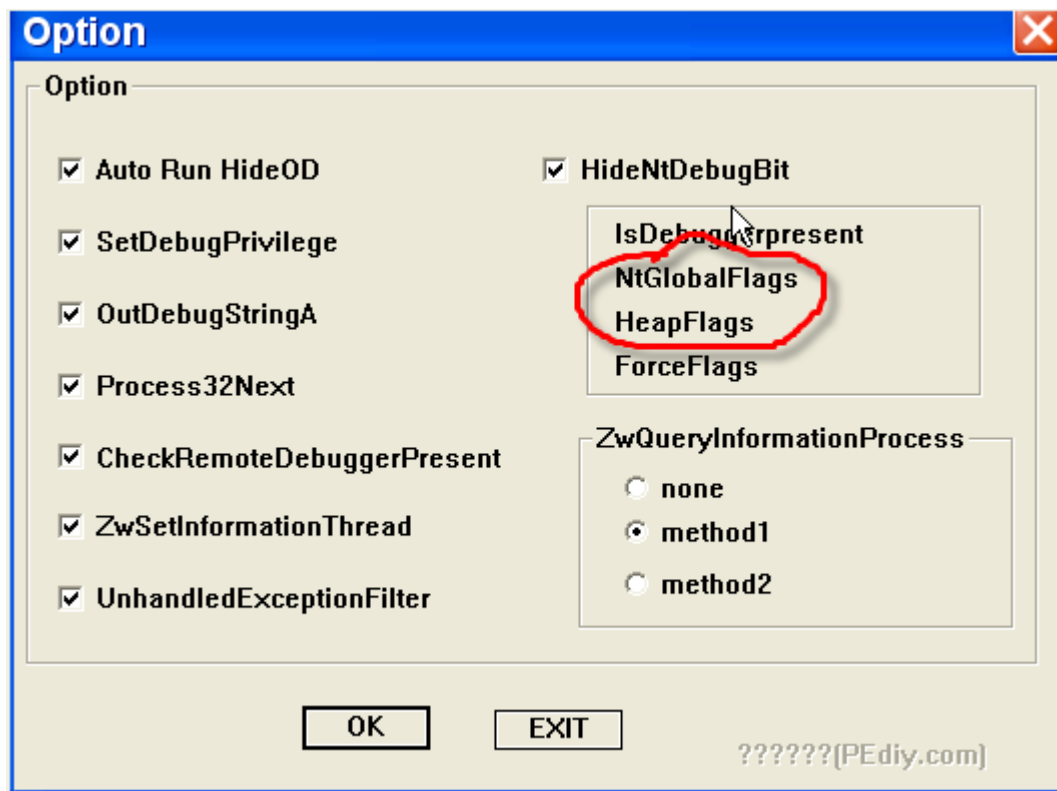


第二十三章-OllyDbg 反调试之 ProcessHeap,NTGlobalFlag,OutputDebugStringA

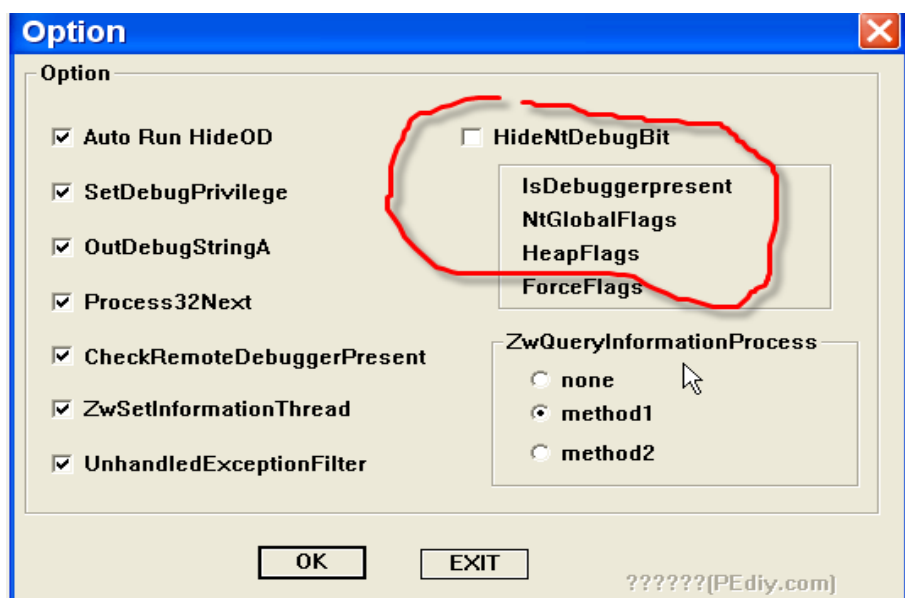
本章是反调试的最后一章,本章将介绍 ProcessHeap 和 NTGlobalFlag 标志位以及如何通过这两个标志位进行反调试,介绍完这部分内容我们就掌握了常见的反调试技巧。反调试的手法还有很多,我们介绍的只是最基本,最常见的。像一些保护壳,比如说 Execryptor 的反调试是比较厉害的,我们后面再介绍。Execryptor 壳除了我们介绍的这几种反调试以外,还有别的反调试手法,它的每个新版本都会增加一些新的反调试选项,这都是后话了,我们先来把常规的反调试手法介绍完。

