# 课前准备

### 做实验前必须先关闭熔断补丁

[HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management]

"FeatureSettingsOverrideMask"=dword:00000003

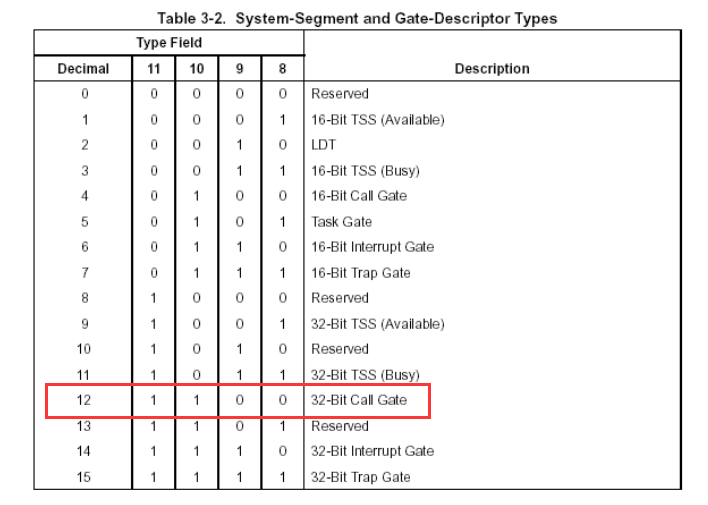
"FeatureSettingsOverride"=dword:00000003



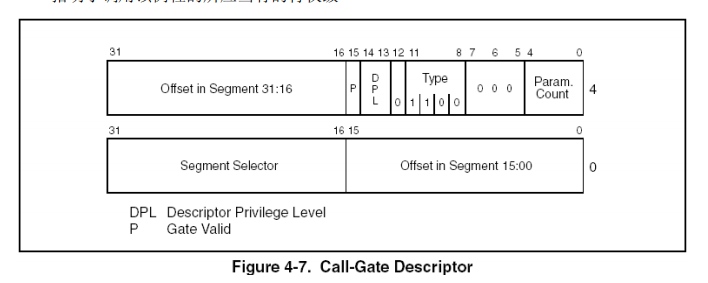
改完重启虚拟机，务必在虚拟机里面重启。

# 课堂笔记

之前讲过S=1时是代码段和数据段。S=0时就是系统段，今天讲了系统段里的调用门段。



看段选择子，S=0，type=c时，就是调用门。调用门的解析方式和之前的数据段代码段又有所不同，如下图：



offset和segment selecctor放在一起讲。Offset是偏移，offset+base就是跨段跳转后要去的地址，但是我们发现调用门描述符里并没有base，这个base其实是segment selector这个段选择子指向的那个段描述符里面给的。

P是有效位，一样的。

DPL，我们从3环提权到0环，这个调用门描述符的DPL必须是3，因为要满足条件：

**CS.DPL == SS.DPL == 调用门描述符.DPL**

S是0，表示系统段，Type是c，表示调用门，这俩是固定的。

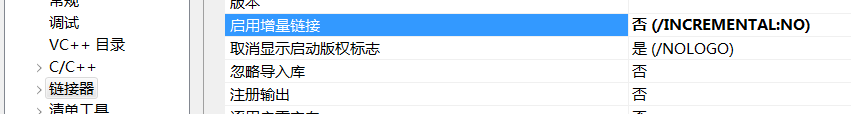
[5,7]位是写死的0

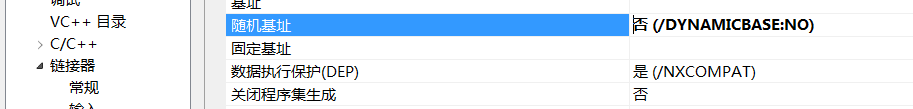
[0,4]位，共5位，表示参数个数，最多是2^5-1==31个参数，每个参数是4字节大小。

