

第四十六章-Patrick 的 CrackMe-Part1

本章我们继续加深难度,实验程序名称为 **Patrick.exe**,该程序的保护较强。大家会发现之前介绍的一些方法对于该保护并不奏效。所有很多时候我们不得不适当变通,特定的保护特定解。

这里要提一句,一些商业壳不可能将所有的保护手段都运用到,因为商业壳的宗旨是要保证目标程序为任意软件。但是恰恰有些保护手段只针对于特定的程序。

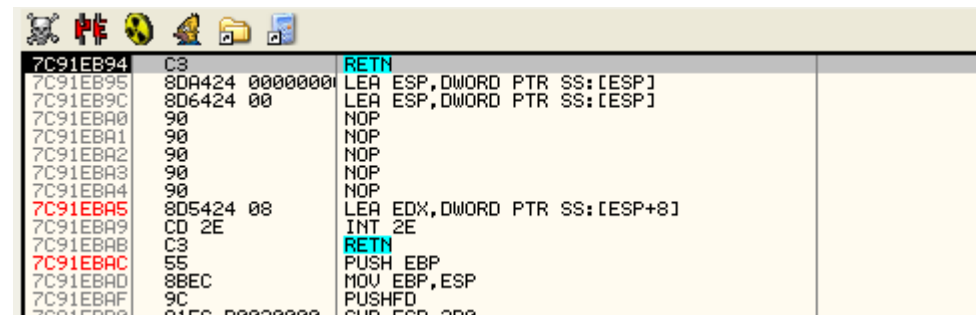
好了,这里我们打开 OD,还记得之前介绍那款修改过的 Patched 4 这款 OD 吧,就它了,配置好反反调试插件。

这个 CrackMe,作者的要求不仅仅是脱壳,还必须将 DLL 剥离,让 EXE 单独正常运行。

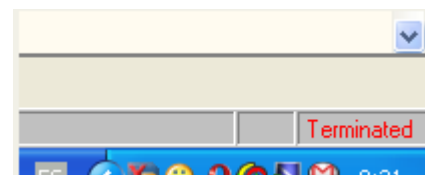


那么就是说 Patrick 主程序目录下的 AntiDebugDll.dll 这是一个核心 DLL,如果将这个 DLL 删除掉的话,主程序将无法正常运行。

好了,现在我们用 Patched4 这款 OD 加载 Patrick.exe。

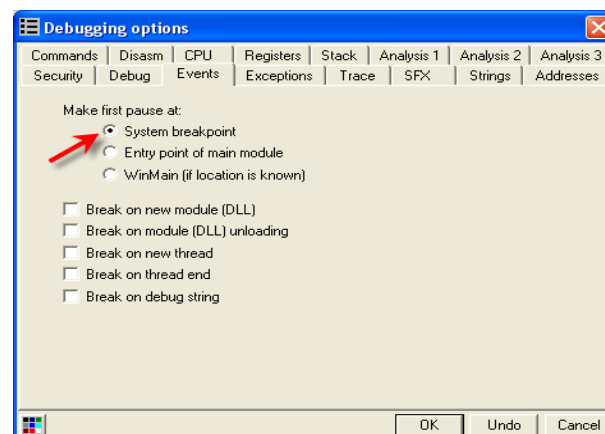


我们可以看到还没有到达入口点,程序就终止了。



好,下面我们来尝试将入口点修改为系统断点处,系统断点会在到达主程序入口点之前断下。

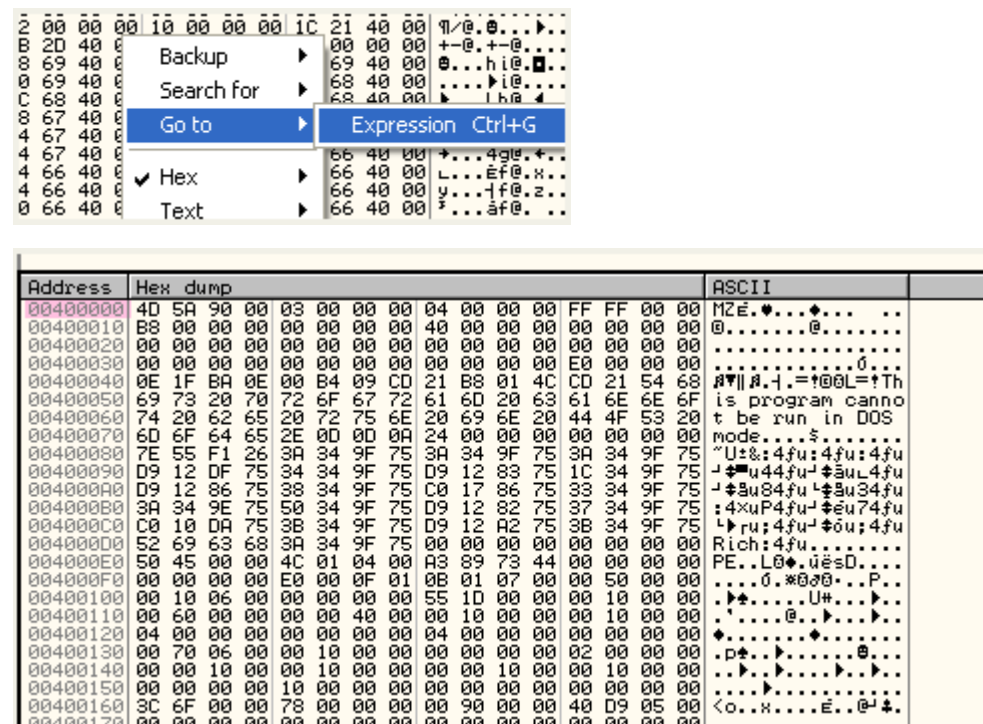
打开主菜单中的 Debugging options-Events,选中 System breakpoint。



这样就可以让程序首次中断在系统断点处,这里要提一句,有些 DLL 会在到达入口点之前被加载,而这些 DLL 会检测主程序是否正在被调试,如果正在被调试的话就立即结束进程。

有一点必须明确,就是系统动态库并不会对 OD 进行检测。

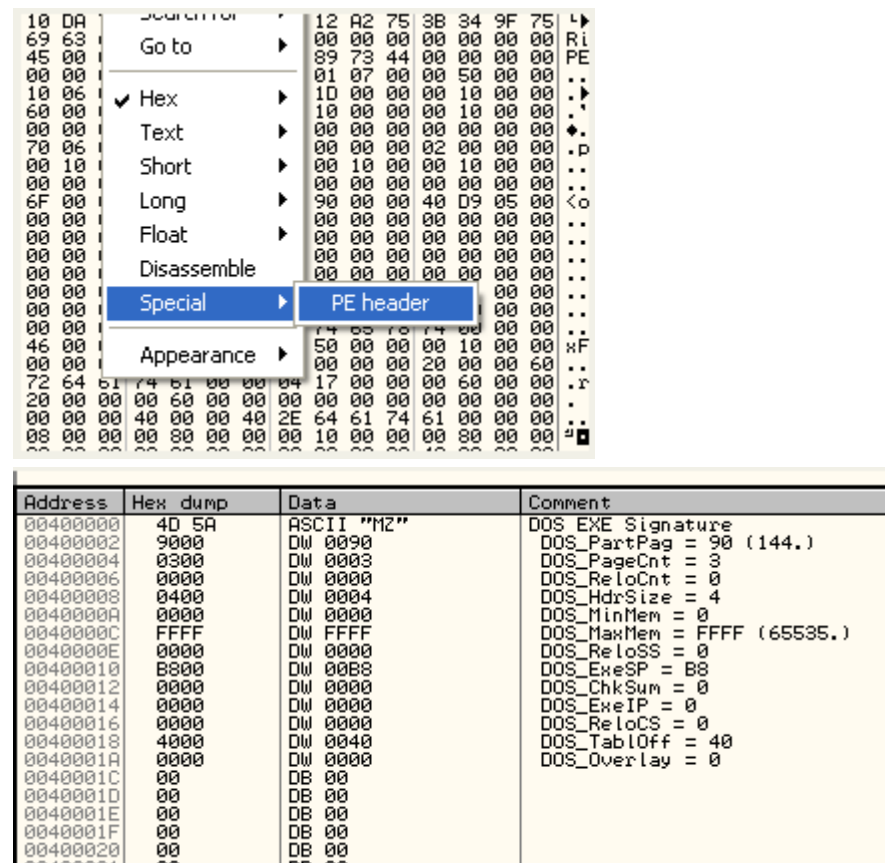
下面我们在数据窗口中单击鼠标右键选择 Goto Expression,输入 400000 定位到 PE 头。



The screenshot shows the 'Go to' menu in OllyDbg with 'Expression Ctrl+G' selected. Below it, a hex dump of memory is displayed, starting at address 00400000. The dump shows various bytes and their corresponding ASCII values, including 'MZ', 'PE', and other system-related strings.

Address	Hex dump	ASCII
00400000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....
00400002	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
00400004	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400006	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400008	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
0040000A	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F
0040000C	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20
0040000E	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00
00400010	7E 55 F1 26 3A 34 9F 75 3A 34 9F 75 3A 34 9F 75
00400012	D9 12 DF 75 34 34 9F 75 D9 12 83 75 1C 34 9F 75
00400014	D9 12 86 75 38 34 9F 75 C0 17 86 75 33 34 9F 75
00400016	3A 34 9E 75 50 34 9F 75 D9 12 82 75 37 34 9F 75
00400018	C0 10 DA 75 38 34 9F 75 D9 12 A2 75 38 34 9F 75
0040001A	52 69 63 68 3A 34 9F 75 00 00 00 00 00 00 00 00
0040001C	50 45 00 00 4C 01 04 00 A3 89 73 44 00 00 00 00
0040001E	00 00 00 00 E0 00 0F 01 0B 01 07 00 00 50 00 00
00400020	00 10 06 00 00 00 00 00 55 1D 00 00 00 10 00 00
00400022	00 60 00 00 00 00 40 00 00 10 00 00 00 10 00 00
00400024	04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
00400026	00 70 06 00 00 10 00 00 00 00 00 00 00 02 00 00
00400028	00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00
0040002A	00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
0040002C	3C 6F 00 00 78 00 00 00 00 90 00 00 40 D9 05 00

接着在数据窗口中单击鼠标右键选择 Special-PE Header,切换到 PE 结构解析模式。



The screenshot shows the 'Special' menu in OllyDbg with 'PE header' selected. Below it, a detailed view of the PE header structure is displayed, showing various fields and their values, including 'DOS EXE Signature', 'DOS_PartPag', 'DOS_PageCnt', 'DOS_ReloCnt', 'DOS_HdrSize', 'DOS_MinMem', 'DOS_MaxMem', 'DOS_ReloSS', 'DOS_ExeSP', 'DOS_ChkSum', 'DOS_ExeIP', 'DOS_ReloCS', 'DOS_Tabloff', and 'DOS_Overlay'.

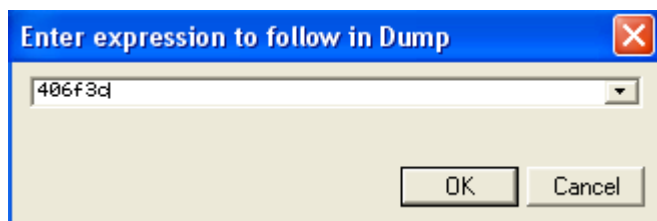
Address	Hex dump	Data	Comment
00400000	4D 5A	ASCII "MZ"	DOS EXE Signature
00400002	9000	DW 0090	DOS_PartPag = 90 (144.)
00400004	0300	DW 0003	DOS_PageCnt = 3
00400006	0000	DW 0000	DOS_ReloCnt = 0
00400008	0400	DW 0004	DOS_HdrSize = 4
0040000A	0000	DW 0000	DOS_MinMem = 0
0040000C	FFFF	DW FFFF	DOS_MaxMem = FFFF (65535.)
0040000E	0000	DW 0000	DOS_ReloSS = 0
00400010	B800	DW 00B8	DOS_ExeSP = B8
00400012	0000	DW 0000	DOS_ChkSum = 0
00400014	0000	DW 0000	DOS_ExeIP = 0
00400016	0000	DW 0000	DOS_ReloCS = 0
00400018	4000	DW 0040	DOS_Tabloff = 40
0040001A	0000	DW 0000	DOS_Overlay = 0
0040001C	00	DB 00	
0040001D	00	DB 00	
0040001E	00	DB 00	
0040001F	00	DB 00	
00400020	00	DB 00	
00400021	00	DB 00	

往下拉。

00400144	00100000	DD 00001000	SizeOfStackCommit = 1000 (4096.)
00400148	00001000	DD 00100000	SizeOfHeapReserve = 100000 (1048576.)
0040014C	00100000	DD 00001000	SizeOfHeapCommit = 1000 (4096.)
00400150	00000000	DD 00000000	LoaderFlags = 0
00400154	10000000	DD 00000010	NumberOfRvaAndSizes = 10 (16.)
00400158	00000000	DD 00000000	Export Table address = 0
0040015C	00000000	DD 00000000	Export Table size = 0
00400160	3C6F0000	DD 00006F3C	Import Table address = 6F3C
00400164	78000000	DD 00000078	Import Table size = 78 (120.)
00400168	00900000	DD 00009000	Resource Table address = 9000
0040016C	40D90500	DD 0005D940	Resource Table size = 5D940 (383296.)
00400170	00000000	DD 00000000	Exception Table address = 0
00400174	00000000	DD 00000000	Exception Table size = 0
00400178	00000000	DD 00000000	Certificate File pointer = 0
0040017C	00000000	DD 00000000	Certificate Table size = 0

这里我们可以看到 Import Table(缩写:IT,俗称:导入表)的RVA(相对虚拟地址)(PS:不要将 Import Table 与 IAT 搞混淆了,IAT 是 Import Address Table(输入函数地址表)的缩写,嘿嘿).