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。

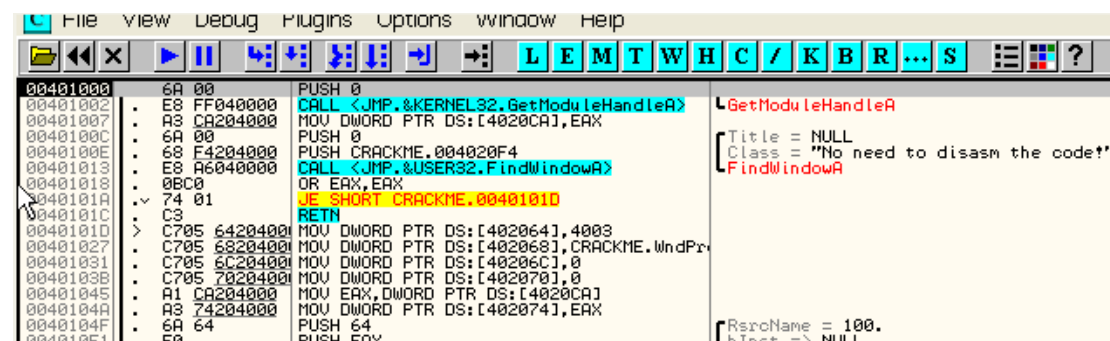
这两个标志位的话表示当前进程正在被调试,很容易定位到这两个标志位。不知道大家还记得不得 IsDebuggerPresent 对应的那个调试标志位是如何定位的,如果你不记得的话,回头去看看第 19 章,如果你弄明白如何定位那个标志位的话,那么这两个标志位也就好定位了,因为这两个标志位就在那个标志位的附近。



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。

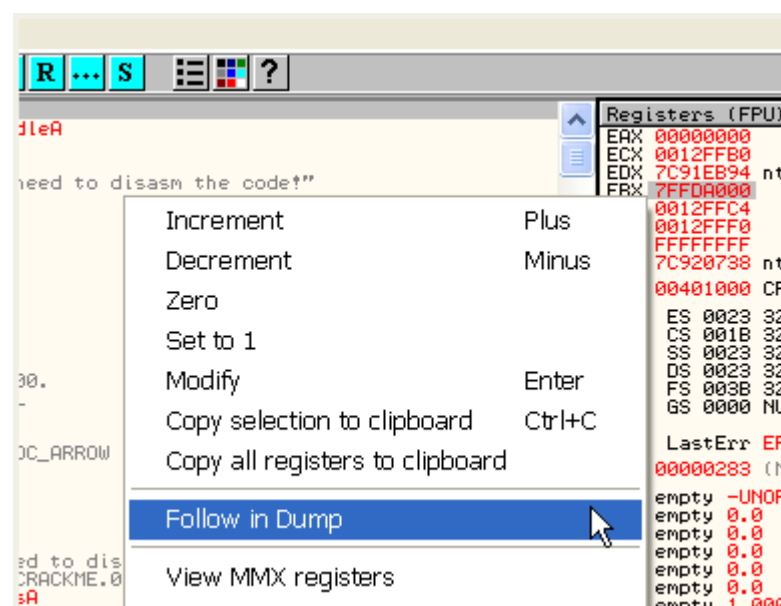
这里我们的实验对象是 Cruehead'a 的 CrackMe,我们首先来定位这两个标志位。