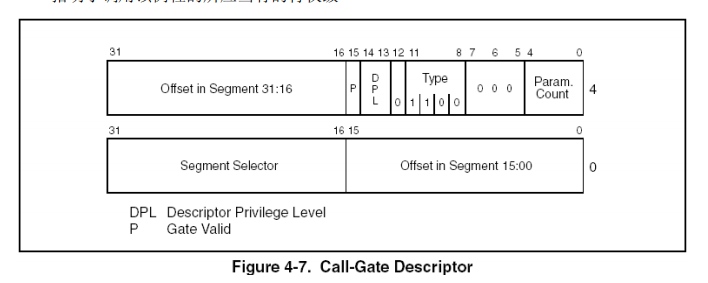
# 实验

#### 跨段call无参数

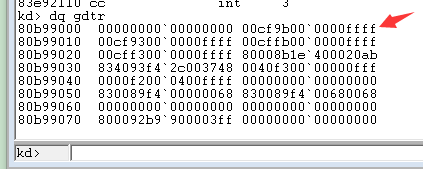
为了方便，我们关掉增量链接，这样就没有多余的jmp。然后关掉随机基址，这样第一个函数的地址就是0x401000







首先肯定是构造一个调用门了，跳转目标是裸函数0x401000，目标段是R0的代码段，观察gdt表，R0代码段的段描述符在第二项，所以段选择子是08.

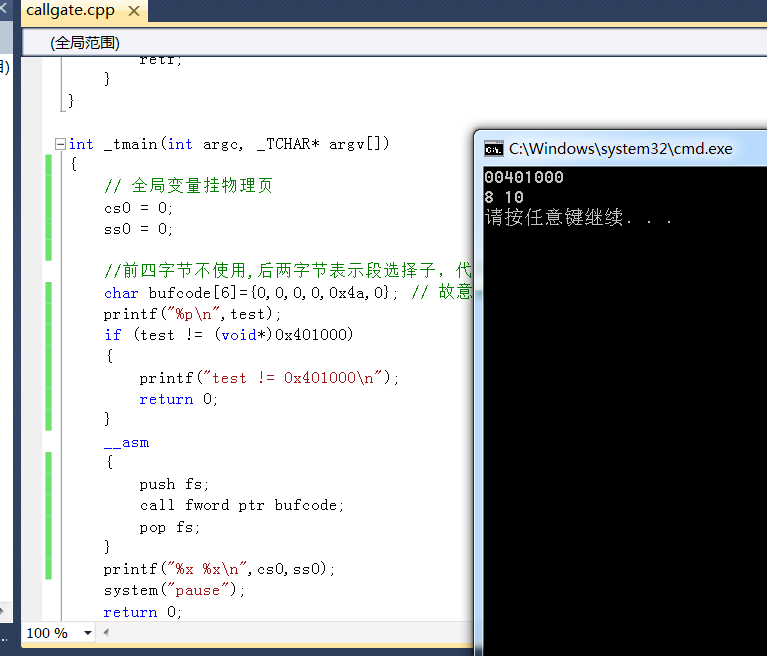


先整个无参的试试，那么调用门描述符就是0040ec00`00081000，放到48位置

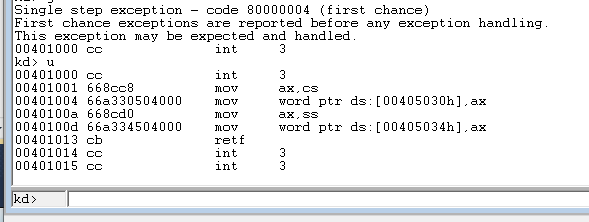


我这里先试了一下常规的call far，测试是没问题的：

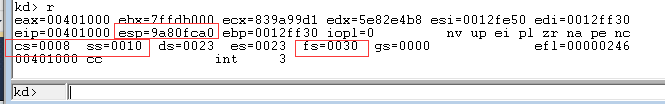
|  |
| --- |
| // callgate.cpp : 定义控制台应用程序的入口点。  //  #include "stdafx.h"  #include <Windows.h>  // 整两个全局变量证明提权成功，CS0应是08，SS0应是10  WORD cs0 = 0;  WORD ss0 = 0;  \_\_declspec(naked) void test()  {  \_\_asm  {  //int 3;    mov ax,cs;  mov cs0,ax;  mov ax,ss;  mov ss0,ax;  retf;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  // 全局变量挂物理页  cs0 = 0;  ss0 = 0;  //前四字节不使用,后两字节表示段选择子，代码段和数据段RPL可以乱给  char bufcode[6]={0,0,0,0,0x4a,0}; // 故意把RPL弄成2也是没问题的  printf("%p\n",test);  if (test != (void\*)0x401000)  {  printf("test != 0x401000\n");  return 0;  }  \_\_asm  {  push fs;  call fword ptr bufcode;  pop fs;  }  printf("%x %x\n",cs0,ss0);  system("pause");  return 0;  } |