这里 6F3C + 映像基地址(400000)就可以定位到 Import Table 了。

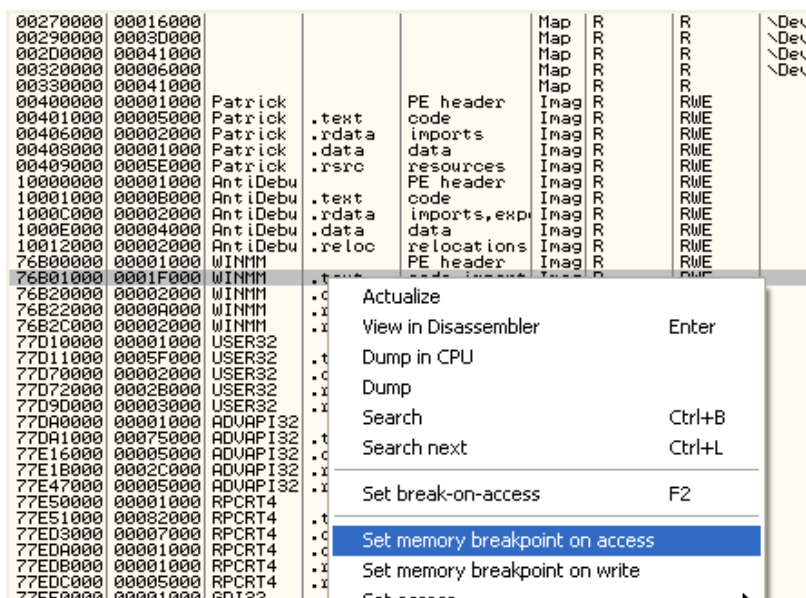


将数据窗口的显示模式由 PE 解析模式切换回十六进制模式。

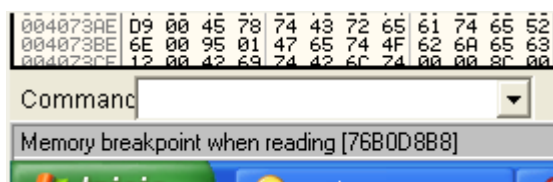
Address	Hex dump	ASCII
00406F3C	18 71 00 00 00 00 00 00 00 00 00 00 2E 71 00 00	td.....q..
00406F4C	64 61 00 00 64 6F 00 00 5C 53 00 00 00 00 00 00	da..fo.....
00406F5C	3C 71 00 00 00 60 00 00 EC 6F 00 00 00 00 00 00	8q...'.yo.....
00406F6C	00 00 00 00 F8 71 00 00 38 60 00 00 00 70 00 00o.q.'8'p..
00406F7C	00 00 00 00 00 00 00 00 2C 73 00 00 00 61 00 00s..la..
00406F8C	BC 6F 00 00 00 00 00 00 00 00 00 00 00 73 00 00	do.....zs..
00406F9C	08 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00406FAC	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
00406FBC	08 73 00 00 64 73 00 00 CE 73 00 00 00 73 00 00	is..ds..fs..ts..
00406FCC	AE 73 00 00 98 73 00 00 88 73 00 00 00 73 00 00	<<s..ys..es..xs..
00406FDC	56 73 00 00 46 73 00 00 38 73 00 00 00 00 00 00	Us..Fs..8s.....
00406FEC	A2 71 00 00 B6 71 00 00 C6 71 00 00 08 71 00 00	6q..Aq..Sq..iq..
00406FFC	E8 71 00 00 D2 76 00 00 C0 76 00 00 AE 76 00 00	6q..Ev..Lv..<v..
0040700C	9E 76 00 00 82 76 00 00 76 76 00 00 6A 76 00 00	xv..ev..vv..jv..
0040701C	5A 76 00 00 4A 76 00 00 3C 76 00 00 2A 76 00 00	Zv..Jv..<v..*v..
0040702C	10 76 00 00 00 76 00 00 E6 75 00 00 CE 75 00 00	v...v..vu..fu..
0040703C	B4 75 00 00 98 75 00 00 88 75 00 00 7C 75 00 00	fu..yu..eu..iu..
0040704C	68 75 00 00 54 75 00 00 8C 71 00 00 F4 76 00 00	hu..Tu..iq..lv..
0040705C	76 71 00 00 66 71 00 00 54 71 00 00 4A 71 00 00	vq..fq..Tq..Jq..
0040706C	E4 76 00 00 8E 76 00 00 EE 73 00 00 FC 73 00 00	8v..Av..ts..*s..
0040707C	0E 74 00 00 1A 74 00 00 26 74 00 00 32 74 00 00	8t..+t..&t..2t..
0040708C	44 74 00 00 56 74 00 00 70 74 00 00 80 74 00 00	Ot..Ut..pt..Ct..
0040709C	96 74 00 00 AC 74 00 00 C6 74 00 00 DC 74 00 00	ut..%t..st.._t..
004070AC	FA 74 00 00 08 75 00 00 16 75 00 00 24 75 00 00	.t..u...u..zu..
004070BC	34 75 00 00 42 75 00 00 00 00 00 00 20 73 00 00	4u..Eu.....s..
004070CC	10 73 00 00 02 73 00 00 F0 72 00 00 E4 72 00 00	ps..0s...r..6r..
004070DC	02 72 00 00 C4 72 00 00 B8 72 00 00 A6 72 00 00	Er...r..@r..@r..
004070EC	9A 72 00 00 8C 72 00 00 7E 72 00 00 6C 72 00 00	ur..ir...r..lr..
004070FC	5A 72 00 00 4C 72 00 00 3A 72 00 00 2A 72 00 00	Zr..Lr...r..*r..
0040710C	18 72 00 00 06 72 00 00 00 00 00 00 20 71 00 00	tr..tr.....q..
0040711C	00 00 00 00 0A 00 50 6C 61 79 53 6F 75 6E 64 41PlaySoundA
0040712C	00 00 57 49 4E 4D 40 2E 64 6C 6C 00 41 6E 74 69	..WINMM.dll.Anti
0040713C	44 65 62 75 67 44 6C 2E 64 6C 6C 00 00 00 EB 00	DebugDll.dll.v.
0040714C	47 65 74 41 43 50 00 00 5D 01 47 65 74 4C 6F 63	GetACP...J0GetLoc

不知道大家是否还记得导入表的格式,每个 DLL 项占 5 个 DWORD。(PS:关于 PE 结构不了解的童鞋,可以参看传说中的小黄书,你懂得!嘿嘿 Windows PE 权威指南)

每个 DLL 项中的第 4 个 DWORD 指向了 DLL 的名称字符串,那么 40712E 这个地址就指向了第一个 DLL 的名称字符串,我们一起来看看。

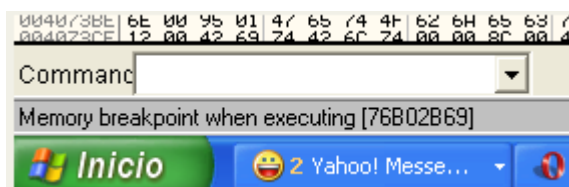


这里断点就设置好了,我们直接运行起来。

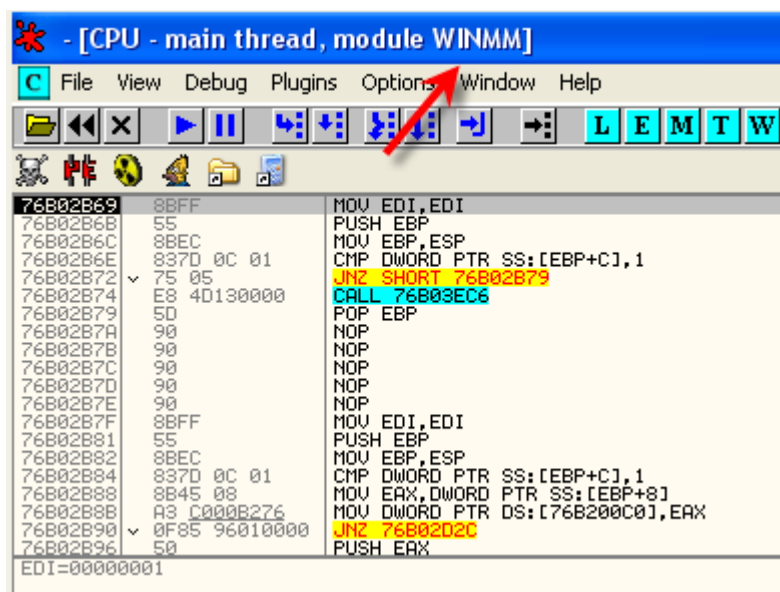


我们看到断了下来,这里是由于读取 76B0D8B8 地址处的内容导致的中断。

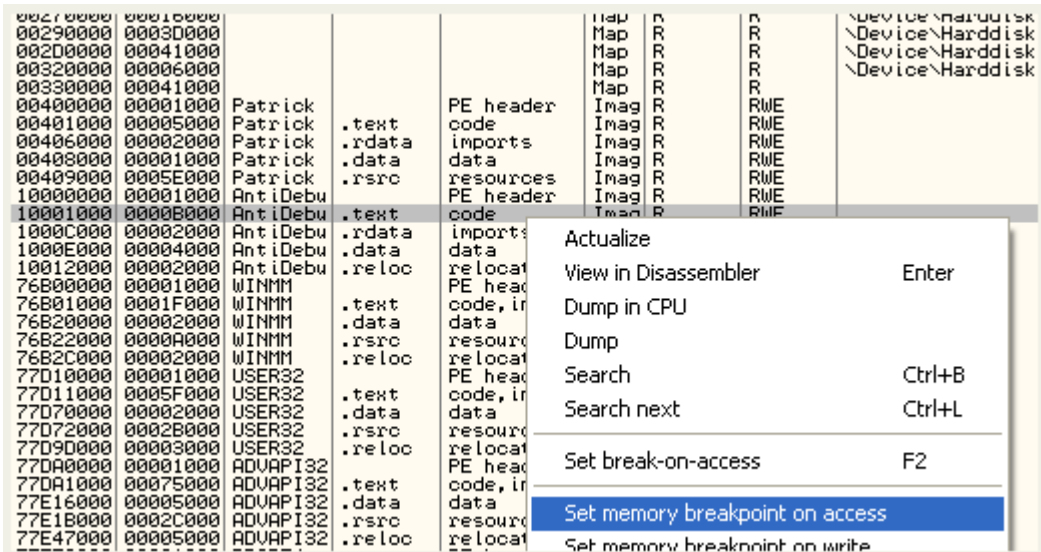
我们继续运行,直到触发内存执行断点为止。



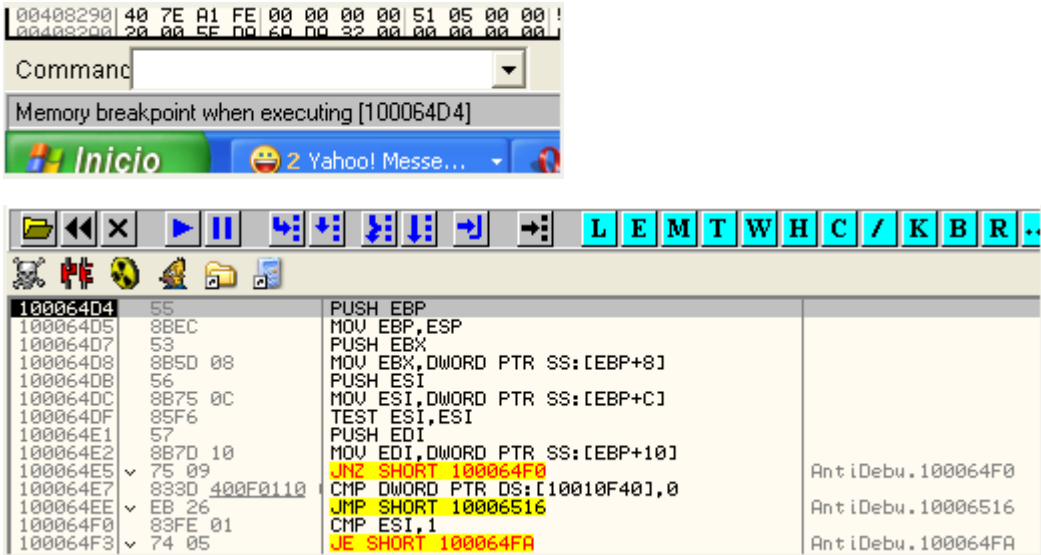
断在了这里,这里就要开始执行该 DLL 的代码了。



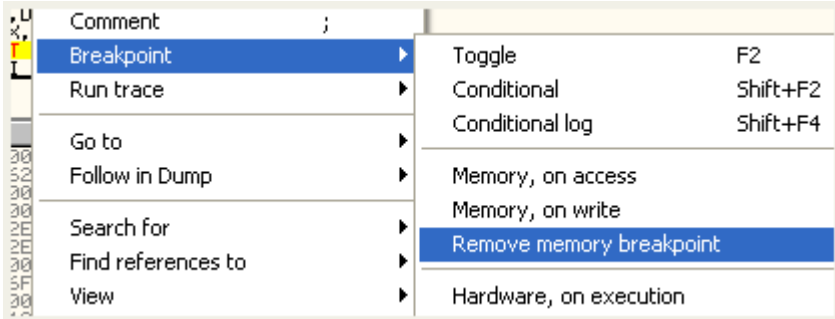
跟踪反调试 DLL 的流程就是这样的,接下来,我们重启 OD,再次重复一遍上面的步骤,这次我们直接对 AntiDebug.dll 的代码段设置内存访问断点。



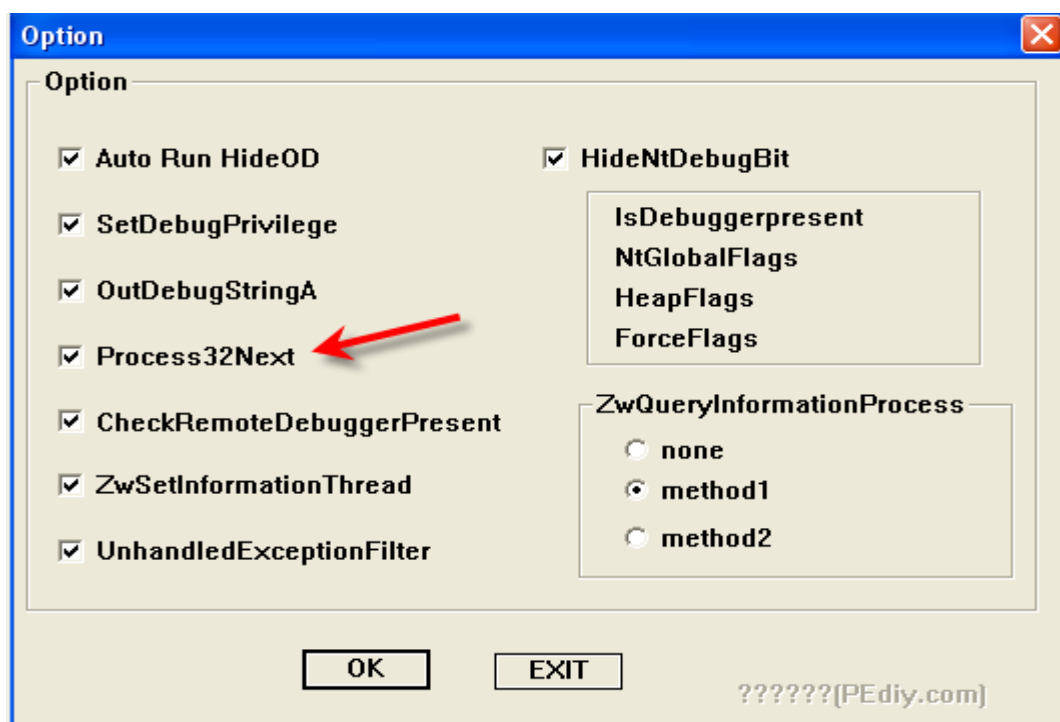
我们运行起来,会发现首次就触发了执行断点。



在运行之前我们首先删除掉之前设置的内存访问断点:

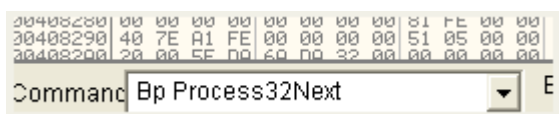


这里提示一下该程序既不是检测 OD 的进程名,也不是检测 OD 窗口名,也不是 HideOD 插件中的涉及的那些检测方法,它是检测该进程是由谁创建的,通过调用 Process32Next 等 API 函数来遍历进程,判断当前进程的父进程是否为桌面进程来达到反调试的目的。

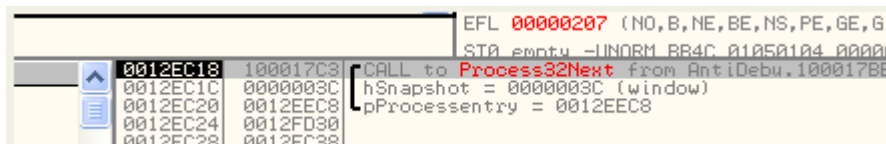


也就说该 CrackMe 会判断当前进程是否由 explorer.exe 启动,如果是的话,那么就说明是用户双击运行的。如果不是的话,那么就说明正在被调试,直接终止进程。

这里我们给 Process32Next 这个 API 函数设置一个断点。



接着运行起来。



断了下来,这里可以看到第二个参数 pProcessentry 的值为 12EEC8。

Address	Hex dump	ASCII
0012EEC8	28 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00	(0.....
0012EED8	00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00	...0.....
0012EEE8	00 00 00 00 5B 53 79 73 74 65 6D 20 50 72 6F 63[System Proc
0012EEF8	65 73 73 5D 00 00 00 00 00 00 00 00 00 00 00 00	ess].....
0012EF08	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

这里我们用 PUPE 这个小工具查看一下进程的 PID。

Procesos en ejecución	ID Proceso	ID Modulo	Nº Threads	Prioridad
ctfmon.exe	00000478	00000000	00000001	Normal
spoolsv.exe	000001D0	00000000	0000000C	Normal
svchost.exe	000007E4	00000000	00000014	Normal
svchost.exe	00000674	00000000	00000008	Normal
svchost.exe	00000640	00000000	00000052	Normal
svchost.exe	00000454	00000000	0000000A	Normal
svchost.exe	0000041C	00000000	00000011	Normal
lsass.exe	0000037C	00000000	00000015	
services.exe	00000370	00000000	00000010	
winlogon.exe	00000344	00000000	00000015	Alta
csrss.exe	0000032C	00000000	0000000D	Alta
smss.exe	00000208	00000000	00000003	
system	00000004	00000000	00000038	Normal
[system process]	00000000	00000000	00000001	

Seleccione el proceso y presione el botón derecho del ratón

Actualizar Salir

这里我们可以看到第一个进程的 PID 为零,我们结合 MSDN 来看。

Archivo	Edición	Marcador	Opciones	Ayuda
Contenido	Índice	Atrás	Imprimir	≤< >≥

Process32Next

Retrieves information about the next process recorded in a system snapshot.

```
BOOL WINAPI Process32Next(HANDLE hSnapshot, LPPROCESSENTRY32 lppe);
```

Parameters

hSnapshot
Handle of the snapshot returned from a previous call to the [CreateToolhelp32Snapshot](#) function.

lppe
Address of a [PROCESSENTRY32](#) structure.

