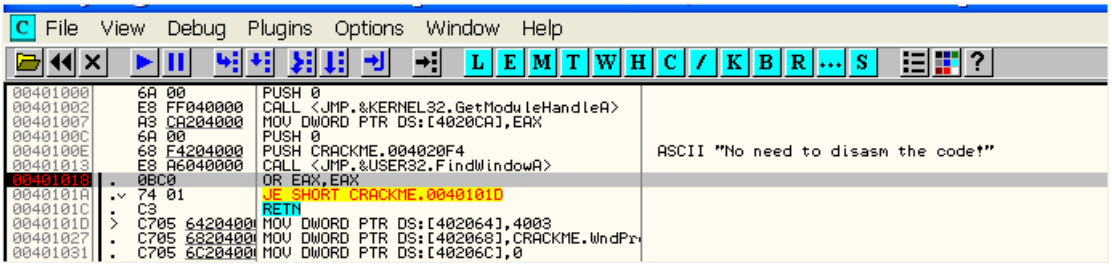
第十章-断点

本章将介绍各种类型的断点。断点可以让你在程序代码执行到合适的时候暂停下来。这次我们的实验的对象还是 CrueHead'a 的 CrackMe。

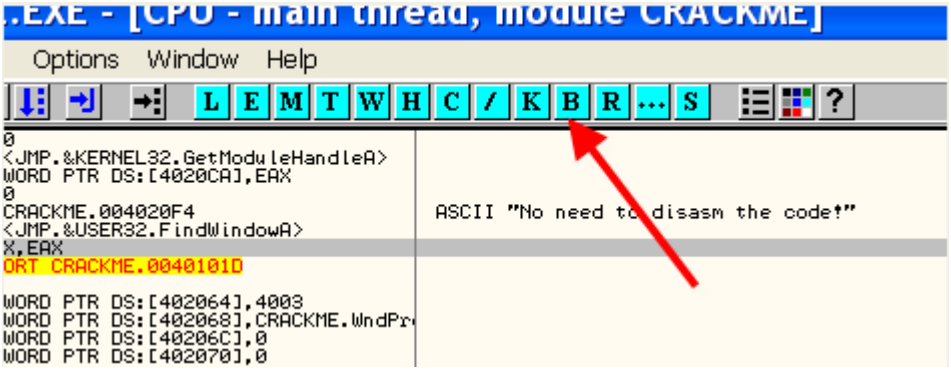
普通断点

这是一个很普通的断点,我们前面章节已经使用了。在 SoftIce 中我们可以使用 BPX 命令来设置断点。OD 中可以使用 BP 命令或者 F2 快捷键来设置断点。也可以再按一次 F2 快捷键来取消断点。

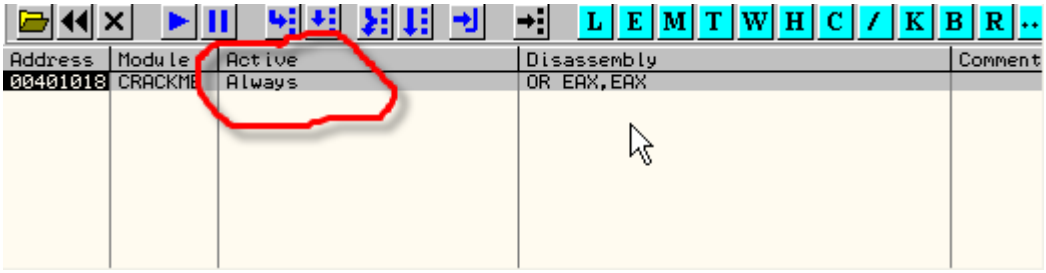
我们来到 CrueHead'a 的 CrackMe 的入口点处。



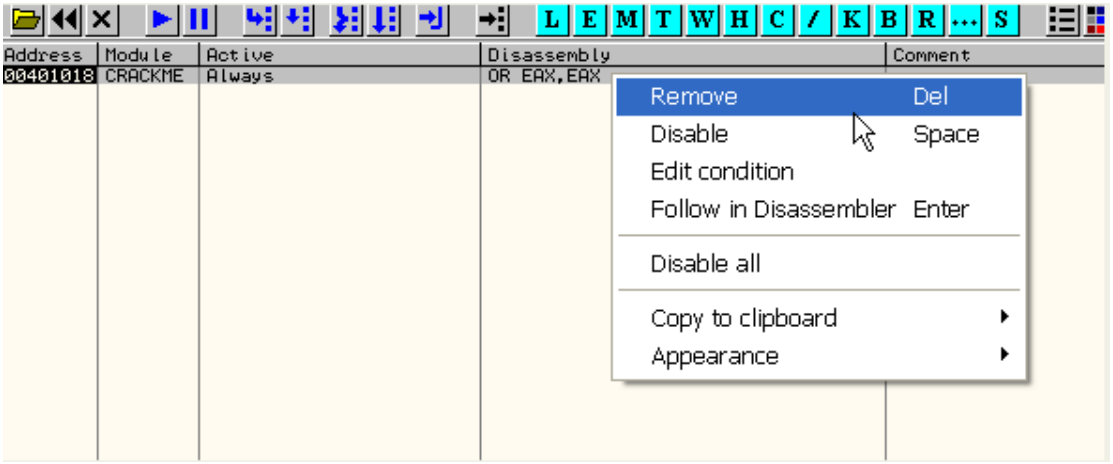
举个例子,拿 401018 这行来说吧,按 F2 键-这行就会以红色突出显示,在该地址处设置的断点就会加入了断点列表中。



