# 作业讲解

### 中断门与陷阱门的区别

进入中断门会清除 efi VM NT IF TF位

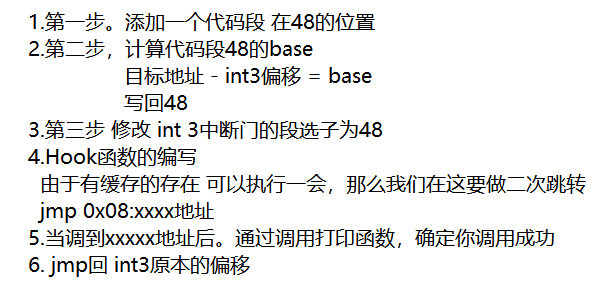
进入陷阱门会清除 efi VM NT TF位

由于中断门会清除IF位，导致一些中断触发会悬挂,陷阱门则不会

### 昨天的作业

昨天观察到一个现象，retf 返回后，如果调用printf，由于缺页，会蓝屏，如果在int 之前先调用一次printf，就不会缺页蓝屏了。还有一种方法是retf前把IF置1,。课上火哥讲了另一种方法：用sti指令，可以解除阻塞并且设置IF=1,。为什么iretd不会蓝呢，因为iretd内部就做了这些工作。

# HOOK INT 3



### 失败的尝试（裸函数版本）

首先查看int3要跳转的地方（这个值每次重启都会变）

int3.base+int3.offset==83e465c0

Hook的思路是让int3.base+int3.offset==401000

offset不能变，只能动base

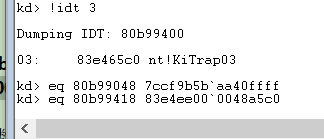
就是说要让 int3.base + 83e465c0==401000

**所以希望int3.base == 0x7C5B AA40**

那么我们在gtd 48的位置构造一个R0代码段，把base改成这个，然后改IDT表int3的段选择子为48，就可以了。

**eq 80b99048 7ccf9b5b`aa40ffff**

**eq 80b99418 83e4ee00`0048a5c0**



接下来编写hook函数，由于进来之后，base已经被改乱了，不是0了，那么只有第一条指令寻址的时候 base+offset==401000的，假设下一条指令的offset是401002，那么加上base之后就直接跑飞了。但是由于指令缓存，我们跳进来hook函数后，有一小段长度的指令被缓存了，不需要进行base+offset的计算就能执行，所以暂时不会跑飞，为了避免后续跑飞，我们进来后最好马上把base改回0，方法是jmp far跨段跳转不提权。我们通过jmp far，将cs改回08.然后调用DbgPrint,最后跳转回int3的函数。

|  |
| --- |
| #include "stdafx.h"  #include <Windows.h>  #include <string.h>  char bufcode[6] = {0};  WCHAR outputstr[32] = {0};  \_\_declspec(naked) void hook()  {  \_\_asm  {  mov eax,haha;  mov dword ptr ds:[bufcode],eax;  jmp far bufcode;  haha:  nop;  //DbgPrint  //lea eax,outputstr;  //push eax;  //mov eax, 0x83e1a41f;  //call eax;  push 0x83e465c0; // go back to int 3  ret;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  printf("%p\n",hook);  if ((int)hook != 0x401000) return 1;  printf("%x\n", \*(DWORD\*)hook);  bufcode[4] = 0x8;  wcscpy(outputstr,L"hook int3");  \_\_asm  {  int 3;  }  return 0;  } |

这样试了好几次都黑屏了。

我折腾了好久，后来写了个驱动，自动HOOK，还是黑屏，突然想起来，是不是应该先把可屏蔽中断打开，干完活再关掉，再跳回int3.就好了。（离成功只差两条指令：sti,cli）

### 驱动代码

|  |
| --- |
| #include <ntifs.h>  #pragma pack(1)  typedef struct \_GDT{  SHORT Limit;  UINT32 Base;  }GDT;  #pragma pack(0)  typedef GDT IDT;  char bufcode[6] = { 0 };  UINT32 int3offset = 0;  VOID DriverUnload(PDRIVER\_OBJECT pDriver)  {  DbgPrintEx(77, 0, "hook int3 unloaded\n");  }  // 跳板，修改CS=8，然后跳转到MyCode  \_\_declspec(naked) VOID Stub()  {  \_\_asm  {  sti;  jmp fword ptr ds:[bufcode];  }    }  // 在这里 DbgPrint，然后跳转到int3原本的地址  \_\_declspec(naked) VOID MyCode()  {  DbgPrintEx(77, 0, "int3 called!\n");  \_\_asm  {  cli;  mov eax, int3offset;  jmp eax;  }  }  // 自动完成HOOK操作：修改int3的段选择子，在GDT 48 写入新的代码段描述符并修改base  VOID HookInt3()  {  GDT gdtr;  IDT idtr;  \_\_asm  {  sgdt gdtr;  sidt idtr;  }  PUINT64 idt = (PUINT64)idtr.Base;  PUINT64 gdt = (PUINT64)gdtr.Base;  int3offset = (UINT32)(idt[3] >> 48) \* 0x10000 + ((UINT32)idt[3] & 0x0000ffff);  DbgPrintEx(77, 0, "int3 offset %x\n", int3offset);    // 拷贝08到48，并修改base，int3时会跳到Stub  gdt[9] = gdt[1];  UINT32 base = (UINT32)Stub - int3offset;  DbgPrintEx(77, 0, "int3 new base: %x == %x - %x\n", base, (UINT32)Stub, int3offset);  ((char\*)(&gdt[9]))[7] = ((char\*)&base)[3];  ((char\*)(&gdt[9]))[4] = ((char\*)&base)[2];  ((char\*)(&gdt[9]))[3] = ((char\*)&base)[1];  ((char\*)(&gdt[9]))[2] = ((char\*)&base)[0];  DbgPrintEx(77, 0, "GDT08 %p: %016llx\n", &gdt[1], gdt[1]);  DbgPrintEx(77, 0, "GDT48 %p: %016llx\n", &gdt[9], gdt[9]);  // 设置bufcode，进入 Stub 后跨段跳转将CS改回08，并且跳转到 MyCode  \*(UINT32\*)bufcode = (UINT32)MyCode;  bufcode[4] = 0x8;  // 修改 int3 的段选择子为48  DbgPrintEx(77, 0, "idt3: %llx\n", idt[3]);  DbgPrintEx(77, 0, "int3 selector %x\n", ((char\*)&idt[3])[2]);  ((char\*)&idt[3])[2] = 0x48;  DbgPrintEx(77, 0, "new int3 selector %x\n", ((char\*)&idt[3])[2]);  }  NTSTATUS DriverEntry(PDRIVER\_OBJECT pDriver, PUNICODE\_STRING pReg)  {  DbgPrintEx(77, 0, "-------------------------------\n");  DbgPrintEx(77, 0, "hook int3 loaded\n");  pDriver->DriverUnload = DriverUnload;  HookInt3();  return STATUS\_SUCCESS;  } |

