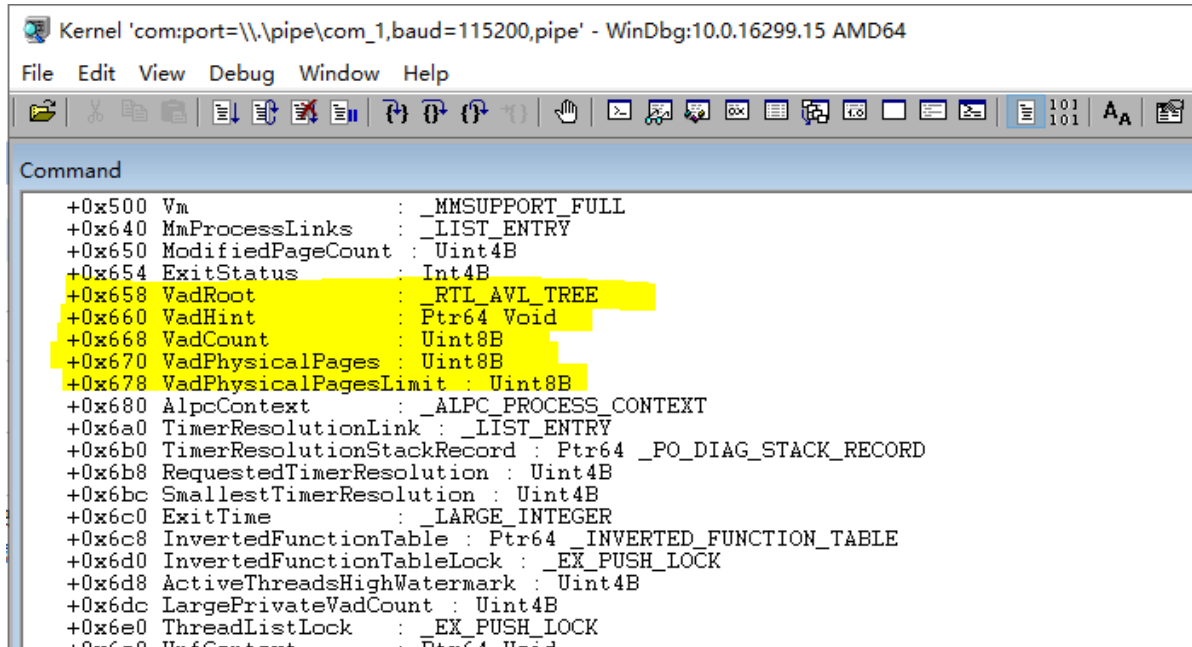


在进程的 `_EPROCESS` 中有一个 `_RTL_AVL_TREE` 类型的 `VadRoot` 成员，它是一个存放进程内存块的二叉树结构，如果我们找到了这个二叉树中我们想要隐藏的内存，直接将这个内存在二叉树中抹去，其实是让上一个节点的 `EndingVpn` 指向下个节点的 `EndingVpn`，类似于摘链隐藏进程，就可以达到隐藏的效果。

通过 `dt _EPROCESS` 得到 `EProcess` 结构 `VadRoot` 如下：



```
Kernel 'com:port=\\.\pipe\com_1,baud=115200,pipe' - WinDbg:10.0.16299.15 AMD64
File Edit View Debug Window Help
+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
+0x680 AlpcContext : _ALPC_PROCESS_CONTEXT
+0x6a0 TimerResolutionLink : _LIST_ENTRY
+0x6b0 TimerResolutionStackRecord : Ptr64 _PO_DIAG_STACK_RECORD
+0x6b8 RequestedTimerResolution : Uint4B
+0x6bc SmallestTimerResolution : Uint4B
+0x6c0 ExitTime : _LARGE_INTEGER
+0x6c8 InvertedFunctionTable : Ptr64 _INVERTED_FUNCTION_TABLE
+0x6d0 InvertedFunctionTableLock : _EX_PUSH_LOCK
+0x6d8 ActiveThreadsHighWatermark : Uint4B
+0x6dc LargePrivateVadCount : Uint4B
+0x6e0 ThreadListLock : _EX_PUSH_LOCK
```

例如当调用 `VirtualAlloc` 分配内存空间。

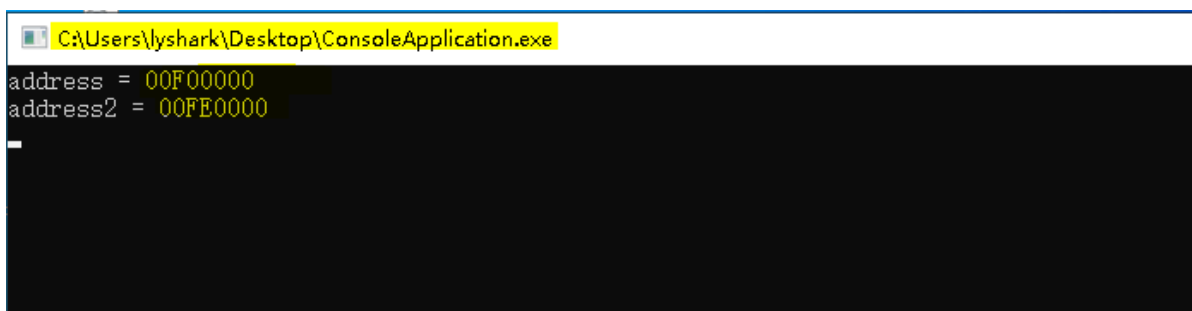
```
#include <iostream>
#include <windows.h>

int main(int argc, char *argv[])
{
    LPVOID p1 = VirtualAlloc(NULL, 0x10000, MEM_COMMIT, PAGE_READWRITE);
    LPVOID p2 = VirtualAlloc(NULL, 0x10000, MEM_COMMIT, PAGE_EXECUTE_READWRITE);

    std::cout << "address = " << p1 << std::endl;
    std::cout << "address2 = " << p2 << std::endl;

    getchar();
    return 0;
}
```

