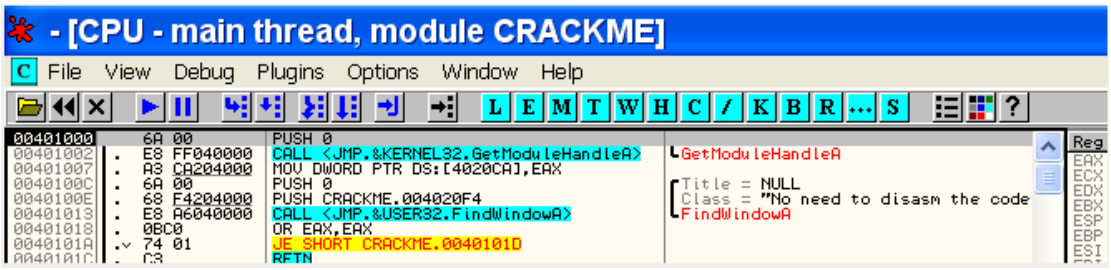
第 25 章-异常处理

本章,我们将介绍异常处理,这一块通常是初学者的绊脚石,但是如果研究深入一点的话,你会发现它并不难。

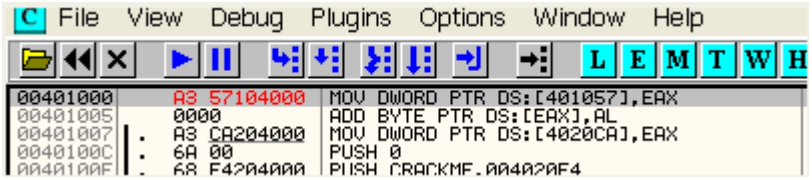
那么异常是如何产生的呢?当处理器执行了一个错误的操作的时候,程序中就会产生异常。好了,我们来看几个异常例子吧,我们用 OD 打开 cruehead'a 的 CrackMe。



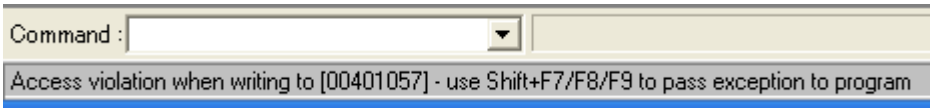
可以看到断在了入口点处,我们在第一行输入会引发异常的指令,这里我们采用 Mr Silver 写的异常例子中指令。

内存访问异常:当线程中尝试访问没有访问权限的内存的时候会发生该类异常。例如:一个线程尝试向只具有读权限的内存写入数据的时候就会产生内存访问异常。

我们在 OD 中输入如下指令:



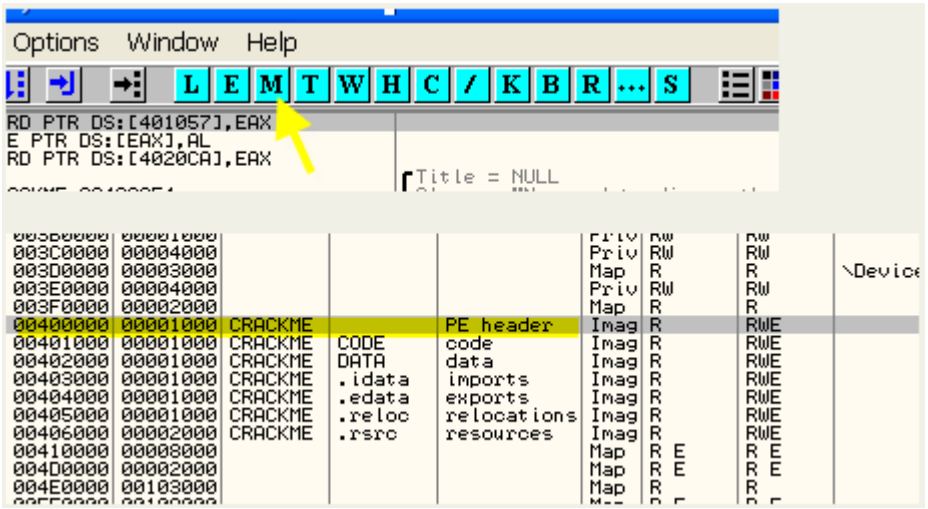
这里我们可以看到 401057 开始的内存单元只具有读取和执行权限,并不具有写入权限。因此如果尝试向该内存单元写入数据的话就会产生异常,我们按 F8 键。



程序会根据 PE 头中的相关信息设置区段的初始权限,当然也可以使用诸如 VirutalProtect 这类 API 函数在运行时修改权限。

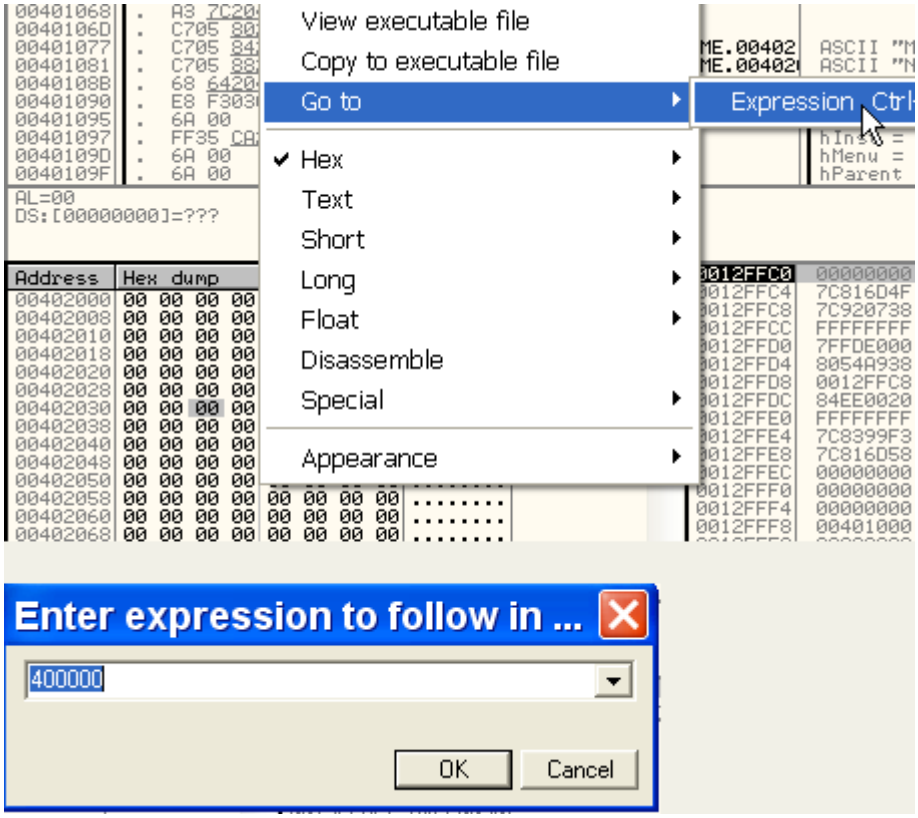
那么在 OllyDbg 中我们如何查看各个区段的初始权限呢,还有就是如何修改这些权限呢?

