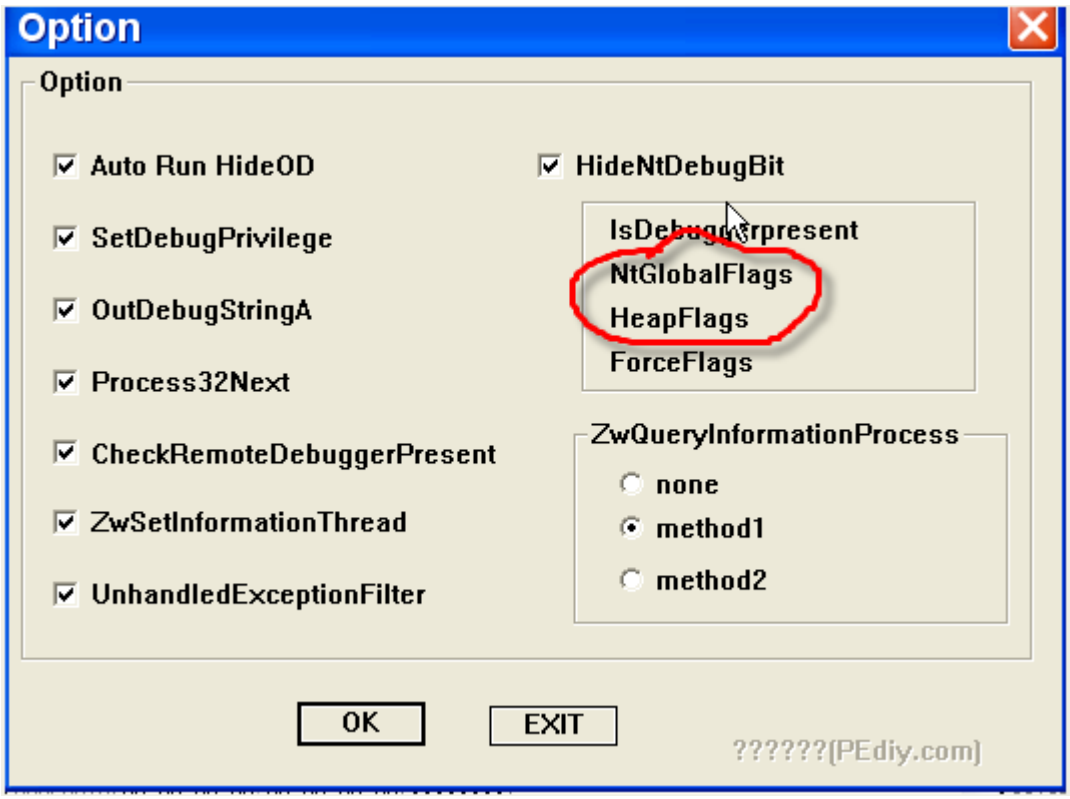
第二十三章-OllyDbg 反调试之 ProcessHeap,NTGlobalFlag,OutputDebugStringA

