在上一篇文章 《驱动开发：内核中实现Dump进程转储》 中我们实现了ARK工具的转存功能，本篇文章继续以内存为出发点介绍 VAD 结构，该结构的全程是 Virtual Address Descriptor 即 虚拟地址描述符 ， VAD是一个 AVL 自 平衡二叉树 ，树的每一个节点代表一段虚拟地址空间。程序中的代码段，数据段，堆段都会各种占用一个或多个 VAD 节点，由一个 MMVAD 结构完整描述。

# VAD结构的遍历效果如下:



那么这个结构在哪？每一个进程都有自己单独的 VAD 结构树，这个结构通常在 EPROCESS 结构里面里面，在内核调试模式下使用 dt \_EPROCESS 可得到如下信息。

lyshark.com 1: kd> dt \_EPROCESS ntdll!\_EPROCESS

+0x500 Vm : \_MMSUPPORT\_FULL

+0x640 MmProcessLinks : \_LIST\_ENTRY

+0x650 ModifiedPageCount : Uint4B

+0x654 ExitStatus : Int4B

+0x658 VadRoot : \_RTL\_AVL\_TREE

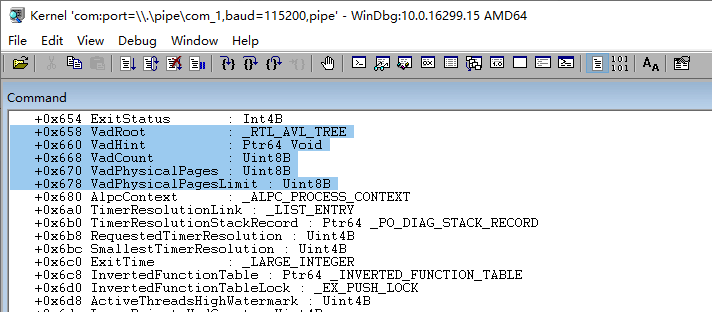
+0x660 VadHint : Ptr64 Void

+0x668 VadCount : Uint8B

+0x670 VadPhysicalPages : Uint8B

+0x678 VadPhysicalPagesLimit : Uint8B

可以看到在本系统中VAD的偏移是 +0x658 紧跟其后的还有 vadCount 的计数等。



# VAD结构是如何被添加的？通常情况下系统调用 VirtualAllocate 等申请一段堆内存时，则会在VAD树上增加一个结点 \_MMVAD 结构体，需要说明的是栈并不受VAD的管理。由系统直接分配空间，并把地址记录在了TEB中。

lyshark.com 0: kd> dt \_MMVAD nt!\_MMVAD

+0x000 Core : \_MMVAD\_SHORT

+0x040 u2 : <anonymous-tag>

+0x048 Subsection : Ptr64 \_SUBSECTION

+0x050 FirstPrototypePte : Ptr64 \_MMPTE

+0x058 LastContiguousPte : Ptr64 \_MMPTE

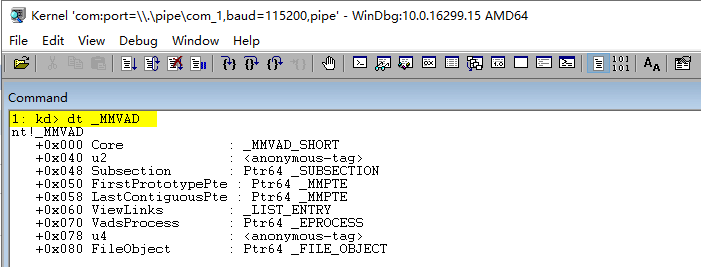
+0x060 ViewLinks : \_LIST\_ENTRY

+0x070 VadsProcess : Ptr64 \_EPROCESS

+0x078 u4 : <anonymous-tag>

+0x080 FileObject : Ptr64 \_FILE\_OBJECT

结构体 MMVAD 则是每一个 VAD 内存块的属性，这个内存结构定义在WinDBG中可看到。



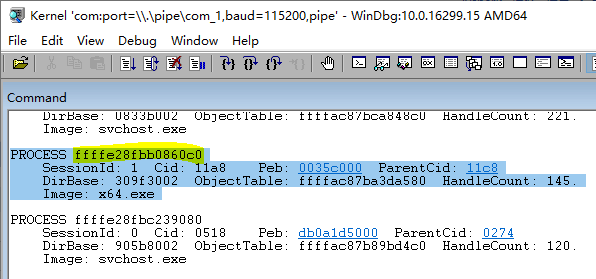
如上在 EPROCESS 结构中可以找到VAD结构的相对偏移 +0x658 以及进程VAD计数偏移 +0x668 ，我们首先通过 !process 0 0 指令得到当前所有进程的 EPROCESS 结构，并选中进程。