我们来看看断点列表刚刚设置的断点,此时该断点是激活状态。



断点列表中 Active 栏显示的是 Always。在该行上单击鼠标右键会弹出一些操作断点的菜单项。



Remove:从列表中删除断点。

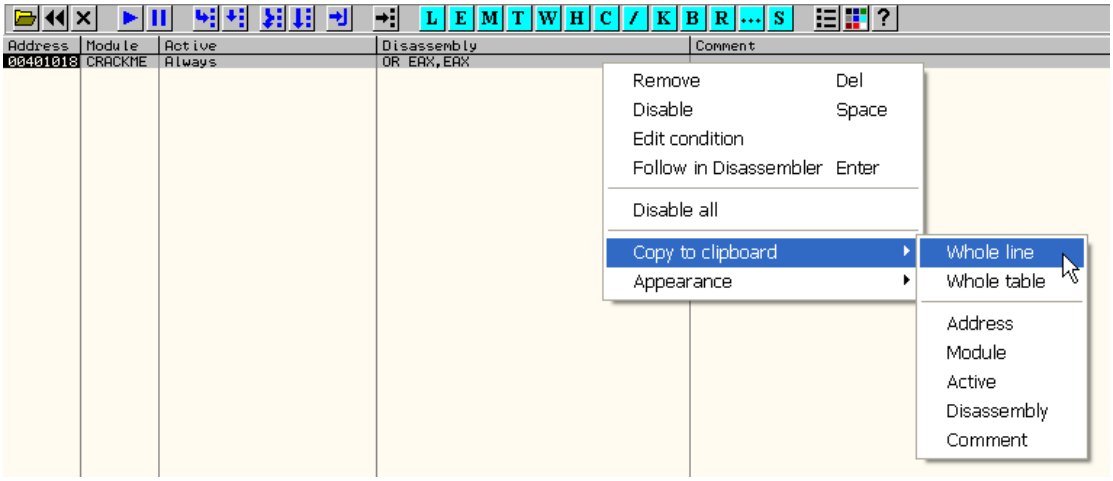
Disable:禁用断点但并不将断点从列表中删除。禁用时,断点并不会触发。

Edit condition:给断点设置触发条件,我们后面再来讨论。

Follow in disassembler:在反汇编窗口中显示断点。

Disable all or enable all:禁用/启用列表中的全部断点。这里没有启用的选项,因为列表中唯一的断点没有被禁用。

Copy to Clipboard:把选中断点的信息复制到剪贴板。我们来实验一下。



我们选择 Whole line 拷贝整行,Whole Table 可以拷贝整个列表的断点信息。

Breakpoints, item 0

Address=00401018

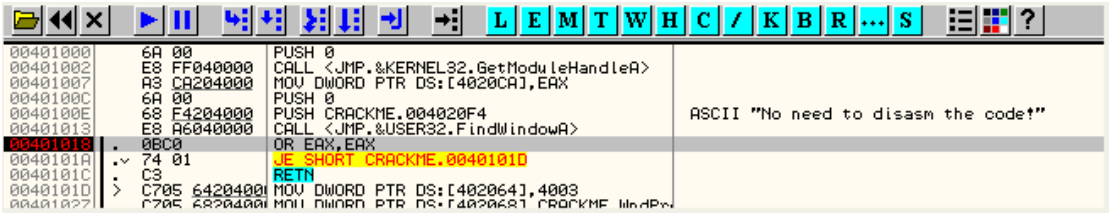
Module=CRACKME

Active=Always

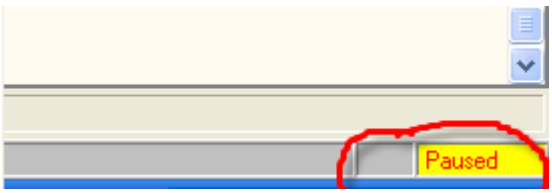
Disassembly=OR EAX,EAX

拷贝下来的信息显示了断点的地址,对应的指令以及激活状态。

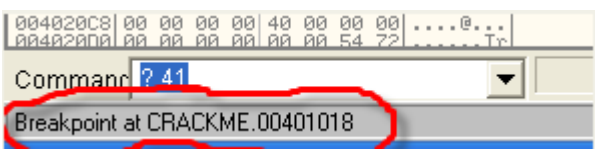
按下 F9 键-CrackMe 运行起来了。然后正如猜想的一样中断了下来。



在状态栏显示暂停状态。

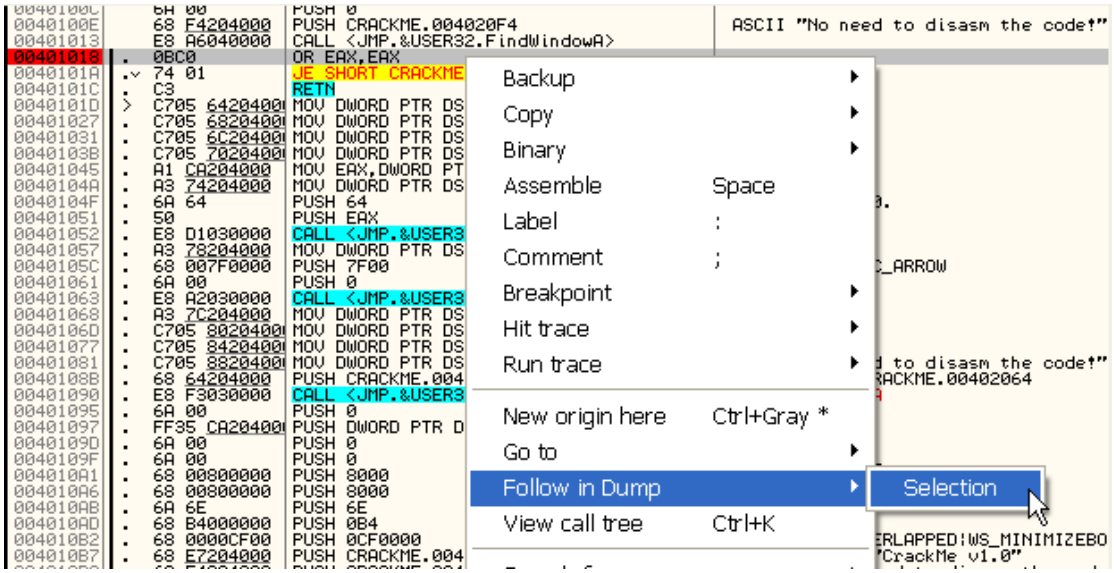


暂停的原因如下:

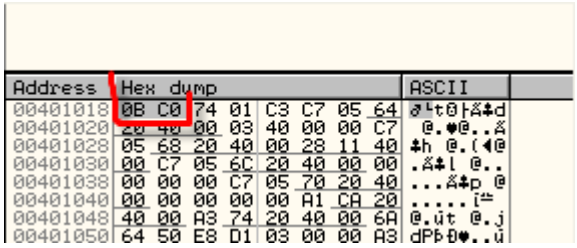


我们来了解一下当设置一个断点以后,二进制代码会发生什么变化。

单击鼠标右键选择-Follow in Dump-Selection



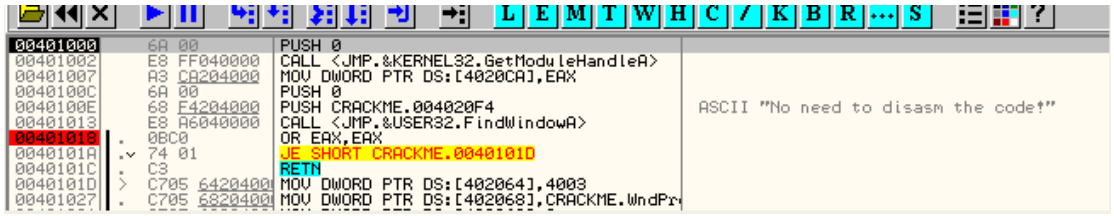
我们看看数据窗口中 401018 地址处的内容:



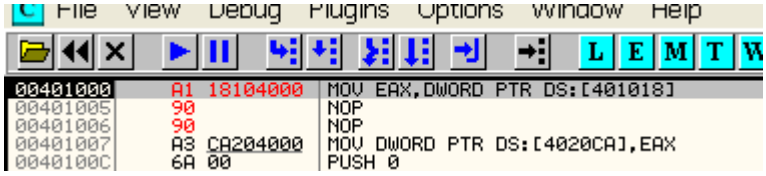
我们初看一下数据窗口中的内容和反汇编代码中代码是一样的:

数据窗口和反汇编代码中我们看到的都是 0B C0,对应的是 OR EAX,EAX。似乎代码没有什么变化,但是真的没有变化吗?

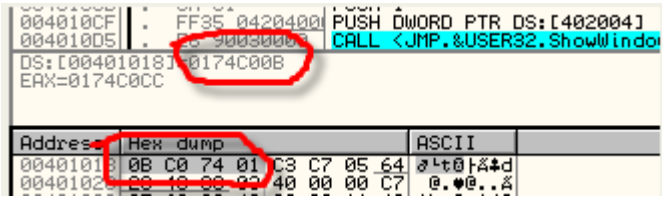
我们保留 401018 处的断点,重新加载 CrackMe。



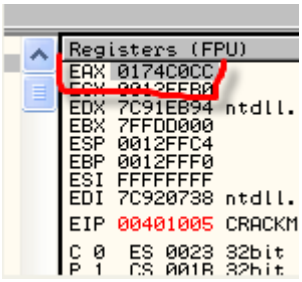
我们将 OR EAX,EAX 对应的机器代码读取出来然后写到别处。



该指令将 401018 地址处的双字值保存到 EAX 中,我们看看 OD 的提示框中提示的信息。



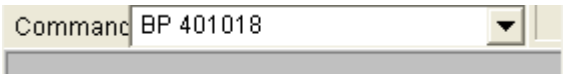
在数据窗口中和提示框中显示的都是相同的内容:0B C0。但是,我们按下 F7 键看看 EAX 显示的内容。



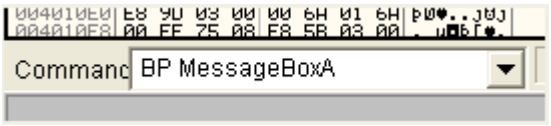
读出来的 401018 处的内容并不是 OD 刚刚显示的 0B C0 74 01(小端存储),按双字取出来是 0174C00B,而现在显示的是 0174C0CC,因此 401018 处字节值是 CC。当我们设置断点后,OD 会将对应指令处第一个字节指令替换成 CC。但是为了不影响界面显示效果,OD 会将 CC 显示为原字节。但是,我们可以在内存单元中读取出其真实的内容,并且可以在反调试中用此方法来检测断点。所以,我们设置的断点有时候莫名其妙的消失了不要感到奇怪,或许说这是调试器的本身的弱点吧。

除了 F2 设置断点以外,我们还可以通过命令栏来设置断点,如下:

BP 401018



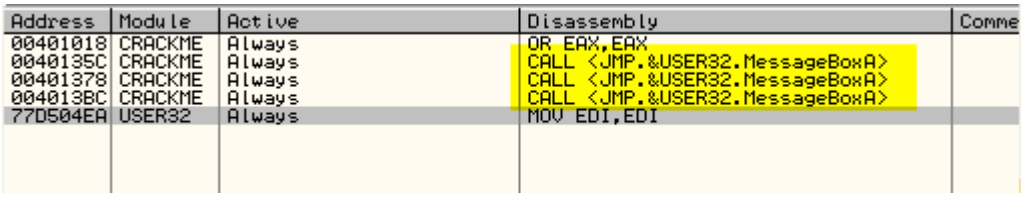
在 NT(2000,XP 和 2003)系统中我们也可以很容易的给 API 函数设置断点-我们前面章节中已经介绍过了。要给 MessageBoxA 设置断点,请输入:



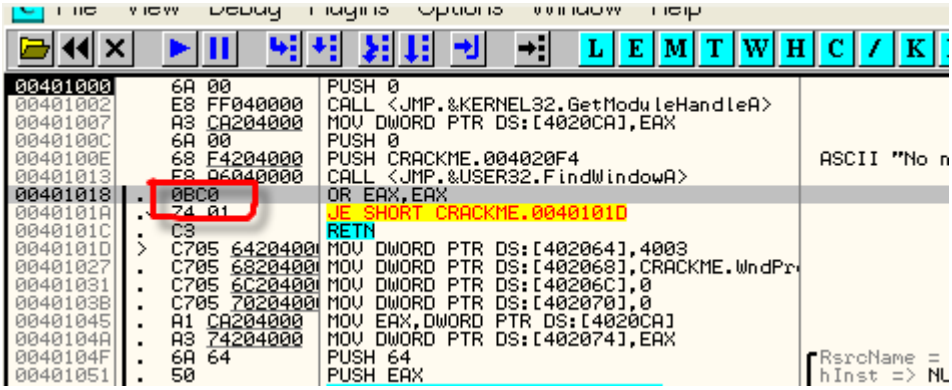
并且你必须指定 API 函数的确切名称,而且大小写敏感。

还有一个比 BP 更加强大的命令 BPX 可以给引用或者调用了指定 API 函数的指令都下断点。

下面是 BPX 给 MessageBoxA 设置的断点列表。正如你所看到的,OD 找到了 3 处地方调用 MessageBoxA,并设置了 3 个断点。



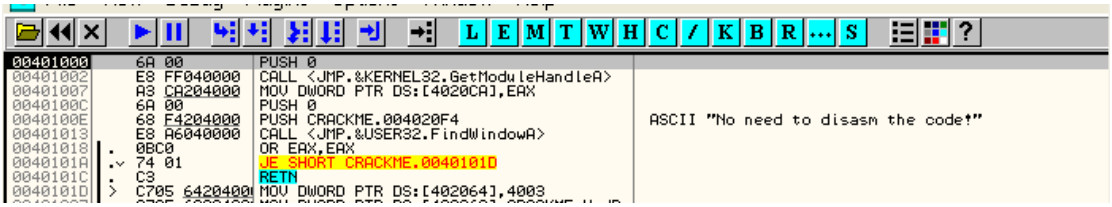
还有一种设置断点的方法:在反汇编窗口中你想设置断点的那一行双击机器码即可。如果想删除的话,再双击一次即可。