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。



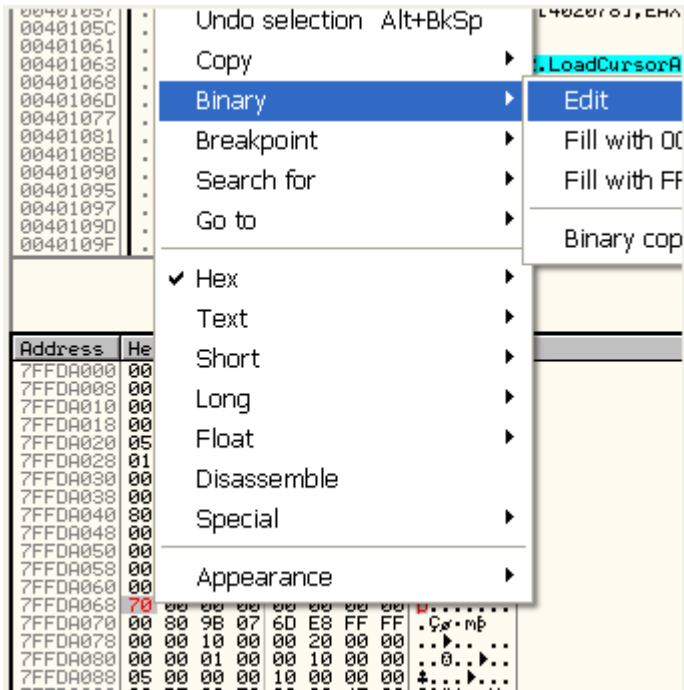
我们在数据窗口中定位到该标志位,我机器上的这个地址可能与你的不同,而且,该程序每次重新启动该地址也可能不同。

Address	Hex dump	ASCII
7FFDA000	00 00 00 00 FF FF FF FF
7FFDA008	00 00 40 00 A0 1E 24 00	...@...\$.
7FFDA010	00 00 02 00 00 00 00 00	...0....
7FFDA018	00 00 14 00 C0 E4 98 7C	...0...C...&
7FFDA020	05 10 91 7C ED 10 91 7C	...0...C...&
7FFDA028	01 00 00 00 80 29 D1 77	...0...C...&
7FFDA030	00 00 00 00 00 00 00 00
7FFDA038	00 00 00 00 00 00 00 00
7FFDA040	80 E4 98 7C FF 03 00 00	...&...&...
7FFDA048	00 00 00 00 00 00 6F 7F	...0...o...
7FFDA050	00 00 6F 7F 88 06 6F 7F	...o...o...&
7FFDA058	00 00 FB 7F 00 10 FC 7F	...0...&...&
7FFDA060	00 20 FD 7F 01 00 00 00	...0...&...
7FFDA068	00 00 00 00 00 00 00 00
7FFDA070	00 80 9B 07 6D E8 FF FF	...&...&...
7FFDA078	00 00 10 00 00 20 00 00	...0...&...
7FFDA080	00 00 01 00 00 10 00 00	...0...&...
7FFDA088	05 00 00 00 10 00 00 00	...0...&...
7FFDA090	80 DE 98 7C 00 00 4E 00	...&...&...N.
7FFDA098	00 00 00 00 14 00 00 00	...0...&...
7FFDA0A0	08 C0 98 7C 05 00 00 00	...&...&...&
7FFDA0A8	01 00 00 00 28 0A 00 02	...0...&...&
7FFDA0B0	02 00 00 00 02 00 00 00	...0...&...
7FFDA0B8	03 00 00 00 0A 00 00 00	...0...&...
7FFDA0C0	00 00 00 00 00 00 00 00