lyshark.com 0: kd> !process 0 0 PROCESS ffffe28fbb0860c0

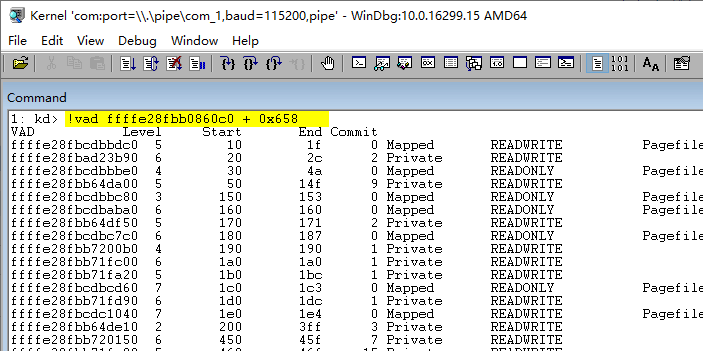
SessionId: 1 Cid: 11a8 Peb: 0035c000 ParentCid: 11c8

DirBase: 309f3002 ObjectTable: ffffac87ba3da580 HandleCount: 145. Image: x64.exe

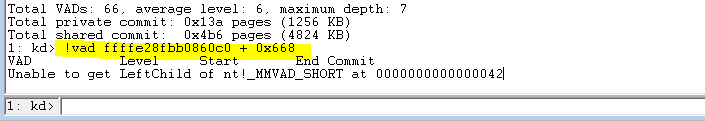
此处的 ffffe28fbb0860c0 正是我们所需要的 EPROCESS 结构。



当需要得到该进程的VAD结构时，只需要使用 !vad ffffe28fbb0860c0 + 0x658 来显示该进程的VAD 树。



至于获取VAD有多少条，则可以直接使用 !vad ffffe28fbb0860c0 + 0x668 来获取到。



# 既然手动可以遍历出来，那么自动化也并不难，首先定义头文件 vad.h 同样这是微软定义，如果想要的到最新的，自己下载WinDBG调试内核输入命令。



#pragma once #include <ntifs.h>

typedef struct \_MM\_GRAPHICS\_VAD\_FLAGS

{

// 15 elements, 0x4 bytes (sizeof)

/ 0x000 / ULONG32 Lock : 1; // 0 BitPosition

/ 0x000 /

ULONG32

LockContended : 1;