Return Value

Returns TRUE if the next entry of the process list has been copied to the buffer or FALSE otherwise. The ERROR_NO_MORE_FILES error value is returned by the [GetLastError](#) function if no processes exist or the snapshot does not contain process information.

Remarks

To retrieve information about the first process recorded in a snapshot, use the [Process32First](#) function.

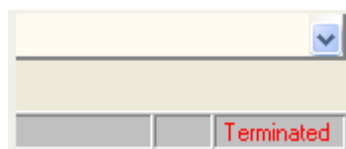
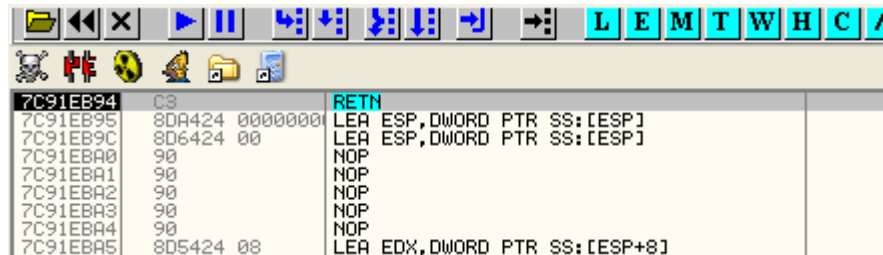
这里我们可以看到关于 Process32Next 这个 API 函数的说明。我们看下关于 PROCESSENTRY32 这个结构体的解释。

PROCESSENTRY32
<pre>typedef struct tagPROCESSENTRY32 { DWORD dwSize; DWORD cntUsage; DWORD th32ProcessID; DWORD th32DefaultHeapID; DWORD th32ModuleID; DWORD cntThreads; DWORD th32ParentProcessID; LONG pcPriClassBase; DWORD dwFlags; char szExeFile[MAX_PATH]; } PROCESSENTRY32; typedef PROCESSENTRY32 PPROCESSENTRY32; typedef PROCESSENTRY32 *LPPROCESSENTRY32;</pre>
<h3>Members</h3> <p>dwSize Specifies the length, in bytes, of the structure. Before calling the Process32First function, set this member to sizeof(PROCESSENTRY32).</p> <p>cntUsage Number of references to the process. A process exists as long as its usage count is nonzero. As soon as its usage count becomes zero, a process terminates.</p> <p>th32ProcessID Identifier of the process. The contents of this member can be used by Win32 API elements.</p> <p>th32DefaultHeapID Identifier of the default heap for the process. The contents of this member has meaning only to the tool help functions. It is not a handle, nor is it usable by Win32 API elements.</p> <p>th32ModuleID Module identifier of the process. The contents of this member has meaning only to the tool help functions. It is not a handle, nor is it usable by Win32 API elements.</p> <p>cntThreads Number of execution threads started by the process.</p> <p>th32ParentProcessID Identifier of the process that created the process being examined. The contents of this member can be used by Win32 API elements.</p>

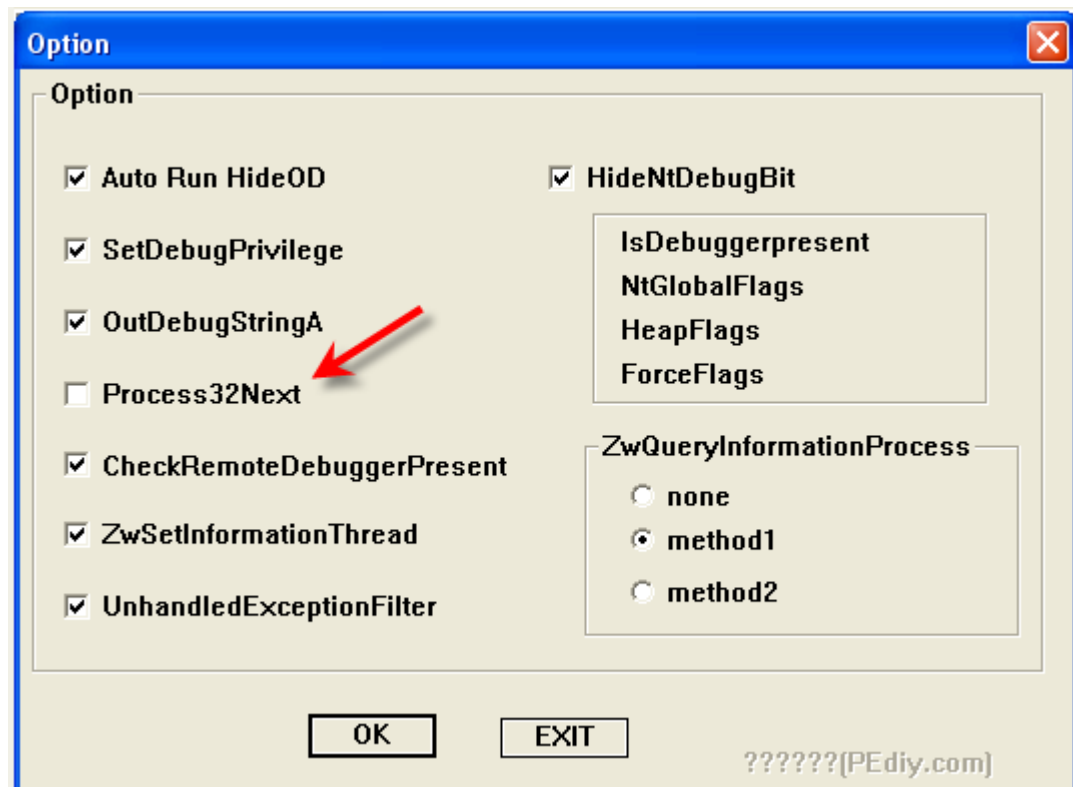
这里我们可以看到第三个字段为 th32ProcessID 即进程 PID,第七个字段为 th32ParentProcessID 即父进程 ID。

Address	Hex dump	ASCII
0012EEC8	28 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00	{0.....
0012EED8	00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 000.....
0012EEF8	00 00 00 00 5B 53 79 73 74 65 6D 20 50 72 6F 63[System Proc
0012EEF8	65 73 73 5D 00 00 00 00 00 00 00 00 00 00 00 00	ess].....
0012EF08	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

这里我用粉红色标注出了 PID,绿色标注出了 PPID。这里我们可以看到当前遍历到的这个进程的 PID(进程 ID)和 PPID(父进程 ID)都为零,代表是[System Process]这个进程,这个进程并不是我们要定位的,我们直接运行起来。



嘿嘿,程序终止了。怎么这样就终止了呢?难道[System Process]这个系统进程有问题?不太可能吧,那么会不会是 HideOD 插件冲突了的原因呢? 我们去掉 HideOD 中 Process32Next 这一项的对勾试试。



现在我们重启 OD,这里我们不需要再给该 DLL 的代码段设置内存访问断点了,我们直接给 Process32Next 下断。

Address	Hex dump	ASCII
0012EEC8	28 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00	(0.....
0012EED8	00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00	...0.....
0012EEF8	00 00 00 00 5B 53 79 73 74 65 6D 20 50 72 6F 63	...[System Proc
0012EEF8	65 73 73 5D 00 00 00 00 00 00 00 00 00 00 00 00	ess].....
0012EF08	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

断了下来,我们可以看到 PID 和 PPID 都为零。

这里就是刚刚出问题的地方,如果勾选了 HideOD 上的 Process32Next 这个选项的话,当该函数执行完毕,EAX 等于零,说明出错了,程序就直接退出了。这里我们不勾选这一项,EAX 非零。函数执行成功,这样程序就不会直接退出了。

Address	Hex dump	ASCII
0012EEC8	28 01 00 00 00 00 00 00 04 00 00 00 00 00 00 00	(0.....
0012EED8	00 00 00 00 3B 00 00 00 00 00 00 00 00 00 00 00	...0.....
0012EEF8	00 00 00 00 53 59 53 54 45 40 00 20 50 72 6F 63	...SYSTEM Proc
0012EEF8	65 73 73 5D 00 00 00 00 00 00 00 00 00 00 00 00	ess].....
0012EF08	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

我们运行起来,这次程序并没有退出,这里遍历到了进程快照中的第二个进程。即 SYSTEM 进程,PID 为 4,我用粉红色标注出来了。PPID 为 0,我用绿色标注出来了。

ctfmon.exe	00000478	00000000	00000001	Normal
spoolsv.exe	000001D0	00000000	0000000C	Normal
svchost.exe	000007E4	00000000	00000014	Normal
svchost.exe	00000674	00000000	00000008	Normal
svchost.exe	00000640	00000000	00000052	Normal
svchost.exe	00000454	00000000	0000000A	Normal
svchost.exe	0000041C	00000000	00000011	Normal
lsass.exe	0000037C	00000000	00000015	
services.exe	00000370	00000000	00000010	
winlogon.exe	00000344	00000000	00000015	Alta
csrss.exe	0000032C	00000000	0000000D	Alta
smss.exe	00000208	00000000	00000003	
system	00000004	00000000	0000003B	Normal
[system process]	00000000	00000000	00000001	

Seleccione el proceso y presione el botón derecho del ratón

Actualizar Salir

其实我们还可以对删除掉 Process32Next 入口处的断点,将断点下在该函数的返回处。

7C863CAC	8946 20	MOV DWORD PTR DS:[ESI+20],EAX	
7C863CAF	8BC3	MOV EAX,EBX	
7C863CB1	5B	POP EBX	
7C863CB2	EB 0C	JMP SHORT 7C863CC0	kernel32.7C863CC0
7C863CB4	68 040000C0	PUSH C00000C0	
7C863CB9	E8 BD56FAFF	CALL 7C80937B	kernel32.7C80937B
7C863CBE	33C0	XOR EAX,EAX	
7C863CC0	8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]	
7C863CC3	5E	POP ESI	
7C863CC4	E8 485AFAFF	CALL 7C809711	kernel32.7C809711
7C863CC9	C9	LEAVE	
7C863CCA	C2 0800	RETN 8	
7C863CCD	90	NOP	
7C863CCE	90	NOP	
7C863CCF	90	NOP	
7C863CD0	90	NOP	

我们继续运行直到出现 Patrick.exe 为止。

Address	Hex dump	ASCII
0012EEC8	28 01 00 00 00 00 00 00 E4 00 00 00 00 00 00 00	(0.....%
0012EED8	00 00 00 00 01 00 00 00 AC 00 00 00 00 00 00 00	...0...%.
0012EEF8	00 00 00 00 50 61 74 72 69 63 68 2E 65 78 65 00	...Patrick.exe
0012EEF8	45 58 45 00 54 45 2E 45 58 45 00 45 00 00 00 00	EXE.TE.EXE.E...
0012EF08	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

这里我们可以看到此时定位到了 Patrick.exe 这个进程。PID 为 0x0AE4,PPID 为 0x8AC,这里我们对比着 PUPE 里面的进程信息来看。我们单击 PUPE 中的 Actualizar(刷新)按钮。

我们按 F7 单步,可以看到 PPID 被保存到了 12EEB8 地址处。

Address	Hex dump	ASCII
0012EEB8	AC 08 00 00 3C 00 00 00 E4 0A 00 00 7C 0C 00 00	%<...%...!
0012EEC8	28 01 00 00 00 00 00 00 E4 0A 00 00 00 00 00 00	(0...%...!
0012EED8	00 00 00 00 01 00 00 00 AC 08 00 00 08 00 00 00	...0...%...!
0012EEF8	00 00 00 00 50 41 54 52 49 43 4B 2E 45 58 45 00	...PATRICK.EXE.
0012EF08	45 58 45 00 54 45 2E 45 58 45 00 45 00 00 00 00	EXE.TE.EXE.E...
0012EF18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

我们对 12EEB8 地址处的内容设置硬件访问断点,来定位何处会获取 PPID 的值。

Address	Hex dump	ASCII
0012EEB8	AC 08 00 00 3C 00 00 00 E4 0A 00 00 7C 0C 00 00	%<...%...!
0012EEC8	28 01 00 00 00 00 00 00 E4 0A 00 00 00 00 00 00	(0...%...!
0012EED8	00 00 00 00 01 00 00 00 AC 08 00 00 08 00 00 00	...0...%...!
0012EEF8	00 00 00 00 50 41 54 52 49 43 4B 2E 45 58 45 00	...PATRICK.EXE.
0012EF08	45 58 45 00 54 45 2E 45 58 45 00 45 00 00 00 00	EXE.TE.EXE.E...
0012EF18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF38	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF48	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF58	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF68	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF78	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF88	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF98	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EFA8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EFB8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EFC8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EFD8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EFE8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

我们运行起来,会发现 PPID 最初保存的地址处的值已经被覆盖掉了。

Address	Hex dump	ASCII
0012EEB8	AC 08 00 00 3C 00 00 00 E4 0A 00 00 7C 0C 00 00	%<...%...!
0012EEC8	28 01 00 00 EC EE 12 00 08 00 00 00 0C EA 12 00	(0...%...!
0012EED8	10 EA 12 00 16 00 00 00 AC 08 00 00 08 00 00 00	...0...%...!
0012EEF8	00 00 00 00 3F 12 0B 00 49 43 4B 2E 45 58 45 00	...PATRICK.EXE.
0012EF08	45 58 45 00 54 45 2E 45 58 45 00 45 00 00 00 00	EXE.TE.EXE.E...
0012EF18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF38	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF48	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EF58	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

我们删除掉之前设置的内存访问断点。

Address	Hex dump	ASCII
100019AF	3B85 C4FDFFFF	
100019B5	74 35	JE SHORT 100019EC
100019B7	B8 7CA01B47	MOV EAX,471BA07C
100019BC	3D E09C95B9	CMP EAX,B9959CE0
100019C1	74 07	JE SHORT 100019CA
100019C3	B8 CB190010	MOV EAX,100019CB
100019C8	FFEB	JMP NEAR EAX
100019CA	DC6A 00	FSUBR QWORD PTR DS:[EDX]
100019CD	FF15 0CC00010	CALL NEAR DWORD PTR DS:[1000C00C]
100019D3	B8 5EAEBE0D	MOV EAX,0DBAE0E5
100019D8	3D 7077F8F2	CMP EAX,F2F87770
100019DD	74 07	JE SHORT 100019E6
100019DF	B8 E7190010	MOV EAX,100019E7
100019E4	FFEB	JMP NEAR EAX
100019E6	DEE9	FSUBP ST(1),ST
100019E8	D6	SALC
100019E9	0300	ADD EAX,DWORD PTR DS:[EAX]
100019EB	00B8 22CA049B	ADD BYTE PTR DS:[EAX+9B04CA22],BH
100019F1	3D 902CBE65	CMP EAX,65BE2C90
100019F6	74 07	JE SHORT 100019FF