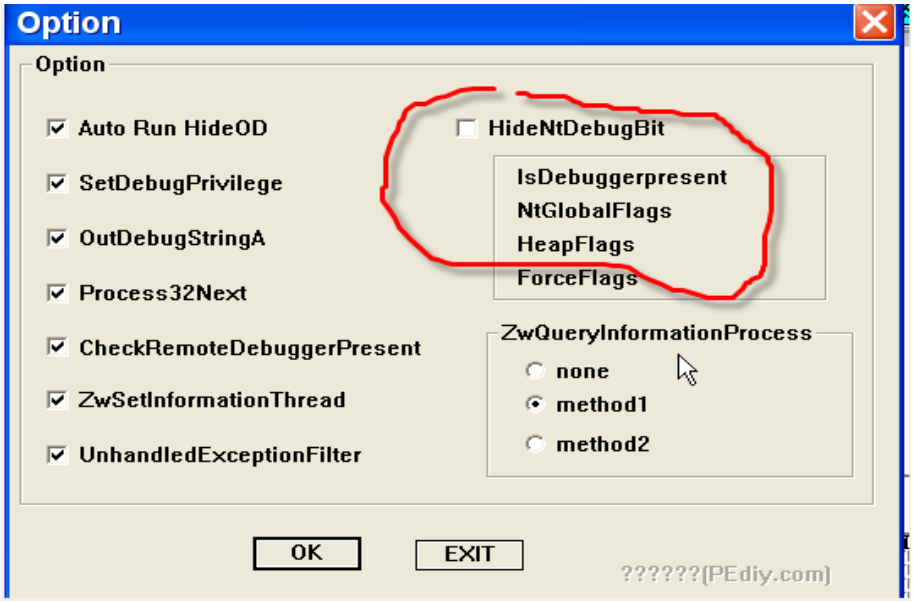
本章是反调试的最后一章,本章将介绍 ProcessHeap 和 NTGlobalFlag 标志位以及如何通过这两个标志位进行反调试,介绍完这部分内容我们就掌握了常见的反调试技巧。反调试的手法还有很多,我们介绍的只是最基本,最常见的。像一些保护壳,比如说 Execryptor 的反调试是比较厉害的,我们后面再介绍。Execryptor 壳除了我们介绍的这几种反调试以外,还有别的反调试手法,它的每个新版本都会增加一些新的反调试选项,这都是后话了,我们先来把常规的反调试手法介绍完。

HideOD 插件中提供绕过这个两个标志检测的选项,我们来看看。



HideOD 插件见附件,上一章也提供了,红圈标注的就是绕过 ProcessHeap 和 NTGlobalFlag 两个标志位的选线,我们首先还是来手工绕过这个两个标志位的检测吧。

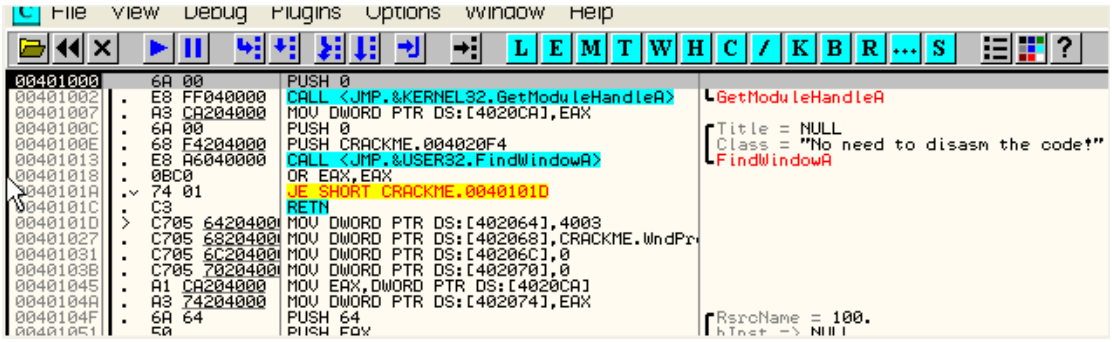
这两个标志置位的话表示当前进程正在被调试,很容易定位到这两个标志位。不知道大家还记得不得 IsDebuggerPresent 对应的那个调试标志位是如何定位的,如果你不记得的话,回头去看看第 19 章,如果你弄明白如何定位那个标志位的话,那么这两个标志位也就好定位了,因为这两个标志位就在那个标志位的附近。