// 1 BitPosition

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| / | 0x000 | / | ULONG32 | DeleteInProgress : 1; | | // | 2 | BitPosition |
| / | 0x000 | / | ULONG32 | NoChange : 1; | | // | 3 | BitPosition |
| / | 0x000 | / | ULONG32 | VadType : 3; | | // | 4 | BitPosition |
| / | 0x000 | / | ULONG32 | Protection : 5; | | // | 7 | BitPosition |
| / | 0x000 | / | ULONG32 | PreferredNode : 6; | | // | 12 | BitPosition |
| / | 0x000 | / | ULONG32 | PageSize : 2; | | // | 18 | BitPosition |
| / | 0x000 | / | ULONG32 | PrivateMemoryAlwaysSet : 1; | | // | 20 | BitPosition |
| / | 0x000 | / | ULONG32 | WriteWatch : 1; | | // | 21 | BitPosition |
| / | 0x000 | / | ULONG32 | FixedLargePageSize : 1; | | // | 22 | BitPosition |
| / | 0x000 | / | ULONG32 | ZeroFillPagesOptional : 1; | | // | 23 | BitPosition |
| / | 0x000 | / | ULONG32 | GraphicsAlwaysSet : 1; | | // | 24 | BitPosition |
| / | 0x000 | / | ULONG32 | GraphicsUseCoherentBus : 1; | | // | 25 | BitPosition |
| / | 0x000 | / | ULONG32 | GraphicsPageProtection : 3; | | // | 26 | BitPosition |
| }MM\_GRAPHICS\_VAD\_FLAGS, PMM\_GRAPHICS\_VAD\_FLAGS;  typedef struct \_MM\_PRIVATE\_VAD\_FLAGS // 15 elements, 0x4 bytes (sizeof)  {  / 0x000 / ULONG32 Lock : 1; // 0 BitPosition  / 0x000 / ULONG32 LockContended : 1; // 1 BitPosition | | | | | | | | |
| / | 0x000 | / | ULONG32 | DeleteInProgress : 1; |  | // | 2 | BitPosition |
| / | 0x000 | / | ULONG32 | NoChange : 1; |  | // | 3 | BitPosition |
| / | 0x000 | / | ULONG32 | VadType : 3; |  | // | 4 | BitPosition |
| / | 0x000 | / | ULONG32 | Protection : 5; |  | // | 7 | BitPosition |
| / | 0x000 | / | ULONG32 | PreferredNode : 6; |  | // | 12 | BitPosition |
| / | 0x000 | / | ULONG32 | PageSize : 2; |  | // | 18 | BitPosition |
| / | 0x000 | / | ULONG32 | PrivateMemoryAlwaysSet : | 1; | // | 20 | BitPosition |
| / | 0x000 | / | ULONG32 | WriteWatch : 1; |  | // | 21 | BitPosition |
| / | 0x000 | / | ULONG32 | FixedLargePageSize : 1; |  | // | 22 | BitPosition |
| / | 0x000 | / | ULONG32 | ZeroFillPagesOptional : | 1; | // | 23 | BitPosition |
| / | 0x000 | / | ULONG32 | Graphics : 1; |  | // | 24 | BitPosition |



/ 0x000 / ULONG32 Enclave : 1; // 25 BitPosition



/ 0x000 / ULONG32 ShadowStack : 1; // 26 BitPosition

}MM\_PRIVATE\_VAD\_FLAGS, PMM\_PRIVATE\_VAD\_FLAGS;

typedef struct \_MMVAD\_FLAGS // 9 elements, 0x4 bytes (sizeof)

{

/ 0x000 / ULONG32 Lock : 1; // 0 BitPosition

/ 0x000 / ULONG32 LockContended : 1; // 1 BitPosition

/ 0x000 / ULONG32 DeleteInProgress : 1; // 2 BitPosition

/ 0x000 / ULONG32 NoChange : 1; // 3 BitPosition

/ 0x000 / ULONG32 VadType : 3; // 4 BitPosition

/ 0x000 / ULONG32 Protection : 5; // 7 BitPosition

/ 0x000 / ULONG32 PreferredNode : 6; // 12 BitPosition

/ 0x000 / ULONG32 PageSize : 2; // 18 BitPosition

/ 0x000 / ULONG32 PrivateMemory : 1; // 20 BitPosition

}MMVAD\_FLAGS, PMMVAD\_FLAGS;

typedef struct \_MM\_SHARED\_VAD\_FLAGS // 11 elements, 0x4 bytes (sizeof)

{

/ 0x000 / ULONG32 Lock : 1; // 0 BitPosition

/ 0x000 / ULONG32 LockContended : 1; // 1 BitPosition

/ 0x000 / ULONG32 DeleteInProgress : 1; // 2 BitPosition

/ 0x000 / ULONG32 NoChange : 1; // 3 BitPosition

/ 0x000 / ULONG32 VadType : 3; // 4 BitPosition

/ 0x000 / ULONG32 Protection : 5; // 7 BitPosition

/ 0x000 / ULONG32 PreferredNode : 6; // 12 BitPosition

/ 0x000 / ULONG32 PageSize : 2; // 18 BitPosition

/ 0x000 / ULONG32 PrivateMemoryAlwaysClear : 1; // 20 BitPosition

/ 0x000 / ULONG32 PrivateFixup : 1; // 21 BitPosition

/ 0x000 / ULONG32 HotPatchAllowed : 1; // 22 BitPosition

}MM\_SHARED\_VAD\_FLAGS, PMM\_SHARED\_VAD\_FLAGS;

typedef struct \_MMVAD\_FLAGS2 // 7 elements, 0x4 bytes (sizeof)