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。

Address	Hex dump	ASCII
7FFDA000	00 00 00 00 FF FF FF FF
7FFDA008	00 00 40 00 A0 1E 24 00	..@.â\$.
7FFDA010	00 00 02 00 00 00 00 00	..@.....
7FFDA018	00 00 14 00 C0 E4 98 7C	..@.tô!
7FFDA020	05 10 91 7C ED 10 91 7C	þa!ýþa!
7FFDA028	01 00 00 00 80 29 D1 77	@...Ç)ðw
7FFDA030	00 00 00 00 00 00 00 00
7FFDA038	00 00 00 00 00 00 00 00
7FFDA040	80 E4 98 7C FF 03 00 00	Çô! ♥..
7FFDA048	00 00 00 00 00 00 6F 7Fô
7FFDA050	00 00 6F 7F 88 06 6F 7Fôô
7FFDA058	00 00 FB 7F 00 10 FC 7F	..¹ð.þð
7FFDA060	00 20 FD 7F 01 00 00 00	..²ð...
7FFDA068	70 00 00 00 00 00 00 00	ð.....
7FFDA070	00 80 9B 07 6D E8 FF FF	..Çø·mþ
7FFDA078	00 00 10 00 00 20 00 00	..ð.ð...
7FFDA080	00 00 01 00 00 10 00 00	..@.ð...
7FFDA088	05 00 00 00 10 00 00 00	þ...ð...
7FFDA090	80 DE 98 7C 00 00 4E 00	Çi!..N.
7FFDA098	00 00 00 00 14 00 00 00@...

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。

Address	Hex dump	ASCII
7FFDA000	00 00 00 00 FF FF FF FF
7FFDA008	00 00 40 00 A0 1E 24 00	..@.â\$.
7FFDA010	00 00 02 00 00 00 00 00	..@.....
7FFDA018	00 00 14 00 C0 E4 98 7C	..@.tô!
7FFDA020	05 10 91 7C ED 10 91 7C	þa!ýþa!
7FFDA028	01 00 00 00 80 29 D1 77	@...Ç)ðw
7FFDA030	00 00 00 00 00 00 00 00
7FFDA038	00 00 00 00 00 00 00 00
7FFDA040	80 E4 98 7C FF 03 00 00	Çô! ♥..
7FFDA048	00 00 00 00 00 00 6F 7Fô
7FFDA050	00 00 6F 7F 88 06 6F 7Fôô
7FFDA058	00 00 FB 7F 00 10 FC 7F	..¹ð.þð
7FFDA060	00 20 FD 7F 01 00 00 00	..²ð...
7FFDA068	00 00 00 00 00 00 00 00
7FFDA070	00 80 9B 07 6D E8 FF FF	..Çø·mþ
7FFDA078	00 00 10 00 00 20 00 00	..ð.ð...
7FFDA080	00 00 01 00 00 10 00 00	..@.ð...
7FFDA088	05 00 00 00 10 00 00 00	þ...ð...
7FFDA090	80 DE 98 7C 00 00 4E 00	Çi!..N.
7FFDA098	00 00 00 00 14 00 00 00@...

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。

Address	Hex dump	ASCII
7FFDA000	00 00 00 00 FF FF FF FF
7FFDA008	00 00 40 00 A0 1E 24 00	..@.â\$.
7FFDA010	00 00 02 00 00 00 00 00	..@.....
7FFDA018	00 00 14 00 C0 E4 98 7C	..@.L\$y!
7FFDA020	05 10 91 7C ED 10 91 7C	..â!yâ!
7FFDA028	01 00 00 00 00 00 00 00	@.....
7FFDA030	00 00 00 00 00 00 00 00
7FFDA038	00 00 00 00 00 00 00 00
7FFDA040	80 E4 98 7C 01 00 00 00	Ç\$y!@..
7FFDA048	00 00 00 00 00 00 6F 7Fo
7FFDA050	00 00 6F 7F 88 06 6F 7F	..oâ+o
7FFDA058	00 00 FB 7F 00 10 FC 7F	..¹â,²
7FFDA060	00 20 FD 7F 01 00 00 00	..²â@...
7FFDA068	00 00 00 00 00 00 00 00
7FFDA070	00 80 9B 07 6D E8 FF FF	..Ç&mb
7FFDA078	00 00 10 00 00 20 00 00	..â.....
7FFDA080	00 00 01 00 00 10 00 00	..@.....
7FFDA088	03 00 00 00 10 00 00 00	...â.....
7FFDA090	80 DE 98 7C 00 00 00 00	Çiy!....
7FFDA098	00 00 00 00 00 00 00 00
7FFDA0A0	08 C0 98 7C 05 00 00 00	i-y!+...
7FFDA0A8	01 00 00 00 28 0A 00 02	@...(!..@
7FFDA0B0	02 00 00 00 02 00 00 00	@...@.....
7FFDA0B8	04 00 00 00 00 00 00 00	â.....
7FFDA0C0	00 00 00 00 00 00 00 00

