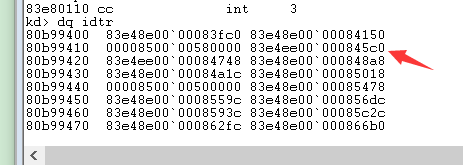
# 课堂笔记

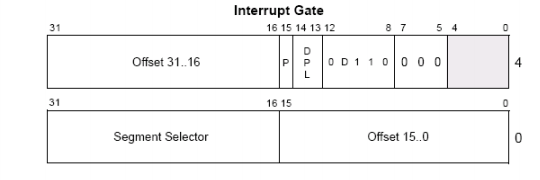
这次课讲了中断门，sgdt,sidt,lgdt,lidt。

中断是外部硬件发起的，异常是软件发起的。

常见的中断有 int 3，这个3表示idt表第三项:



中断门描述符长这样：



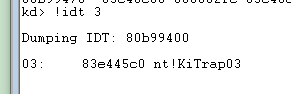
我们以int 3为例说明。

Offset是要去的偏移，83e445c0

Segment selector 段选择子是08，表示gdt表的第二项；

其他的位没啥说的，type里的D位32位下是1,16位是0；

我们还可以用 !idt 3 来看看offset拆的对不对：



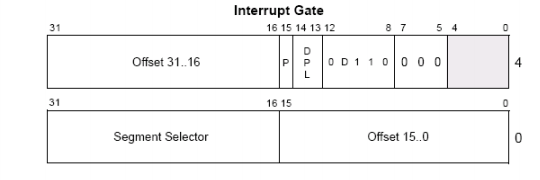
没毛病，就是这个地址。

课上还演示了在IDT表中找空白项，填入自己构造的中断门描述符，然后用这个中断门提权，以下进行复现。

# 

在IDT表中找到空白位置，(0x500-0x400)/8 == 0x20,接下来可以用 int 0x20 调用这个中断。

然后构造一个中断门描述符，因为关闭了随机基址，test 函数就是 401000



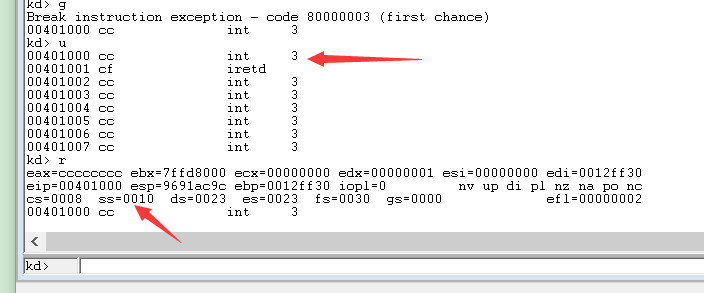
构造中断门描述符 0040ee00`00081000



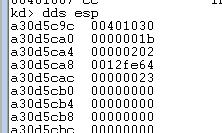
编写代码如下：

|  |
| --- |
| \_\_declspec(naked) void test()  {  \_\_asm  {  int 3;  iretd;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  \_\_asm  {  int 0x20;  push 0x3b;  pop fs;  }  return 0;  } |

执行代码，成功提权，在R0断下：



观察一下栈，看看int指令干了啥：



**可以看到，依次压入了三环的ss,esp,eflag,cs,返回地址。**

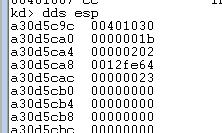
此外还讲了 sgdt等指令，用法比较简单，不演示了。S是store，表示读取，R3也可以用。Lgdt的L表示load，是写入，他可以用来替换GDT表，是R0特权指令。

# 课后作业

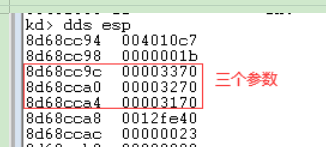
### 1.返回一定要用iretd嘛？ retf

因为retf也可以降权回R3，所以我们可以自己模拟一下栈，用retf代替iretd。

先来对比一下retf和iretd之前，在R0的情况：



Iretd前



Retf前（call far带了三个参数进来的）

我们发现，中断进来的栈其实跟call far带1个参数进来是差不多的，我们就当那个参数是eflags，就行了，所以我们可以用 retf 4 来替代iretd，这里要注意一点，iretd不会平衡R3的栈，而retf 4 会，所以我们要在R3也模拟一下压栈的操作，也就是push 1那句，不加会蓝屏的。

eq 80b99500 0040ee00`00081000

|  |
| --- |
| #include "stdafx.h"  #include <Windows.h>  \_\_declspec(naked) void test()  {  \_\_asm  {  int 3;  retf 4;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  if ((int)test != 0x401000)  {  printf("test is not 401000\n");  return 0;  }  \_\_asm  {  pushfd;  push 1;  int 0x20;  popfd;  push 0x3b;  pop fs;  }  return 0;  } |

后来我换了一个更简单的方法，直接把esp和ss往上挪，把eflag覆盖掉就完事了，修改代码如下：

|  |
| --- |
| #include "stdafx.h"  #include <Windows.h>  \_\_declspec(naked) void test()  {  \_\_asm  {  int 3;  mov eax,[esp+12];  mov [esp+8],eax;  mov eax,[esp+16];  mov [esp+12],eax;  retf;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  \_\_asm  {  pushfd;  int 0x20;  popfd;  push 0x3b;  pop fs;  }  return 0;  } |