再次运行,断了下来,这里就是进行比较的地方。这个将 parcheado4(Patched 4)的 PID 与另一个 PID 0xC74 进行比较。

Process name	PID	Process ID	Module ID	Thread ID	Module
extra buttons lite.exe	00000980	00000000	00000001	Normal	
sched.exe	00000E40	00000000	00000007	Normal	
avgnt.exe	000005A8	00000000	00000002	Normal	
avguard.exe	00000C08	00000000	00000017	Normal	
ypager.exe	00000098	00000000	0000001C	Normal	
opera.exe	00000FBC	00000000	00000014	Normal	
wmplayer.exe	00000BFC	00000000	00000004	Normal	
googledesktopcrawl.exe	00000BB8	00000000	0000000A	Normal	
googledesktopdisplay.exe	00000CA0	00000000	0000000C	Normal	
googledesktopindex.exe	00000870	00000000	00000010	Normal	
googledesktop.exe	000003D4	00000000	00000002	Normal	
explorer.exe	00000C7C	00000000	00000015	Normal	
fdm.exe	00000FCC	00000000	00000006	Normal	
kfwserv.exe	00000F20	00000000	00000005	Normal	

Seleccione el proceso y presione el botón derecho del ratón

Actualizar Salir

正如所料,是 explorer.exe 的 PID。

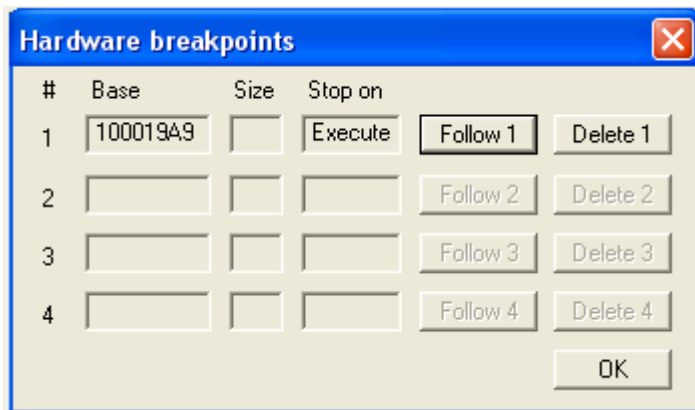
100019AF	3B85 C4DFFFF	CMP EAX, DWORD PTR SS:[EBP-23C]	AntiDebu.100019EC
100019B5	74 35	JE SHORT 100019EC	AntiDebu.100019EC
100019B7	B8 7CA01B47	MOV EAX, 471BA07C	AntiDebu.100019CA
100019BC	3D E09C95B9	CMP EAX, B9959CE0	AntiDebu.100019CA
100019C1	74 07	JE SHORT 100019CA	AntiDebu.100019CA
100019C3	B8 CB190010	MOV EAX, 100019CB	kernel32.ExitProcess
100019C8	FFEB	JMP NEAR EAX	kernel32.ExitProcess
100019CA	DC6A 00	FSUBR QWORD PTR DS:[EDX]	kernel32.ExitProcess
100019CD	FF15 0CC00010	CALL NEAR DWORD PTR DS:[1000C00C]	kernel32.ExitProcess
100019D3	B8 5EAE8E00	MOV EAX, 00BEAE8E	AntiDebu.100019E6
100019D8	3D 7077F8F2	CMP EAX, F2F87770	AntiDebu.100019E6
100019DD	74 07	JE SHORT 100019E6	AntiDebu.100019E6
100019DF	B8 E7190010	MOV EAX, 100019E7	AntiDebu.100019E6
100019E4	FFEB	JMP NEAR EAX	AntiDebu.100019E6
100019E6	DEE9	FSUBP ST(1), ST	AntiDebu.100019E6
100019E8	D6	SALC	AntiDebu.100019E6
100019E9	0300	ADD EAX, DWORD PTR DS:[EAX]	AntiDebu.100019E6
100019EB	00B8 22CA049B	ADD BYTE PTR DS:[EAX+9B04CA22], BH	AntiDebu.100019E6
100019F1	3D 902CBE65	CMP EAX, 65BE2C90	AntiDebu.100019E6
100019F6	74 07	JE SHORT 100019FF	AntiDebu.100019E6

这里我们可以看到,判断 Patrick.exe 父进程的 PID 是否与 explorer.exe 的 PID 相同,相同的话,继续往下运行,不同的话则调用 ExitProcess 退出进程。这是其中一处反调试,如果我们将此处修改并保存到文件的话,那么如果还其他其他反调试的话,程序还是无法正常运行。所以这里我们不进行修改,我们删除掉之前设置的硬件访问断点。给进行比较的这一行指令设置硬件执行断点。

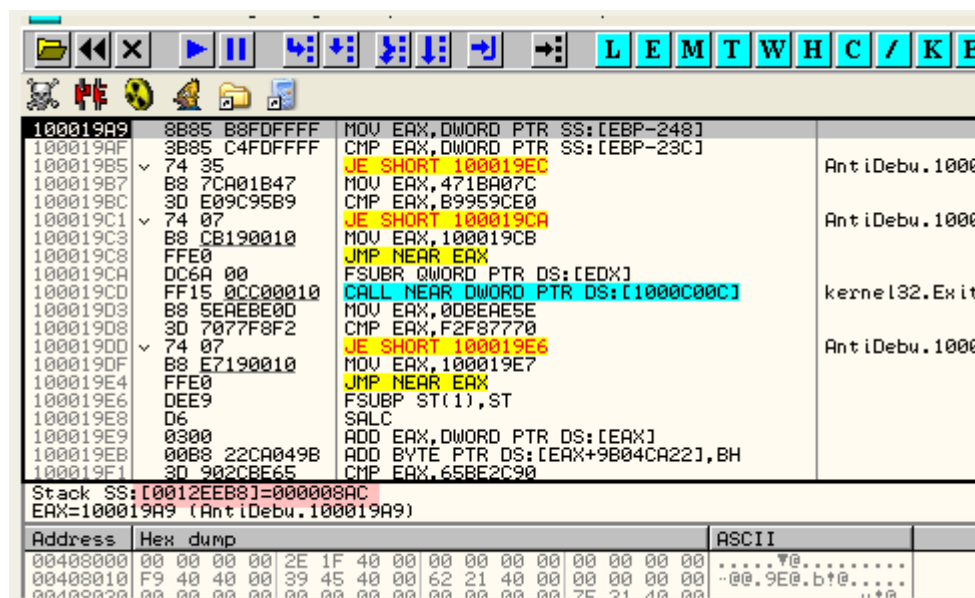
100019AF	3B85 C4DFFFF	CMP EAX, DWORD PTR SS:[EBP-23C]	Backup	00019EC
100019B5	74 35	JE SHORT 100019EC	Copy	00019CA
100019B7	B8 7CA01B47	MOV EAX, 471BA07C	Binary	00019CA
100019BC	3D E09C95B9	CMP EAX, B9959CE0	Assemble	Space
100019C1	74 07	JE SHORT 100019CA	Label	:
100019C3	B8 CB190010	MOV EAX, 100019CB	Comment	;
100019C8	FFEB	JMP NEAR EAX	Breakpoint	Toggle
100019CA	DC6A 00	FSUBR QWORD PTR DS:[EDX]	Run trace	Conditional
100019CD	FF15 0CC00010	CALL NEAR DWORD PTR DS:[1000C00C]	Go to	Shift+F2
100019D3	B8 5EAE8E00	MOV EAX, 00BEAE8E	Follow in Dump	Conditional log
100019D8	3D 7077F8F2	CMP EAX, F2F87770	Search for	Shift+F4
100019DD	74 07	JE SHORT 100019E6	Find references to	Memory, on access
100019DF	B8 E7190010	MOV EAX, 100019E7	View	Memory, on write
100019E4	FFEB	JMP NEAR EAX		Hardware, on execution
100019E6	DEE9	FSUBP ST(1), ST		
100019E8	D6	SALC		
100019E9	0300	ADD EAX, DWORD PTR DS:[EAX]		
100019EB	00B8 22CA049B	ADD BYTE PTR DS:[EAX+9B04CA22], BH		
100019F1	3D 902CBE65	CMP EAX, 65BE2C90		
100019F6	74 07	JE SHORT 100019FF		

当程序断在这里的时候,我们可以手动修改 Patrick.exe 父进程的 PID,将其修改为 explorer.exe 的 PID 值。

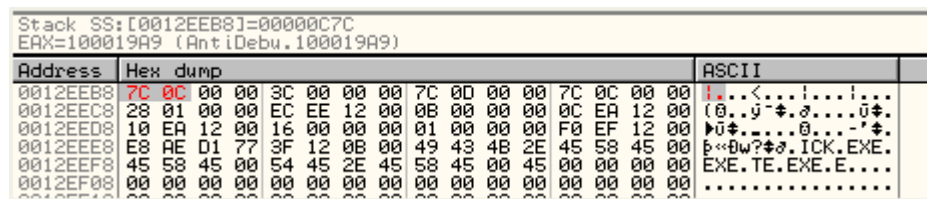
更加方便的方法是断在这一行的时候修改 EAX 的值。



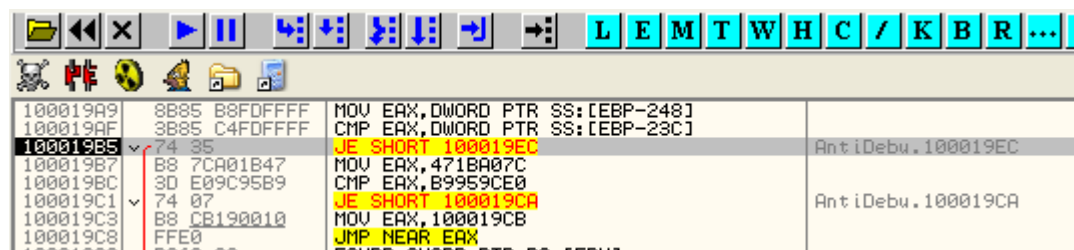
现在我们删除之前设置的 Process32Next 这个函数的断点。重启 OD。



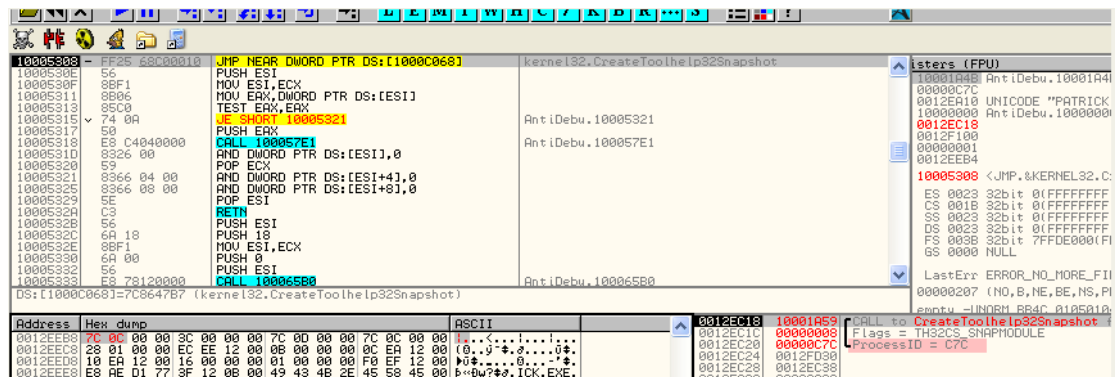
断在了这里,12EEB8 地址处保存的 PID 为 0x8AC。而另一个与之比较的 PID 为 0xC7C。



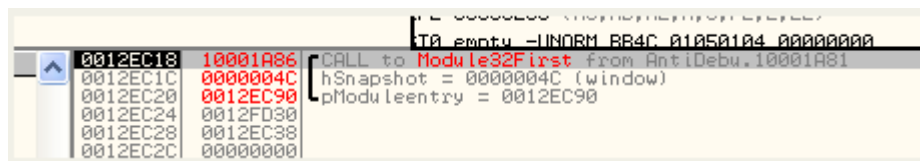
这里我们将 12EEB8 地址处的值修改为 0xC7C。这样两个 PID 就相等了,程序也就不会直接退出了。我们按 F7 键单步。



这样第一处反调试就绕过了。我们继续。



这里往下跟一点,就可以看到该 CrackMe 会获取 PID 为 0xC7C 进程(即 explorer.exe)的模块快照。进而检查模块的一些字段信息,用于判断 explorer 是不是重命名过的。



我们继续往下跟,就会到达 Module32First 函数调用处。这个 API 函数可以配合 Module32Next 这个 API 函数来遍历进程模块信息。

```
typedef struct _MODULEENTRY32 {
    DWORD dwSize;
    DWORD th32ModuleID;
    DWORD th32ProcessID;
    DWORD GblcntUsage;
    DWORD ProccntUsage;
    DWORD modBaseAddr;
    DWORD modBaseSize;
    HMODULE hModule;
    LPCTSTR szModule;
    LPCTSTR szExePath;
} MODULEENTRY32;
```

Members

dwSize
Specifies the length, in bytes, of the structure. Before calling the [Module32First](#) function, set this member to sizeof(MODULEENTRY32).

th32ModuleID
Module identifier in the context of the owning process. The contents of this member has meaning only to the tool help functions. It is not a handle, nor is it usable by Win32 API elements.

th32ProcessID
Identifier of the process being examined. The contents of this member can be used by Win32 API elements.

GblcntUsage
Global usage count on the module.

ProccntUsage
Module usage count in the context of the owning process.

modBaseAddr
Base address of the module in the context of the owning process.

modBaseSize
Size, in bytes, of the module.

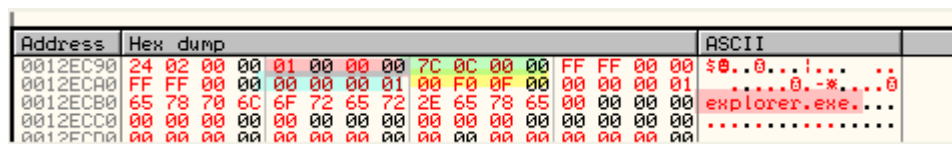
hModule
Handle of the module in the context of the owning process.

szModule
String containing the module name.

szExePath
String containing the location (path) of the module.

Note **modBaseAddr** and **hModule** are valid *only* in the context of the process specified by **th32ProcessID**.