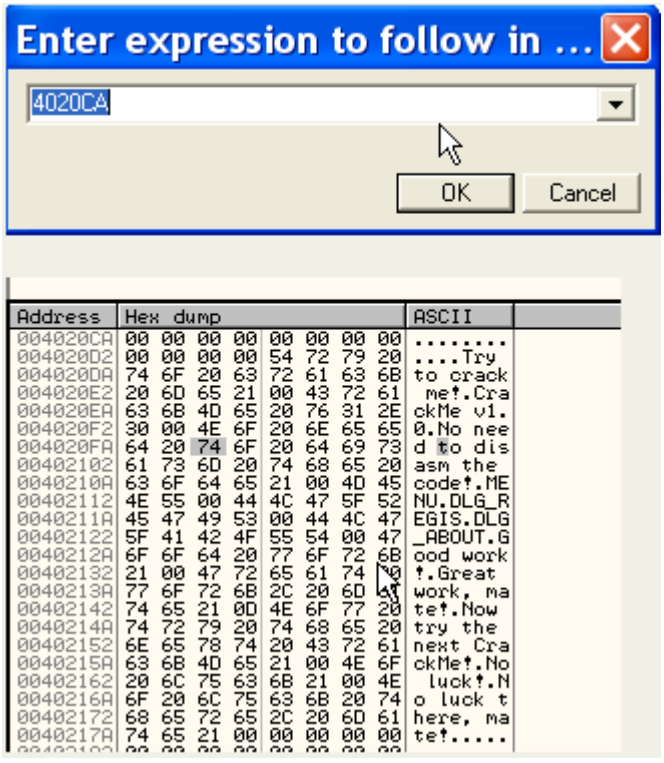
内存断点

内存访问断点有时候也称之为 BPM,但是不要与 SoftIce 中的 BPM 弄混淆了,这二者是完全不同的。

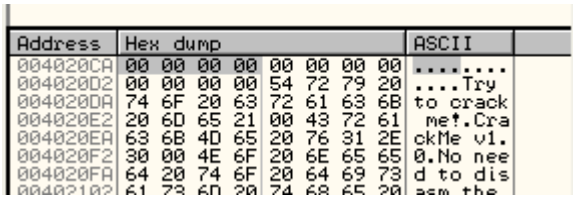
这种类型的断点修改内存页的访问属性。当前我们设置了内存断点。任何代码访问(读,写或者执行代码)了该处代码的话,都会触发异常。我们来看一个例子:



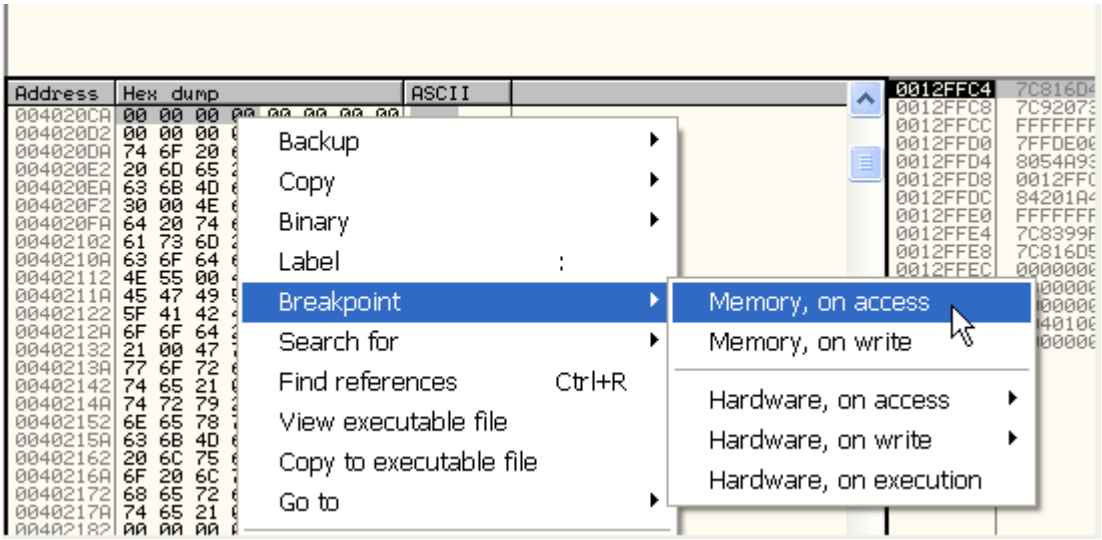
现在我们在 CrackMe 的入口点处，我们尝试设置一个内存断点。单击鼠标右键选择-Goto-Expression 输入 4020CA,转到这个地址。



我们在 4020CA 处设置 4 个字节的内存断点。当前有指令尝试读取这几个字节的时候,就会中断下来。

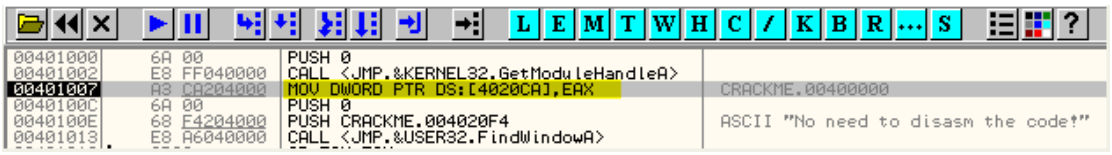


我们这 4 字节上单击鼠标右键选择-Breakpoint-Memory,on access,这里不一定要设置的 4 个字节,你也可以设置长一点,也可以设置短一点。

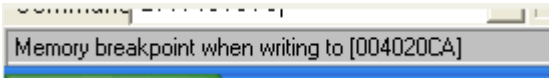


内存访问断点有两个缺点:1.它们不会出现[B]断点列表中和其他的地址。所以,你必须记得设置在什么地址处。2.不能同时设置多个内存断点。如果你设置了一个那么你之前设置的就会被自动删除。

当运行到 401007 地址处的时候,该地址处指令试图写入内容到 4020CA 内存单元中。

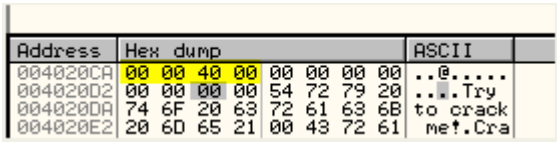


下面状态栏清楚了描述了暂停的原因:

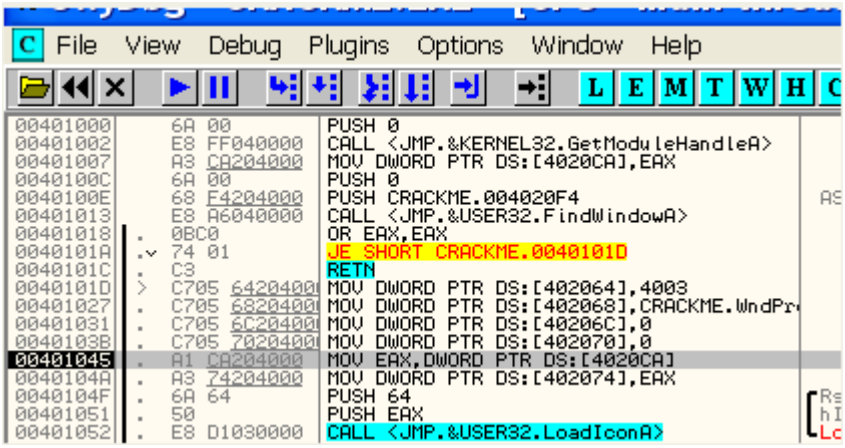


当指令尝试将 EAX 的值写入 4020CA 内存单元的时候,OD 会断下来。记住,当我们对指定内存单元没有写权限,尝试写入的时候会触发异常,OD 会拦截到这个异常,并中断下来,我们看到断下来的时候,OD 已经将内存页的访问属性设置正常了。

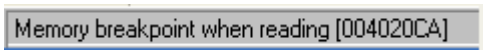
如果此时我们按 F7 键,EAX 的值会被写入到 4020CA 内存单元中,此处的异常不会再次发生。



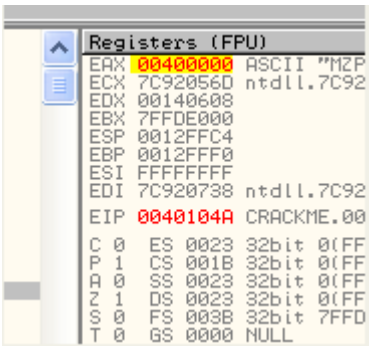
400000h 的值成功写入。我们运行起来,由于内存访问断点仍然存在,如果程序尝试访问 4020CA 内存单元的话,会再次触发异常。



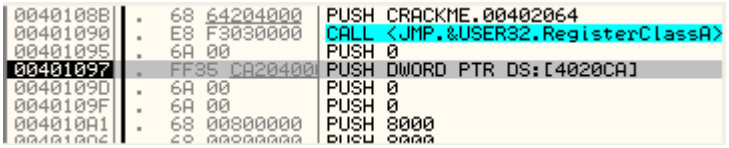
正如你看到的,该指令尝试读取 4020CA 内存单元的内容,证明内存访问断点又触发了。



再次按 F7 键运行一步,将 4020CA 内存单元内容读取出来并保存到 EAX 中。

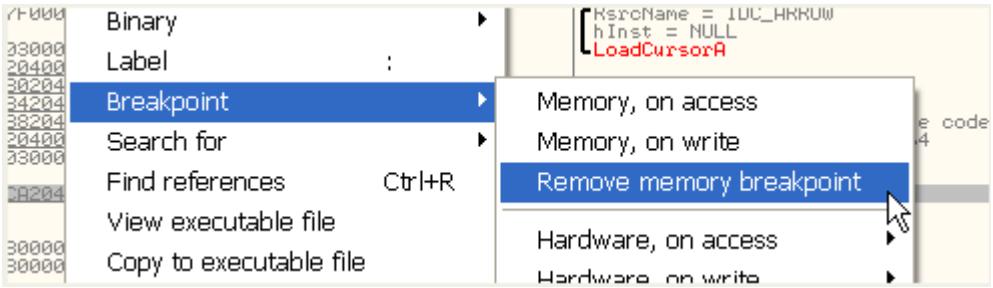


再次 F9 键运行起来,还会不会出发内存断点呢?对,依然会。



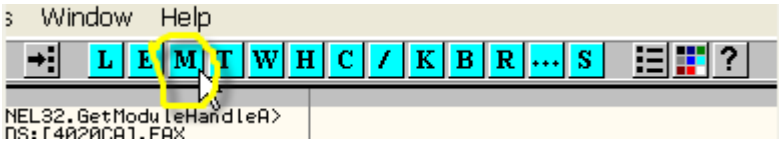
OD 会再次中断在尝试读取 4020CA 内存单元的指令处,这是一个 push 指令,该指令会尝试将 4020CA 内存单元的内容压入堆栈。

要删除内存断点的话,可以数据窗口中单击鼠标右键选择-Breakpoint-Remove memory breakpoint。你还可以设置一个新的内存断点,旧的内存断点会自动被删除。