我们可以通过单击工具栏中的 M 按钮来查看各个区段的情况。

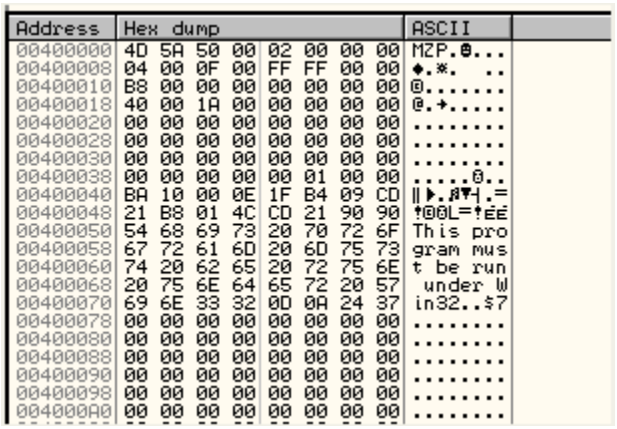


我们可以看到主模块的所在的区段开始于 400000,首先是 PE 头,占 1000 个字节,PE 头中保存了各个区段的名称,长度以及程序运行所必须的一些信息。

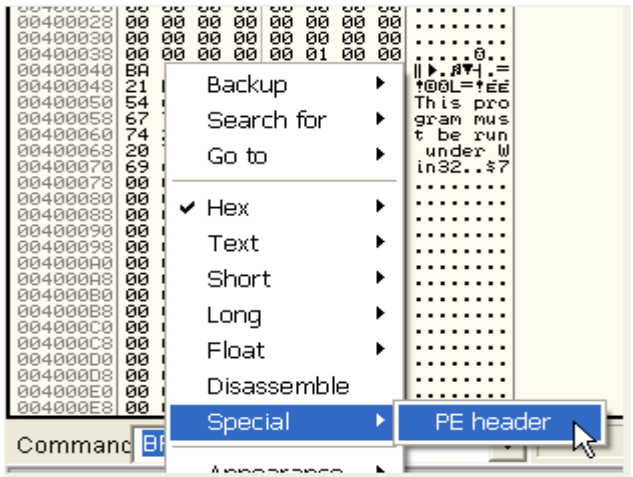
我们在数据窗口中定位到 PE 头。

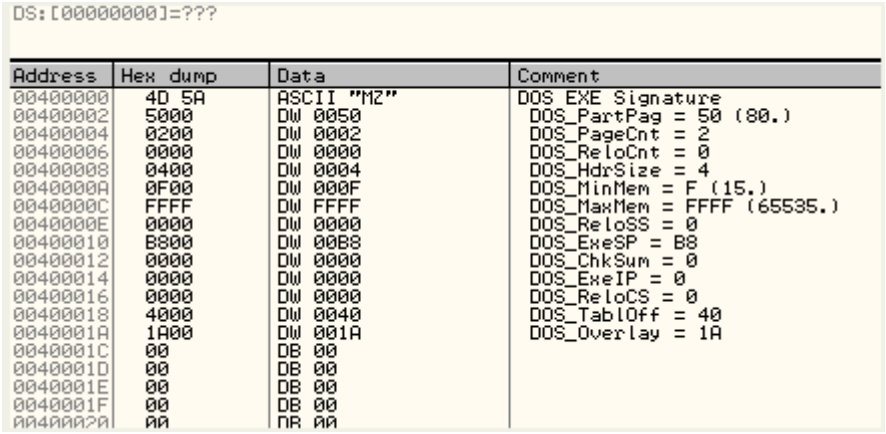


由于 PE 头开始于 400000,所以我们输入 400000。

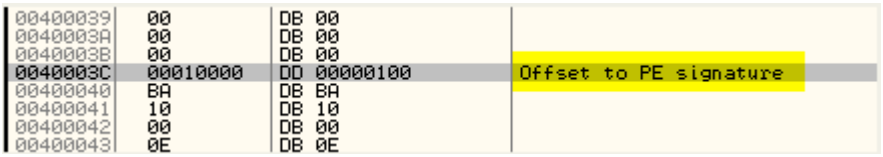


其实 OllyDbg 中有一个可以解析 PE 头的各个字段的选项,我们可以在数据窗口中单击鼠标右键。

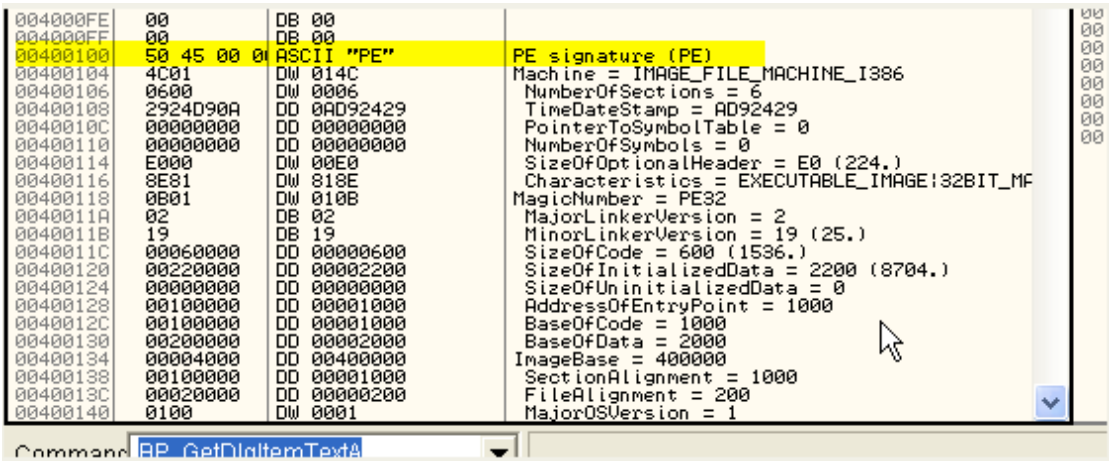




我们可以看到显示出了 DOS 头的各个字段信息。

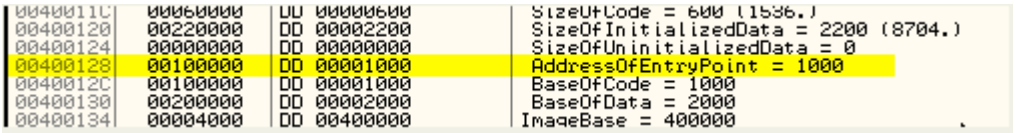


如果我们继续往下看,我们首先会看到 Offset to PE signature,这是告诉我们 PE 头的偏移量,我们可以看到偏移量为 100。那么起始地址 400000 加上 100 就是 400100,我们定位到 400100 处。

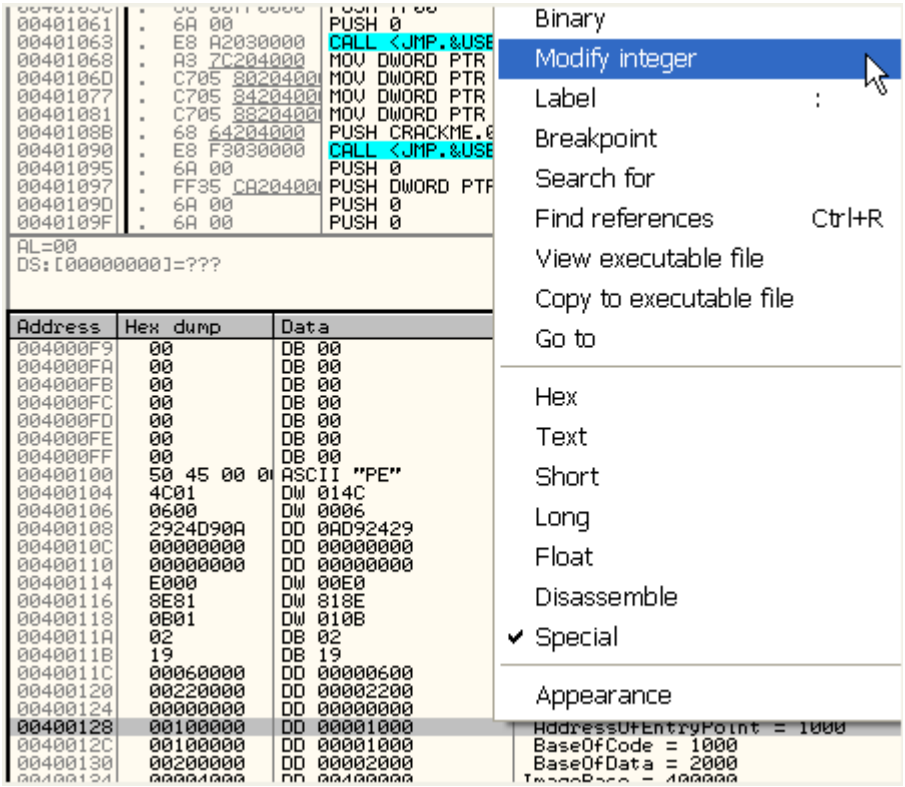


这里你所看到的,这里是关于程序的重要信息。

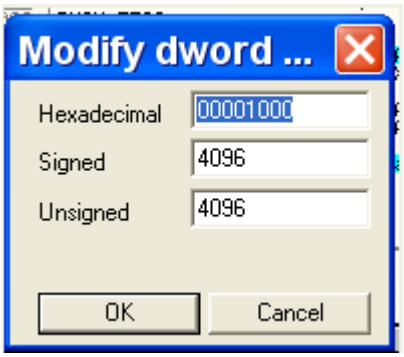
下面让我们来看看部分字节详细的解释。



也就是说基地址 400000 加上 1000 就是程序的入口点。如果你想修改入口点的话,例如把入口点修改为 2000,我们可以在入口点这一行上单击鼠标右键。

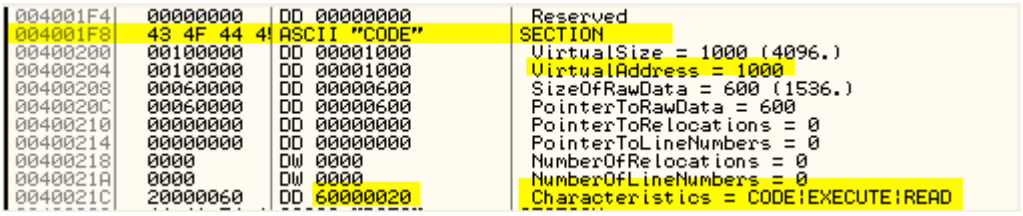


这里我们可以输入任意数值,例如:如果你想让程序从 402000 处开始执行,我们可以输入 2000。这个 2000 是相对于映像基址 400000 的偏移量。



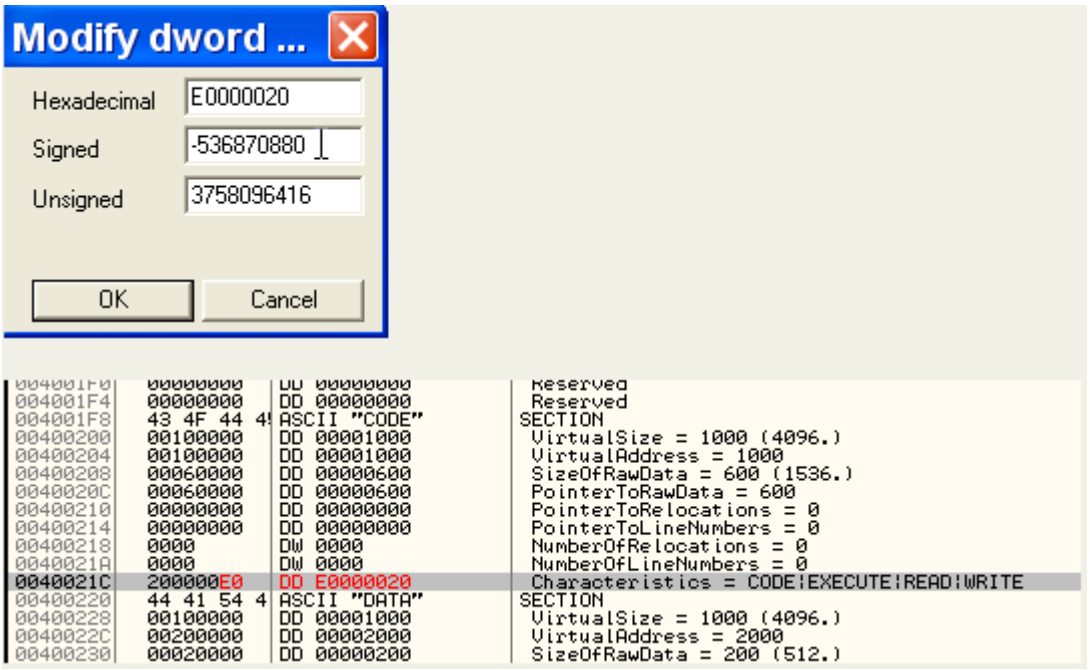
修改完毕以后,我们可以单击鼠标右键选择-Copy to executable file,然后在弹出的窗口中单击鼠标右键选择-Save file,这样就可以保存到文件了,我们并不修改入口点,只要知道可以这么做就行了。