{

/ 0x000 / ULONG32 FileOffset : 24; // 0 BitPosition

/ 0x000 / ULONG32 Large : 1; // 24 BitPosition

/ 0x000 / ULONG32 TrimBehind : 1; // 25 BitPosition

/ 0x000 / ULONG32 Inherit : 1; // 26 BitPosition

/ 0x000 / ULONG32 NoValidationNeeded : 1; // 27 BitPosition

/ 0x000 / ULONG32 PrivateDemandZero : 1; // 28 BitPosition

/ 0x000 / ULONG32 Spare : 3; // 29 BitPosition

}MMVAD\_FLAGS2, PMMVAD\_FLAGS2;

typedef struct \_MMVAD\_SHORT

{

RTL\_BALANCED\_NODE VadNode;

UINT32 StartingVpn; / 0x18 /

UINT32 EndingVpn; / 0x01C / UCHAR StartingVpnHigh;

UCHAR EndingVpnHigh;

UCHAR CommitChargeHigh; UCHAR SpareNT64VadUChar; INT32 ReferenceCount;

EX\_PUSH\_LOCK PushLock; / 0x028 / struct

{

union

{

/ 0x030 /

ULONG\_PTR flag;

MM\_PRIVATE\_VAD\_FLAGS PrivateVadFlags;

MMVAD\_FLAGS VadFlags; MM\_GRAPHICS\_VAD\_FLAGS GraphicsVadFlags; MM\_SHARED\_VAD\_FLAGS SharedVadFlags;

}Flags;



}u1;

PVOID EventList; / 0x038 /

}MMVAD\_SHORT, PMMVAD\_SHORT;

typedef struct \_MMADDRESS\_NODE

{

ULONG64 u1;

struct \_MMADDRESS\_NODE LeftChild; struct \_MMADDRESS\_NODE RightChild; ULONG64 StartingVpn;

ULONG64 EndingVpn;

}MMADDRESS\_NODE, PMMADDRESS\_NODE;

typedef struct \_MMEXTEND\_INFO // 2 elements, 0x10 bytes (sizeof)

{

/ 0x000 / UINT64 CommittedSize;

/ 0x008 / ULONG32 ReferenceCount;

/ 0x00C / UINT8 \_PADDING0\_[0x4];

}MMEXTEND\_INFO, PMMEXTEND\_INFO;

struct \_SEGMENT

{

struct \_CONTROL\_AREA ControlArea; ULONG TotalNumberOfPtes;

ULONG SegmentFlags;

ULONG64 NumberOfCommittedPages; ULONG64 SizeOfSegment;

union

{

}u;

struct \_MMEXTEND\_INFO ExtendInfo; void BasedAddress;

ULONG64 SegmentLock; ULONG64 u1;



ULONG64 u2;

PVOID PrototypePte; ULONGLONG ThePtes[0x1];

};

typedef struct \_EX\_FAST\_REF

{

union

{

PVOID Object; ULONG\_PTR RefCnt : 3;

ULONG\_PTR Value;

};

} EX\_FAST\_REF, PEX\_FAST\_REF;

typedef struct \_CONTROL\_AREA // 17 elements, 0x80 bytes (sizeof)

{

/ 0x000 / struct \_SEGMENT Segment;

union // 2 elements, 0x10 bytes

(sizeof)

{

/ 0x008 / struct \_LIST\_ENTRY ListHead; // 2 elements, 0x10 bytes (sizeof)

/ 0x008 / VOID AweContext;

};

/ 0x018 / UINT64 NumberOfSectionReferences;

/ 0x020 / UINT64 NumberOfPfnReferences;

/ 0x028 / UINT64 NumberOfMappedViews;

/ 0x030 / UINT64 NumberOfUserReferences;

/ 0x038 / ULONG32 u; // 2 elements, 0x4 bytes (sizeof)

/ 0x03C / ULONG32 u1; // 2 elements, 0x4 bytes (sizeof)



