在论坛2位高手mm的基础上谈谈结构化异常处理SEH (理论+CrackMe例子分析)(发表于看雪,这里是做个收藏)

原创

rageliu 于 2007-02-15 11:30:00 发布 2152 收藏
分类专栏： 汇编 文章标签： exception 数据结构 c struct byte os

由于不是破文，也不知道前面该加点什么

最近刚好看到riijj mm的CrackMe11，原帖：
http://bbs.pediy.com/showthread.php?s=&threadid=38021
和绫濑遥 mm的破文，原帖：
http://bbs.pediy.com/showthread.php?s=&threadid=38125

2位mm都讲的很到位，偶原本也没什么好说的了，不过偶看到网上很多关于SEH的资料都是理论居多，较少有和实际分析的结合，刚好这里又有个例子，所以我在2 mm的基础上说几句，我力求把关键的东东讲的明白。水平有限，不对的地方欢迎大家指出。

为了说清楚问题，文章可能会有点长，如果你想对SEH有更深的理解，我想看了应该是对你有些帮助的。当然大牛可以跳过，完全不用理偶。

首先一步一步讲几个SEH相关的概念，会出现的数据结构和需要注意的地方，我认为这是必须要知道的：

1.所谓结构化异常处理SEH,我们要知道它属于OS给我们提供的一种机制，也就是说，windows系统给所有程序都默认做了这个处理，和程序的开发环境开发语言（vc,vb,c#...）是无关的。平时我们没有捕获异常代码的时候，系统就做默认处理，比如程序崩溃时常见的那个错误提示框，就是系统的处理。

2.我们知道cpu是以线程为单位来分配运行时间片的，所以os也是以线程为单位来处理异常的。不同线程一般有不同异常处理函数。当然也可以多个线程使用同一个函数来处理。

3.异常发生的时候，系统到底做了什么呢：当cpu执行出现异常的时候，它会产生一个中断，通过中断将控制权转交到操作系统，操作系统就调用异常处理函数做处理。注意异常处理函数可能不止一个，而是以链表的形式有很多个。这里就会有2个问题：
a.要是有很多个异常处理函数，是全部都要调用吗？
b.操作系统是如何取得异常处理函数的地址的？
下面的4和5分别来回答这2个疑问。

4.异常处理函数是有一个返回值的，系统通过这个返回值来决定是否还要调用后面的异常处理函数。比如第一个异常处理函数它的返回值表示“我成功处理了这个异常，程序可以继续执行”(这里我用汉语，方便大家理解)，那系统就不用调用后面的异常处理函数了，而是将控制权交还程序，让它继续执行。而如果第一个异常处理函数返回值为“郁闷，偶没搞定这个异常，你让后面的兄弟来处理吧”，那么操作系统就会取下一个异常处理函数，让它来处理。最后如果该线程链表上的所有异常处理函数都没能成功处理这个异常，那操作系统就会释放对应的资源，然后毫不犹豫地kill掉这个程序。

5.我们来回答3中的问题b，这里会引出几个重要的数据结构，希望大家好好理解。我们经常说任何事情都有一个环境，线程也不例外。操作系统在启动一个线程的时候，也会保存该线程的一个运行环境。这就是我们经常说的线程的环境块TEB,早些时候的操作系统称为TIB,这个TIB好象很神秘，那么它到底是个什么东东呢？？其实它就是一个数据结构，里面保存了与本线程相关的一些信息而已。可能到这里你已经想到了，对，我们的异常处理函数的链表指针就保存在这里。哈哈，到此我们b问题基本搞明白了，下面讲解一下TIB这个结构，本来该结构早有大牛分析过了，不过由于它很重要，不弄明白它后面会比较吃力，所以我这里再说下：

```
typedef struct _TIB
{
PEXCEPTION_REGISTRATION_RECORD pvExcept;//我们只关心这里，注意是一个结构的指针
...
后面还有很多，由于我们这里不用，我省略之
...
} TIB, *PTIB;
```