好了,我们继续往下看。

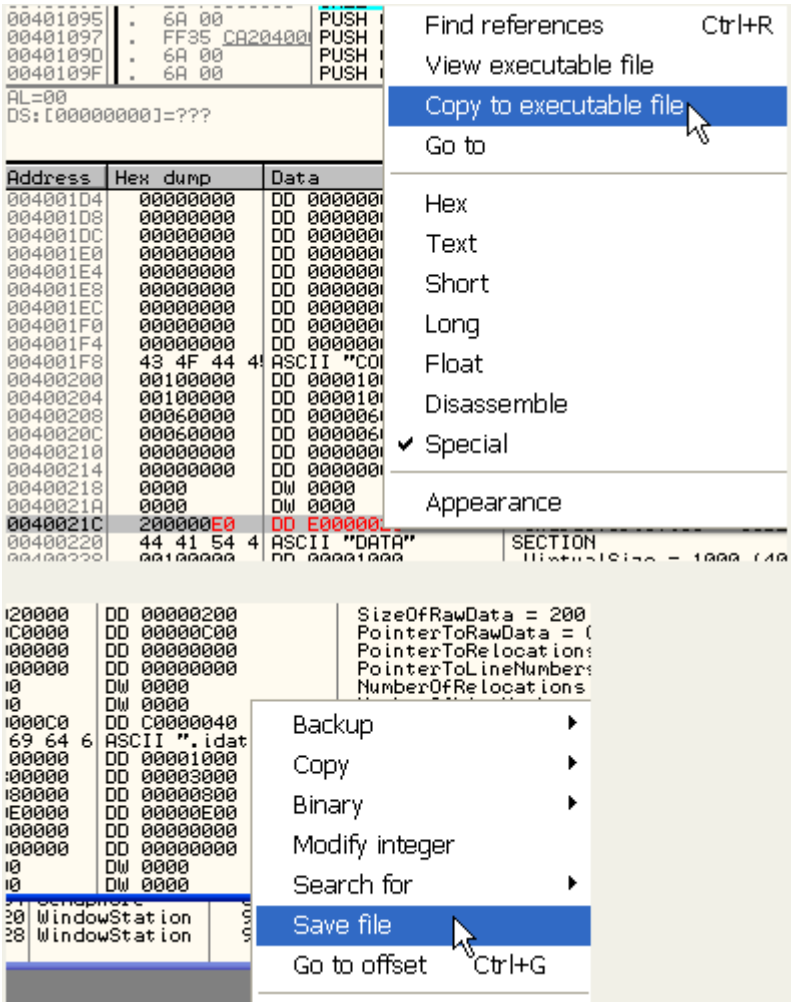


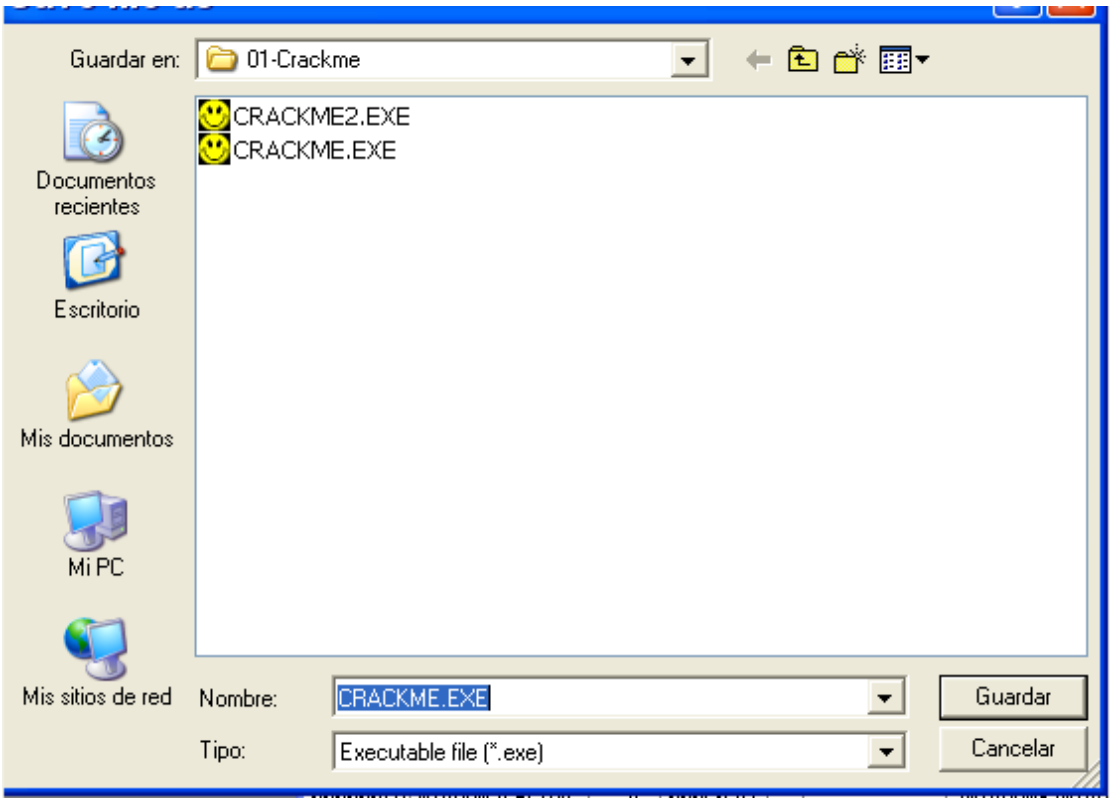
下面各个区段的信息,首先我们看到的这个区段起始虚拟地址为 1000,注意这里是偏移量,实际上内存地址为 401000,并且 Characteristics(特征)为 CODE,EXECUTE,READ(代码段,可执行,可读)。

如果我们想让该区段具有可写权限的话,我们可以将 Characteristics 这个字段的 60000020 修改为 E0000020,这样该区段就具有了所有权限,嘿嘿,我们来验证一下。

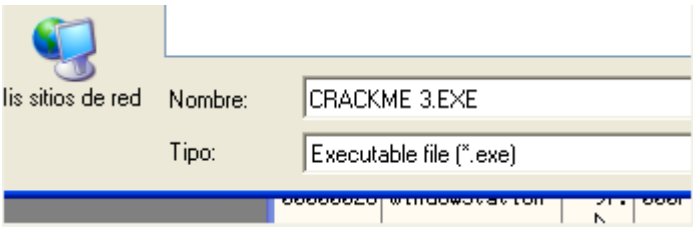


好了,我们现在来将修改保存到文件。

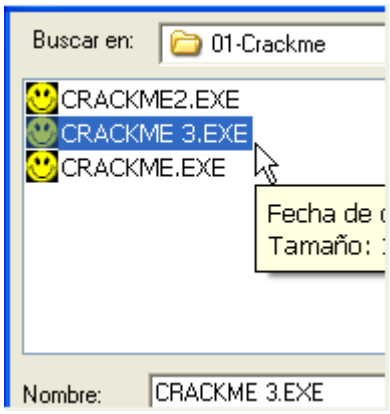


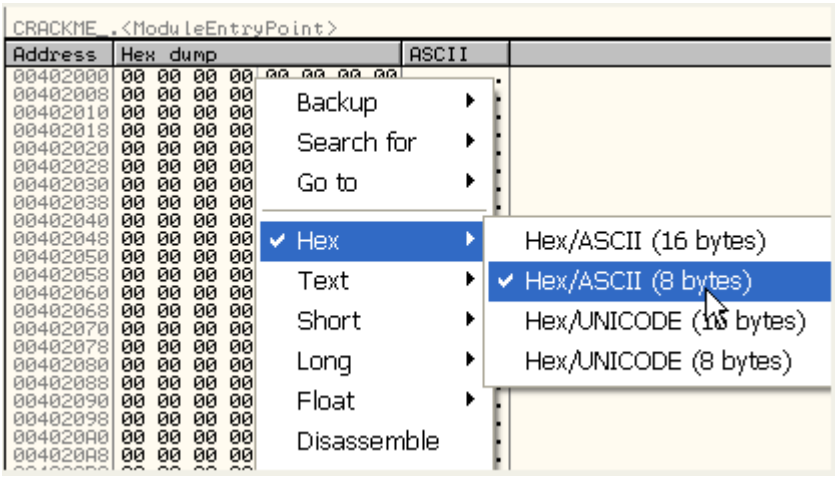


我们将名称修改为 CRACKME 3,标识这个文件是修改版。

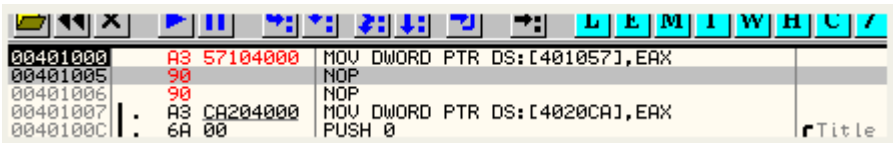


好了,我们现在用 OllyDbg 打开这个 CRACKME3.EXE。





现在我们将数据窗口显示模式切换为正常模式。

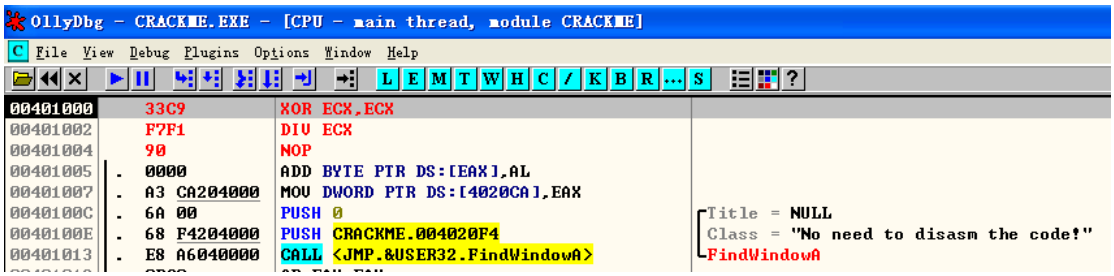


我们按 F8 键单步,会发生并没有产生异常,EAX 的值被成功写入了 401057 内存单元中了。

下面介绍另外一种异常:

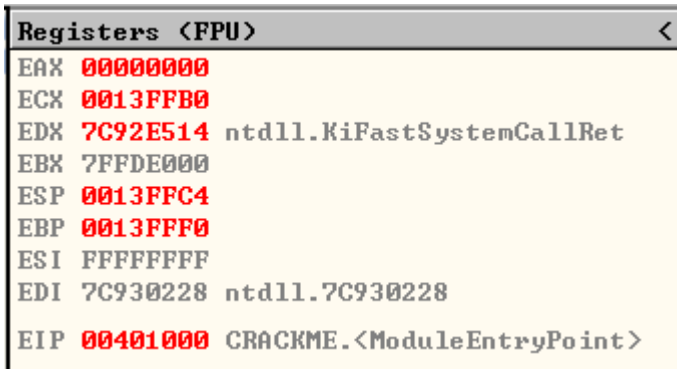
除 0 异常:试图除以 0 时会产生该异常。

例如,我们在 OllyDbg 中输入如下指令:



(PS:原作者这个地方讲错了,他是直接 DIV ECX,ECX 这个时候并不为零,EAX 为零,EDX 指向了 7C92E514,所以按照作者的做法,只会产生整数溢出异常,并不会产生除 0 异常。)

寄存器的如下:



好了,我们 F8 单步一次。

寄存器的情况如下:

```
Registers (FPU)
EAX 00000000
ECX 00000000
EDX 7C92E514 ntdll.KiFastSystemCallRet
EBX 7FFDE000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C930228 ntdll.7C930228

EIP 00401002 CRACKME.00401002
```

这里时候除数 ECX 为 0 了。我们继续 F8 单步。

```
00402078 00 00 00 00 00 00 00 00 ........
Command :
Integer division by zero - use Shift+F7/F8/F9 to pass exception to program
```

我们可以看到提示 Integer division by zero(整数除 0)异常。

无效指令,尝试执行特权指令异常:

当 CPU 试图执行越权指令的话就会产生该异常。

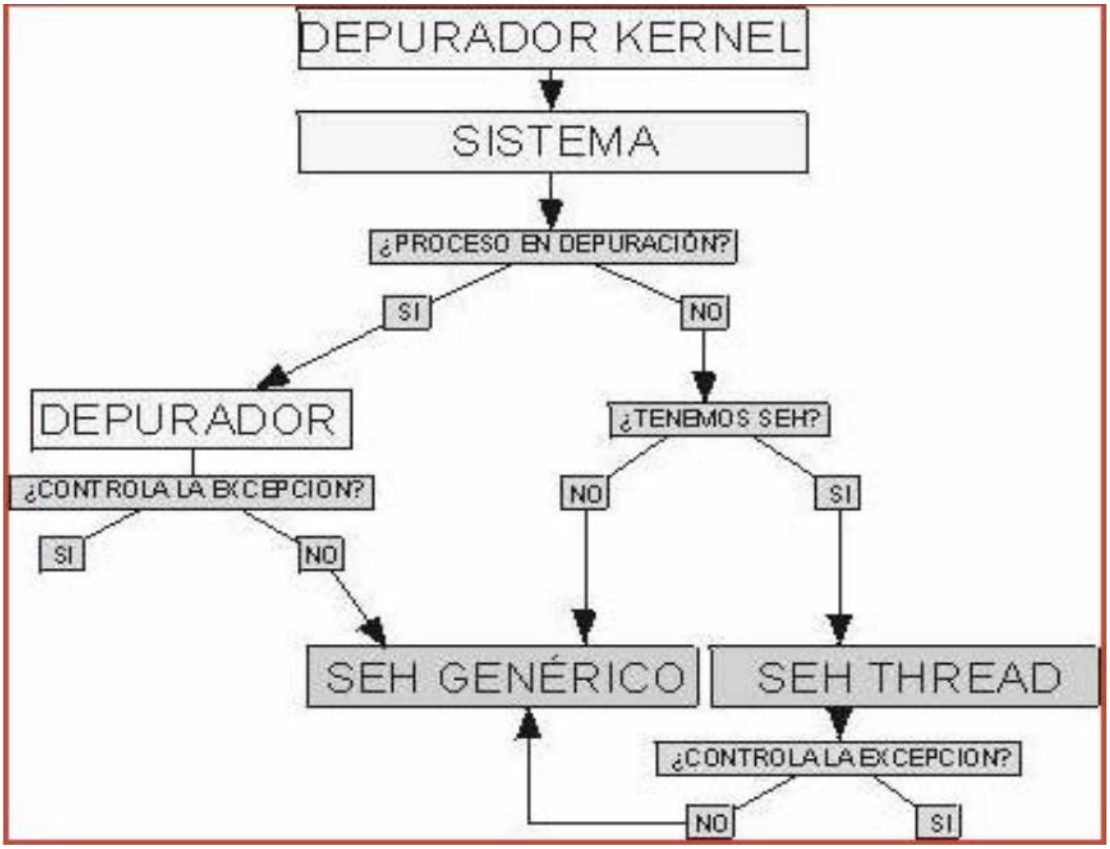
由于 OllyDbg 不允许我们输入 CPU 不可识别的指令,所以我们无法验证。但是程序员可以自己设计一些处理器并不支持的指令,当执行到指令时显示相应的错误即可。

最为典型就是 INT 3 指令,INT 3 指令会产生一个异常,并且该异常会被调试器捕获到,比如,我们可以设置 BPX 断点来让程序中断下来,然后就可以对该程序进行相应的控制了。

另外,有一些程序会直接写入 INT 3 指令,所以说 INT 3 产生的异常是最常见的。

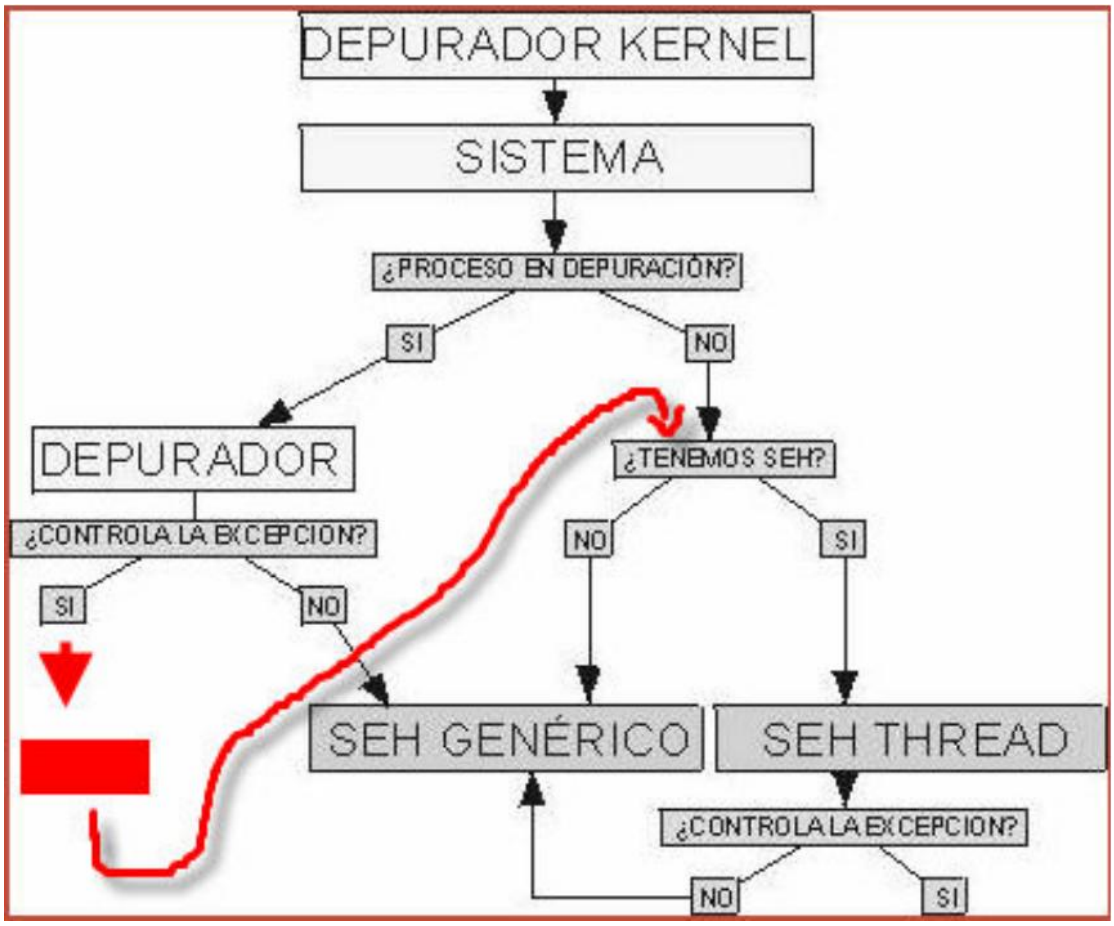
其实还有很多其他的异常,这里我们就不一一介绍了。下面我们看一个简单的例子。

现在我们只知道该程序会产生异常,但是到底会产生哪种异常呢?我们现在先来看看下面这个示意图:



这个图是我从 Mr Silver 的教程中截取出来的,这里我们可以看到一个异常是被处理的流程,首先系统会判断当前进程是否正在被调试。

根据上图来看可以知道异常发生后,如果当前程序正在被调试的话,那么此时控制权就会交予调试器。如果调试器的调试选项勾选了跳过对应的异常类型的话,这个时候控制权又会重新归还给当前程序,如果没有勾选跳过对应的异常的话,那么我们就需要按 Shift + F9 键来跳过该异常并将控制权交予程序了。但是控制权交予程序以后的流程该如何走上图中并没有标注出来。所以我们接着来看下面的流程图。



这里我补全了整个流程图,见上图的红色箭头。

接下来我们可以看到,当前控制权由调试器归还给程序以后,系统会检查当前程序是否安装了 SEH,如果安装了 SEH,就转向 SEH 的异常处理程序执行,如果没有安装 SEH,就会调用系统默认的异常处理程序。以上介绍听起来可能有点复杂,其实并不复杂,我们再来详细介绍一下什么是 SEH。

什么是 SEH 呢?

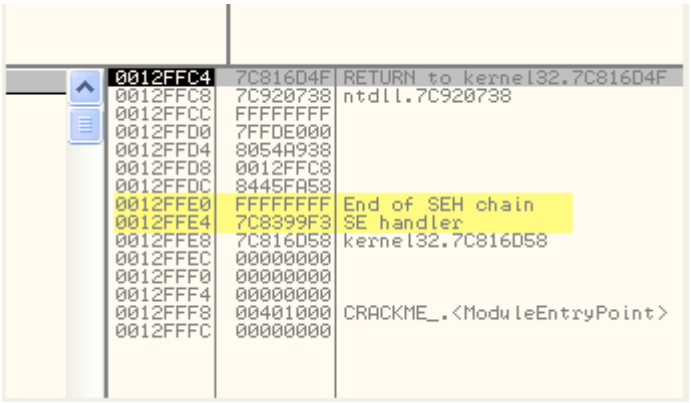
SEH 或者结构化异常处理,它是用来确保该程序可以从错误中恢复,也就是说,如果你没有设置 SEH,那么当程序中有异常发生时,程序就会弹出一个错误信息框,告诉我们程序即将关闭。如果我们设置了 SEH 的话,异常处理程序就能够捕获到程序中发生的异常,进行相应的处理后,就会把控制权重新交予程序继续执行,程序并不会终止也不会弹出那个烦人的错误消息框。