由于勾选上 HideNtDebugBit 选项,就会绕过 ProcessHeap 和 NTGlobalFlag 的检测,这里我们不勾选。

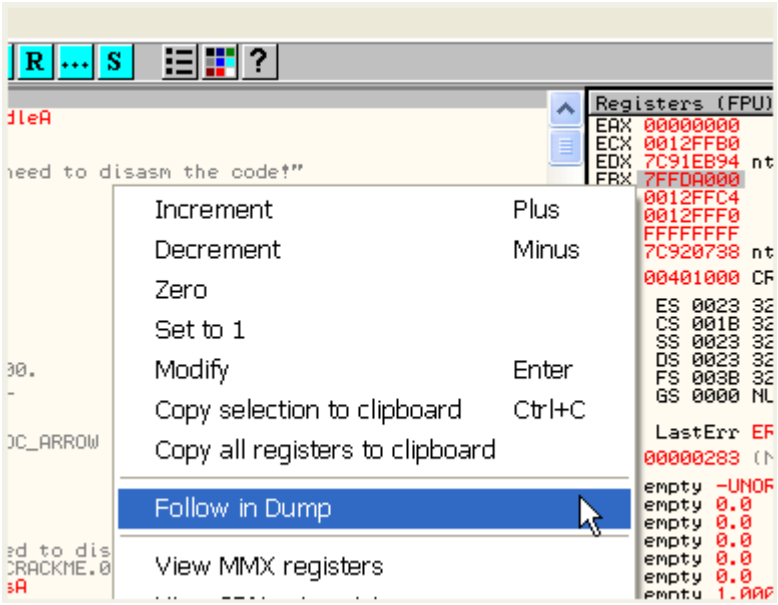
这里我们的实验对象是 Cruehead'a 的 CrackMe,我们首先来定位这两个标志位。

我们用 OD 加载该 CrackMe,并确保 HideOD 插件的配置如上图所示。

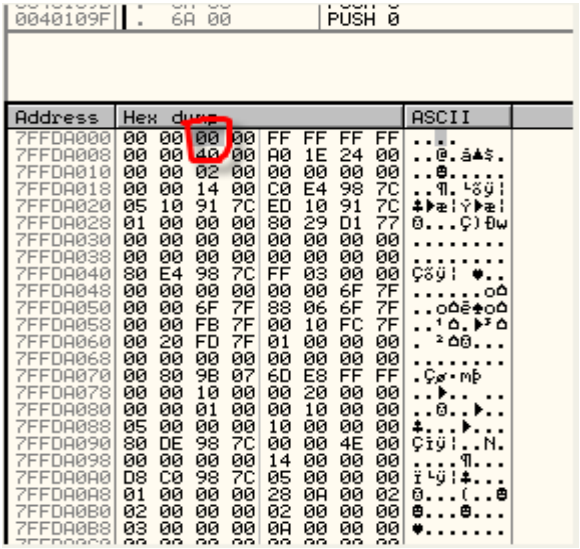


好了,我们现在来看看如何手工定位和修改 ProcessHeap 和 NTGlobalFlag 这两个标志位。

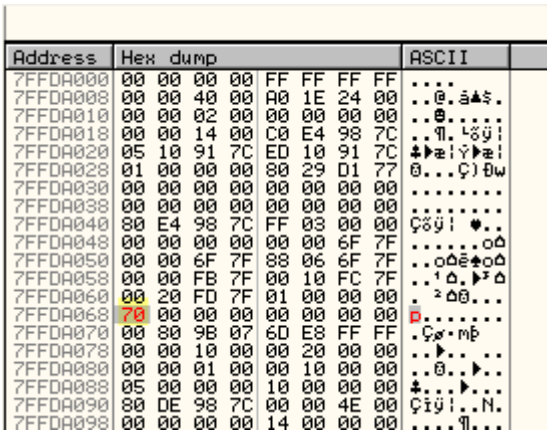
我们先定位到第 19 章介绍过的 IsDebuggerPresent 的那个标志,最简单的做法就是 EIP 在入口点处时找到 EBX 寄存器的值,然后单击鼠标右键选择-Follow in Dump。完整的定位流程你可以回头看第 19 章。



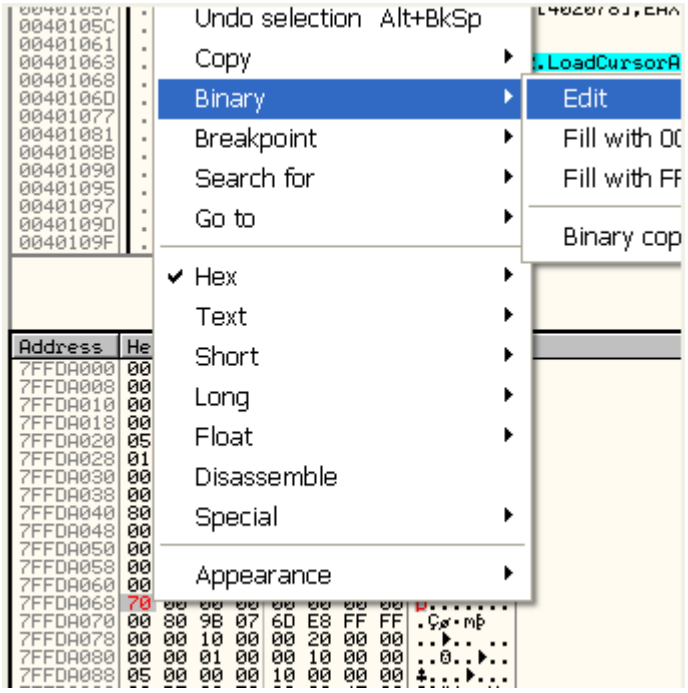
我们在数据窗口中定位到该标志位,我机器上的这个地址可能与你的不同,而且,该程序每次重新启动该地址也可能不同。



