# 进程结构

讲进程的第一课。进程是一块空间，执行代码是线程干的。32位下是4GB，64位的地址只用了48位，所以是2^48=256TB.

创建进程后，会分配一些必要的内存，如EXE,DLL拉伸内存，PEB等。

为啥会有KPROCESS, EPROCESS？Windows设计之初想搞微内核，后面又往宏内核发展了，所以结构上有点奇怪，大概是这样。

简单过了一遍EPROCESS, KPROCESS的关键字段。

### KPROCESS

|  |  |
| --- | --- |
| 字段 | 含义 |
| DirectoryTableBase | CR3 |
| ThreadListHead | 线程链表 |
| ProcessLock | 写锁 |
| Affinity | CPU亲和性，某个位置1表示可以被某个核调度 |
| ReadyListHead | 就绪线程列表，可以被调度 |
| SwapListEntry | 交换链表，blablabla不知道是啥 |
| ActiveProcessors | 当前有哪些线程在哪些核上跑？ |
| 0x05c 是个位域 | 具体是啥没听清 |
| BasePriority | 线程的默认优先级，从KPROCESS里取 |
| QuantumReset | 时间片基础值，后面逆线程要用 |
| Flags | \_KEXECUTE\_OPTIONS类型，可以用来关闭SEH,DEP之类的 |
|  |  |

### EPROCESS

|  |  |
| --- | --- |
| 字段 | 含义 |
| ProcessLock | 写锁 |
| CreateTime | 创建时间 |
| ExitTime | 结束时间（EPROCESS根据计数决定是否释放） |
| RundownProtect | 锁？ |
| UniqueProcessId | Pid咯 |
| ActiveProcessLinks | 进程链表，作业是遍历它 |
| SessionProcessLinks | 当前用户的进程链表 |
| DebugPort | 调试端口，抹了它，反调试 |
| ExceptionPortData | 异常报告 |
| ObjectTable | 句柄表 |
| Token | 权限 |
| WorkingSetPage | 工作集相关，不知道是啥 |
| AddressCreationLock | 申请内存锁 |
| Win32Process | 是否UI进程 |
| Job | 工作相关，《windows核心编程》 |
| SectionObject | 忘记是啥了，createsection啥啥啥的 |
| SectionBaseAddress |  |
| Cookie | Cookie |
| InheritedFromUniqueProcessId | 父进程ID |
| ImageFileName | Exe名，长度不大于15 |
| ThreadListHead | 线程链表 |
| Peb | Peb进程控制块 |
| SeAuditProcessCreationInfo | 全路径 |
|  |  |
|  |  |
|  |  |
|  |  |

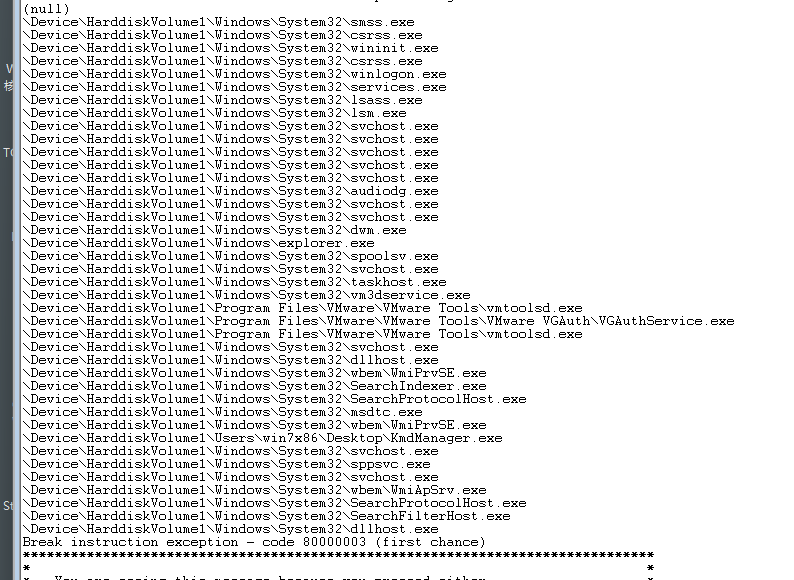
还有几个字段，隐藏，保护相关的，明天讲。

# 遍历进程链表

获取 ActiveProcessLinksOffset 的姿势有很多，我这写死了。比如先获取pid然后+sizeof(void\*)，或者解析系统版本，换个姿势写死，还有下载符号解析的，不太懂。