此外,我们需要知道每个线程都可以有自己的异常处理程序,如果当前异常处理程序不予处理的话,可以将异常将于 SEH 链中的其他异常处理程序来处理。

如何定位异常处理程序

好了,我们用 OD 重新加载 cruehead'a 的 CrackMe。

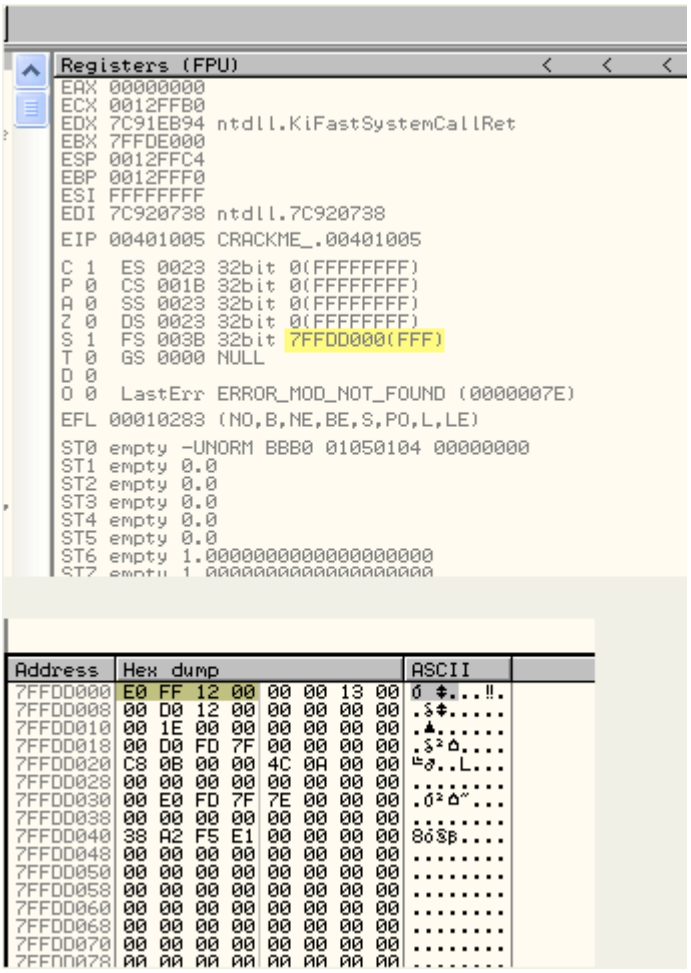
我们来看看堆栈的情况。



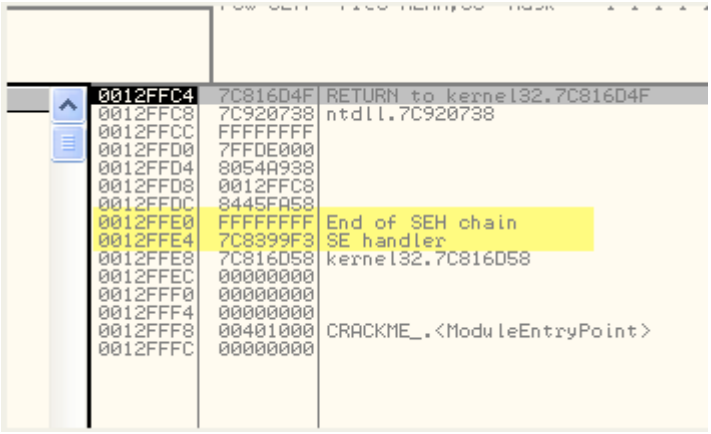
这是系统默认安装的异常处理程序,无论什么异常交予该默认异常处理程序处理的话,它都会弹出错误消息框。现在我们来定位到该默认异常处理函数。

我们看到,FS:[0]就是指向了当前异常处理程序。

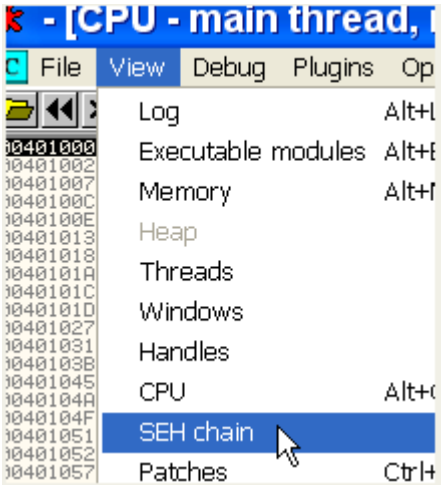
我们在数据窗口中定位到 FS:[0]。



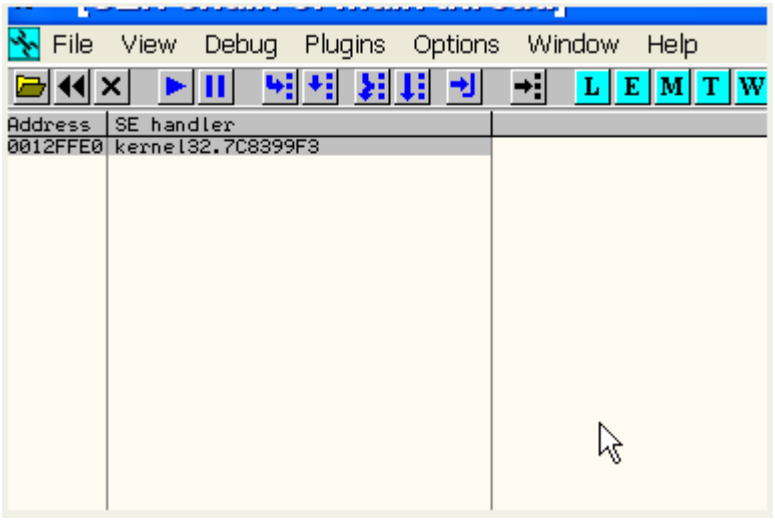
这里我们可以看到 FS:[0]指向的内存单元中内容是多少,可能不同的机器上这个值会不一样,我这里 FS:[0]指向内存单元中保存的值是 12FFE0。



从堆栈中我们可以看到,12FFE0 指向的是 SEH 链的最后一个结点,当前异常被交予该结点的异常处理程序的话,就会弹出一个我们熟悉的错误消息框。



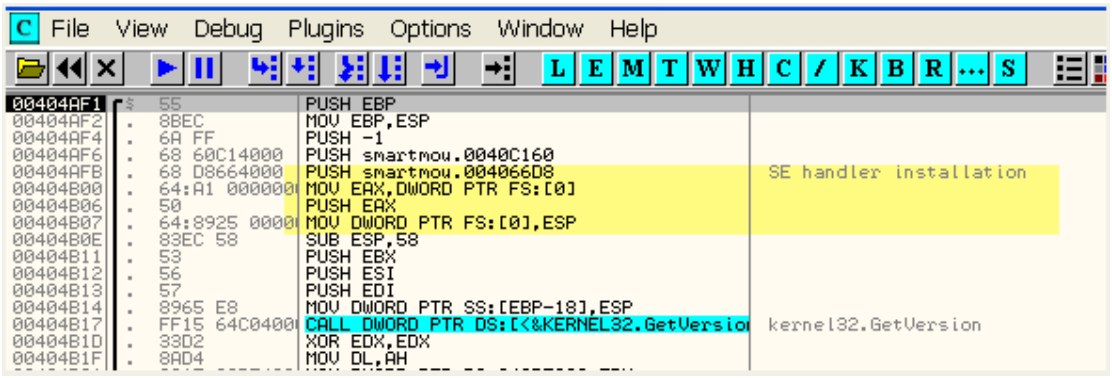
我们来查看一下 SEH 链的情况。



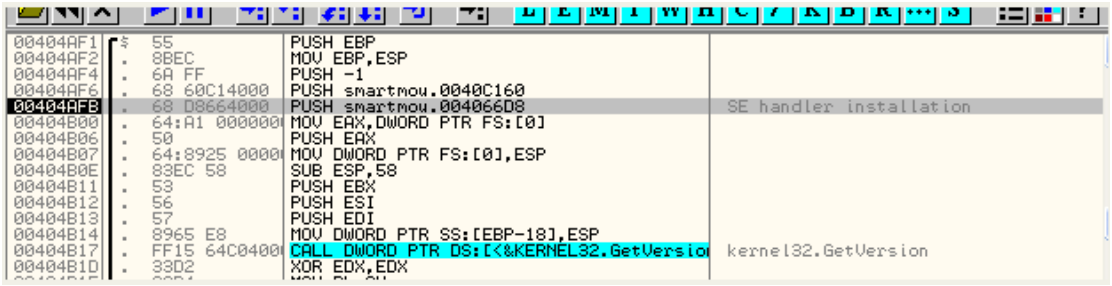
我们可以看到只有系统默认的异常处理程序被安装了,当有多个异常处理程序的话,当捕获到异常的话,异常会依次由 SEH 链的顶部向底部传递。

由于 cruehead'a 的 CrackMe 并没有安装自己的异常处理程序,所以这里我们再看另一个例子 smartmouse111。

我们用 OD 加载这个例子。

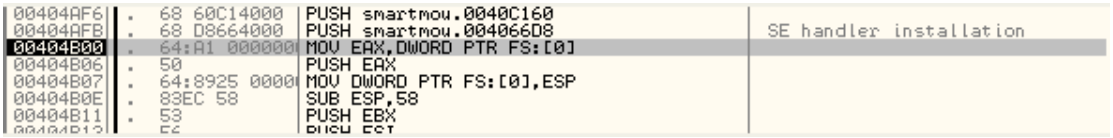


你可以看到程序开始处在安装自己的异常处理程序,OllyBbg 中也以注释标注出来了 SE handler installation。

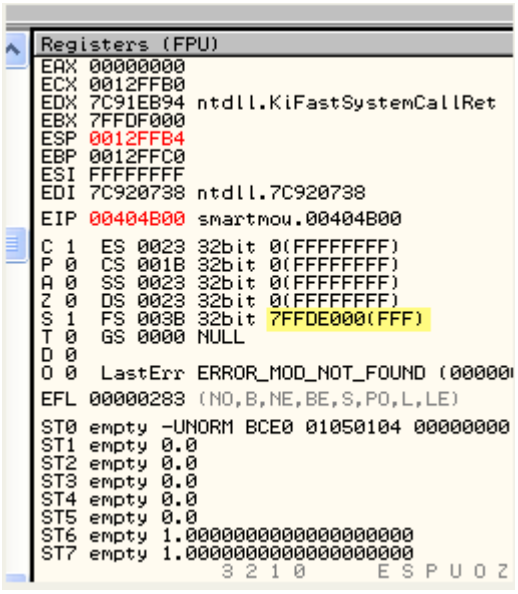


我们到达 OllyDbg 提示 SE handler installation(安装异常处理程序)指令处,首先是一个 PUSH 4066D8 指令,当程序发生异常时,将会调用 4066D8 地址处的异常处理程序。

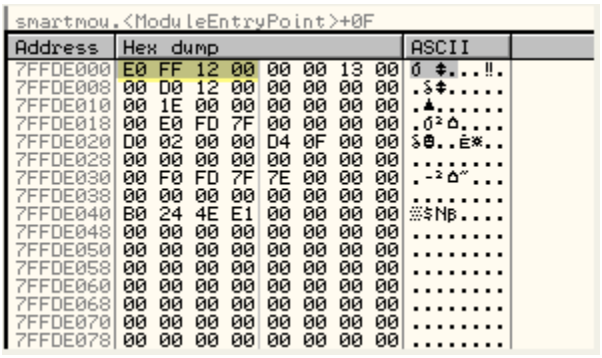
执行 PUSH 指令后,4066D8 被保存到堆栈中了。



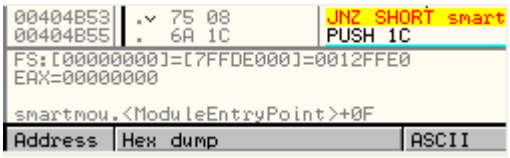
接下来一行是将 FS:[0]的值保存到 EAX 中,我们在数据窗口中来看看 FS:[0]指向的内存单元中保存的内容是多少。