我们来看看堆中保存了些什么。

Address	Hex dump	ASCII
7FFDA000	00 00 00 00 FF FF FF FF
7FFDA008	00 00 40 00 A0 1E 24 00	..@.â\$.
7FFDA010	00 00 02 00 00 00 00 00	..@.....
7FFDA018	00 00 14 00 C0 E4 98 7C	..@.L\$y!
7FFDA020	05 10 91 7C ED 10 91 7C	..â!yâ!
7FFDA028	01 00 00 00 00 00 00 00	@.....
7FFDA030	00 00 00 00 00 00 00 00
7FFDA038	00 00 00 00 00 00 00 00
7FFDA040	80 E4 98 7C 01 00 00 00	Ç\$y!@..
7FFDA048	00 00 00 00 00 00 6F 7Fo
7FFDA050	00 00 6F 7F 88 06 6F 7F	..oâ+o
7FFDA058	00 00 FB 7F 00 10 FC 7F	..¹â,²
7FFDA060	00 20 FD 7F 01 00 00 00	..²â@...
7FFDA068	00 00 00 00 00 00 00 00
7FFDA070	00 80 9B 07 6D E8 FF FF	..Ç&mb
7FFDA078	00 00 10 00 00 20 00 00	..â.....
7FFDA080	00 00 01 00 00 10 00 00	..@.....
7FFDA088	03 00 00 00 10 00 00 00	...â.....
7FFDA090	80 DE 98 7C 00 00 00 00	Çiy!....

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。

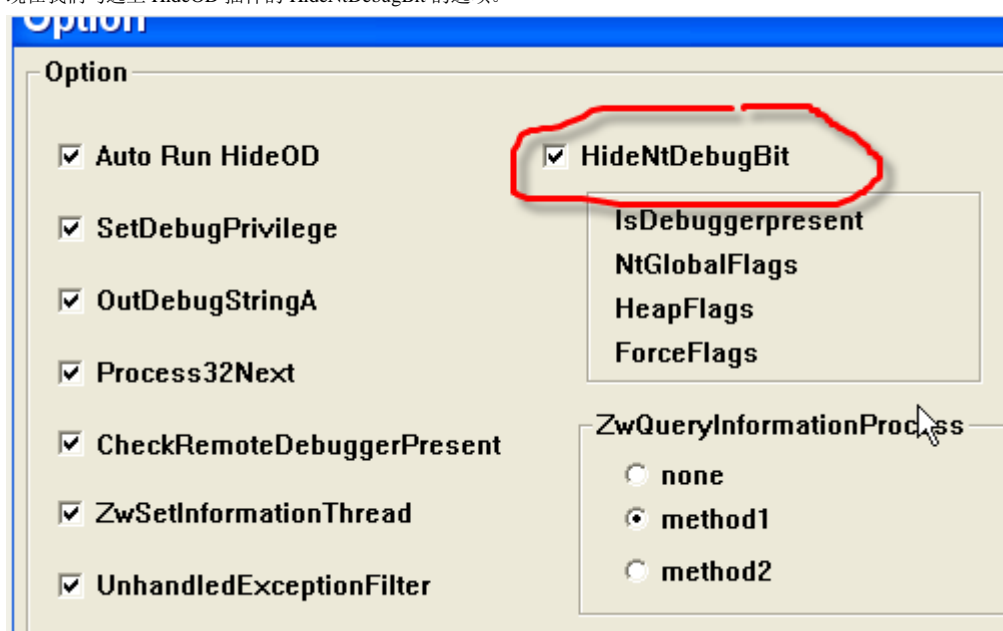
Address	Hex dump	ASCII
00140000	C8 00 00 00 4F 01 00 00	..00..
00140008	FF EE FF EE 02 00 00 00	..-@...
00140010	00 00 00 00 00 FE 00 00
00140018	00 00 10 00 00 20 00 00	..â.....
00140020	00 02 00 00 00 20 00 00	@.....
00140028	C3 00 00 00 FF EF FD 7F	..²â
00140030	01 00 08 06 00 00 00 00	@.â+...
00140038	00 00 00 00 00 00 00 00
00140040	00 00 00 00 98 01 14 00	...yâ!