运行程序得到两个内存地址 `0xf00000` 和 `0xfe0000`



```
C:\Users\lyshark\Desktop\ConsoleApplication.exe
address = 00F00000
address2 = 00FE0000
```

通过 !process 0 0 枚举所有进程，并得到我们所需进程的EProcess地址。

```
PROCESS ffffff28fbb6be080
  SessionId: 1 Cid: 0b8c Peb: d433d45000 ParentCid: 0310
  DirBase: b53ed002 ObjectTable: fffffac87bb1dfc40 HandleCount: 318.
  Image: smartscreen.exe

PROCESS ffffff28fbb451080
  SessionId: 1 Cid: 0584 Peb: 011f4000 ParentCid: 11c8
  DirBase: 2e56b002 ObjectTable: fffffac87bca84240 HandleCount: 59.
  Image: ConsoleApplication.exe

PROCESS ffffff28fbc210080
  SessionId: 0 Cid: 14d8 Peb: 1d79d86000 ParentCid: 0274
  DirBase: 91157002 ObjectTable: fffffac87bca85340 HandleCount: 258.
  Image: svchost.exe

PROCESS ffffff28fbb7e3080
  SessionId: 1 Cid: 12b4 Peb: 9c5eba8000 ParentCid: 0584
  DirBase: 2ca1b002 ObjectTable: fffffac87bca87640 HandleCount: 253.
  Image: conhost.exe
```

1: kd>

Ln 0, Col 0 Sys 0:KdSrv:S Pr

检查进程 !process ffffff28fbb451080 得到 VAD 地址 ffffff28fbc0b7e40

Kernel 'com:port=\\.\pipe\com_1,baud=115200,pipe' - WinDbg:10.0.16299.15 AMD64

File Edit View Debug Window Help

Command

```
1: kd> !process ffffff28fbb451080
PROCESS ffffff28fbb451080
  SessionId: 1 Cid: 0584 Peb: 011f4000 ParentCid: 11c8
  DirBase: 2e56b002 ObjectTable: fffffac87bca84240 HandleCount: 59.
  Image: ConsoleApplication.exe
  VadRoot ffffff28fbc0b7e40 Vads 35 Clone 0 Private 134. Modified 8. Locked 2.
  DeviceMap fffffac87b7ef86f0
  Token fffffac87baeea5f0
  ElapsedTime 00:02:18.452
  UserTime 00:00:00.000
  KernelTime 00:00:00.000
  QuotaPoolUsage[PagedPool] 28056
  QuotaPoolUsage[NonPagedPool] 4952
  Working Set Sizes (now,min,max) (846, 50, 345) (3384KB, 200KB, 1380KB)
  PeakWorkingSetSize 838
  VirtualSize 13 Mb
  PeakVirtualSize 17 Mb
  PageFaultCount 923
  MemoryPriority BACKGROUND
  BasePriority 8
  CommitCharge 198
  Job ffffff28fbc288060
```

此处以 0xf00000 为例，这里我们看到 windbg 中的值和进程中分配的内存地址并不完全一样，这是因为 x86 cpu 默认内存页大小 4k 也就是 0x1000，所以这里还要再乘以 0x1000 才是真正的内存地址。

Kernel 'com:port=\\.\pipe\com_1,baud=115200,pipe' - WinDbg:10.0.16299.15 AMD64

File Edit View Debug Window Help

Command

```
1: kd> !vad ffffff28fbc0b7e40
VAD Level Start End Commit
fffff28fbc0b7da0 4 3b0 3b6 2 Mapped Exe EXECUTE_WRITECOPY \Users\
fffff28fbc0b8f20 3 ee0 eef 0 Mapped READWRITE Pagefil
fffff28fbc0b9cf880 4 ef0 ef7 1 Private READWRITE
fffff28fbc0b9e1f60 5 f00 f0f 16 Private READWRITE
fffff28fbc0b8c00 2 f10 f2a 0 Mapped READONLY Pagefil
fffff28fbc0b9dcf10 4 f30 f6f 11 Private READWRITE
fffff28fbc0b8ca0 3 f70 f73 0 Mapped READONLY Pagefil
fffff28fbc0b92e0 1 f80 f80 0 Mapped READONLY Pagefil
fffff28fbc0b9ddd70 5 f90 f91 2 Private READWRITE
fffff28fbc0b9e24b0 4 fe0 fef 16 Private EXECUTE_READWRITE
```

所以计算结果刚好等于 0xf00000



而隐藏进程内特定内存段核心代码在于 `p1->EndingVpn = p2->EndingVpn`; 将VAD前后节点连接。

```
PMMVAD p1 = vad_enum((PMMVAD)VadRoot, 0x3a0); // 遍历第一个结点
PMMVAD p2 = vad_enum((PMMVAD)VadRoot, 0x3b0); // 遍历找到第二个结点
if (p1 && p2)
{
    p1->EndingVpn = p2->EndingVpn; // 将第二个结点完全隐藏起来
}
```

本书作者：王瑞 (LyShark)

作者邮箱：me@lyshark.com

作者博客：<https://lyshark.cnblogs.com>

团队首页：www.lyshark.com