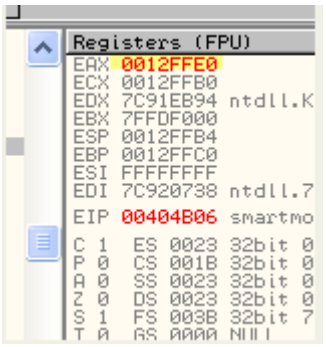
我们在数据窗口中定位到 7FFDE000 地址处。



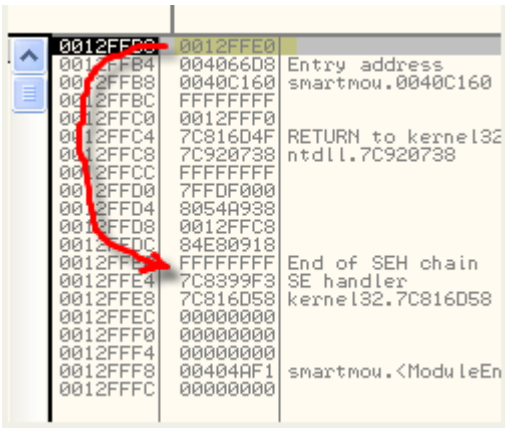
我们可以看到 FS:[0]内存单元中的值为 12FFE0。OD 提示窗口中也显示了。



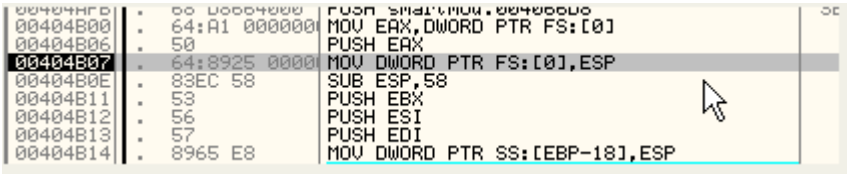
我们执行这条指令。



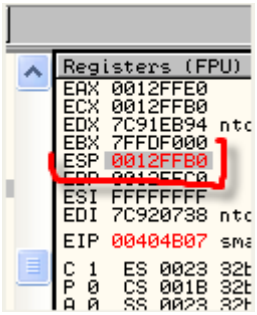
EAX 的值变为了 12FFE0,接着这个值被压入堆栈。

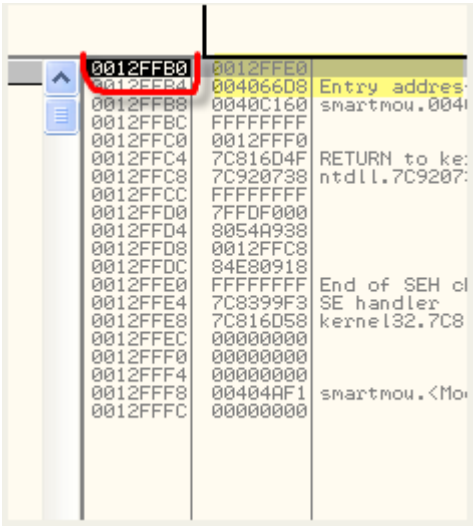


我们可以看到之所以叫 SEH 链,因为它是一个链表，这里的 12FFE0 就指向了上一个异常处理程序。

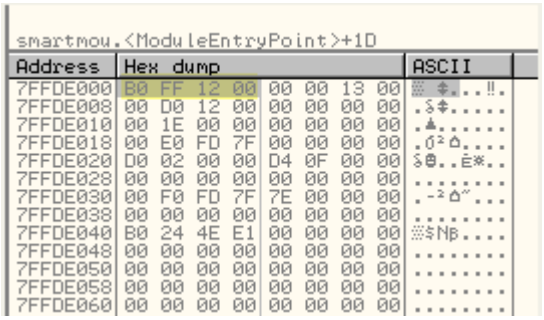


下面一行指令就是将 FS:[0]的内容设置为 ESP 的内容,这样一个异常处理程序就被安装好了。

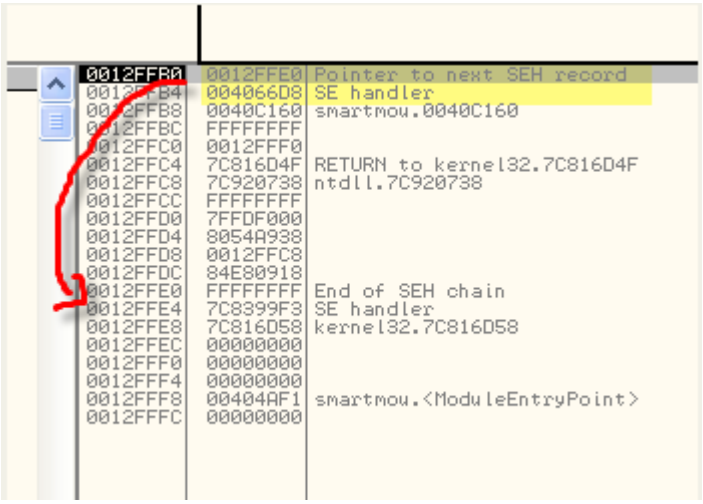




我们执行这一行指令。

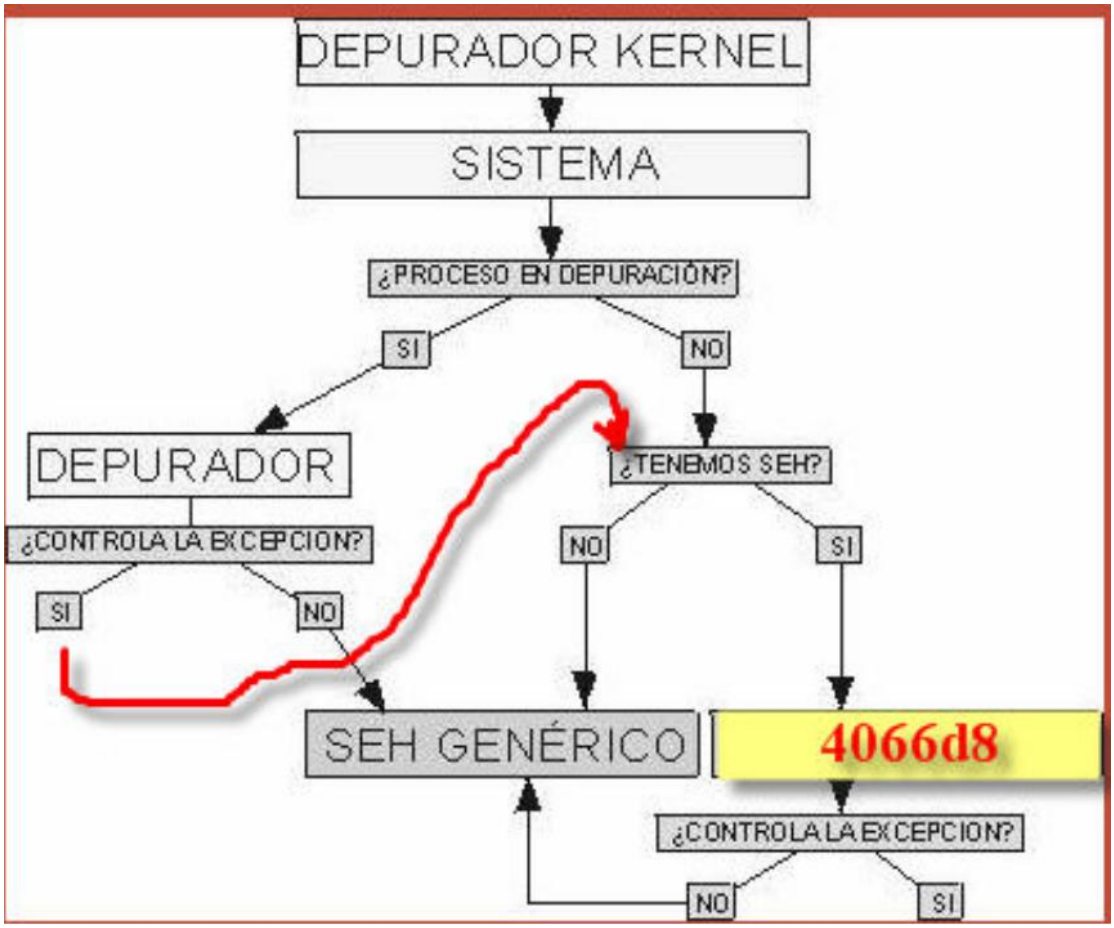


这样 12FFB0 处就是我们安装的 新的 SEH 结点了,也就是 FS:[0]指向了我们新安装的 SEH 结点。



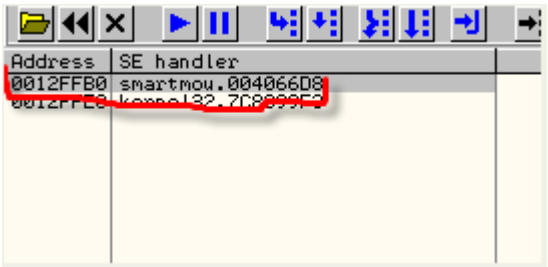
OllyDbg 也标注出来了,提示这是一个 SEH 结点,首先的 4 个字节的值指向了老的 SEH 结点,接下来的 4 个字节值即当前的异常处理程序入口地址。

所以当该程序发生异常后,异常被处理流程如下:

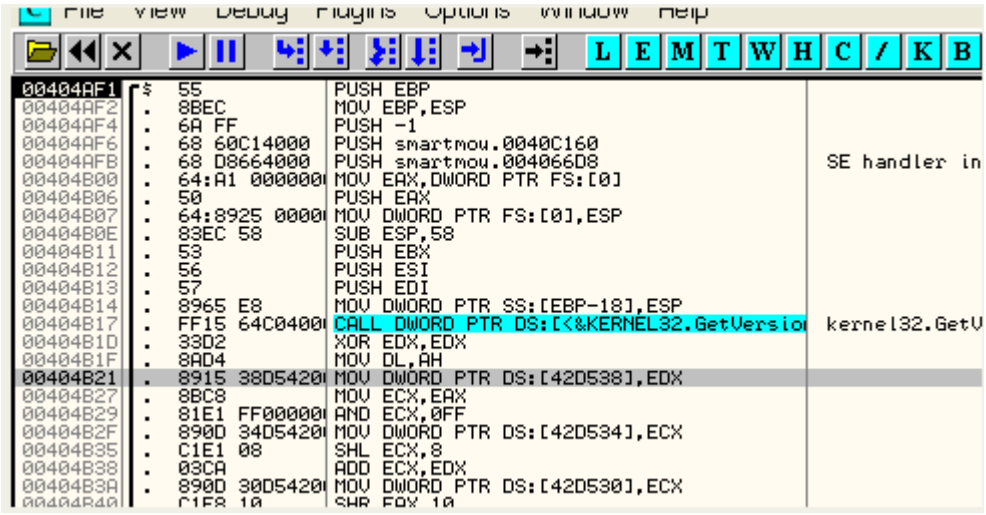


判断当前是否被调试,由于当前正在被调试,所以系统将控制权交予调试器,然后如果你勾选了忽略对应异常的选项的话(如果你没有勾选忽略对应异常选项的话,你也可以按 Shift + F9 键来忽略异常),那么控制权将重新交予程序,如图所示,接着判断是否安装了 SEH,这里安装了,所以将会执行 4066D8 处的异常处理程序。