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。

Address	Hex dump	ASCII
00140000	C8 00 00 00 75 01 00 00	..u0..
00140008	FF EE FF EE 62 00 00 50	..b..P
00140010	60 00 00 40 00 FE 00 00	..@..
00140018	00 00 10 00 00 20 00 00	..
00140020	00 02 00 00 00 20 00 00	..
00140028	16 00 00 00 FF EF FD 7F	..z
00140030	01 00 08 06 00 00 00 00	..
00140038	00 00 00 00 00 00 00 00
00140040	00 00 00 00 98 05 14 00	...\$.
00140048	17 00 00 00 F8 FF FF FF	...°
00140050	50 00 14 00 50 00 14 00	P..P..
00140058	40 06 14 00 00 00 00 00	@..
00140060	00 00 00 00 00 00 00 00
00140068	00 00 00 00 00 00 00 00

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。

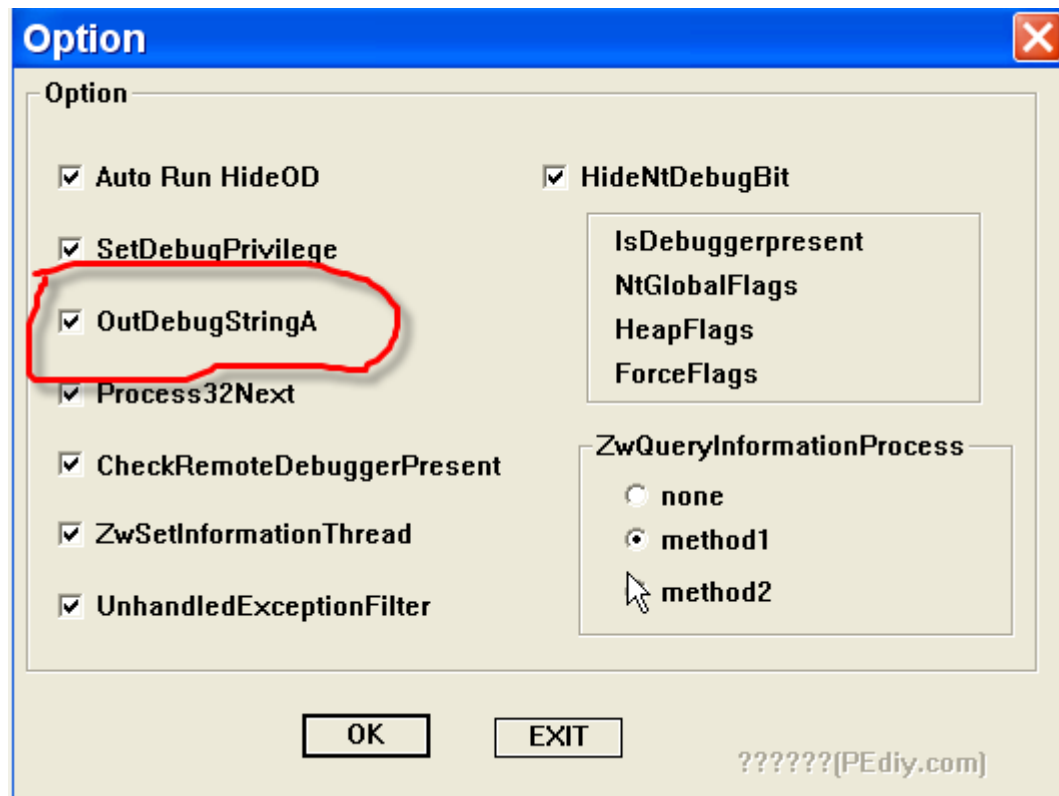
Address	Hex dump	ASCII
7FFD4000	00 00 00 00 FF FF FF FF
7FFD4008	00 00 00 00 A0 1E 24 00	..0.â\$.
7FFD4010	00 00 02 00 00 00 00 00	..0....
7FFD4018	00 00 14 00 C0 E4 98 7C	..0.48y!
7FFD4020	05 10 91 7C ED 10 91 7C	510917CED10917C
7FFD4028	01 00 00 00 80 29 D1 77	0...Ç)0w
7FFD4030	00 00 00 00 00 00 00 00
7FFD4038	00 00 00 00 00 00 00 00
7FFD4040	80 E4 98 7C FF 03 00 00	Ç8y! ..
7FFD4048	00 00 00 00 00 00 6F 7Fo
7FFD4050	00 00 6F 7F 88 06 6F 7F	..oöæo
7FFD4058	00 00 FB 7F 00 10 FC 7F	..¹0.º
7FFD4060	00 20 FD 7F 01 00 00 00	..²00...
7FFD4068	00 00 00 00 00 00 00 00
7FFD4070	00 80 9B 07 6D E8 FF FF	..Ç·mb
7FFD4078	00 00 10 00 00 20 00 00	..º....
7FFD4080	00 00 01 00 00 10 00 00	..0.º....
7FFD4088	05 00 00 00 10 00 00 00	50000010000000
7FFD4090	80 DE 98 7C 00 00 4E 00	Çiy!..N.
7FFD4098	00 00 00 00 14 00 00 00	...0....
7FFD40A0	D8 C0 98 7C 05 00 00 00	i0y!º....
7FFD40A8	01 00 00 00 28 0A 00 02	0...(.0
7FFD40B0	02 00 00 00 02 00 00 00	0...0....