下面看看EXCEPTION_REGISTRATION_RECORD结构的定义：

```
typedef struct _EXCEPTION_REGISTRATION_RECORD
{
    struct _EXCEPTION_REGISTRATION_RECORD * pNext;
    FARPROC                                 pfnHandler;
} EXCEPTION_REGISTRATION_RECORD, *PEXCEPTION_REGISTRATION_RECORD;
```

这个结构就十分简单了，一个指向自身EXCEPTION_REGISTRATION_RECORD结构的指针，一个FARPROC，也就是函数的指针。pNext指向下一个异常处理结构的地址，通过它就构成了上面4中所说的异常处理函数的链表(当然其实是一个EXCEPTION_REGISTRATION_RECORD结构指针的链表，为了便于理解，请原谅我使用“异常处理函数链表”)，而pfnHandler也就是当前异常处理函数的指针了。

这里特别说明TIB结构的指针是保存在FS:[0]这里的！！！！！！！！！！！！

罗罗嗦嗦的说了这么多，也不知道你是否理解。明白了我们就接着讲，要是没理解请再看一遍。。。如果还不理解，请相信是我没讲明白，你可以休息休息再来研究。

Go on...

6.我们在写程序的时候，为了程序的健壮性，可能会不止一处用__try __except __finally这样的代码来捕获并处理异常。当这样的语句太多的时候，如果我们为每个__try写一个异常处理函数，那异常处理函数就会非常多，这是我们不愿意看到的。到这里不得不佩服一下MS,它早想到了这个问题，并且给我们完美的解决了。说白了挺简单：不管有多少个__try，我们只写一个异常处理函数，不同的__try我们给这唯一的一个异常处理函数传递不同的参数，根据参数做不同处理。那么这个参数都需要包含什么信息呢？这里又要引入一个数据结构了，我们传的参数就是这个数据结构的指针，一切一切需要的信息都在这个结构中。下面是这个叫SCOPETABLE的结构的定义：

```
typedef struct _SCOPETABLE
{
DWORD previousTryLevel; //外层try链表指针
DWORD lpfnFilter;  //异常处理代码地址
DWORD lpfnHandler;  //如果异常成功处理后程序接着执行的地址
} SCOPETABLE, *PSCOPETABLE;
```

上面结构中的3个变量我都做了说明，可是似乎没有哪一个是标志特定__try的啊。聪明！！！还真是没有，所以还需要一个参数来唯一确定是哪个__try中抛出的异常。所以我们引入一个和__try一一对应的标识做为另一个参数，叫TryLevel吧，听起来还是蛮专业的，嘿嘿...

操作系统如何处理这个TryLevel呢？在没有__try的时候，初始化TryLevel=-1。以后第1个__try时TryLevel=0，第2个__try时TryLevel=1，第2个__try时TryLevel=2 ...以次类推。我们仔细想象，这样不但区别了不同的__try,是不是还能有其他的作用，其他的什么作用呢？？注意到0 1 2 3 4...这样的东东是不是很象数组的下标，哈哈，对，操作系统正是用了它来做为SCOPETABLE结构的索引。

理论先说这么多，基本说完一个SEH的轮廓了，如果你坚持看到了这里，那么恭喜，你成功一半了。接下来我们以riijj mm的CrackMe11为例子结合绫濑遥 mm的破文来对上面提到的几点东东做实际代码级的分析。

OD打开CrackMe11.exe，程序停在入口点，这里就出现了对SEH的初始化，这就是系统默认的异常处理设置，看下面代码：

```
004022AA >/$  55            PUSH EBP     ；这是程序入口点，标准初始化操作
004022AB  |.  8BEC          MOV EBP,ESP
004022AD  |.  6A FF         PUSH -1      ；上面6中说的，初始化TryLevel=-1
004022AF  |.  68 40A24000   PUSH riijjcm1.0040A240  ；6中2个参数的SCOPETABLE的结构指针
004022B4  |.  68 E81F4000   PUSH riijjcm1.00401FE8      ；00401FE8地址是异常处理函数入口
004022B9  |.  64:A1 0000000>MOV EAX,DWORD PTR FS:[0]
004022BF  |.  50            PUSH EAX     ；原始指针
004022C0  |.  64:8925 00000>MOV DWORD PTR FS:[0],ESP ；设置新指针
```