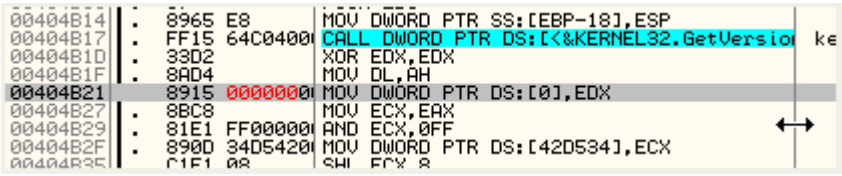
我们来看看 SEH 链的情况:



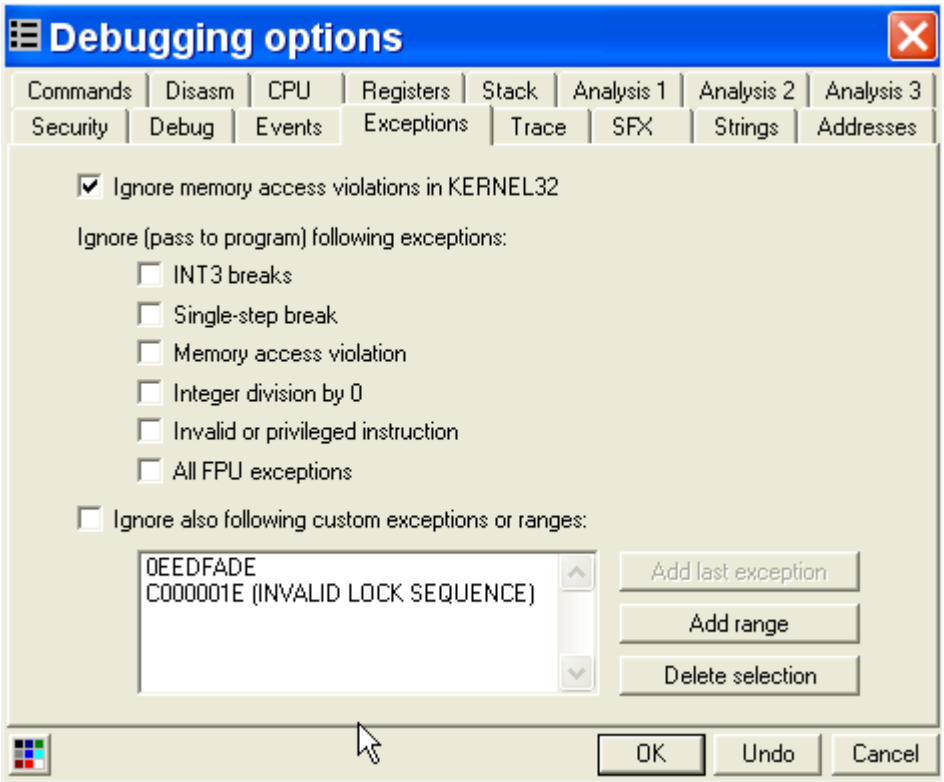
我们可以看到 SEH 链的顶部是程序自己安装的异常处理程序,接下来才是系统默认的异常处理程序,如果发生异常,应该是调用 4066D8 处异常处理程序,而并不是调用系统默认的异常处理程序,我们来手工制造一个异常试试。



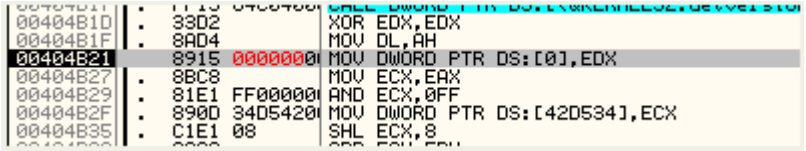
我们来将该行修改为如下指令:



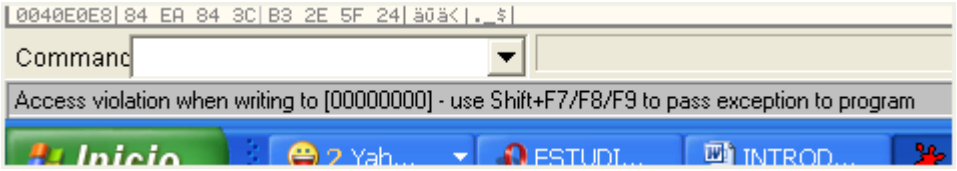
这样会产生一个异常,因为 0 地址不能写入。我们确保调试选项中忽略各类异常的选项没有被勾选,但是第一个选项还是要勾选的。



我们运行起来。

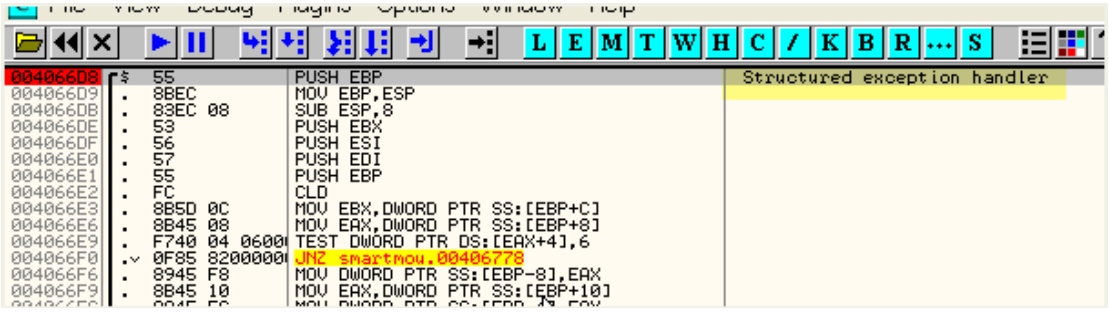


OD 左下方显示错误,程序将被终止。

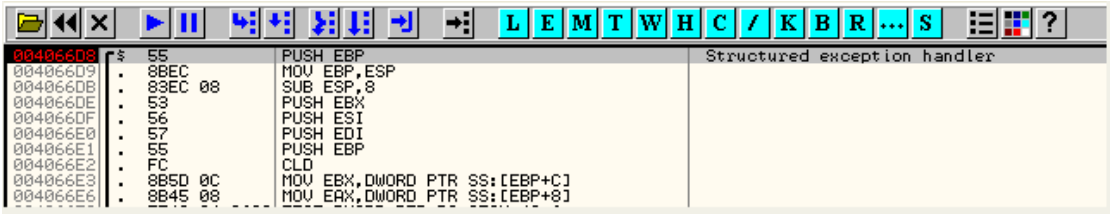


现在程序继续执行的话可以尝试从错误中恢复。

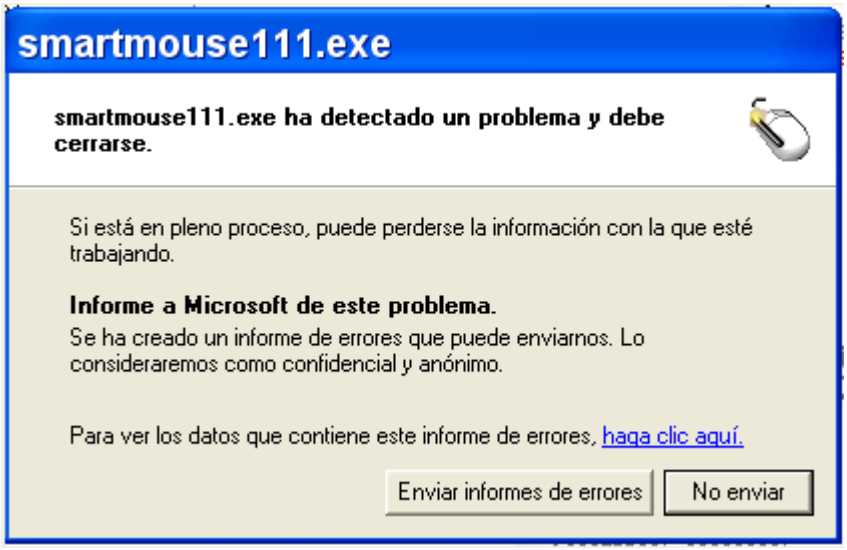
我们在 4066D8 指向的异常处理程序入口处设置一个断点。



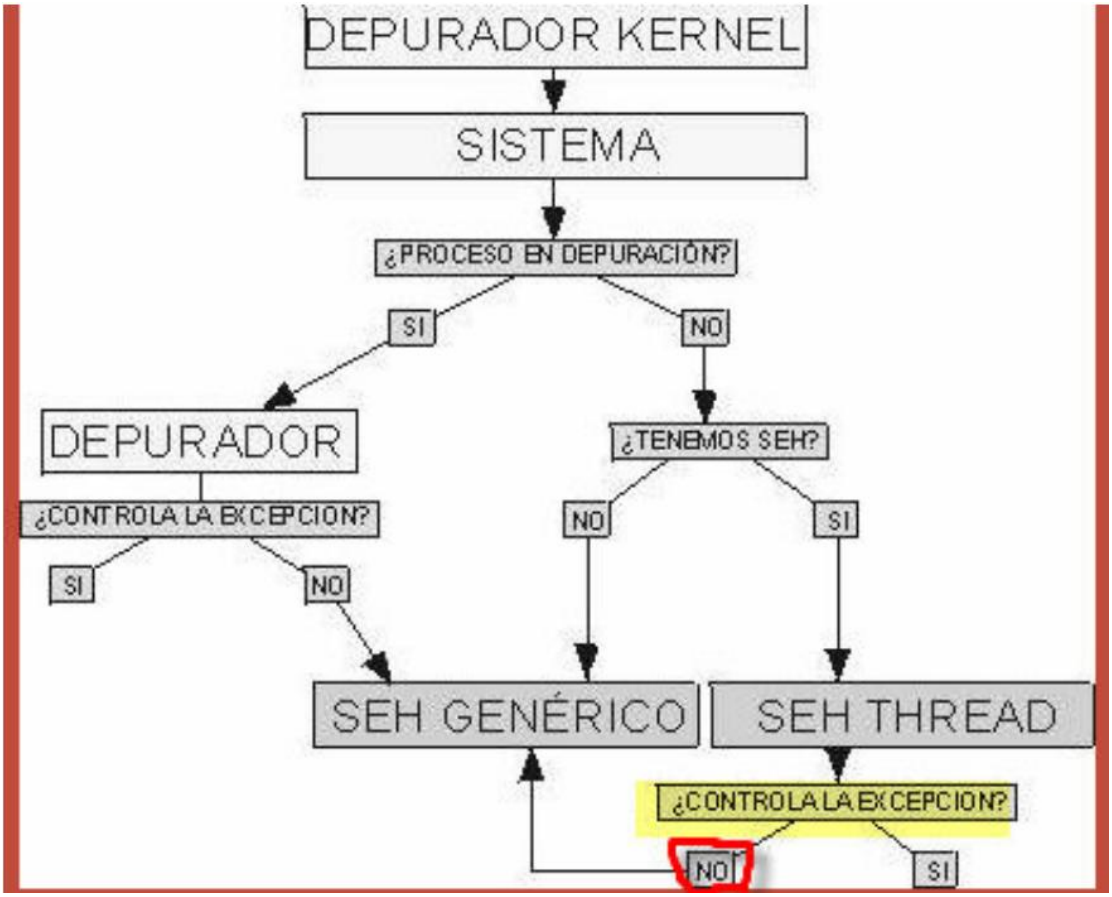
我们可以看到 OD 提示我们,按 Shift + F9 键可以忽略异常,继续执行程序,嘿嘿。