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。

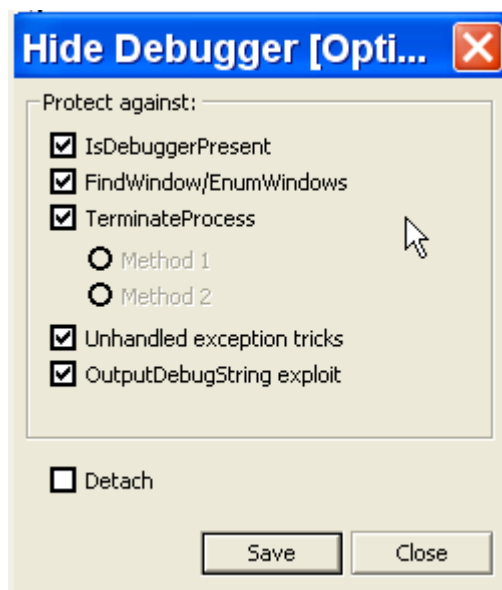
Address	Hex dump	ASCII
00020000	00 10 00 00 A4 09 00 00	.>..&...
00020008	01 00 00 00 00 00 00 00	@.....
00020010	00 00 00 00 00 00 00 00
00020018	00 00 00 00 00 00 00 00
00020020	00 00 00 00 5A 00 08 02	...Z..
00020028	90 02 02 00 0C 00 00 00	...&...
00020030	86 03 88 03 98 04 02 00	...&...
00020038	70 00 72 00 20 00 02 00	p.r. ...
00020040	74 00 76 00 94 00 02 00	t.v. ...

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



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:

Debug string: :-)

```
0013FDE0 004F1757 CALL to OutputDebugStringM  
0013FDE4 0013FDE8 String = "%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s"  
0013FDE8 73257325  
0013FDEC 73257325
```

“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 `HideDebugger` 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。”

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)