/ 0x040 / struct \_EX\_FAST\_REF FilePointer; // 3 elements,

0x8 bytes (sizeof)

// 4 elements, 0x8 bytes (sizeof)

}CONTROL\_AREA, PCONTROL\_AREA;

typedef struct \_SUBSECTION\_

{

struct \_CONTROL\_AREA ControlArea;

}SUBSECTION, PSUBSECTION;

typedef struct \_MMVAD

{

MMVAD\_SHORT Core;

union / 0x040 /

{

UINT32 LongFlags2;

//现在用不到省略MMVAD\_FLAGS2 VadFlags2;

}u2;

PSUBSECTION Subsection; / 0x048 / PVOID FirstPrototypePte; / 0x050 / PVOID LastContiguousPte; / 0x058 / LIST\_ENTRY ViewLinks; / 0x060 /

PEPROCESS VadsProcess; / 0x070 /

PVOID u4; / 0x078 /

PVOID FileObject; / 0x080 /

}MMVAD, PMMVAD;

typedef struct \_RTL\_AVL\_TREE // 1 elements, 0x8 bytes (sizeof)

{

/ 0x000 / struct \_RTL\_BALANCED\_NODE Root;

}RTL\_AVL\_TREE, PRTL\_AVL\_TREE;

typedef struct \_VAD\_INFO\_

{

ULONG\_PTR pVad;

ULONG\_PTR startVpn; ULONG\_PTR endVpn; ULONG\_PTR pFileObject; ULONG\_PTR flags;

}VAD\_INFO, PVAD\_INFO;

typedef struct \_ALL\_VADS\_

{

ULONG nCnt;

VAD\_INFO VadInfos[1];

}ALL\_VADS, PALL\_VADS;

typedef struct \_MMSECTION\_FLAGS // 27 elements, 0x4 bytes (sizeof)

{

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| / 0x000  BitPosition | / | UINT32 | BeingDeleted : 1; | // | 0 |
| / 0x000 | / | UINT32 | BeingCreated : 1; | // | 1 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | BeingPurged : 1; | // | 2 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | NoModifiedWriting : 1; | // | 3 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | FailAllIo : 1; | // | 4 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | Image : 1; | // | 5 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | Based : 1; | // | 6 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | File : 1; | // | 7 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | AttemptingDelete : 1; | // | 8 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | PrefetchCreated : 1; | // | 9 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | PhysicalMemory : 1; | // | 10 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | ImageControlAreaOnRemovableMedia : 1; | // | 11 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | Reserve : 1; | // | 12 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | Commit : 1; | // | 13 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | NoChange : 1; | // | 14 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | WasPurged : 1; | // | 15 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | UserReference : 1; | // | 16 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | GlobalMemory : 1; | // | 17 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | DeleteOnClose : 1; | // | 18 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | FilePointerNull : 1; | // | 19 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | ULONG32 | PreferredNode : 6; | // | 20 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | GlobalOnlyPerSession : 1; | // | 26 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | UserWritable : 1; | // | 27 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | SystemVaAllocated : 1; | // | 28 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | PreferredFsCompressionBoundary : 1; | // | 29 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | UsingFileExtents : 1; | // | 30 |
| BitPosition |  |  |  |  |  |
| / 0x000 | / | UINT32 | PageSize64K : 1; | // | 31 |
| BitPosition |  |  |  |  |  |
| }MMSECTION\_FLAGS, PMMSECTION\_FLAGS; | | | | | |



typedef struct \_SECTION (sizeof)

{

// 9 elements, 0x40 bytes

/ 0x000 / struct \_RTL\_BALANCED\_NODE SectionNode; // 6 elements,

0x18 bytes (sizeof)

/ 0x018 /

/ 0x020 /

/ 0x028 /

UINT64 UINT64

union {

StartingVpn;

EndingVpn;

PCONTROL\_AREA ControlArea;

PVOID FileObject;

}u1;

/ 0x030 /

/ 0x038 /

// 4 elements, 0x8 bytes (sizeof) UINT64 SizeOfSection;

union {

ULONG32 LongFlags; MMSECTION\_FLAGS Flags;