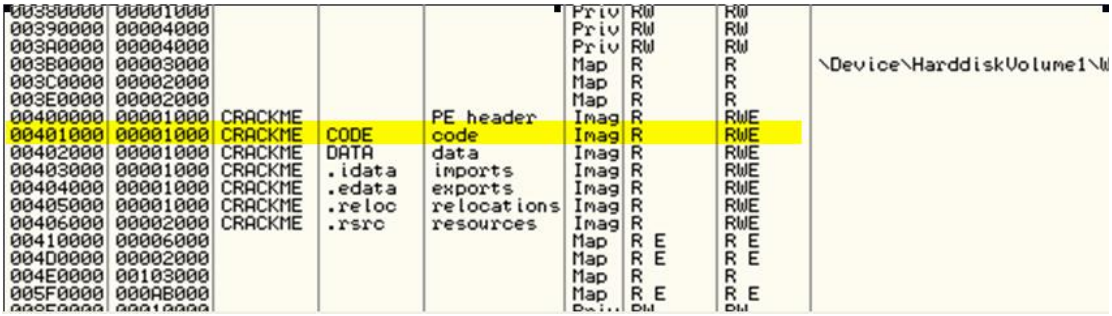


“Memory,on access”是内存访问断点(读或者写),“Memory,on write”是内存写断点。

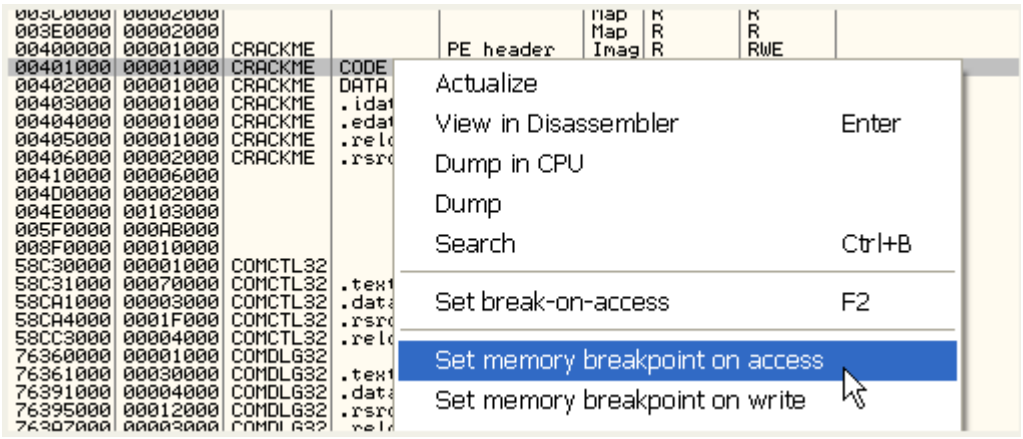
OD 也可以对区段设置内存断点,我们选择菜单项 View-Memory,也可以按工具栏中的[M]按钮打开内存窗口。



这个列表包含了 CrackMe 加载的一些区段以及其加载的一些 DLL 的区段,如图我们选中了以 401000 开头的区段。

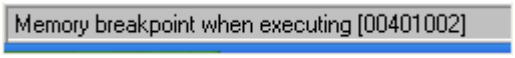


在选中部分上单击鼠标右键选择-Set Memory breakpoint on access。

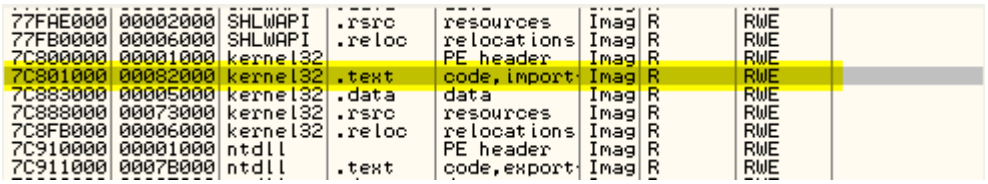


下面还有 Set memory breakpoint on write 的选项,但是这里我们选择 Set memory breakpoint on access 并且运行起来。

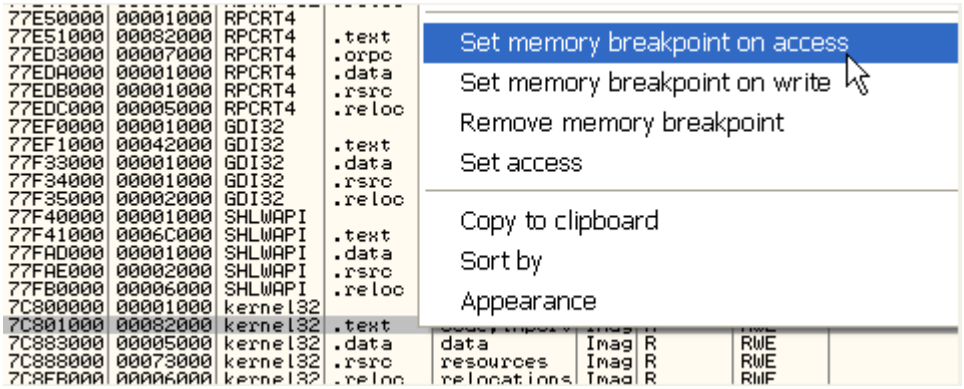
中断在了下一行指令上。



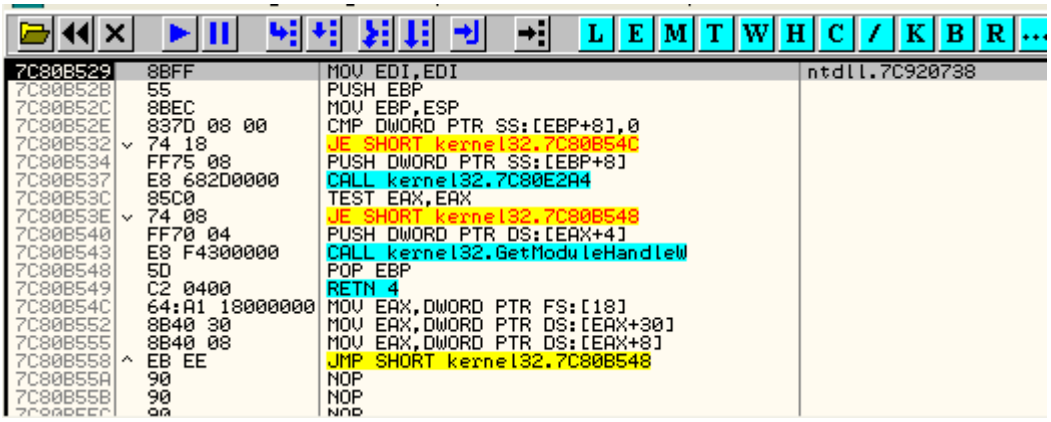
因为试图执行 401002 处的代码,而当前代码段设置了内存访问断点,尝试执行当前代码段的任何一条指令都会触发内存访问断点。



我们对 kernel32 的代码段设置内存访问断点,列表的下面可以找到。



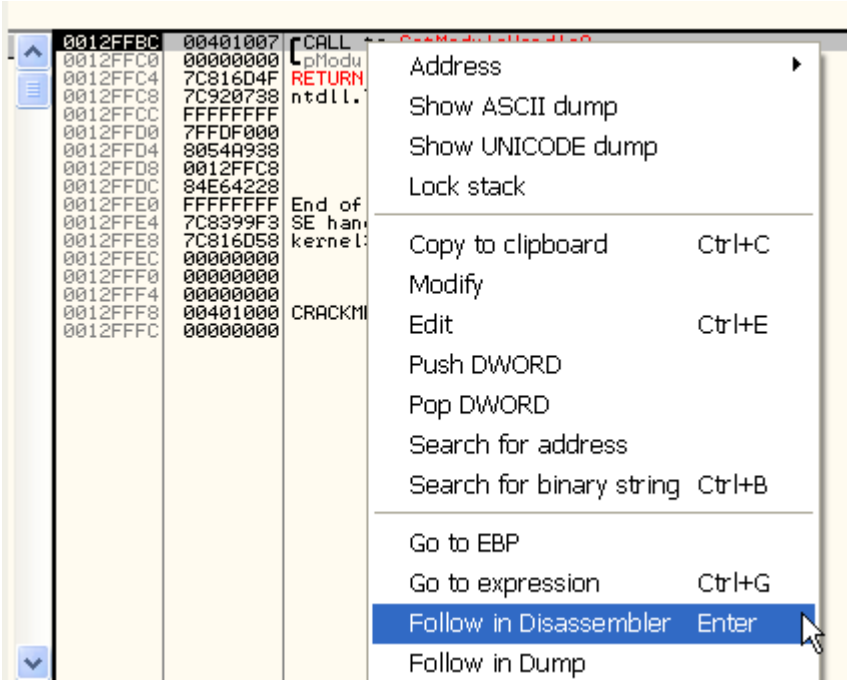
我们选择-Set memory breakpoint on access,当试图读取/写入/执行 Kernel32 代码段的时候就会中断下来，我们运行起来。