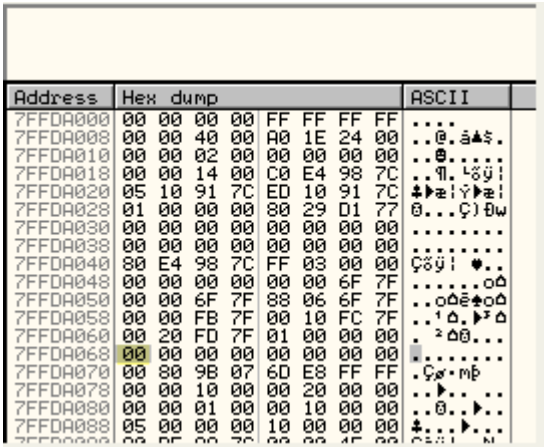
我们知道 IsDebuggerPresent 是获取该标志位来检测是否被调试的,NTGlobalFlag 就在它的隔壁,嘿嘿,我们只需要将 EBX 的值加上 0x68 就可以定位 NTGlobalFlag 标志位,当前,EBX 的值为 7FFDA000,加上 0x68 等于 7FFDA068。



这就是 NTGlobalFlag 标志位,当前不为零表示正在被调试,我们来手工将其修改为零。



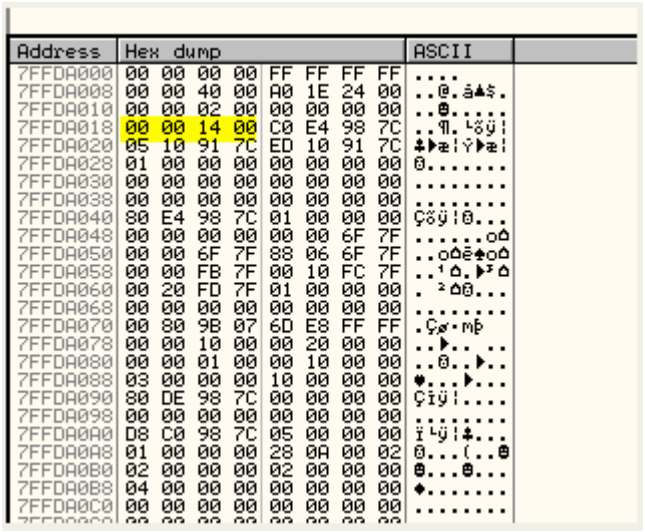
这里把该标志修改为零了。



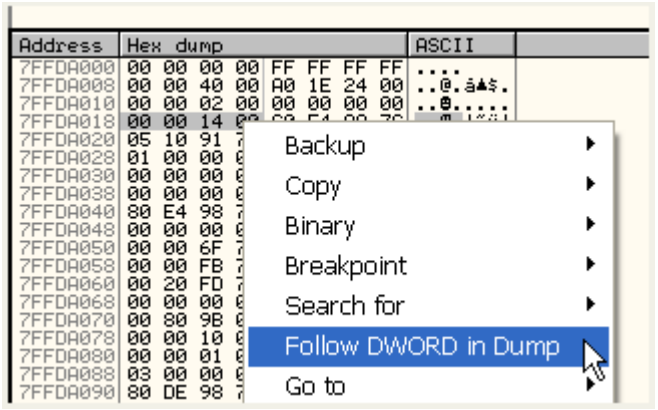
我们可以看到 NtGlobalFlag 标志清零了。

现在我们来定位另一个标志 ProcessHeap,也很容易定位。