我们来看看 MSDN 中的关于该函数的说明。



这里模块信息依次是:MODULEENTRY32 结构大小,模块标示符,进程 ID,全局模块引用计数,所在进程范围内模块引用计数,模块基地址,模块大小,模块句柄,模块名称,模块全路径,预留标志。这里我们可以看到模块基地址为 0x10000000。

Address	Hex dump	ASCII
0012EC90	24 02 00 00 01 00 00 00 7C 0C 00 00 FF FF 00 00	\$0..0...!... ..
0012ECA0	FF FF 00 00 00 00 00 01 00 F0 0F 00 00 00 00 010.-*....0
0012ECB0	65 78 70 6C 6F 72 65 72 2E 65 78 65 00 00 00 00	explorer.exe....
0012ECC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ECD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ECE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ECF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED50	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED80	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED90	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EDA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EDB0	43 3A 5C 57 49 4E 44 4F 57 53 5C 65 78 70 6C 6F	C:\WINDOWS\explo
0012EDC0	72 65 72 2E 65 78 65 00 00 00 00 00 00 00 00 00	rer.exe.....
0012EDD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EDE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EDF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012FF00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

这里我们可以看到关于模块的相关信息,我们来一起看看通过该 CrackMe 通过模块信息会干什么事情。

10001AB6	8B8D C4FDFFFF	MOV ECX,DWORD PTR SS:[EBP-23C]	
10001ABC	3B8D 98FBFFFF	CMP ECX,DWORD PTR SS:[EBP-468]	
10001AC2	0F85 CB020000	JNZ 10001D93	AntiDebu.10001D93
10001AC8	B8 41B2CB6C	MOV EAX,6CCBB241	
10001ACD	3D C8932394	CMP EAX,942393C8	

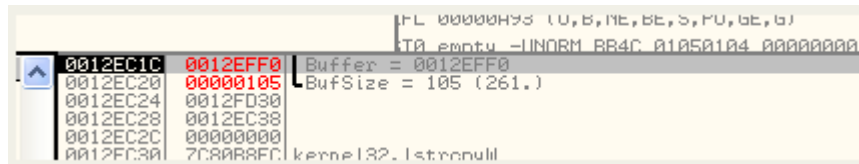
往下跟一点就可以看到读取 0xC7C 与模块信息中的进程 PID 进行比较。判断该模块是不是 explorer.exe。

10001AB6	8B8D C4FDFFFF	MOV ECX,DWORD PTR SS:[EBP-23C]	
10001ABC	3B8D 98FBFFFF	CMP ECX,DWORD PTR SS:[EBP-468]	
10001AC2	0F85 CB020000	JNZ 10001D93	AntiDebu.10001D93
10001AC8	B8 41B2CB6C	MOV EAX,6CCBB241	
10001ACD	3D C8932394	CMP EAX,942393C8	
10001AD2	74 07	JE SHORT 10001ADB	AntiDebu.10001ADB
10001AD4	B8 DC1A0010	MOV EAX,10001ADC	
10001AD9	FF00	JMP NEAR EAX	
10001ADB	D968 05	FILDQ WORD PTR DS:[EAX+5]	
10001ADE	0100	ADD DWORD PTR DS:[EAX],EAX	
10001AE0	008D 95F0FEFF	ADD BYTE PTR SS:[EBP+FFFEF095],CL	
10001AE6	FF52 FF	CALL NEAR DWORD PTR DS:[EDX-1]	
10001AE9	15 58C00010	ADC EAX,1000C058	
10001AEE	B8 23C06E33	MOV EAX,336EC023	
10001AF3	3D 586E86CD	CMP EAX,CD866E58	
10001AF8	74 07	JE SHORT 10001B01	AntiDebu.10001B01
10001AFA	B8 021B0010	MOV EAX,10001B02	
10001AFF	FF00	JMP NEAR EAX	
10001B01	DB	???	Unknown command
10001B02	8D85 F0FEFFFF	LEA EAX,DWORD PTR SS:[EBP-110]	
Stack SS:[0012EC98]=00000C7C			
ECX=00000C7C			
Address	Hex dump	ASCII	
0012EC90	24 02 00 00 01 00 00 00 7C 0C 00 00 FF FF 00 00	\$0..0...!... ..	
0012ECA0	FF FF 00 00 00 00 00 01 00 F0 0F 00 00 00 00 010.-*....0	
0012ECB0	65 78 70 6C 6F 72 65 72 2E 65 78 65 00 00 00 00	explorer.exe....	
0012ECC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

我们继续往下跟。

10001ADC	68 05010000	PUSH 105	
10001AE1	8D95 F0FEFFFF	LEA EDX,DWORD PTR SS:[EBP-110]	
10001AE7	52	PUSH EDX	
10001AE8	FF15 58C00010	CALL NEAR DWORD PTR DS:[1000C058]	kernel32.GetWindowsDirectoryA
10001AEE	B8 23C06E33	MOV EAX,336EC023	
10001AF3	3D 586E86CD	CMP EAX,CD866E58	
10001AF8	74 07	JE SHORT 10001B01	AntiDebu.10001B01
10001AFA	B8 021B0010	MOV EAX,10001B02	

这里我们可以看到调用 GetWindowsDirectory 这个 API 函数获取 Windows 的目录,也就是 explorer.exe 所在目录。也就是说该 CrackMe 会检测父进程的路劲。



这里我们可以看到 Windows 所在目录字符串将要保存的这个缓冲区中。

Address	Hex dump	ASCII
0012EFF0	43 3A 5C 57 49 4E 44 4F 57 53 00 20 DC 02 22 21	C:\WINDOWS.
0012F000	60 01 3A 20 52 01 9D 00 7D 01 78 01 A0 00 A1 00	'0: R00.00x00.i.
0012F010	A2 00 A3 00 A4 00 A5 00 A6 00 A7 00 A8 00 A9 00	0.0.0.0.0.0.0.0.
0012F020	AA 00 AB 00 AC 00 AD 00 AE 00 AF 00 B0 00 B1 00	1.0.0.0.0.0.0.0.
0012F030	B2 00 B3 00 B4 00 B5 00 B6 00 B7 00 B8 00 B9 00	2.0.0.0.0.0.0.0.

这里我们可以看到保存到了这里。下面就进行比较路径是否一致了。这里路径是一致的,我们继续跟。

10001B20	75 E2	JNZ SHORT 10001B11	AntiDebu.10001B11
10001B2F	8BBD 60FBFFFF	MOV EDI,DWORD PTR SS:[EBP-498]	
10001B35	66:A1 04C20010	MOV AX,WORD PTR DS:[1000C204]	
10001B3E	66:8907	MOV WORD PTR DS:[EDI],AX	
10001B3E	B8 05CE11FA	MOV EAX,FA11CE05	
10001B43	3D E848E906	CMP EAX,6E948E8	
10001B48	74 07	JE SHORT 10001B51	AntiDebu.10001B51
10001B4A	B8 521B0010	MOV EAX,10001B52	
10001B4F	FF07	JMP NEAR EAX	
AX=005C Stack DS:[0012EFA1]=2000			
Address	Hex dump	ASCII	
0012EFAA	00 00 00 00 00 00 43 3A 5C 57 49 4E 44 4F 57 53C:\WINDOWS	
0012EFBA	00 20 DC 02 22 21 60 01 3A 20 52 01 9D 00 7D 01	.'0: R00.00x00.i.	
0012F00A	78 01 A0 00 A1 00 A2 00 A3 00 A4 00 A5 00 A6 00	x00.i.0.0.0.0.0.	
0012F01A	A7 00 A8 00 A9 00 AA 00 AB 00 AC 00 AD 00 AE 00	0.0.0.0.0.0.0.0.	

我们继续往下,可以看到该程序将 C:\WINDOWS 与 explorer.exe 进行字符串拼接。

10001B76	75 E2	JNZ SHORT 10001B61	AntiDebu.10001B61
10001B7F	8BBD 60FBFFFF	MOV EDI,DWORD PTR SS:[EBP-4A0]	
10001B85	8BBD 08C20010	MOV ECX,DWORD PTR DS:[1000C208]	
10001B8B	890F	MOV DWORD PTR DS:[EDI],ECX	
10001B8D	8B15 0CC20010	MOV EDX,DWORD PTR DS:[1000C20C]	
10001B93	8957 04	MOV DWORD PTR DS:[EDI+4],EDX	
10001B96	A1 10C20010	MOV EAX,DWORD PTR DS:[1000C210]	
10001B9B	8947 08	MOV DWORD PTR DS:[EDI+8],EAX	
10001B9E	8A0D 14C20010	MOV CL,BYTE PTR DS:[1000C214]	
10001BA4	884F 0C	MOV BYTE PTR DS:[EDI+C],CL	
10001BA7	B8 E7DBB4C0	MOV EAX,C0B4DBE7	
10001BAC	3D 78234C40	CMP EAX,404C2378	
10001BB1	74 07	JE SHORT 10001BB8	AntiDebu.10001BB8
10001BB3	B8 BB1B0010	MOV EAX,10001BBB	
DS:[1000C208]=4C505845			
Address	Hex dump	ASCII	
1000C208	45 58 50 4C 4F 52 45 52 2E 45 58 45 00 00 00 00	EXPLORER.EXE....	
1000C218	52 45 41 4C 49 47 4E 2E 44 4C 4C 00 50 52 4F 43	REALIGN.DLL.PROC	
1000C228	53 2E 44 4C 4C 00 00 00 43 41 44 54 2E 44 4C 4C	S.DLL...CAOT.DLL	
1000C238	00 00 00 00 55 55 50 44 41 54 45 53 59 53 54 45UUPDATESYS	

这里就得到了 explorer.exe 的全路径。

10001B76	75 E2	JNZ SHORT 10001B61	AntiDebu.10001B61
10001B7F	8BBD 60FBFFFF	MOV EDI,DWORD PTR SS:[EBP-4A0]	
10001B85	8BBD 08C20010	MOV ECX,DWORD PTR DS:[1000C208]	
10001B8B	890F	MOV DWORD PTR DS:[EDI],ECX	
10001B8D	8B15 0CC20010	MOV EDX,DWORD PTR DS:[1000C20C]	
10001B93	8957 04	MOV DWORD PTR DS:[EDI+4],EDX	
10001B96	A1 10C20010	MOV EAX,DWORD PTR DS:[1000C210]	
10001B9B	8947 08	MOV DWORD PTR DS:[EDI+8],EAX	
10001B9E	8A0D 14C20010	MOV CL,BYTE PTR DS:[1000C214]	
10001BA4	884F 0C	MOV BYTE PTR DS:[EDI+C],CL	
10001BA7	B8 E7DBB4C0	MOV EAX,C0B4DBE7	
10001BAC	3D 78234C40	CMP EAX,404C2378	
10001BB1	74 07	JE SHORT 10001BB8	AntiDebu.10001BB8
10001BB3	B8 BB1B0010	MOV EAX,10001BBB	
10001BB8	FF07	JMP NEAR EAX	
EAX=C0B4DBE7			
Address	Hex dump	ASCII	
0012EFAA	00 00 00 00 00 00 43 3A 5C 57 49 4E 44 4F 57 53C:\WINDOWS	
0012EFBA	5C 45 58 50 4C 4F 52 45 52 2E 45 58 45 00 7D 01	\EXPLORER.EXE.00	
0012F00A	78 01 A0 00 A1 00 A2 00 A3 00 A4 00 A5 00 A6 00	x00.i.0.0.0.0.0.	
0012F01A	A7 00 A8 00 A9 00 AA 00 AB 00 AC 00 AD 00 AE 00	0.0.0.0.0.0.0.0.	

好了,我们来看看接下来要干什么。

10001BB8	FFE0	JMP NEAR EAX	
10001BBA	DF	???	Unknown command
10001BBB	8095 B0FCFFFF	LEA EDX,DWORD PTR SS:[EBP-350]	
10001BC1	8995 58FBFFFF	MOV DWORD PTR SS:[EBP-4A8],EDX	
10001BC7	8B85 58FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4A8]	
10001BCD	40	INC EAX	
10001BCE	8985 54FBFFFF	MOV DWORD PTR SS:[EBP-4AC],EAX	
10001BD4	8B8D 58FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4A8]	
10001BDA	8A11	MOV DL,BYTE PTR DS:[ECX]	
10001BDC	8895 53FBFFFF	MOV BYTE PTR SS:[EBP-4AD],DL	
10001BE2	FF85 58FBFFFF	INC DWORD PTR SS:[EBP-4A8]	
10001BE8	80BD 53FBFFFF	CMP BYTE PTR SS:[EBP-4AD],0	
10001BEF	75 E3	JNZ SHORT 10001BD4	AntiDebu.10001BD4
10001BF1	8B85 58FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4A8]	
10001BF7	2B85 54FBFFFF	SUB EAX,DWORD PTR SS:[EBP-4AC]	
10001BFD	8985 4CFBFFFF	MOV DWORD PTR SS:[EBP-4B4],EAX	
10001C03	8B8D 4CFBFFFF	MOV ECX,DWORD PTR SS:[EBP-4B4]	
10001C09	51	PUSH ECX	
10001C0A	8095 B0FCFFFF	LEA EDX,DWORD PTR SS:[EBP-350]	
10001C10	52	PUSH EDX	
10001C11	FF15 74C10010	CALL NEAR DWORD PTR DS:[1000C174]	USER32.CharUpperBuff
Stack address=0012EDB0, (ASCII "C:\WINDOWS\explorer.exe")			
EDX=5245524F			
Address	Hex dump	ASCII	
0012EC90	24 02 00 00 01 00 00 00 7C 0C 00 00 FF FF 00 00	\$0..0...!... ..	
0012ECA0	FF FF 00 00 00 00 00 01 00 F0 0F 00 00 00 000.-*....0	
0012ECB0	65 78 70 6C 6F 72 65 72 2E 65 78 65 00 00 00 00	explorer.exe....	
0012ECC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ECD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ECE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ECF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED50	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED80	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012ED90	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012EDA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012EDB0	43 3A 5C 57 49 4E 44 4F 57 53 5C 65 78 70 6C 6F	C:\WINDOWS\explo	
0012EDC0	72 65 72 2E 65 78 65 00 00 00 00 00 00 00 00 00	rer.exe.....	
0012EDD0	AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA	

下面将开始与快照中的进程路径进行比较。

10001C03	8B8D 4CFBFFFF	MOV ECX,DWORD PTR SS:[EBP-4B4]	
10001C09	51	PUSH ECX	
10001C0A	8095 B0FCFFFF	LEA EDX,DWORD PTR SS:[EBP-350]	
10001C10	52	PUSH EDX	
10001C11	FF15 74C10010	CALL NEAR DWORD PTR DS:[1000C174]	USER32.CharUpperBuffA
10001C17	B8 C9E95787	MOV EAX,8757E9C9	
10001C1C	3D 08FEAE79	CMP EAX,79AEFE08	
10001C21	74 07	JE SHORT 10001C2A	AntiDebu.10001C2A
10001C23	B8 2B1C0010	MOV EAX,10001C2B	
10001C28	FFE0	JMP NEAR EAX	

这里我们到了 CharUpperBuffA 这个 API 函数的调用处。这里是将目标缓冲区中的字符串由小写转大写。

10001C17	B8 C9E95787	MOV EAX,8757E9C9	
10001C1C	3D 08FEAE79	CMP EAX,79AEFE08	
10001C21	74 07	JE SHORT 10001C2A	AntiDebu.10001C2A
10001C23	B8 2B1C0010	MOV EAX,10001C2B	
10001C28	FFE0	JMP NEAR EAX	
10001C2A	D9	???	Unknown command
10001C2B	8085 F0FEFFFF	LEA EAX,DWORD PTR SS:[EBP-110]	
10001C31	8985 48FBFFFF	MOV DWORD PTR SS:[EBP-4B8],EAX	
10001C37	8B8D 48FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4B8]	
10001C3D	41	INC ECX	
10001C3E	898D 44FBFFFF	MOV DWORD PTR SS:[EBP-4BC],ECX	
10001C44	8B95 48FBFFFF	MOV EDX,DWORD PTR SS:[EBP-4B8]	
EAX=00000017			
Address	Hex dump	ASCII	
0012EDB0	43 3A 5C 57 49 4E 44 4F 57 53 5C 45 58 50 4C 4F	C:\WINDOWS\EXPLO	
0012EDC0	52 45 52 2E 45 58 45 00 00 00 00 00 00 00 00 00	RER.EXE.....	
0012EDD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012EDE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

这里没看出什么特别的,就是将遍历到的进程全路径小写转大写。然后通过 GetWindowsDirectory 获取 windows 目录,接着与 explorer.exe 进行字符串连接,然后进行比较。