这里结果是对的，无参调用门提权成功了，这里不妨把断点打开，观察一下R0里面的栈和寄存器：

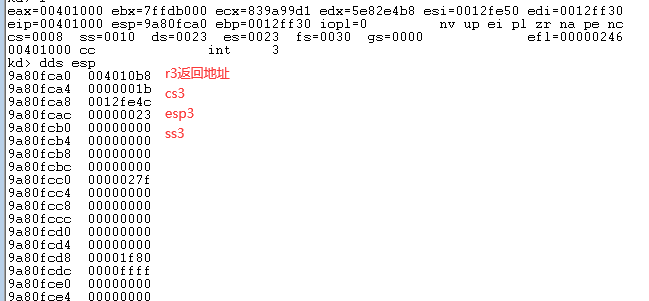


在R0断下，看看寄存器：



重点看的寄存器我框出来了，ESP变成了R0的栈顶，发生栈切换了，FS变成了30。CS变成了08，SS是CS+8==10.

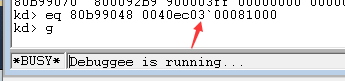
看看栈的内容：



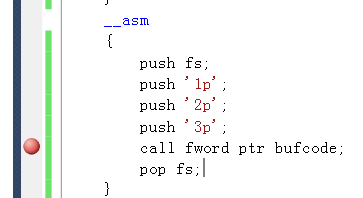
地址从低到高依次是 r3返回地址、cs3、esp3、ss3.说明call far 跨段提权call不仅会发生栈的切换，还会把调用者的上下文信息（返回地址和寄存器）保存到栈中，retf回R3时用来恢复上下文。

#### 跨段call带参数

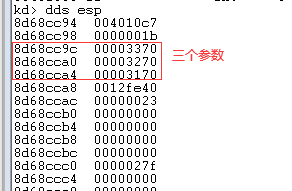
这个我就简单记一下，其实就是在无参数的基础上。简单做个实验，改一下调用门描述符，比如我要传3个参数：