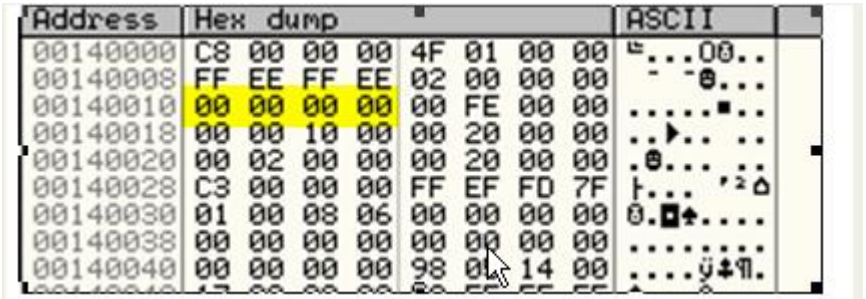
同样是 EIP 在入口点处时定位到 EBX 的值,然后将 EBX 的值加上 0x18,我机器上 ProcessHeap 的值为 0x140000,这是程序刚启动的时候创建的一块堆内存空间,该内存是用来保存一些重要的数据的,好了,我们知道这些就够了。



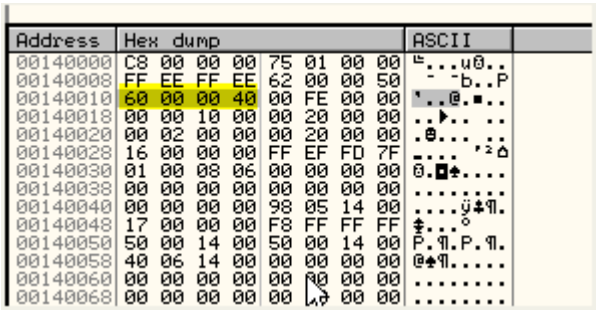
我们来看看堆中保存了些什么。



我们选中这 4 个字节,单击鼠标右键选择-Follow DWORD in Dump 就可以在数据窗口中定位该堆空间了。

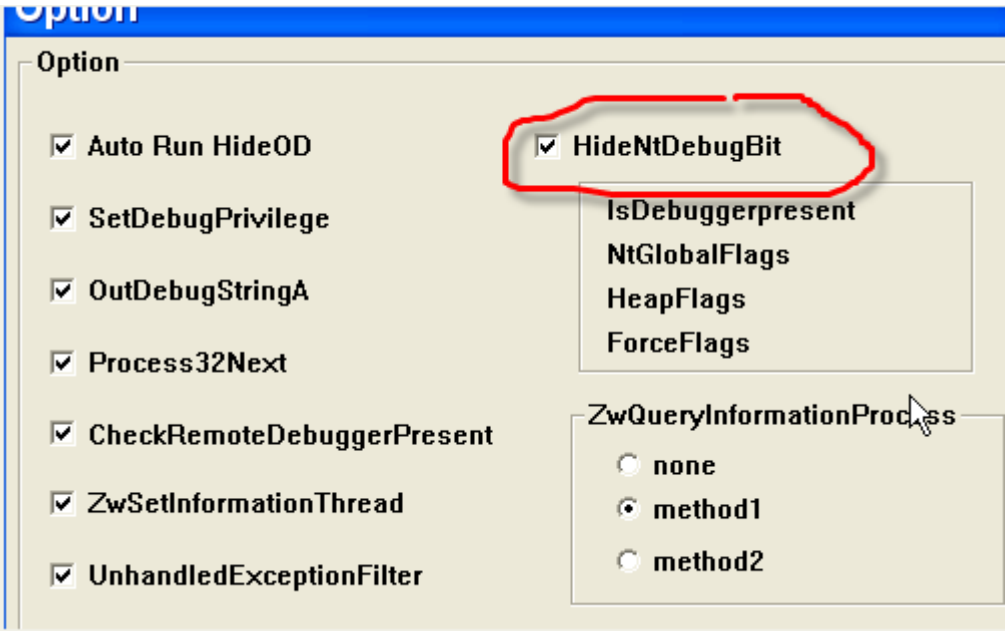


偏移 0x10 的位置的 4 个字节就是 HeapFlags 标志了,当前为零,表示当前没有被调试,这是加载了 HideOD 和 HideDebugger 插件的原因,我们不加载这两个插件然后打开该 CrackMe。