}u; // 2 elements, 0x4 bytes (sizeof)

struct (sizeof)

{

/ 0x03C / BitPosition

/ 0x03C / BitPosition

/ 0x03C /

BitPosition

};

}SECTION, PSECTION;

// 3 elements, 0x4 bytes

ULONG32

InitialPageProtection : 12; // 0

ULONG32

SessionId : 19;

// 12

ULONG32

NoValidationNeeded : 1;

// 31

引入 vad.h 头文件，并写入如下代码，此处的 eprocess\_offset\_VadRoot 以及

eprocess\_offset\_VadCount 则是上方得出的相对于 EPROCESS 结构的偏移值，每个系统都不一样，版本不同偏移值会不同。

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

// 定义VAD相对于EProcess头部偏移值#define eprocess\_offset\_VadRoot 0x658 #define eprocess\_offset\_VadCount 0x668

VOID EnumVad(PMMVAD Root, PALL\_VADS pBuffer, ULONG nCnt)

{

if (!Root || !pBuffer || !nCnt)

{

return;

}

try

{

if (nCnt > pBuffer->nCnt)

{

// 得到起始页与结束页

ULONG64 endptr = (ULONG64)Root->Core.EndingVpnHigh; endptr = endptr << 32;



ULONG64 startptr = (ULONG64)Root->Core.StartingVpnHigh; startptr = startptr << 32;

// 得到根节点

pBuffer->VadInfos[pBuffer->nCnt].pVad = (ULONG\_PTR)Root;

// 起始页: startingVpn 0x1000

pBuffer->VadInfos[pBuffer->nCnt].startVpn = (startptr | Root-

>Core.StartingVpn) << PAGE\_SHIFT;

// 结束页: EndVpn 0x1000 + 0xfff

pBuffer->VadInfos[pBuffer->nCnt].endVpn = ((endptr | Root-

>Core.EndingVpn) << PAGE\_SHIFT) + 0xfff;

// VAD标志 928 = Mapped 1049088 = Private ....

pBuffer->VadInfos[pBuffer->nCnt].flags = Root->Core.u1.Flags.flag;

// 验证节点可读性

if (MmIsAddressValid(Root->Subsection) && MmIsAddressValid(Root-

>Subsection->ControlArea))

{

if (MmIsAddressValid((PVOID)((Root->Subsection->ControlArea-

>FilePointer.Value >> 4) << 4)))

{

pBuffer->VadInfos[pBuffer->nCnt].pFileObject = ((Root-

>Subsection->ControlArea->FilePointer.Value >> 4) << 4);

}

}

pBuffer->nCnt++;

}

if (MmIsAddressValid(Root->Core.VadNode.Left))

{

// 递归枚举左子树

EnumVad((PMMVAD)Root->Core.VadNode.Left, pBuffer, nCnt);

}

if (MmIsAddressValid(Root->Core.VadNode.Right))

{

// 递归枚举右子树

EnumVad((PMMVAD)Root->Core.VadNode.Right, pBuffer, nCnt);

}

}

except (1)

{

}

}

BOOLEAN EnumProcessVad(ULONG Pid, PALL\_VADS pBuffer, ULONG nCnt)

{

PEPROCESS Peprocess = 0; PRTL\_AVL\_TREE Table = NULL; PMMVAD Root = NULL;

// 通过进程PID得到进程EProcess



if (NT\_SUCCESS(PsLookupProcessByProcessId((HANDLE)Pid, &Peprocess)))

{

// 与偏移相加得到VAD头节点

Table = (PRTL\_AVL\_TREE)((UCHAR )Peprocess + eprocess\_offset\_VadRoot); if (!MmIsAddressValid(Table) || !eprocess\_offset\_VadRoot)

{

return FALSE;

}

try

{

// 取出头节点

Root = (PMMVAD)Table->Root;

if (nCnt > pBuffer->nCnt)

{

// 得到起始页与结束页

ULONG64 endptr = (ULONG64)Root->Core.EndingVpnHigh; endptr = endptr << 32;

ULONG64 startptr = (ULONG64)Root->Core.StartingVpnHigh; startptr = startptr << 32;

pBuffer->VadInfos[pBuffer->nCnt].pVad = (ULONG\_PTR)Root;