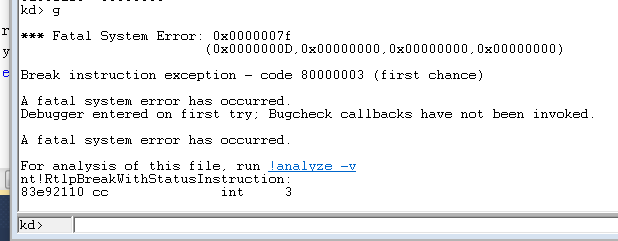
修改代码如下：

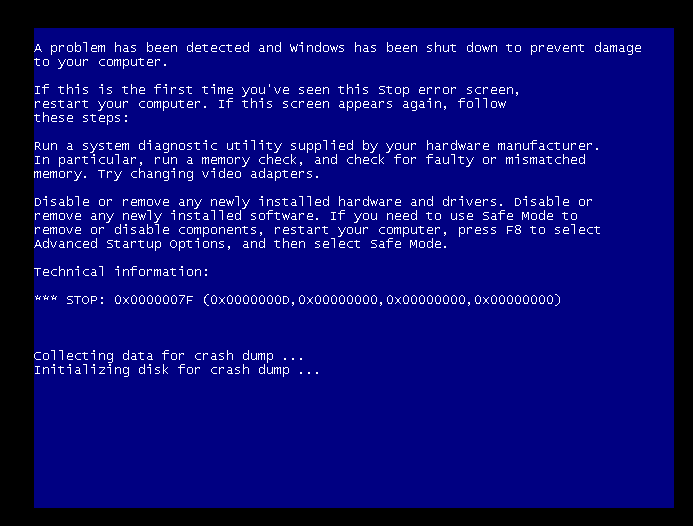


执行，在R0断下，观察栈：

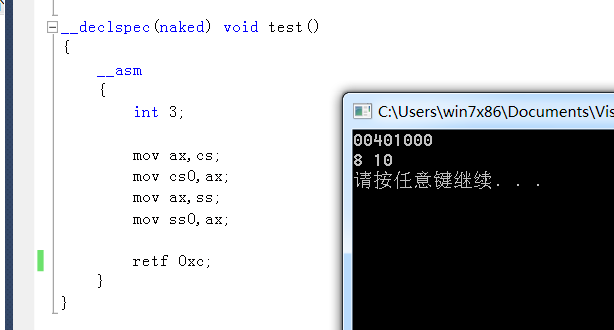


我这里比较操蛋，忘记 retf 12了，我直接retf，所以蓝屏了。





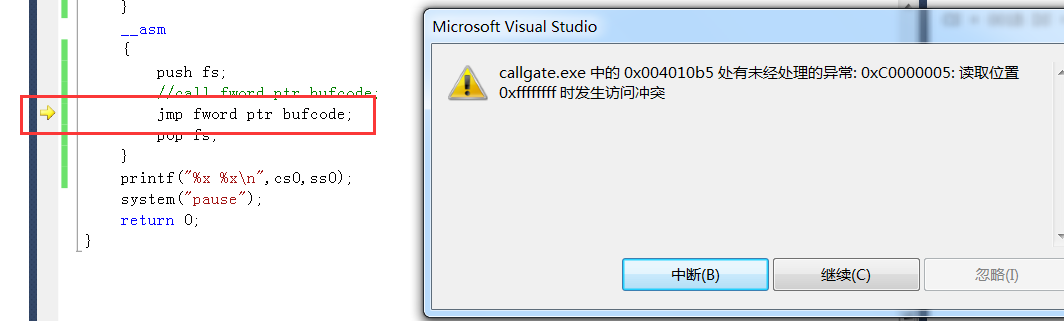
这里要记好了，传了3个参数，retf的时候就要 retf 12



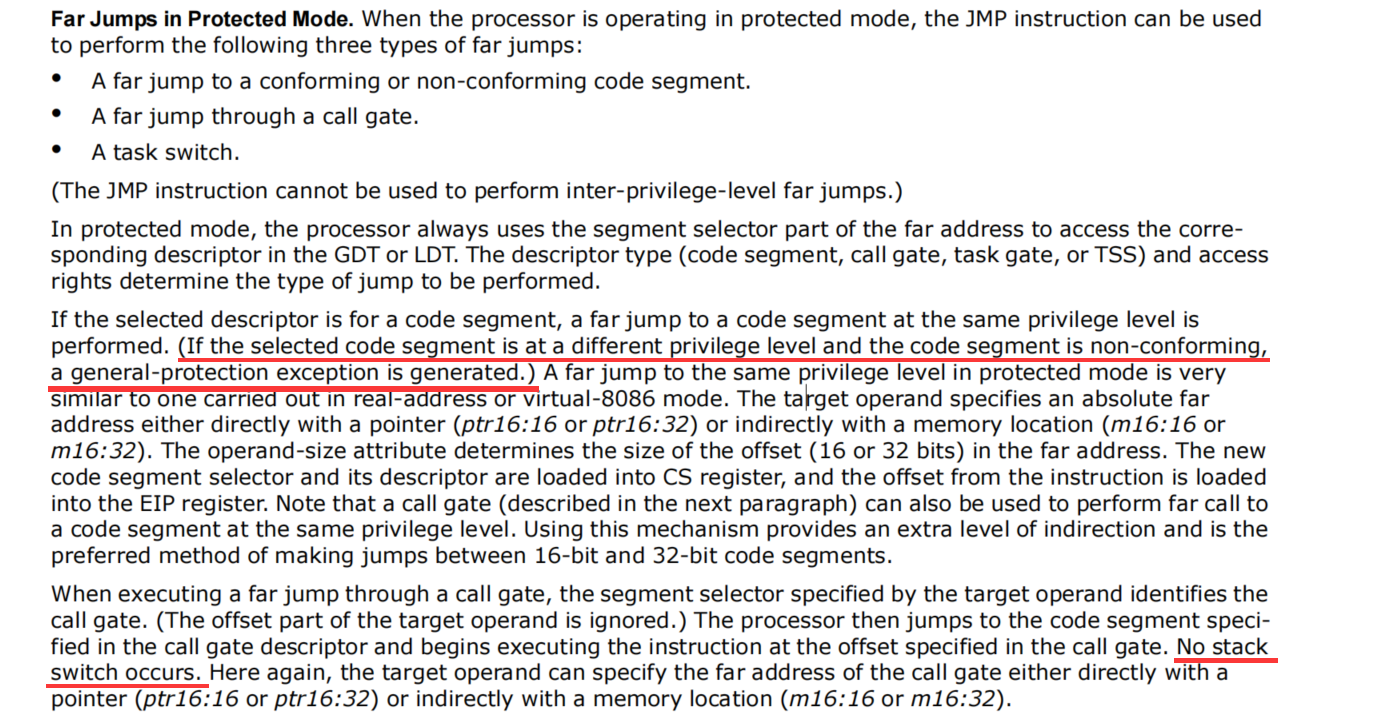
# 作业

#### 1.用JMP跨段跳转调用门 (记录自己的实验是否成，如果不成功 想一想为什么)

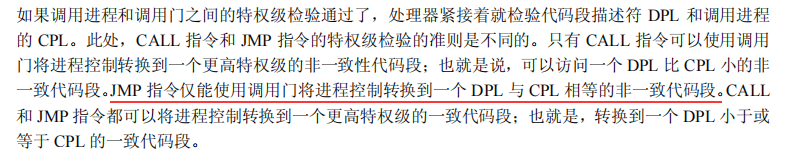
把call far 改成 jmp far，执行，结果是C0000005，在裸函数加上int3，也没断下，证明根本没进R0：



为什么不可以，JMP FAR不能像CALL FAR那样同时修改CS和SS。详细可以看白皮书卷2对JMP FAR 和 CALL FAR 的介绍。



白皮书卷2第三章 INSTRUCTION SET REFERENCE, A-L