上面的代码也就是在栈上构造了一个5中讲到的EXCEPTION_REGISTRATION_RECORD结构，并使这个结构的pNext指向了原始的EXCEPTION_REGISTRATION_RECORD结构。也就是在原来的异常处理函数链的头上加上了我们构造的这个新的EXCEPTION_REGISTRATION_RECORD结构，从而有异常的时候我们的异常处理函数会有优先处理该异常的机会，谁让偶在链表头呢？？嘿嘿。至于偶处理后返回什么值，是否还要向链表后面的异常处理函数传递该异常，程序是否还能继续执行等等，就看偶心情了(如不明白偶在说什么，请看上面的第5点，那个引号里面的汉语说明)，哈哈。

接着我们看WinMain代码部分，前面是热身，这才是重点。WinMain的入口在00401BB0,别问偶是怎么看出来的，这不属于本文要讲解的范围。看代码：

```
00401BB0  $ 55          PUSH EBP                    ;函数标准入口处理方式
00401BB1  . 8BEC        MOV EBP,ESP
00401BB3  . 6A FF       PUSH -1                     ;初始化trylevel=-1
00401BB5  . 68 18A24000  PUSH riijjcm1.0040A218     ;6中SCOPETABLE结构指针
00401BBA  . 68 E81F4000  PUSH riijjcm1.00401FE8     ;异常处理函数地址
00401BBF  . 64:A1 0000000>MOV EAX,DWORD PTR FS:[0]
00401BC5  . 50          PUSH EAX                    ;新的结构中指向原来结构的指针
00401BC6  . 64:8925 00000>MOV DWORD PTR FS:[0],ESP  ;设置FS[0]为上面构造的结构的指针
00401BCD  . 83EC 08     SUB ESP,8                   ;给局部变量分配空间
00401BD0  . 53          PUSH EBX
00401BD1  . 56          PUSH ESI
00401BD2  . 57          PUSH EDI
00401BD3  . 8965 E8     MOV DWORD PTR SS:[EBP-18],ESP
00401BD6  . C745 FC 00000>MOV DWORD PTR SS:[EBP-4],0   ;设置trylevel=0，也就是 try{ 开始SEH
00401BDD  . 33C0        XOR EAX,EAX                 ;EAX=0
00401BDF  . C600 00     MOV BYTE PTR DS:[EAX],0      ;mov [0],0这里会抛异常
00401BE2  . C745 FC FFFFF>MOV DWORD PTR SS:[EBP-4],-1
00401BE9  . EB 4C       JMP SHORT riijjcm1.00401C37  ;这里跳了就完蛋
00401BEB    B8 01000000  MOV EAX,1                  ;__except(1)
00401BF0  . C3          RETN
00401BF1  . 8B65 E8     MOV ESP,DWORD PTR SS:[EBP-18]
00401BF4  . C745 FC 01000>MOV DWORD PTR SS:[EBP-4],1
00401BFB  . 33C0        XOR EAX,EAX
00401BFD  . 33D2        XOR EDX,EDX
00401BFF  . F7F0        DIV EAX                     ;__try{0/0}
00401C01  . 83C8 FF     OR EAX,FFFFFFFF
00401C04  . EB 2B       JMP SHORT riijjcm1.00401C31  ;这里跳了就完蛋
00401C06  . B8 01000000  MOV EAX,1                  ;__except(1)
00401C0B  . C3          RETN
00401C0C  . 8B65 E8     MOV ESP,DWORD PTR SS:[EBP-18]
00401C0F  . C745 FC 02000>MOV DWORD PTR SS:[EBP-4],2
00401C16  . E8 55FBFFFF  CALL riijjcm1.00401770  ;该call调试模式下不会抛异常
00401C1B    EB 0E       JMP SHORT riijjcm1.00401C2B  ;这里跳了就完蛋
00401C1D  . B8 01000000  MOV EAX,1
00401C22  . C3          RETN
00401C23  . 8B65 E8     MOV ESP,DWORD PTR SS:[EBP-18]
00401C26  . E8 75FDFFFF  CALL riijjcm1.004019A0
00401C2B  > 83C8 FF     OR EAX,FFFFFFFF
00401C2E  . 8945 FC     MOV DWORD PTR SS:[EBP-4],EAX
00401C31  > 8945 FC     MOV DWORD PTR SS:[EBP-4],EAX
00401C34  . 8945 FC     MOV DWORD PTR SS:[EBP-4],EAX
00401C37  > 33C0        XOR EAX,EAX
00401C39  . 8B4D F0     MOV ECX,DWORD PTR SS:[EBP-10]
00401C3C  . 64:890D 00000>MOV DWORD PTR FS:[0],ECX
00401C43  . 5F          POP EDI
00401C44  . 5E          POP ESI
00401C45  . 5B          POP EBX
00401C46  . 8BE5        MOV ESP,EBP
00401C48  . 5D          POP EBP
00401C49  . C2 1000     RETN 10
```