// 起始页: startingVpn 0x1000

pBuffer->VadInfos[pBuffer->nCnt].startVpn = (startptr | Root-

>Core.StartingVpn) << PAGE\_SHIFT;

// 结束页: EndVpn 0x1000 + 0xfff

pBuffer->VadInfos[pBuffer->nCnt].endVpn = (endptr | Root-

>Core.EndingVpn) << PAGE\_SHIFT;

pBuffer->VadInfos[pBuffer->nCnt].flags = Root-

>Core.u1.Flags.flag;

if (MmIsAddressValid(Root->Subsection) && MmIsAddressValid(Root-

>Subsection->ControlArea))

{

if (MmIsAddressValid((PVOID)((Root->Subsection->ControlArea-

>FilePointer.Value >> 4) << 4)))

{

pBuffer->VadInfos[pBuffer->nCnt].pFileObject = ((Root-

>Subsection->ControlArea->FilePointer.Value >> 4) << 4);

}

}

pBuffer->nCnt++;

}

// 枚举左子树

if (Table->Root->Left)

{

EnumVad((MMVAD )Table->Root->Left, pBuffer, nCnt);

}

// 枚举右子树



if (Table->Root->Right)

{

EnumVad((MMVAD )Table->Root->Right, pBuffer, nCnt);

}

}

finally

{

ObDereferenceObject(Peprocess);

}

}

else

{

return FALSE;

}

return TRUE;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

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

}

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

{

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

typedef struct

{

ULONG nPid; ULONG nSize;

PALL\_VADS pBuffer;

}VADProcess;

try

{

VADProcess vad = { 0 };

vad.nPid = 4520;

// 默认有1000个线程

vad.nSize = sizeof(VAD\_INFO) 0x5000 + sizeof(ULONG);

// 分配临时空间

vad.pBuffer = (PALL\_VADS)ExAllocatePool(PagedPool, vad.nSize);

// 根据传入长度得到枚举数量

ULONG nCount = (vad.nSize - sizeof(ULONG)) / sizeof(VAD\_INFO);

// 枚举VAD

EnumProcessVad(vad.nPid, vad.pBuffer, nCount);

// 输出VAD

for (size\_t i = 0; i < vad.pBuffer->nCnt; i++)

{

DbgPrint("StartVPN = %p | ", vad.pBuffer->VadInfos[i].startVpn); DbgPrint("EndVPN = %p | ", vad.pBuffer->VadInfos[i].endVpn); DbgPrint("PVAD = %p | ", vad.pBuffer->VadInfos[i].pVad); DbgPrint("Flags = %d | ", vad.pBuffer->VadInfos[i].flags); DbgPrint("pFileObject = %p \n", vad.pBuffer-

>VadInfos[i].pFileObject);

}

}

except (1)

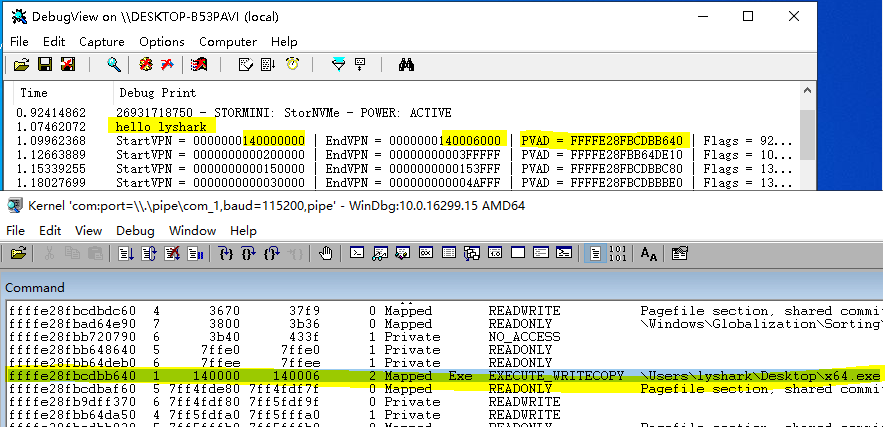
{

}

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

}

# 程序运行后输出效果如下：



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