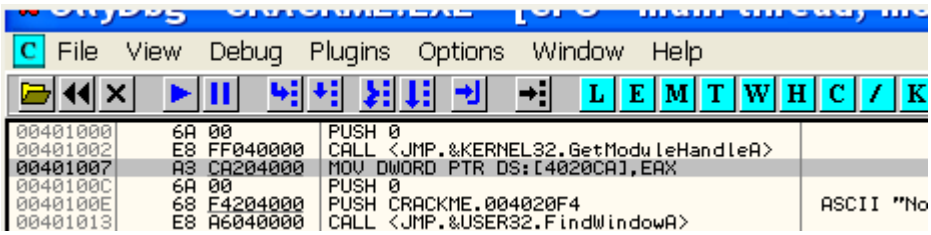
从堆栈的顶部来看,调用 API 函数的时候中断下来了。

当前执行 kernel32.dll 中的第一个 API 函数的时候中断下来了,从堆栈中我们可以看到返回地址:

在返回地址上单击鼠标右键选择-Follow in Disassembler。



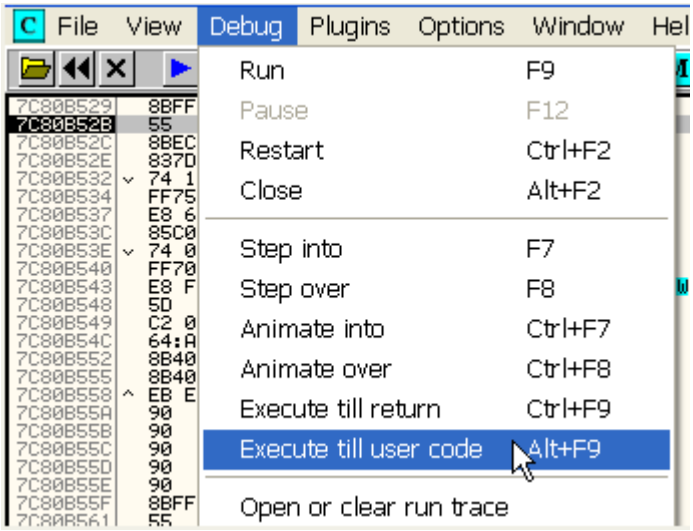
返回地址是 401007,是调用 GetModuleHandleA 后面的一行。



正下方是调用 FindWindowA,调用这个函数的时候不会触发内存访问断点,应该这个函数是属于另一个 DLL-user32.dll 的。如果你再次运行的话,会中断在下一条指令处,因为下一条指令也属于 kernel32 的代码段,

所以执行 GetModuleHandleA 的每一条指令的时候内存访问断点都会触发,所以这个时候我们可以选择-Remove memory breakpoint 删除内存断点。

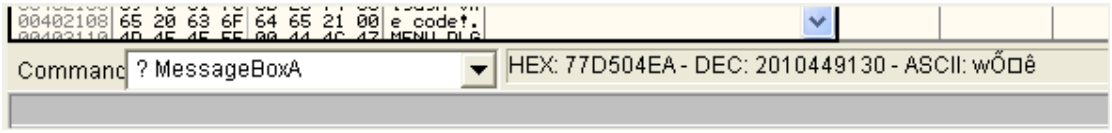
如果你想要返回到主程序模块中,可以选择主菜单项-Debug-Execute till user code。有的时候,这个方法不起作用。我们可以在函数返回指令处设置断点,然后运行起来,等中断下来以后,再 F7 或者 F8 单步到主程序的代码中。



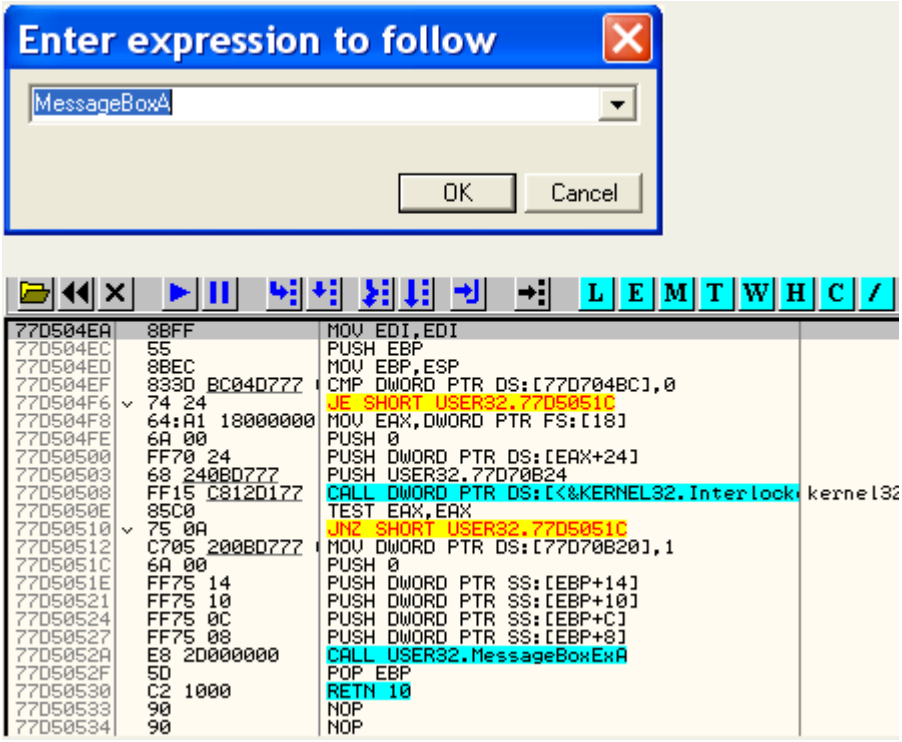
这里 Execute till user code 起作用了,我们回到了主程序模块的代码中。后面我们会介绍哪些情况下这种方式不奏效。