上面是WinMain的全部代码，有些注释，还很多都是绫濑遥 mm破文中的，下面我们来一句一句的分析：

```
00401BB0   $  55           PUSH EBP                    ;函数标准入口处理方式
00401BB1   .  8BEC         MOV EBP,ESP
00401BB3   .  6A FF        PUSH -1                     ;初始化trylevel=-1
00401BB5   .  68 18A24000  PUSH riijjcm1.0040A218      ;6中SCOPETABLE结构指针
00401BBA   .  68 E81F4000  PUSH riijjcm1.00401FE8      ;异常处理函数地址
00401BBF   .  64:A1 0000000>MOV EAX,DWORD PTR FS:[0]
00401BC5   .  50           PUSH EAX                    ;新的结构中指向原来结构的指针
00401BC6   .  64:8925 00000>MOV DWORD PTR FS:[0],ESP   ;设置FS[0]为上面构造的结构的指针
00401BCD   .  83EC 08      SUB ESP,8                   ;给局部变量分配空间
00401BD0   .  53           PUSH EBX
00401BD1   .  56           PUSH ESI
00401BD2   .  57           PUSH EDI
00401BD3   .  8965 E8      MOV DWORD PTR SS:[EBP-18],ESP
```

这里的

```
00401BB3   .  6A FF        PUSH -1                     ;初始化trylevel=-1
```

由于初始化trylevel=-1，记得上面6中我们说了我们的索引是从0开始的，所以这个-1也就说明了这里还没有__try这样的的语句。

下面这2句非常重要，主要是入栈的2个地址，后面__try到异常后需要用到

```
00401BB5   .  68 18A24000  PUSH riijjcm1.0040A218      ;6中SCOPETABLE结构指针
00401BBA   .  68 E81F4000  PUSH riijjcm1.00401FE8      ;异常处理函数地址
```

其他就不罗嗦了，就是在系统默认的异常处理函数链表头加上了我们自己的异常处理函数(结构指针)。接着向下看，这就到了第一个出现__try异常的部分了：

```
00401BD6   .  C745 FC 00000>MOV DWORD PTR SS:[EBP-4],0    ;设置trylevel=0，也就是 try{ 开始SEH
00401BDD   .  33C0         XOR EAX,EAX                 ;EAX=0
00401BDF   .  C600 00      MOV BYTE PTR DS:[EAX],0     ;mov [0],0这里会抛异常
00401BE2   .  C745 FC FFFFF>MOV DWORD PTR SS:[EBP-4],-1
00401BE9   .  EB 4C        JMP SHORT riijjcm1.00401C37 ;这里跳了就完蛋
00401BEB      B8 01000000  MOV EAX,1                   ;__except(1)
00401BF0   .  C3           RETN
00401BF1   .  8B65 E8      MOV ESP,DWORD PTR SS:[EBP-18]
```