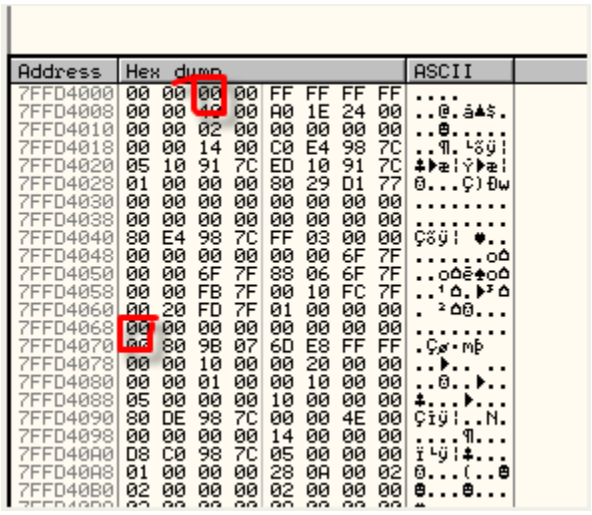


现在该 DWORD 就不为零,表示当前正在被调试,但是使用了某些插件的话,即使我们不设置绕过 ProcessHeap 的选项,该 DWORD 也会变为零。

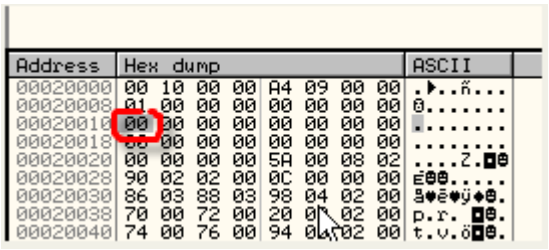
现在我们勾选上 HideOD 插件的 HideNtDebugBit 的选项。



我们重新启动 cruehead'a 的 CrackMe。

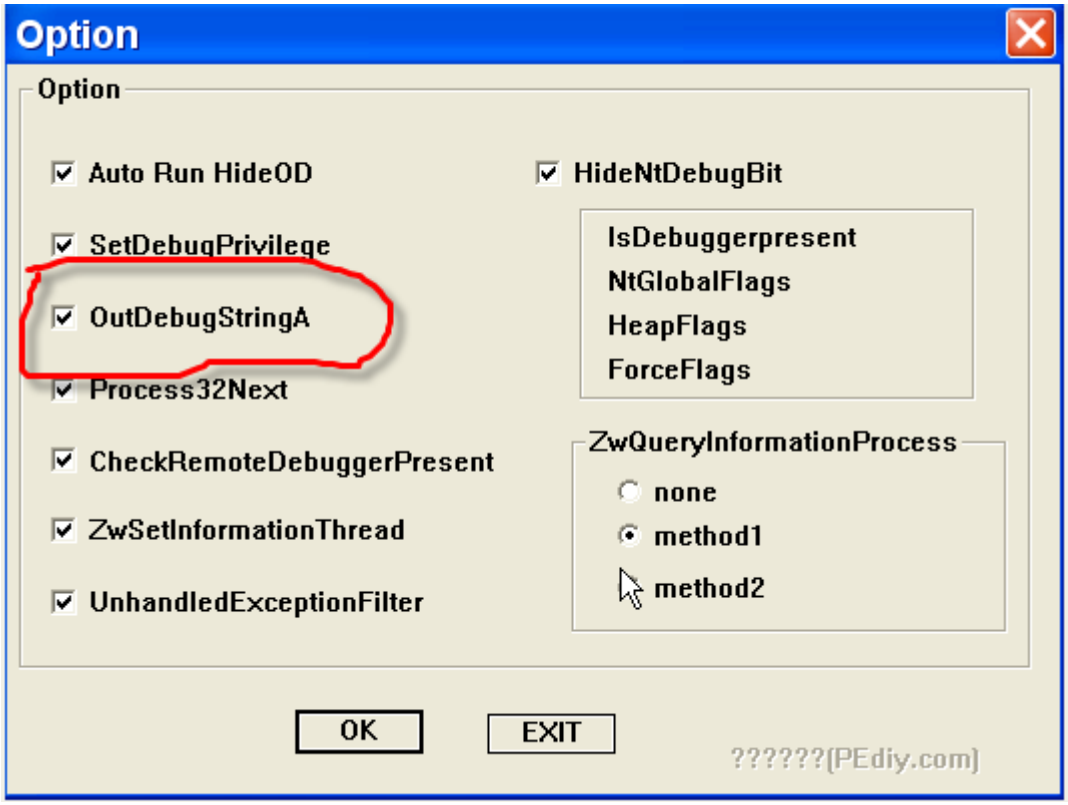


断在入口点处,我们定位到 IsDebuggerPresent 以及 NtGlobalFlag 标志,可以看到都是零,我们再来看看 ProcessHeap 标志。

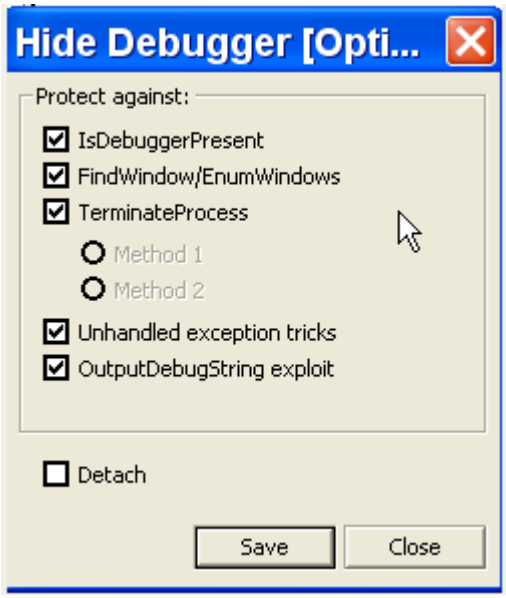


也是零,说明 HideOD 插件起作用了。至此,我们就学会了如何手工定位和修改 ProcessHeap 和 NTGlobalFlag 标志。

接下来我们看看 OutputDebugStringA 选项。



和



这里要提到一点就是:

OllyDbg 存在一个 bug-当被调试程序通过 OutputDebugString 输出超长的一串调试字符串的时候,OllyDbg 无法处理导致崩溃。我们可以通过上面的插件中的 OutputDebugStringA 选项来修复 OD 的这个 bug。这里我引用 Juan Jose 的 execryptor 脱壳教程中一段话来解释:

> Bueno, me dejo de rollos, de esta manera me di cuenta que el problema venia con el truco antiOlly de **OutputDebugString**, que justamente utiliza este programa y no los demás