现在,你可以再次对 kernel32.dll 的代码段设置内存访问断点了,让程序调用 Kernel32.dll 中的 API 函数的时候再次中断下来。

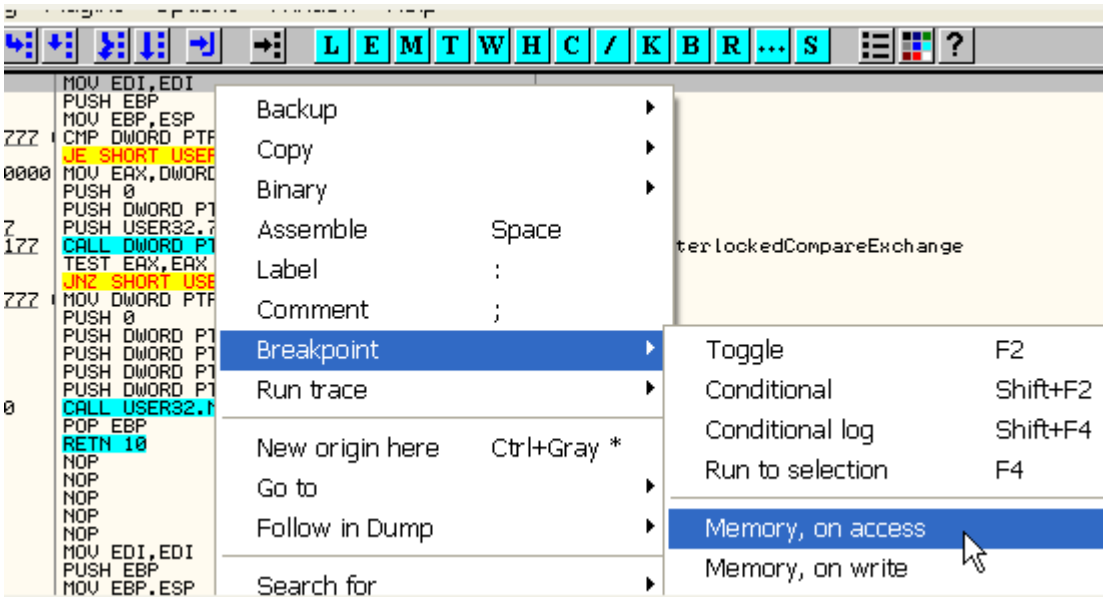
此外,如果程序会检测函数首字节是否为 0xCC 的话,这个时候我们使用 bp MessageBoxA 命令下断点就无效了,这个时候我们可以尝试一下内存访问断点。我们来看个例子:



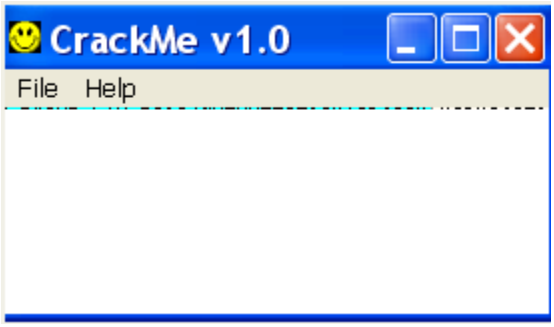
MessageBoxA 在我的机器上的对应的地址是 77D504EA。通过单击鼠标右键选择-Goto-Expression 输入该函数的名称:



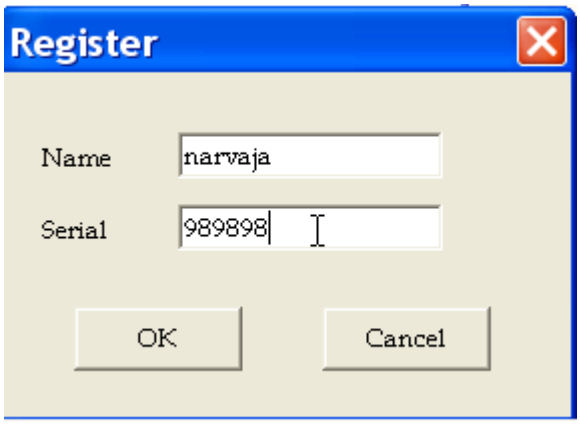
第一条指令被高亮显示:



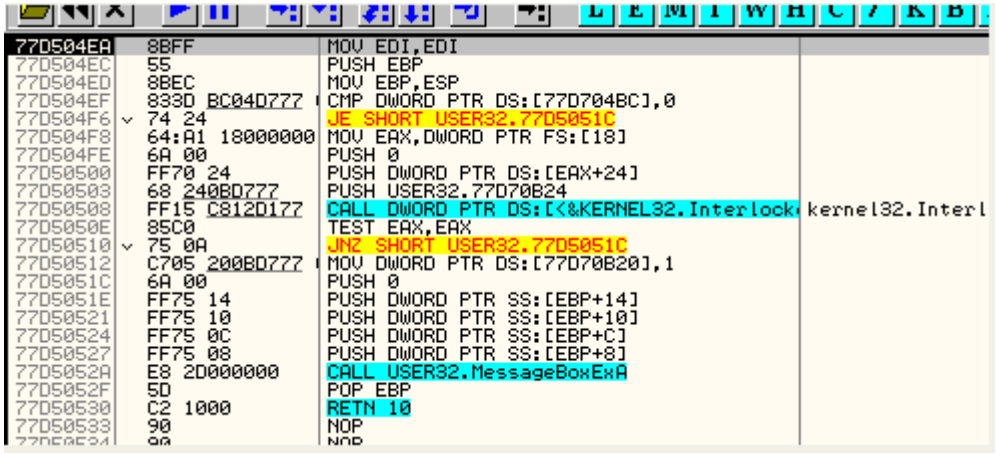
这里我们单击鼠标右键选择-Breakpoint-Memory,on access 或者 memory,on write,这里我选择 Memory,on access 然后运行起来。



选择 help-Register,然后输入任意用户名和序列号:



单击 OK。



在同一个 API 函数中,如果通过 bp 命令设置断点会被程序检测而导致断点失效的话,也许设置内存访问断点可以绕过这个检测。

设置内存访问断点这个方法也可以通过检测内存页的属性并恢复内存页的属性来进行保护,但是这在反调试技巧中并不常见。

下一章,我们将介绍硬件断点和消息断点,条件断点我们会在后面介绍。