上面的代码如果顺序执行下来，有个JMP SHORT riijjcm1.00401C37，要是跳了WinMain就会结束，也就是说跳了程序就结束了。可是我们执行的时候程序是正常的，并没有结束，那说明JMP SHORT riijjcm1.00401C37这句没跳。哈哈，大家可能要问：JMP是无条件的跳转哦，怎么会不跳呢？？？？？问得好，那唯一的解释就是这个JMP没有被执行到，别跳过了，这就是SEH发挥的作用，看JMP的前面有这样的代码：

```
00401BDF   .  C600 00      MOV BYTE PTR DS:[EAX],0     ;mov [0],0这里会抛异常
```

这句显然是会抛异常的，请复习前面的第3点：“当cpu执行出现异常的时候，它会产生一个中断，通过中断将控制权转交到操作系统，操作系统就调用异常处理函数做处理”，说明什么？？到这句的时候操作系统获得了控制权，它要去调用我们的异常处理函数。前面说了注意的那2个地址，其中的riijjcm1.00401FE8就是我们的异常处理函数，如果你对它下断，Shift+F9过异常就会到riijjcm1.00401FE8这里的，呵呵，你可以分析下riijjcm1.00401FE8函数的功能，它是根据6中SCOPETABLE结构指针做处理，我这里就不说了。我重点说下riijjcm1.0040A218 这个地址，我们前面说了它是6中SCOPETABLE结构指针，异常处理代码也就是在这个结构中，异常处理完后如果程序还能继续执行，那么操作系统将控制权转交给程序的时候程序接着执行的代码的地址也在这里。

用DB 0040A218命令来看看这里都存储了些什么东东，还记得前面定义的SCOPETABLE结构吗？？它有3个变量，这里都是DWORD类型的，那我们从0040A218地址开始取3个DWORD大小的内容来看看，如下：

FF FF FF FF EB 1B 40 00 F1 1B 40 00

这里你可能看不出来什么，我将来它转换成SCOPETABLE结构结构来看看(注意高高低低的转换顺序)：

typedef struct _SCOPETABLE
{
DWORD previousTryLevel; //外层try链表指针 (就是：FF FF FF FF)
DWORD lpfnFilter; //异常处理代码地址 (就是: 00 40 1B EB)
DWORD lpfnHandler; //如果异常成功处理后程序接着执行的地址 (就是:00 40 1B F1)
} SCOPETABLE, *PSCOPETABLE;

注意到什么了吗？？对，都2个DWORD都是代码的地址，就是根据这2个地址来做了很多了操作：
//异常处理代码地址 (就是: 00 40 1B EB)说明我们这个异常的处理代码在00401BEB，看到这个地址了吗？？居然刚好是上面那个该死的JMP的下一个指令，哈哈！！这样是不是就跳过了那个JMP指令了。嘿嘿

00401BEB B8 01000000 MOV EAX,1 ;__except(1)
00401BF0 . C3 RETN

大家知道函数的返回值是在EAX里面，上面2句不就是说异常处理函数返回了1吗？？1代表什么呢？偶调试发现该异常处理了后程序能继续执行，是不是1就代表了我们上面第4点中的“我成功处理了这个异常，程序可以继续执行”，为了测试，偶将那个1改成了-1和0，两种情况下程序都一直在异常和异常处理代码间死循环，你可以试试，呵呵，我们继续...

注意上面那个RETN并不是说WinMain函数返回了，而是代码执行完了返回系统，将控制权交给系统，系统根据那个返回值（EAX中的1）看到我们成功处理了这个异常，程序还可以继续执行。OS应该是把控制权交还给程序的时候了。控制权是回来了，可是应该从哪里接着执行呢？？？看上面SCOPETABLE结构中的第3个变量，这就是接着执行的地址DWORD lpfnHandler; //如果异常成功处理后程序接着执行的地址 (就是:00 40 1B F1)，也就是00401BF1,却不管00401BF1这个地址是做什么，到这里我们的一个异常处理就完成了，却是成功搞定了这个异常，所以程序得以继续执行，也成功跳过了那个该死的JMP。

到这里写了1个多小时，累死偶了。也不知道说没说明白，再不走我就坐不到公交了，如果对大家有些帮助偶就找时间把后面的接上。

再次感谢上面提到的2位mm，嘿嘿.....走人