这里我们可以看到接下来这个模块是 `ntdll.dll`。其模块标示符为 `0x01`，PID 为 `0xC7C`，以及其他相关的模块信息。

10001AB6	8B8D C4FDFFFF	MOV ECX,DWORD PTR SS:[EBP-23C]
10001ABC	3B8D 98FBFFFF	CMP ECX,DWORD PTR SS:[EBP-468]
10001AC2	0F85 CB020000	JNZ 10001D93
10001AC8	B8 41B2CB6C	MOV EAX,6CCBB241
10001ACD	3D C8932394	CMP EAX,942393C8
10001AD2	74 07	JE SHORT 10001AD8
10001AD4	B8 DC1A0010	MOV EAX,10001ADC
10001AD9	FFEB	JMP NEAR EAX
10001ADB	D968 05	FLOP WORD PTR DS:[EAX+5]
10001ADE	0100	ADD DWORD PTR DS:[EAX],EAX
10001AE0	008D 95F0FEFF	ADD BYTE PTR SS:[EBP+FFFEF095],CL
10001AE6	FF52 FF	CALL NEAR DWORD PTR DS:[EDX-1]
10001AE9	15 58C00010	ADC EAX,1000C058
10001AEE	B8 23C06E33	MOV EAX,336EC023
10001AF3	3D 586E86CD	CMP EAX,CD866E58
10001AF8	74 07	JE SHORT 10001B01
10001AFA	B8 021B0010	MOV EAX,10001B02
10001AFF	FFEB	JMP NEAR EAX
10001B01	DB	22
10001B02	8D85 F0FEFFFF	LEA EAX,DWORD PTR SS:[EBP-110]
10001B08	83C0 FF	ADD EAX,-1
10001B0B	9885 68FBFFFF	MOV DWORD PTR SS:[EBP-498],EAX
10001B11	8B8D 68FBFFFF	MOV ECX,DWORD PTR SS:[EBP-498]
10001B17	8A51 01	MOV DL,BYTE PTR DS:[ECX+1]
10001B1A	8895 67FBFFFF	MOV BYTE PTR SS:[EBP-499],DL
10001B20	FF85 68FBFFFF	INC DWORD PTR SS:[EBP-498]
10001B26	00BD 67FBFFFF	CMP BYTE PTR SS:[EBP-499],0
10001B2D	75 E2	JNZ SHORT 10001B11
10001B2F	8BBD 68FBFFFF	MOV EDI,DWORD PTR SS:[EBP-498]
10001B35	66:A1 04C20010	MOV AX,WORD PTR DS:[1000C204]
10001B38	66:8907	MOV WORD PTR DS:[EDI],AX
10001B3E	B8 05CE11FA	MOV EAX,FA11CE05
10001B43	3D E848E906	CMP EAX,6E948E8
10001B48	74 07	JE SHORT 10001B51
Stack SS:[0012EC98]=00000C7C		
ECX=00000C7C		
Address	Hex dump	ASCII

接下来继续进行,依次判断 explorer.exe 的所有模块。

10001ADC	68 05010000	PUSH 105
10001AE1	8D95 F0FEFFFF	LEA EDI,DWORD PTR SS:[EBP-110]
10001AE7	52	PUSH EDI
10001AE8	FF15 58C00010	CALL NEAR DWORD PTR DS:[1000C058]
10001AEE	B8 23C06E33	MOV EAX,336EC023
10001AF3	3D 586E86CD	CMP EAX,CD866E58
10001AF8	74 07	JE SHORT 10001B01
10001AFA	B8 021B0010	MOV EAX,10001B02
10001AFF	FFEB	JMP NEAR EAX

这里再次到了 GetWindowsDirectoryA 的调用处。

10001B51	8B95 60FBFFFF	MOV EDI,DWORD PTR SS:[EBP-4A0]
10001B67	8A42 01	MOV AL,BYTE PTR DS:[EDI+1]
10001B6D	8B95 5FEFEFF	MOV BYTE PTR SS:[EBP-4A1],AL
DS:[1000C058]=7C82293B (kernel32.GetWindowsDirectoryA)		
Address	Hex dump	ASCII
0012EC90	24 02 00 00 01 00 00 00 7C 0C 00 00 FF FF 00 00	\$.0...i... ..
0012ECA0	FF FF 00 00 00 00 91 7C 00 60 00 00 00 00 91 7C	...e!'.0...e!
0012ECB0	6E 74 64 6C 6C 2E 64 6C 6C 00 78 65 00 00 00 00	ntdll.dll.xe...
0012ECC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ECD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ECE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ECF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED50	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED80	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ED90	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EDA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EDB0	43 3A 5C 57 49 4E 44 4F 57 53 5C 73 79 73 74 65	C:\WINDOWS\sys...
0012EDC0	6D 33 32 5C 6E 74 64 6C 6C 2E 64 6C 6C 00 00 00	32\ntdll.dll...
0012EDD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EDE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EDF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EC1C	0012EFF0	Buffer = 0012EFF0
0012EC20	00000105	BufSize = 105 (261,)
0012EC24	0012FD30	
0012EC28	0012EC38	
0012EC2C	00000000	
0012EC30	7C8088EC	kernel32.lstrcpw
0012EC34	76B029F9	RETURN to WINMM.76B029F9 from W
0012EC38	00000022	
0012EC3C	00000017	
0012EC40	0012F90C	
0012EC44	0012EFF1	ASCII ":\WINDOWS\EXPLORER.EXE"
0012EC48	0012F008	
0012EC4C	00000017	
0012EC50	00000001	
0012EC54	0012ED81	ASCII ":\WINDOWS\system32\ntdll
0012EC58	0012EDC8	ASCII "i.dll"
0012EC5C	0061006D	
0012EC60	0012EFFB	ASCII "EXPLORER.EXE"
0012EC64	0064002E	
0012EC68	0012EFAA	ASCII "EXPLORER.EXE"
0012EC6C	00000001	
0012EC70	00000001	
0012EC74	5050FFFF	
0012EC78	0012EEEC	

我们继续往下。

10001ADC	68 05010000	PUSH 105	
10001AE1	8D95 F0FEFFFF	LEA EDX,DWORD PTR SS:[EBP-110]	
10001AE7	52	PUSH EDX	
10001AE8	FF15 58C00010	CALL NEAR DWORD PTR DS:[1000C058]	kernel32.GetWindowsDirectoryA
10001AE9	B8 23C06E53	MOV EAX,336EC023	
10001AF3	3D 586E86C0	CMP EAX,CD866E58	
10001AF9	74 07	JE SHORT 10001B81	AntiDebu.10001B01
10001AFA	B8 021B0010	MOV EAX,10001B02	
10001AFF	FFE0	JMP NEAR EAX	
10001B01	DB	???	
10001B02	8D85 F0FEFFFF	LEA EAX,DWORD PTR SS:[EBP-110]	
10001B08	83C0 FF	ADD EAX,-1	Unknown command
10001B0B	8985 68FBFFFF	MOV DWORD PTR SS:[EBP-498],EAX	
10001B11	8B8D 68FBFFFF	MOV ECX,DWORD PTR SS:[EBP-498]	
10001B17	8A51 01	MOV DL,BYTE PTR DS:[ECX+1]	
10001B1A	8B95 67FBFFFF	MOV BYTE PTR SS:[EBP-499],DL	
10001B20	FF85 68FBFFFF	INC DWORD PTR SS:[EBP-498]	
10001B26	80BD 67FBFFFF	CMP BYTE PTR SS:[EBP-499],0	
10001B2D	75 E2	JNZ SHORT 10001B11	AntiDebu.10001B11
10001B2F	8BBD 68FBFFFF	MOV EDI,DWORD PTR SS:[EBP-498]	
10001B35	66:81 04C20010	MOV AX,WORD PTR DS:[1000C204]	
10001B38	66:8907	MOV WORD PTR DS:[EDI],AX	
10001B3E	B8 05CE11FA	MOV EAX,FA11CE05	
10001B43	3D E848E906	CMP EAX,6E948E8	
10001B48	74 07	JE SHORT 10001B51	AntiDebu.10001B51
10001B4A	B8 521B0010	MOV EAX,10001B52	
10001B4F	FFE0	JMP NEAR EAX	
10001B51	DD	???	
10001B52	8D8D F0FEFFFF	LEA ECX,DWORD PTR SS:[EBP-110]	
10001B58	83C1 FF	ADD ECX,-1	Unknown command
10001B5B	898D 60FBFFFF	MOV DWORD PTR SS:[EBP-4A0],ECX	
10001B61	8B95 60FBFFFF	MOV EDX,DWORD PTR SS:[EBP-4A0]	
10001B67	8A42 01	MOV AL,BYTE PTR DS:[EDX+1]	
10001B69	8B85 5FBFFFFF	MOV BYTE PTR SS:[EBP-4A1],AL	
EAX=0000000A			
Address	Hex dump	ASCII	
0012EFFF	43 3A 5C 57 49 4E 44 4F 57 53 00 45 58 50 4C 4F	C:\WINDOWS\EXPLO	0012EC24 0012FD30
0012F000	52 45 52 2E 45 58 45 00 7D 01 78 01 A0 00 A1 00	RER.EXE.30x0A.i.	0012EC28 0012EC38
0012F010	02 0A 03 00 04 0A 05 00 06 0A 07 00 08 0A 09 0A	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0012EC2C 00000000
			0012EC30 7C80B8EC kernel32.L

这里又是获取 Windows 的所在目录,重复前面的比较步骤,这里我们就不再赘述了,继续跟踪。

10001C80	50	PUSH EAX	
10001C81	8D8D B0FCFFFF	LEA ECX,DWORD PTR SS:[EBP-350]	
10001C87	51	PUSH ECX	
10001C88	E8 433C0000	CALL 100058D0	AntiDebu.100058D0
10001C8D	83C4 0C	ADD ESP,0C	
10001C90	85C0	TEST EAX,EAX	
10001C92	0F84 FB000000	JE 10001D93	AntiDebu.10001D93
10001C98	B8 9C7E4CB1	MOV EAX,B14C7E9C	
10001C9D	3D E045C34F	CMP EAX,4FC345E0	
10001CA2	74 07	JE SHORT 10001CAB	AntiDebu.10001CAB
10001CA4	B8 AC1C0010	MOV EAX,10001CAC	
10001CA9	FFE0	JMP NEAR EAX	
10001CAB	DC8D 95B0FCFF	FHUL QWORD PTR SS:[EBP+FFFCB095]	
10001CB1	FF52 E8	CALL NEAR DWORD PTR DS:[EDX+18]	
10001CB4	09 F5FFFF	ENTER 0FFFF,0FF	
10001CB8	83C4 04	ADD ESP,4	
10001CBB	B8 7E8CEFF7	MOV EAX,77EF8C7E	
10001CBF	3D 7B202683	CMP EAX,83262070	
100058D0-AntiDebu.100058D0			
Address	Hex dump	ASCII	
0012EDB0	43 3A 5C 57 49 4E 44 4F 57 53 5C 53 59 53 54 45	C:\WINDOWS\SVSTE	0012EC18 0012EDB0 ASCII "C:\WINDOWS\SVSTE\NTDLL.DLL"
0012EDC0	4D 33 32 5C 4E 54 44 4C 4C 2E 44 4C 4C 00 00 00	M32\NTDLL.DLL...	0012EC1C 0012EDF0 ASCII "C:\WINDOWS\EXPLORER.EXE"
0012EDD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012EC20 00000017
			0012EC24 0012FD30
			0012EC28 0012FD30

我们再次到了字符串比较函数处。这里两个字符串明显不相等。

Registers (FPU)	
EAX	00000001
ECX	00000001
EDX	00000017
EBX	10000000 AntiDet
ESP	0012EC18
EBP	0012F100
ESI	00000001
EDI	0012EFFF ASCII "
EIP	10001C8D AntiDet
C 1	ES 0023 32bit 0
D 0	CS 001D 32bit 0

EAX 返回 1,说明比较的两个字符串不相等。下面的跳转不成立。

10001C80	50	PUSH EAX	
10001C81	8D8D B0FCFFFF	LEA ECX,DWORD PTR SS:[EBP-350]	
10001C87	51	PUSH ECX	
10001C88	E8 433C0000	CALL 100058D0	AntiDebu.100058D0
10001C8D	83C4 0C	ADD ESP,0C	
10001C90	85C0	TEST EAX,EAX	
10001C92	0F84 FB000000	JE 10001D93	AntiDebu.10001D93
10001C98	B8 9C7E4CB1	MOV EAX,B14C7E9C	
10001C9D	3D E045C34F	CMP EAX,4FC345E0	
10001CA2	74 07	JE SHORT 10001CAB	AntiDebu.10001CAB
10001CA4	B8 AC1C0010	MOV EAX,10001CAC	
10001CA9	FFE0	JMP NEAR EAX	
10001CAB	DC8D 95B0FCFF	FHUL QWORD PTR SS:[EBP+FFFCB095]	

下面的代码是经过混淆了的。(PS:也就是常说的花指令)

10001C8D	83C4 0C	ADD ESP,0C	
10001C90	85C0	TEST EAX,EAX	
10001C92	0F84 FB000000	JE 10001D93	AntiDebu.10001D93
10001C98	B8 9C7E4CB1	MOV EAX,B14C7E9C	
10001C9D	3D E045C34F	CMP EAX,4FC345E0	
10001CA2	74 07	JE SHORT 10001CAB	AntiDebu.10001CAB
10001CA4	B8 AC1C0010	MOV EAX,10001CAC	
10001CA9	FFEB	JMP NEAR EAX	
10001CAB	DC8D 95B0FCFF	FMUL QWORD PTR SS:[EBP+FFFCB095]	
10001CB1	FF52 E8	CALL NEAR DWORD PTR DS:[EDX-18]	
10001CB4	C8 F5FFFF	ENTER 0FFFF5,0FF	
10001CB8	83C4 04	ADD ESP,4	
10001CBB	B8 7E8CE777	MOV EAX,77EF8C7E	
10001CC0	3D 70202689	CMP EAX,89262070	
10001CC5	74 07	JE SHORT 10001CCE	AntiDebu.10001CCE
10001CC7	B8 CF1C0010	MOV EAX,10001CCF	
10001CCC	FFEB	JMP NEAR EAX	
10001CCE	DEC7	FADD ST(7),ST	
10001CD0	8538	TEST DWORD PTR DS:[EAX],EDI	
10001CD2	FB	ST1	
10001CD3	FFFF	?	Unknown command
10001CD5	08C2	OR DL,AL	
10001CD7	0010	ADD BYTE PTR DS:[EAX],DL	
10001CD9	8D85 B0FCFFFF	LEA EAX,DWORD PTR SS:[EBP-350]	
10001CDF	8985 34FBFFFF	MOV DWORD PTR SS:[EBP-4CC],EAX	
10001CE5	8B8D 34FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4CC]	
10001CEB	8A11	MOV DL,BYTE PTR DS:[ECX]	
10001CED	8B95 33FBFFFF	MOV BYTE PTR SS:[EBP-4CD],DL	
10001CF3	8B85 38FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4C8]	
10001CF9	3A10	CMP DL,BYTE PTR DS:[EAX]	
10001CFB	75 46	JNZ SHORT 10001D43	AntiDebu.10001D43
10001CFD	80BD 33FBFFFF	CMP BYTE PTR SS:[EBP-4CD],0	
10001D04	74 31	JE SHORT 10001D37	AntiDebu.10001D37
10001D06	8B8D 34FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4CC]	
10001D0C	8A51 01	MOV DL,BYTE PTR DS:[ECX+1]	
10001D0F	8B95 32FBFFFF	MOV BYTE PTR SS:[EBP-4CE],DL	
10001D15	8B85 38FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4C8]	
10001D1B	3A50 01	CMP DL,BYTE PTR DS:[EAX+1]	
10001D1E	75 23	JNZ SHORT 10001D43	AntiDebu.10001D43
10001D20	8385 34FBFFFF	ADD DWORD PTR SS:[EBP-4CC],2	
10001D27	8385 38FBFFFF	ADD DWORD PTR SS:[EBP-4C8],2	
10001D2E	80BD 32FBFFFF	CMP BYTE PTR SS:[EBP-4CE],0	
10001D35	75 AE	JNZ SHORT 10001CE5	AntiDebu.10001CE5
10001D37	C785 2CFBFFFF	MOV DWORD PTR SS:[EBP-4D4],0	
10001D41	EB 0B	JMP SHORT 10001D4E	AntiDebu.10001D4E
10001D43	1BC9	SBB ECX,ECX	
10001D45	83D9 FF	SBB ECX,-1	

Jump is NOT taken
10001D93=AntiDebu.10001D93

这里我们可以看到,首先将一个常量赋值给 EAX,然后与另一个常量进行比较,这里由于两个常量并不相等,所有下面 JE 条件跳转永远不会成立。接下来又是将一个常量赋值给 EAX,大家注意到了我标红了的地址没?紧接着下面通过 JMP NEAR EAX 来跳转的刚刚标红的常量所表示地址处。这里可以看到这部分代码被混淆了,我们如何看到定位其真正要执行的代码处呢?我们选中标红了的地址,单击鼠标右键选择 Follow immediate constant(跟随立即数)。

