrint("BeingDebugged = %d \n", pPeb64->BeingDebugged);

// 脱离进程

KeUnstackDetachProcess(&kpc);

}

}

except (EXCEPTION\_EXECUTE\_HANDLER)

{

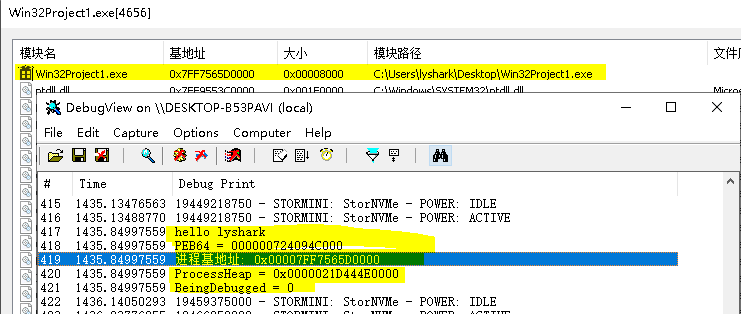
Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

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

}

# PEB64代码运行后，我们加载驱动即可看到如下结果:



而相对于64位进程来说，获取 32位 进程的PEB信息可以直接调用 PsGetProcessWow64Process() 函数得到，该函数已被导出可以任意使用，获取PEB代码如下。

#include "peb.h" #include <ntifs.h>

// 定义导出

NTKERNELAPI PVOID NTAPI PsGetProcessPeb(\_In\_ PEPROCESS Process);

VOID UnDriver(PDRIVER\_OBJECT driver)

{

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

}

// LyShark

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

{

DbgPrint("hello lyshark \n");

NTSTATUS status = STATUS\_UNSUCCESSFUL; PEPROCESS eproc = NULL;

KAPC\_STATE kpc = { 0 };

PPEB32 pPeb32 = NULL;

try

{

// HANDLE)4656 进程PID

status = PsLookupProcessByProcessId((HANDLE)6164, &eproc);

// 得到32位PEB

pPeb32 = (PPEB32)PsGetProcessWow64Process(eproc);

DbgPrint("PEB32 = %p \n", pPeb32);

if (pPeb32 != 0)

{

// 验证可读性

ProbeForRead(pPeb32, sizeof(PEB32), 1);

// 附加进程

KeStackAttachProcess(eproc, &kpc);

DbgPrint("进程基地址: 0x%p \n", pPeb32->ImageBaseAddress); DbgPrint("ProcessHeap = 0x%p \n", pPeb32->ProcessHeap); DbgPrint("BeingDebugged = %d \n", pPeb32->BeingDebugged);

// 脱离进程

KeUnstackDetachProcess(&kpc);

}

}

except (EXCEPTION\_EXECUTE\_HANDLER)

{

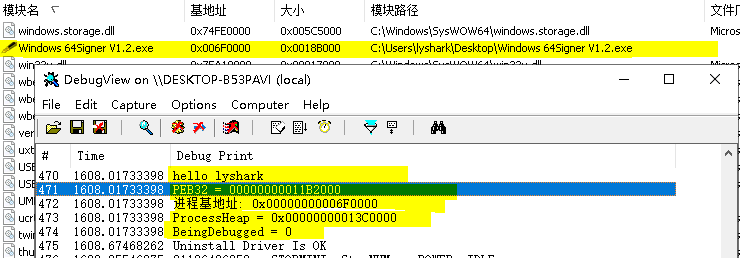
Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

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

}

# PEB32代码运行后，我们加载驱动即可看到如下结果:



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！