execryptos que había probado. Este truco consiste en mandarle un string preparado al Olly, que no lo sabe aceptar, y te da un error que cierra el Olly (creo ¿?) porque en realidad nunca me he encontrado con él, y si algún programa lo ha tenido el **plugin HideDebugger** lo neutraliza, y te manda una linda sonrisa:
>
> 
>
>
> La forma de realizarlo es muy simple tienes que llamar a **OutputDebugStringA** con una cadena de caracteres **%s** bien larga, yo he contado 100d:
>
> 
>

上图中 Juan Jose 的说法是:

“可以通过 OutputDebugStringA 输出一长串%s 字符串,OD 无法处理这么长一串字符串,就会发生错误。我们可以使用 HideDebuggger 插件来修复这个 bug。这里我给 OutputDebugStringA 传递的参数是长度为 100 的%s 字符串。”

以上就是 Juan Jose 的教程中描述。通过 HideDebuggger 插件的 OutputDebugString exploit 选项我们可以修复 OD 的这个 bug。

好了,这里给大家留一个小练习,名字叫做 antisocial1。在这里例子中大家可以看到我们前面介绍过的反调试技巧,同时还夹杂着其他的反调试,嘿嘿,大家发挥自己的想象来解决这个反调试吧。

(下面一点点是作者关于这个练习的提示,这里就不做翻译了,挺绕的,嘿嘿,下一章里面对这个例子会有详细介绍)