10001C90	85C0	TEST EAX,EAX	
10001C92	0F84 FB000000	JE 10001D93	AntiDebu
10001C98	B8 9C7E4CB1	MOV EAX,B14C7E9C	
10001C9D	3D E045C34F	CMP EAX,4FC345E0	
10001CA2	74 07	JE SHORT 10001CAB	AntiDebu
10001CA4	B8 AC1C0010	MOV EAX,10001CAC	
10001CA9	FFEB	JMP NEAR EAX	Backup
10001CAB	DC8D 95B0FCFF	FMUL QWORD PTR	Copy
10001CB1	FF52 E8	CALL NEAR DWO	Binary
10001CB4	C8 F5FFFF	ENTER 0FFFF5,0	Assemble
10001CB8	83C4 04	ADD ESP,4	Space
10001CBB	B8 7E8CE777	MOV EAX,77EF8C7E	
10001CC0	3D 70202689	CMP EAX,89262070	
10001CC5	74 07	JE SHORT 10001CCE	Label
10001CC7	B8 CF1C0010	MOV EAX,10001CCF	:
10001CCC	FFEB	JMP NEAR EAX	;
10001CCE	DEC7	FADD ST(7),S	Comment
10001CD0	8538	TEST DWORD PTR	Breakpoint
10001CD2	FB	ST1	Run trace
10001CD3	FFFF	?	
10001CD5	08C2	OR DL,AL	
10001CD7	0010	ADD BYTE PTR	
10001CD9	8D85 B0FCFFFF	LEA EAX,DWORD	Follow immediate constant
10001CDF	8985 34FBFFFF	MOV DWORD PTR	
10001CF3	8B8D 34FBFFFF	MOV ECX,DWORD	

这样我们将能够不执行混淆过的代码,直接定位到实际的功能代码了。

10001CAC	8D95 B0FCFFFF	LEA EDX,DWORD PTR SS:[EBP-350]	
10001CB2	52	PUSH EDX	
10001CB3	E8 C8F5FFFF	CALL 10001280	AntiDebu.10001280
10001CB8	83C4 04	ADD ESP,4	
10001CBB	B8 7E8CEF77	MOV EAX,77EF8C7E	
10001CC0	3D 70202689	CMP EAX,89262070	
10001CC5	74 07	JE SHORT 10001CCE	AntiDebu.10001CCE
10001CC7	B8 CF1C0010	MOV EAX,10001CCF	
10001CCC	FFEB	JMP NEAR EAX	
10001CCE	DEC7	FADD ST(7),ST	
10001CD0	8538	TEST DWORD PTR DS:[EAX],EDI	
10001CD2	FB	STI	
10001CD3	FFFF	222	Unknown command
10001CD5	08C2	OR DL,AL	
10001CD7	0010	ADD BYTE PTR DS:[EAX],DL	

这里我们就能看到其实要执行的代码了。下一个 MOV 指令处同样通过这种方式来查看。

10001CAC	8D95 B0FCFFFF	LEA EDX,DWORD PTR SS:[EBP-350]	
10001CB2	52	PUSH EDX	
10001CB3	E8 C8F5FFFF	CALL 10001280	AntiDebu
10001CB8	83C4 04	ADD ESP,4	
10001CBB	B8 7E8CEF77	MOV EAX,77EF8C7E	
10001CC0	3D 70202689	CMP EAX,89262070	
10001CC5	74 07	JE SHORT 10001CCE	AntiDebu
10001CC7	B8 CF1C0010	MOV EAX,10001CCF	
10001CCC	FFEB	JMP NEAR EAX	
10001CCE	DEC7	FADD ST(7),ST	Backup
10001CD0	8538	TEST DWORD PTR DS:[EAX],EDI	Copy
10001CD2	FB	STI	Binary
10001CD3	FFFF	222	Assemble
10001CD5	08C2	OR DL,AL	Space
10001CD7	0010	ADD BYTE PTR DS:[EAX],DL	:
10001CD9	8D85 B0FCFFFF	LEA EAX,DWORD PTR SS:[EBP-350]	
10001CDF	8985 34FBFFFF	MOV DWORD PTR SS:[EBP-4CC],EAX	
10001CE5	8B8D 34FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4CC]	
10001CEB	8A11	MOV DL,BYTE PTR DS:[ECX]	
10001CED	8B95 33FBFFFF	MOV BYTE PTR SS:[EBP-4CD],DL	
10001CF3	8B85 38FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4C8]	
10001CF9	3A10	CMP DL,BYTE PTR DS:[EAX]	
10001CFB	75 46	JNZ SHORT 10001D43	Run trace
10001CFD	80BD 33FBFFFF	CMP BYTE PTR SS:[EBP-4CD],0	
10001D04	74 31	JE SHORT 10001D37	Follow immediate constant
10001D06	8B8D 34FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4CC]	
10001D0C	8A51 01	MOV DL,BYTE PTR DS:[ECX]	
10001D0F	8B95 32FBFFFF	MOV BYTE PTR SS:[EBP-4CD],DL	
10001D15	8B85 38FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4C8]	

10001CCF	C785 38FBFFFF	MOV DWORD PTR SS:[EBP-4C8],1000C208	ASCII "EXPLORER.EXE"
10001CD9	8D85 B0FCFFFF	LEA EAX,DWORD PTR SS:[EBP-350]	
10001CDF	8985 34FBFFFF	MOV DWORD PTR SS:[EBP-4CC],EAX	
10001CE5	8B8D 34FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4CC]	
10001CEB	8A11	MOV DL,BYTE PTR DS:[ECX]	
10001CED	8B95 33FBFFFF	MOV BYTE PTR SS:[EBP-4CD],DL	
10001CF3	8B85 38FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4C8]	
10001CF9	3A10	CMP DL,BYTE PTR DS:[EAX]	
10001CFB	75 46	JNZ SHORT 10001D43	AntiDebu.10001D43
10001CFD	80BD 33FBFFFF	CMP BYTE PTR SS:[EBP-4CD],0	
10001D04	74 31	JE SHORT 10001D37	AntiDebu.10001D37
10001D06	8B8D 34FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4CC]	

我们再次定位到了实际代码处,通过这样方式就可以不执行混淆过的代码直接定位实际代码了,比较方便。

我们继续跟踪。

10001340	8B40 F8	MOV ECX,DWORD PTR SS:[EBP-8]	
10001343	8B75 FC	MOV ESI,DWORD PTR SS:[EBP-4]	
10001346	8B7D 08	MOV EDI,DWORD PTR SS:[EBP+8]	
10001349	8BD1	MOV EDX,ECX	
1000134B	C1E9 02	SHR ECX,2	
1000134E	F3:A5	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[EDI]	
10001350	8BCA	MOV ECX,EDX	
10001352	83E1 03	AND ECX,3	
10001355	F3:A4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[EDI]	
10001357	B8 7AB44716	MOV EAX,1647B47A	
1000135C	3D 501905EA	CMP EAX,EA051950	
10001361	74 07	JE SHORT 1000136A	AntiDebu.1000136A
10001363	B8 6B130010	MOV EAX,1000136B	
10001368	FFEB	JMP NEAR EAX	
1000136A	DAEB	FISUBR EBX	Illegal use of register
1000136C	14 B8	ADC AL,0B8	
1000136E	4D	DEC EBP	
1000136F	49	DEC ECX	
10001370	3C 40	CMP AL,40	
10001372	3D 286119C0	CMP EAX,C0196128	
10001377	74 07	JE SHORT 10001380	AntiDebu.10001380
10001379	B8 81130010	MOV EAX,10001381	
1000137E	FFEB	JMP NEAR EAX	
10001380	DD5F 5E	FSTP QWORD PTR DS:[EDI+5E]	
10001383	8BE5	MOV ESP,EBP	
10001385	5D	POP EBP	
10001386	C3	RETN	
10001387	CC	INT3	
10001388	CC	INT3	
10001389	CC	INT3	
1000138A	CC	INT3	
1000138B	CC	INT3	
1000138C	CC	INT3	
1000138D	CC	INT3	
1000138E	CC	INT3	
1000138F	CC	INT3	
10001390	55	PUSH EBP	
10001391	8BEC	MOV EBP,ESP	
10001393	83EC 14	SUB ESP,14	
10001396	B8 106F1835	MOV EAX,35186F10	
1000139B	3D 80D416CB	CMP EAX,CB16D480	
100013A0	74 07	JE SHORT 100013A9	AntiDebu.100013A9
100013A2	B8 BA130010	MOV EAX,100013AA	
100013A7	FFEB	JMP NEAR EAX	
100013A9	D8C6	FADD ST,ST(6)	
100013AB	45	INC EBP	
100013AC	E3	PREFIX REP.	Superfluous prefix
EAX=10001340 (AntiDebu.10001340)			
Address	Hex dump	ASCII	
0012EDBA	5C 53 59 53 54 45 40 33 32 5C 4E 54 44 4C 4C 2E	SYSTEM32\NTDLL.	001
0012EDCA	44 4C 4C 00 00 00 00 00 00 00 00 00 00 00 00	DLL.....	001
0012EDDA	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		001

这里我们可以看到拼接字符串得到 C:\WINDOWS\SYSTEM32\NTDLL.DLL,我们继续。

10001CCF	C795 3FBFFFFF	MOV DWORD PTR SS:[EBP-4C8],1000C208	ASCII "EXPLORER.EXE"
10001CD9	8D85 B0FCFFFF	LEA EAX,DWORD PTR SS:[EBP-350]	
10001CDF	8985 34FBFFFF	MOV DWORD PTR SS:[EBP-4CC],EAX	
10001CE5	8B8D 34FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4CC]	
10001CEB	8A11	MOV DL,BYTE PTR DS:[ECX]	
10001CED	8B95 33FBFFFF	MOV BYTE PTR SS:[EBP-4CD],DL	
10001CF3	8B85 38FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4C8]	
10001CF9	3A10	CMP DL,BYTE PTR DS:[EAX]	
10001CFB	75 46	JNZ SHORT 10001D43	AntiDebu.10001D43

这里再次看到了 explorer.exe 这个名称。

10001CDF	8985 34FBFFFF	MOV DWORD PTR SS:[EBP-4CC],EAX	
10001CE5	8B8D 34FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4CC]	
10001CEB	8A11	MOV DL,BYTE PTR DS:[ECX]	
10001CED	8B95 33FBFFFF	MOV BYTE PTR SS:[EBP-4CD],DL	
10001CF3	8B85 38FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4C8]	
10001CF9	3A10	CMP DL,BYTE PTR DS:[EAX]	
10001CFB	75 46	JNZ SHORT 10001D43	AntiDebu.
10001CFD	80BD 33FBFFFF	CMPL BYTE PTR SS:[EBP-4CD],0	AntiDebu.
10001D04	74 31	JE SHORT 10001D35	
10001D06	8B8D 34FBFFFF	MOV ECX,DWORD PTR SS:[EBP-4CC]	
10001D0C	8A51 01	MOV DL,BYTE PTR DS:[ECX+1]	
10001D0F	8B95 32FBFFFF	MOV BYTE PTR SS:[EBP-4CE],DL	
10001D15	8B85 38FBFFFF	MOV EAX,DWORD PTR SS:[EBP-4C8]	
10001D1B	3A50 01	CMP DL,BYTE PTR DS:[EAX+1]	
10001D1E	75 23	JNZ SHORT 10001D43	AntiDebu.
10001D20	8395 34FBFFFF	ADD DWORD PTR SS:[EBP-4CC],2	
10001D27	8385 38FBFFFF	ADD DWORD PTR SS:[EBP-4C8],2	
10001D2E	80BD 32FBFFFF	CMPL BYTE PTR SS:[EBP-4CE],0	
10001D35	75 4E	JNZ SHORT 10001D4E	AntiDebu.
10001D37	C795 2CFBFFFF	MOV DWORD PTR SS:[EBP-4D4],0	AntiDebu.
10001D41	EB 0B	JMP SHORT 10001D4E	
10001D43	1BC9	SBB ECX,ECX	
10001D45	83D9 FF	SBB ECX,-1	
10001D48	839D 2CFBFFFF	ADD DWORD PTR SS:[EBP-4D4],ECX	
10001D4E	8B95 2CFBFFFF	MOV EDX,DWORD PTR SS:[EBP-4D4]	
10001D54	8395 28FBFFFF	MOV DWORD PTR SS:[EBP-4D8],EDX	
10001D5A	838D 28FBFFFF	CMPL DWORD PTR SS:[EBP-4D8],0	
10001D61	75 30	JNZ SHORT 10001D6B	AntiDebu.
10001D63	B8 5121E4A1	MOV EAX,A1E42151	
10001D68	3D 48683A5F	CMP EAX,5F3A6848	
10001D6D	74 47	JE SHORT 10001D75	AntiDebu.
10001D6F	B8 771D0010	MOV EAX,10001D77	
10001D74	FFEB	JMP NEAR EAX	
10001D76	D96A 00	FILDQ WORD PTR DS:[EDX]	
DS:[1000C208]=45 ("E")			
DL=4E ("N")			

下面就将 EXPLORER.EXE 的全路径与 NTDLL.DLL 的全路径进行比较。

10001054	8395 28FBFFFF	MOV DWORD PTR SS:[EBP-408],EAX	
1000105A	83B0 28FBFFFF	CMP DWORD PTR SS:[EBP-408],0	
10001061	75 30	JNZ SHORT 10001093	AntiDebu.10001093
10001063	B8 5121E4A1	MOV EAX,A1E42151	
10001068	3D 48683A5F	CMP EAX,5F3A6848	
1000106D	74 07	JE SHORT 10001076	AntiDebu.10001076
1000106F	B8 771D0010	MOV EAX,10001D77	
10001074	FFE0	JMP NEAR EAX	
10001076	D96A 00	FLDCW WORD PTR DS:[EDX]	
10001079	FF15 0CC00010	CALL NEAR DWORD PTR DS:[1000C00C]	kernel32.ExitProcess
1000107F	B8 332F8768	MOV EAX,68872F33	
10001084	3D 08429D98	CMP EAX,989D42D8	
10001089	74 07	JE SHORT 10001092	AntiDebu.10001092
1000108B	B8 931D0010	MOV EAX,10001D93	
10001090	FFE0	JMP NEAR EAX	
10001092	DB	???	Unknown command
10001093	8D85 90FBFFFF	LEA EAX,DWORD PTR SS:[EBP-470]	
10001099	50	PUSH EAX	
1000109A	8B80 BCF0FFFF	MOV ECX,DWORD PTR SS:[EBP-244]	
1000109B	51	PUSH ECX	
100010A1	E8 4A350000	CALL 100052F0	<JMP.&KERNEL32.Module32Next>
100010A6	85C0	TEST EAX,EAX	
100010A8	0F85 F4FCFFFF	JNC 100010A2	AntiDebu.100010A2
100010AE	B8 D95870BC	MOV EAX,BC7058D9	
100010B3	3D 88D2C544	CMP EAX,44C5D288	
100010B8	74 07	JE SHORT 100010C1	AntiDebu.100010C1
100010BA	B8 C21D0010	MOV EAX,10001DC2	
100010BE	FFE0	JMP NEAR EAX	

两者并不相等,并且会跳过了中间的 ExitProcess,继续。

10001084	3D 08429D98	CMP EAX,989D42D8	
10001089	74 07	JE SHORT 10001092	AntiDebu.10001092
1000108B	B8 931D0010	MOV EAX,10001D93	
10001090	FFE0	JMP NEAR EAX	
10001092	DB	???	Unknown command
10001093	8D85 90FBFFFF	LEA EAX,DWORD PTR SS:[EBP-470]	
10001099	50	PUSH EAX	
1000109A	8B80 BCF0FFFF	MOV ECX,DWORD PTR SS:[EBP-244]	
1000109B	51	PUSH ECX	
100010A1	E8 4A350000	CALL 100052F0	<JMP.&KERNEL32.Module32Next>
100010A6	85C0	TEST EAX,EAX	
100010A8	0F85 F4FCFFFF	JNC 100010A2	AntiDebu.100010A2
100010AE	B8 D95870BC	MOV EAX,BC7058D9	
100010B3	3D 88D2C544	CMP EAX,44C5D288	
100010B8	74 07	JE SHORT 100010C1	AntiDebu.100010C1
100010BA	B8 C21D0010	MOV EAX,10001DC2	
100010BE	FFE0	JMP NEAR EAX	