但是这两种做法都有一个问题，如果在主函数 return 0 前调用一下 printf 就会蓝屏。如果我们在 int 之前先 printf 一下，他又不会蓝了，很神奇，代码如下：

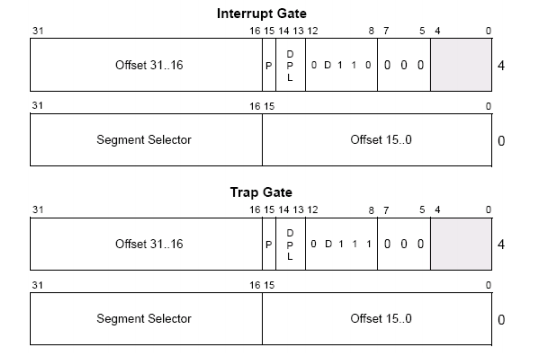
|  |
| --- |
| #include "stdafx.h"  #include <Windows.h>  \_\_declspec(naked) void test()  {  \_\_asm  {  retf 4;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  if ((int)test != 0x401000) return 1;  printf("11111\n");  \_\_asm  {  push 1;  int 0x20;  push 0x3b;  pop fs;  }  printf("no bsod\n");  return 0;  } |

我猜测是和IF位有关，所以在retf之前把IF置1，代码如下：

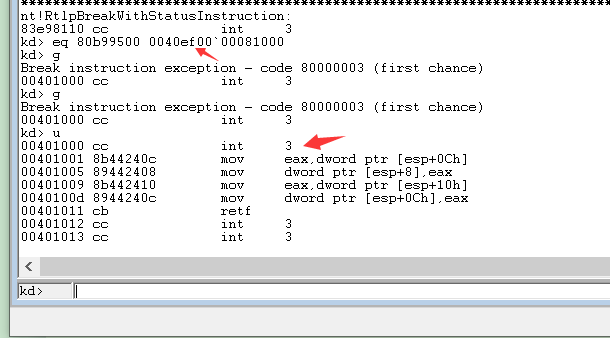
|  |
| --- |
| #include "stdafx.h"  #include <Windows.h>  \_\_declspec(naked) void test()  {  \_\_asm  {  // let IF = 1  pushfd;  pop eax;  xor eax,0x100;  push eax;  popfd;  retf 4;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  if ((int)test != 0x401000) return 1;  \_\_asm  {  push 1;  int 0x20;  push 0x3b;  pop fs;  }  printf("no bsod\n");  return 0;  } |

这样就不会蓝了。

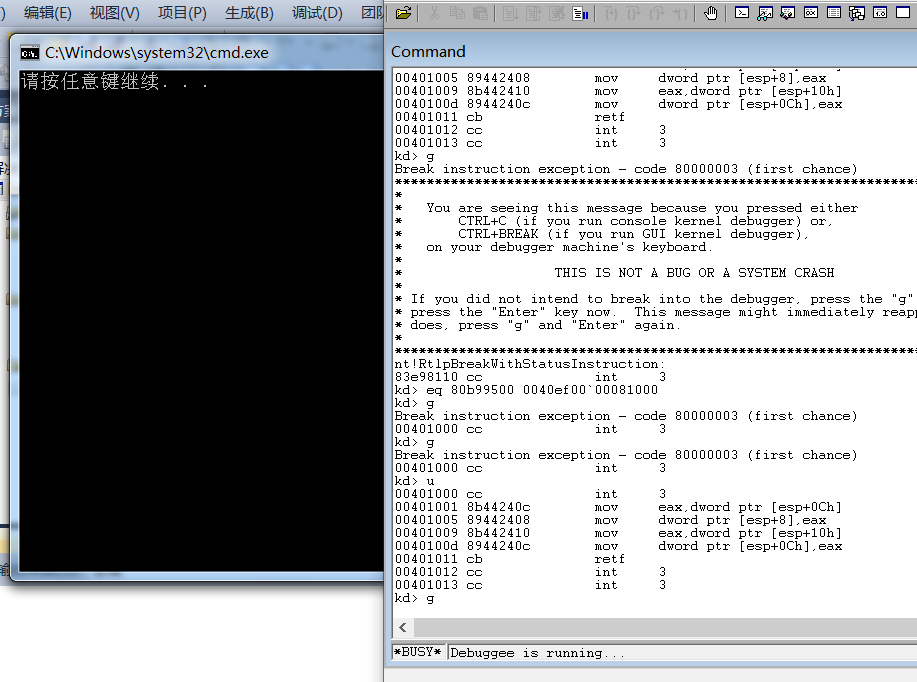
### 2.构建陷阱门 自己调用



火哥都没讲就让我们做，证明这个陷阱门跟中断门没啥区别，观察陷阱门描述符，只有type和中断门的不一样，E改成F就行了，还是用刚才的代码，稍微修改一下INT 0x20



成功断下



正常返回。

陷阱门和中断门唯一的区别是，陷阱门不会把 IF 位置0，即陷阱门可以相应可屏蔽中断。对比一下中断门和陷阱门进R0后，EFLAG的值：

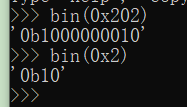


陷阱门



中断门

IF 是eflag的下标9



0x000202和0x00002的区别就是第九位是否置1.