注意用非文档化API SeLocateProcessImageName 前要判断一下是不是死的进程，不然会蓝。

|  |
| --- |
| #include<ntifs.h>  EXTERN\_C NTSTATUS SeLocateProcessImageName(  \_\_in PEPROCESS Process,  \_\_deref\_out PUNICODE\_STRING\* pImageFileName  );  VOID Unload(PDRIVER\_OBJECT pDriver)  {  }  NTSTATUS DriverEntry(PDRIVER\_OBJECT pDriver, PUNICODE\_STRING pReg)  {  PUNICODE\_STRING ImageName = NULL;  NTSTATUS status = 0;  PEPROCESS SysProcess = PsInitialSystemProcess;  //PEPROCESS SysProcess = PsGetCurrentProcess();  PEPROCESS CurProcess = SysProcess;  ULONG\_PTR ActiveProcessLinksOffset = 0xb8;  do  {  if (PsGetProcessExitStatus(CurProcess) == STATUS\_PENDING)  {  status = SeLocateProcessImageName(CurProcess, &ImageName);  if (NT\_SUCCESS(status))  {  DbgPrintEx(77, 0, "%wZ\r\n", ImageName);  ExFreePool(ImageName);  }  }  CurProcess = (PEPROCESS)(\*(PULONG\_PTR)((ULONG\_PTR)CurProcess + ActiveProcessLinksOffset) - ActiveProcessLinksOffset);  } while (SysProcess != CurProcess);  pDriver->DriverUnload = Unload;  return STATUS\_SUCCESS;  } |



# 抹调试端口

循环抹除调试端口实现的反调试。

|  |
| --- |
| #include<ntifs.h>  #include<strsafe.h>  BOOLEAN Loop = TRUE;  EXTERN\_C NTSTATUS SeLocateProcessImageName(  \_\_in PEPROCESS Process,  \_\_deref\_out PUNICODE\_STRING\* pImageFileName  );  VOID KSleep(ULONG ms, BOOLEAN Alert)  {  LARGE\_INTEGER larTime = { 0 };  larTime.QuadPart = ((ULONG64)-10000) \* ms;  KeDelayExecutionThread(KernelMode, Alert, &larTime);  }  VOID Unload(PDRIVER\_OBJECT pDriver)  {  Loop = FALSE;  KSleep(5000, FALSE);  }  VOID FuckDbgPort(  \_In\_ PVOID StartContext  )  {    NTSTATUS status = 0;  PEPROCESS SysProcess = PsInitialSystemProcess;  ULONG\_PTR ActiveProcessLinksOffset = 0xb8;  UNICODE\_STRING ProtectProcessName = RTL\_CONSTANT\_STRING(L"\\Device\\HarddiskVolume1\\Users\\win7x86\\Desktop\\KmdManager.exe");  ULONG\_PTR DebugPortOffset = 0xec;  while (Loop)  {  PEPROCESS CurProcess = SysProcess;  PUNICODE\_STRING ImageName = NULL;  do  {  if (PsGetProcessExitStatus(CurProcess) == STATUS\_PENDING)  {  status = SeLocateProcessImageName(CurProcess, &ImageName);  if (NT\_SUCCESS(status))  {  if (0 == RtlCompareUnicodeString(ImageName, &ProtectProcessName, TRUE))  {  DbgPrintEx(77, 0, "保护: %wZ\r\n", ImageName);  \*(ULONG\*)((ULONG\_PTR)CurProcess + DebugPortOffset) = 0; // 别调了  }  ExFreePool(ImageName);  }  }  CurProcess = (PEPROCESS)(\*(PULONG\_PTR)((ULONG\_PTR)CurProcess + ActiveProcessLinksOffset) - ActiveProcessLinksOffset);  } while (SysProcess != CurProcess);  KSleep(1000, FALSE);  }  }  NTSTATUS DriverEntry(PDRIVER\_OBJECT pDriver, PUNICODE\_STRING pReg)  {  HANDLE hThread = NULL;  NTSTATUS st = PsCreateSystemThread(&hThread,THREAD\_ALL\_ACCESS,NULL,NULL,NULL, FuckDbgPort, pDriver);    if (NT\_SUCCESS(st))  {  NtClose(hThread);  }  pDriver->DriverUnload = Unload;  return STATUS\_SUCCESS;  } |