我们可以看到断在了异常处理程序的入口处,我们运行起来看看程序会不会从错误中恢复过来。



我们可以看到程序崩溃,弹出了错误消息框,这该程序表明调用了系统默认的异常处理程序。

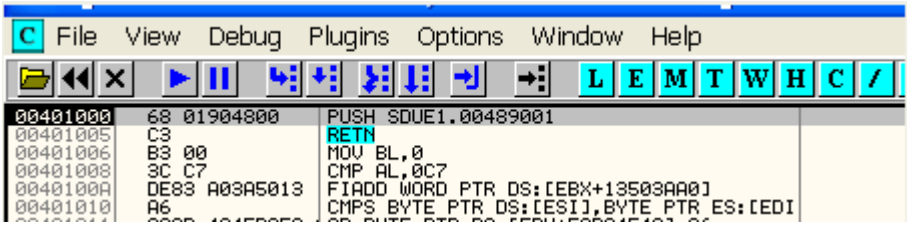


显示这个错误提示框是因为程序自己安装 SEH 异常处理程序中并没有修复刚刚那个异常,所以异常继续传递,最后交予了系统默认的异常处理程序,将弹出了一个错误消息框,程序就终止掉了。

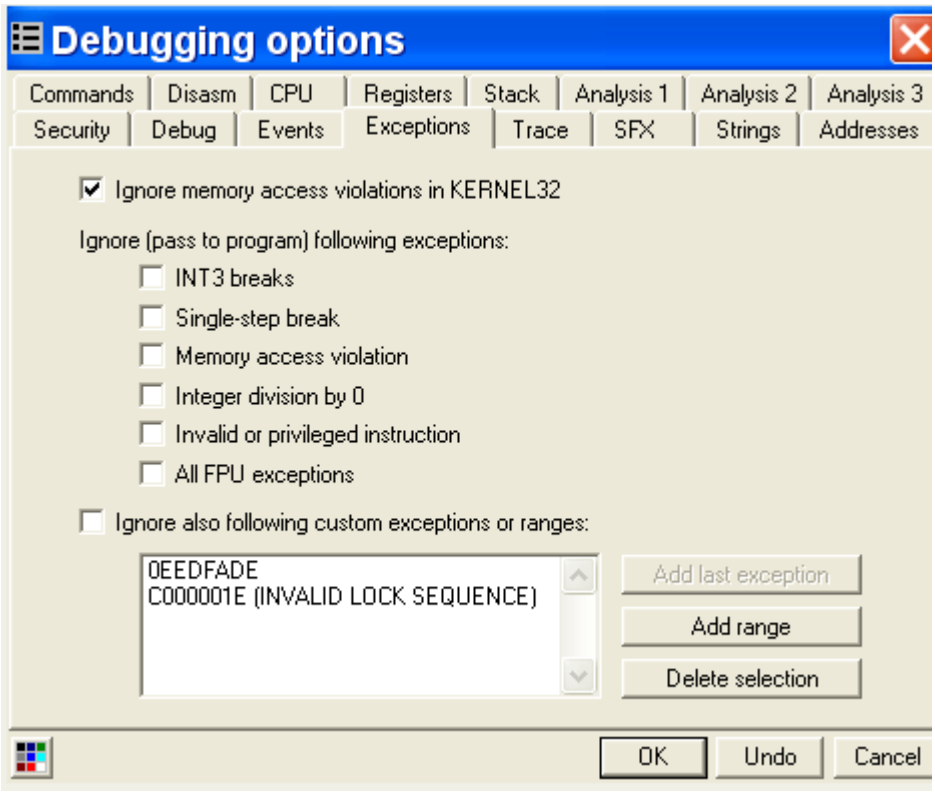
显然,程序自己安装的异常处理程序是用来处理别的类型的异常的,并不能处理向 0 地址处写入导致的异常。

为了能看到异常被成功处理的效果,我们再来看一个例子 SDUE1。

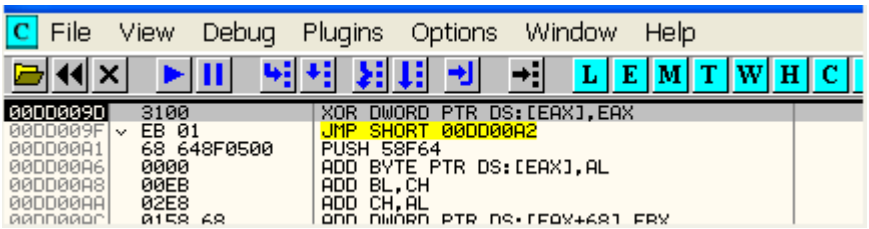
我们用 OD 加载该程序,可以看到 OD 提示说该程序可能被加壳了。



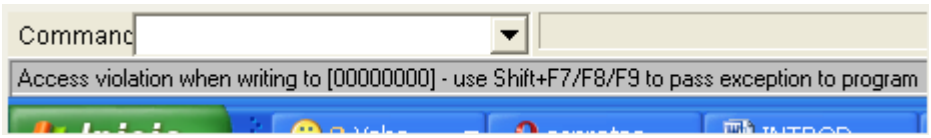
我们依然不勾选调试选项中的忽略各类异常的选项,除了忽略第一个异常以外。



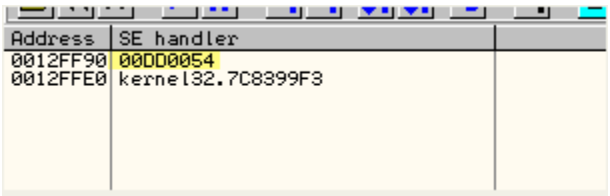
我们运行起来。



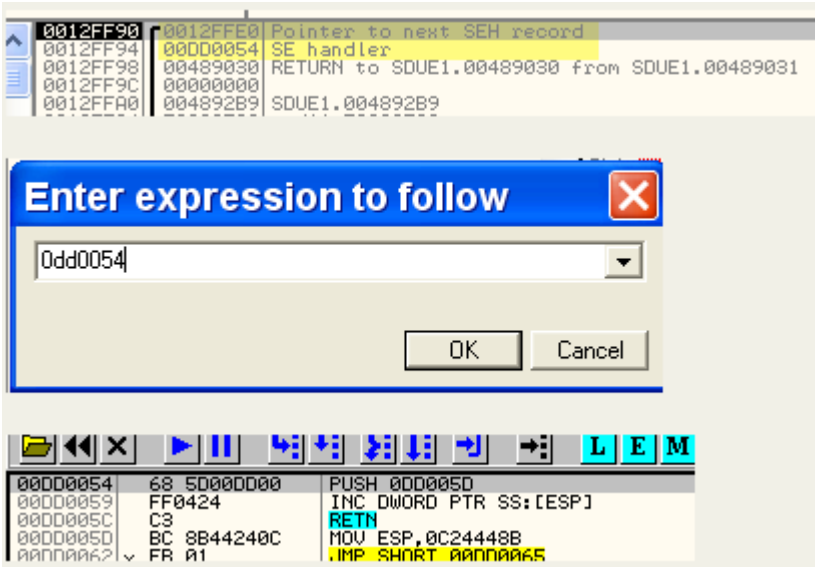
我们可以看到发生了异常,断了下来。



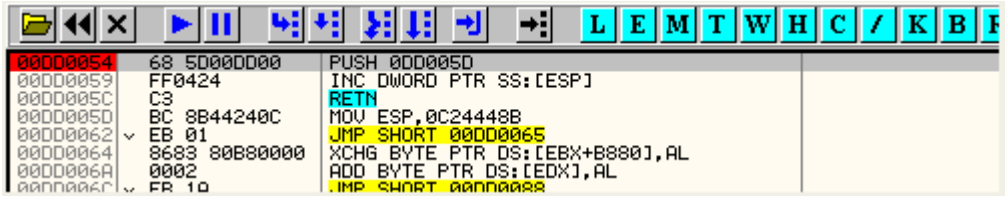
我们来看看异常处理程序在哪里。



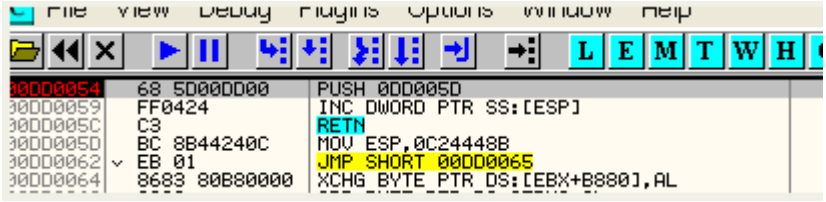
在你的机器上,这个地址可能会不一样,因为该地址属于一个动态创建的区段。我们现在来给该异常处理程序设置一个断点。



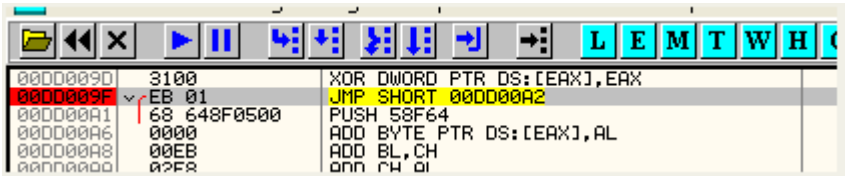
我们定位到了该异常处理程序的入口地址,现在我们该它设置一个断点。



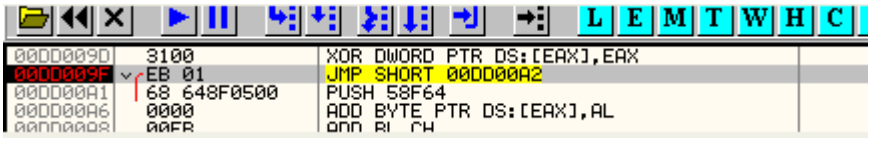
我们按 Shift + F9 键运行起来。



我们可以看到断在了异常处理程序的入口处,如果成功从异常中恢复了的话,那么程序将会从刚刚发生异常的指令的下一条指令处继续往下执行。



我们给产生异常的指令的下一条指令设置一个断点,然后运行起来。



我们可以看到程序继续执行起来了,并弹出提示错误消息框,说明异常已经成功被修复了。

其实设置异常处理程序还可以使用 SetUnhandledExceptionFilter 这个 API 函数,可以通过其参数来设置异常处理程序的入口地址。

好了,本章介绍了我们以后破解过程中会用到的一些知识点,下一章开始我们将介绍 VB 相关的内容。