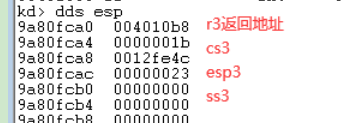
白皮书中文版卷3第4.8.4节 通过调用门访问代码段

#### 2.既然ret也能跳转 那么是否可以提权？

ret就是普通的jmp+pop，jmp不能提权，ret肯定也不能。

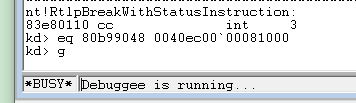
#### 3.既然retf也能跳转，那么是否提权？

以无参数call far提权为例，进入R0后，栈的情况是这样的：

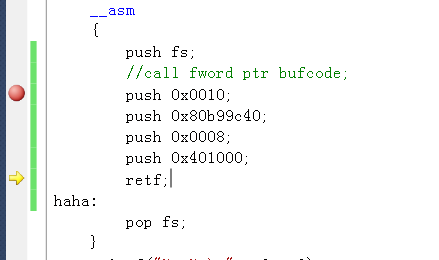


如果我想在r3用retf模拟call far，我可以依次把 ss0, esp0, cs0, test的地址压栈，然后retf.

ss0==10,cs0==08,test==401000,主要是这个esp0比较难搞，我去gdt表靠后的位置随便找个地址作为esp0，比如 80b99c40



设置调用门描述符



模拟一下R0返回R3时的栈，只不过换成从R3返回R0。

试了下不太行，谷歌也没搜到retf提权进R0的姿势，查白皮书也云里雾里，最后看了一下火哥五期的视频，说是retf只能用来平权和降权，jmp只能同权限，call只能平权和提权。我感觉retf没有读gdt表，如果像我这么写就能提权那就太离谱了。