我们看到这里继续遍历其他模块。说实话这部分我没看出有多大意义。

我们继续耐心跟吧。

10001C81	8D8D B0FCFFFF	LEA ECX,DWORD PTR SS:[EBP-350]	
10001C87	51	PUSH ECX	
10001C88	E8 433C0000	CALL 100058D0	AntiDebu.100058D0
10001C8D	83C4 0C	ADD ESP,0C	
10001C90	85C0	TEST EAX,EAX	
10001C92	0F84 FB000000	JE 10001093	AntiDebu.10001093
10001C98	B8 9C7F4C81	MOV EAX,B14C7F9C	

0012EC18	0012EDB0	ASCII "C:\WINDOWS\SYSTEM32\KERNEL32.DLL"
0012EC1C	0012EFF0	ASCII "C:\WINDOWS\EXPLORER.EXE"
0012EC20	00000017	
0012EC24	0012FD30	
0012EC28	00000001	

我们可以看到继续一个模块一个模块的遍历,都与 EXPLORER.EXE 的全路径进行比较。只有首次比较是相等的,其余的比较都不相等。

那么为了解决时间,我们直接给进行字符串比较的 CALL 处设置硬件执行断点。

0012EC18	0012EDB0	ASCII "C:\WINDOWS\SYSTEM32\MSVCRT.DLL"
0012EC1C	0012EFF0	ASCII "C:\WINDOWS\EXPLORER.EXE"
0012EC20	00000017	
0012EC24	0012FD30	
0012EC28	00000001	

这里我们可以通过 PUPE 查看 explorer.exe 进程的模块列表信息对照的看。

Caja de herramientas		
Proceso: explorer.exe		
Módulos del Proceso	Dirección (Hex)	Tamaño (Hex)
c:\windows\explorer.exe	01000000	000FF000
c:\windows\system32\ntdll.dll	7C910000	000B6000
c:\windows\system32\kernel32.dll	7C800000	00101000
c:\windows\system32\msvcrt.dll	77BE0000	00058000
c:\windows\system32\advapi32.dll	77DA0000	000AC000

[illegible]

1000569A	CC	INT3	
1000569B	3B0D BC050110	CMP ECX,DWORD PTR DS:[100105BC]	
100056A1	75 01	JNZ SHORT 100056A4	AntiDebu.100056A4
100056A3	C3	RETN	
100056A4	E9 C1FFFFFF	JMP 1000566A	AntiDebu.1000566A
100056A5	F2	RUNTIME_ERROR	

我们可以看到这里 ExitProcess 又被跳过了。继续往下跟。

100026D6	68 2C010000	PUSH 12C	
100026D8	8D95 1CFCFFFF	LEA EDX,DWORD PTR SS:[EBP-3E4]	
100026E1	52	PUSH EDX	
100026E2	6A 00	PUSH 0	
100026E4	FF15 88C00010	CALL NEAR DWORD PTR DS:[1000C088]	kernel32.GetModuleFileNameA
100026E8	B8 E6E54A28	MOV EAX,284AE5E6	
100026EF	3D B0E183D8	CMP EAX,D883E1B0	
100026F4	74 07	JE SHORT 100026FD	AntiDebu.100026FD

这里调用 GetModuleFileNameA。

Address	Hex dump	ASCII
0012F5EC	43 3A 5C 44 6F 63 75 6D 65 6E 74 73 20 61 6E 64	C:\Documents and
0012F5FC	20 53 65 74 74 69 6E 67 73 5C 52 69 63 61 72 64	Settings\Ricard
0012F60C	6F 5C 45 73 63 72 69 74 6F 72 69 6F 5C 43 4F 4E	o\Escritorio\CON
0012F61C	43 55 52 53 4F 20 38 37 5C 63 61 72 70 65 74 61	CURSO 87\carpet
0012F62C	20 73 69 6E 20 74 2B A1 74 75 6C 6F 5C 50 61 74	sin t+itulo\Pat
0012F63C	72 69 63 68 6F 72 69 5C 50 61 74 72 69 63 68 2E	rickori\Patrick.
0012F64C	65 78 65 00 C0 2F 3F 00 00 00 3F 00 00 00 00 00	exe. L/?...?...?

这里就得到了该 CrackMe 的全路径。

10002712	68 24C30010	PUSH 1000C324	ASCII "WAIT"
10002717	6A 01	PUSH 1	
10002719	6A 00	PUSH 0	
1000271B	FF15 14C00010	CALL NEAR DWORD PTR DS:[1000C014]	kernel32.CreateMutexA
10002721	8985 F4FEFFFF	MOV DWORD PTR SS:[EBP-10C],EAX	
10002727	B8 AA0191B5	MOV EAX,B59101AA	
1000272C	3D D096494B	CMP EAX,4B4996D0	
10002731	74 07	JE SHORT 1000273A	AntiDebu.1000273A
10002733	BA 3B270010	MOV EAX,1000273A	

接下来就到了 CreateMutexA 的调用处,该函数用于创建互斥体,可用于防止多个实例运行。(PS:最简单的防多开)

0012F0FC	00000000	pSecurity = NULL
0012F100	00000001	InitialOwner = TRUE
0012F104	1000C324	MutexName = "WAIT"
0012F108	0012FD30	
0012F10C	00210020	

这里是该函数的参数。

Registers (FPU)	
EAX	00000050
ECX	0012F090
EDX	7C91EB94
EBX	10000000
ESP	0012F108
EBP	0012F900
ESI	00000003
EDI	0012F9B0
EIP	10002721
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 0	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FDF000(FFF)
T 0	GS 0000 NULL
D 0	
O 0	LastErr ERROR_SUCCESS (00000000)
EFL	00000206 (NO,NB,NE,A,NS,PE,GE,G)
ST0	empty -UNORM BB4C 01050104 00000000
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0

这里我们直接按 F8 执行该函数就得到了句柄 50,OD 中 LastErr(PS:也就是调用 GetLastError 得到的值)的返回值为 ERROR_SUCCESS,也就是创建互斥体成功。

我们单击工具栏中的 H 按钮查看句柄列表窗口,可以看到其中有两个互斥体的句柄,一个叫做 MYFIRSTINSTANCE,另一个叫做 WAIT。

Address	Type	Count	Value	Name
00000024	Desktop	2441	000F01FF	\Default
00000008	Directory	75	00000003	\KnownDlls
00000014	Directory	43	000F000F	\Windows
00000030	Directory	311	0002000F	\BaseNamedObjects
0000001C	Event	3	001F0003	
00000044	Event	2	001F0003	
0000000C	File (dir)	2	00100020	c:\Documents and Settings\Ricardo\Escritorio\CONCURSO 8
0000002C	Key	2	000F003F	HKEY_LOCAL_MACHINE
00000040	Key	2	00020019	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\Current
00000004	KeyedEvent	41	000F0003	\KernelObjects\CritSecOutOfMemoryEvent
00000048	Mutant	3	001F0001	\BaseNamedObjects\MYFIRSTINSTANCE
00000050	Mutant	3	001F0001	\BaseNamedObjects\WAIT
00000018	Port	3	001F0001	
00000010	Section	40	000F001F	
0000003C	Section	2	000F0007	
0000004C	Section	2	000F0007	
00000034	Semaphore	2	00100003	
00000038	Semaphore	2	00100003	
00000020	WindowStation	76	000F037F	\Windows\WindowStations\WinSta0
00000028	WindowStation	76	000F037F	\Windows\WindowStations\WinSta0

MYFIRSTINSTANCE 这个互斥体是之前创建的,这里我就不跟踪了。

1000273B	FF15 10C00010	CALL NEAR DWORD PTR DS:[1000C010]	ntdll.RtlGetLastWin32Error
10002741	8945 98	MOV DWORD PTR SS:[EBP-68],EAX	
10002744	B8 8C0F347C	MOV EAX,7C340F8C	
10002749	3D 6071AC84	CMP EAX,84AC7160	

接下来调用 RtlGetLastWin32Error 这个函数获取 API 函数执行的错误码,这里我们可以对照着 OD 来看。

这里 EAX 的值为 0,也就是 OD 中显示的 LastErr 的值。

Registers (FPU)	
EAX	00000000
ECX	0012F090
EDX	7C91EB94 ntdll.KiFastSystemCallRet
EBX	10000000 AntiDebu.10000000
ESP	0012F108
EBP	0012F9D0
ESI	00000003
EDI	0012F9B0
EIP	10002741 AntiDebu.10002741
C 0	ES 0023 32bit 0(FFFFFFFF)
P 0	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 0	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFDF000(FFF)
T 0	GS 0000 NULL
O 1	LastErr ERROR_SUCCESS (00000000)
EFL	00000A02 (O,NB,NE,A,NS,PO,L,LE)
ST0	empty -UNORM BB4C 01050104 00000000
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.0
ST5	empty 0.0

10002758	817D 98 B70000	CMP DWORD PTR SS:[EBP-68],0B7	
1000275F	75 1C	JNZ SHORT 1000277D	AntiDebu.1000277D
10002761	B8 5FA428A6	MOV EAX,A628A45F	
10002766	3D 38B9C05A	CMP EAX,5AC0B938	
1000276B	74 07	JE SHORT 10002774	AntiDebu.10002774
1000276D	B8 75270010	MOV EAX,10002775	
10002772	FFE0	JMP NEAR EAX	
10002774	DF6A 00	FILD QWORD PTR DS:[EDX]	
10002777	FF15 0CC00010	CALL NEAR DWORD PTR DS:[1000C00C]	kernel32.ExitProcess
1000277D	B8 32391DD0	MOV EAX,D01D3932	
10002782	3D 1001D530	CMP EAX,30050110	
10002787	74 07	JE SHORT 10002790	AntiDebu.10002790
10002789	B8 91270010	MOV EAX,10002791	

这里我们可以看到这个 WAIT 互斥体是第一次创建,所以跳过了 ExitProcess,如果 WAIT 不是第一次创建的话,那么就会调用 ExitProcess 结束进程。这样就可以防止多个程序实例运行。

继续。

10002791	8D45 E4	LEA EAX,DWORD PTR SS:[EBP-1C]	
10002794	50	PUSH EAX	
10002795	8D4D 9C	LEA ECX,DWORD PTR SS:[EBP-64]	
10002798	51	PUSH ECX	
10002799	6A 00	PUSH 0	
1000279B	6A 00	PUSH 0	
1000279D	6A 20	PUSH 20	
1000279F	6A 00	PUSH 0	
100027A1	6A 00	PUSH 0	
100027A3	6A 00	PUSH 0	
100027A5	6A 00	PUSH 0	
100027A7	8D95 1CFCFFFF	LEA EDX,DWORD PTR SS:[EBP-3E4]	
100027AD	52	PUSH EDX	
100027AE	FF15 84C00010	CALL NEAR DWORD PTR DS:[1000C084]	kernel32.CreateProcessA
100027B4	B8 1447C096	MOV EAX,96C04714	
100027B9	3D A0B376A0	CMP EAX,6A37DBA0	
100027BE	74 07	JE SHORT 100027C7	AntiDebu.100027C7
100027C0	B8 C8270010	MOV EAX,100027C8	
100027C5	FFE0	JMP NEAR EAX	
100027C7	DC68 C8	FSUBR QWORD PTR DS:[EAX-38]	
100027CA	0000	ADD BYTE PTR DS:[EAX],AL	
100027CB	0000		

这里可以看到该程序将调用 CreateProcessA 创建第二个进程。

0012F0E0	0012F5EC	ModuleFileName = "C:\Documents and Settings\Ricardo\Escrit
0012F0E4	00000000	CommandLine = NULL
0012F0E8	00000000	pProcessSecurity = NULL
0012F0EC	00000000	pThreadSecurity = NULL
0012F0F0	00000000	InheritHandles = FALSE
0012F0F4	00000020	CreationFlags = NORMAL_PRIORITY_CLASS
0012F0F8	00000000	pEnvironment = NULL
0012F0FC	00000000	CurrentDir = NULL
0012F100	0012F96C	pStartupInfo = 0012F96C
0012F104	0012F9B4	pProcessInfo = 0012F9B4

这里我们可以看到该函数的参数,如果我们按 F8 键执行该函数的话,那么第二个进程将会被创建,但是 WAIT 互斥体已经存在了,所以被创建的进程会检测是否存在 WAIT 这个互斥体,显示这个互斥体是存在的,所有程序将调用 ExitProcess 退出。

这里如果我们按 F8 键创建该进程的话,就算被创建了,该进程也会立即结束掉,所以这里我们可以尝试将进程挂起,我们可以通过修改 CreateFlags 这个参数将进程挂起,这里我们将 CreateFlags 的值修改 0x4。

0012F0E0	0012F5EC	ModuleFileName = "C:\Documents and Settings\Ricardo\
0012F0E4	00000000	CommandLine = NULL
0012F0E8	00000000	pProcessSecurity = NULL
0012F0EC	00000000	pThreadSecurity = NULL
0012F0F0	00000000	InheritHandles = FALSE
0012F0F4	00000004	CreationFlags = CREATE_SUSPENDED
0012F0F8	00000000	pEnvironment = NULL
0012F0FC	00000000	CurrentDir = NULL
0012F100	0012F96C	pStartupInfo = 0012F96C
0012F104	0012F9B4	pProcessInfo = 0012F9B4
0012F108	0012FD30	
0012F10C	00210020	

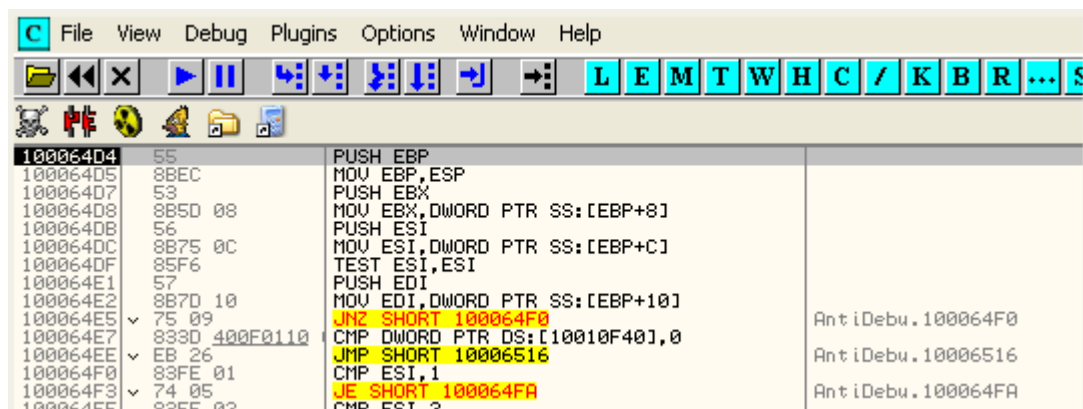
我们来看看会发生什么。

这里我们按 F8 键执行该函数,EAX 返回的是 0,也就是说创建进程失败,这是为什么呢?

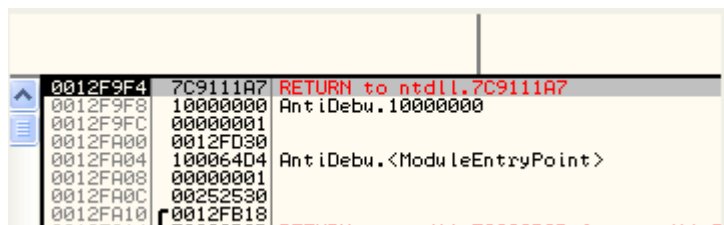
好,我们直接给 CreateProcessA 这个函数调用处设置硬件执行断点,来看看为什么创建进程失败。也就是说此时有两个硬件访问断点,一个是父进程 PID 与 explorer.exe PID 进行比较处,另一个就是该 CreateProcessA 的调用处。

这里有可能是 HideOD 插件的影响,可能是该插件某些与权限相关的选项导致不能创建其他进程,我们尝试去掉这些选项。

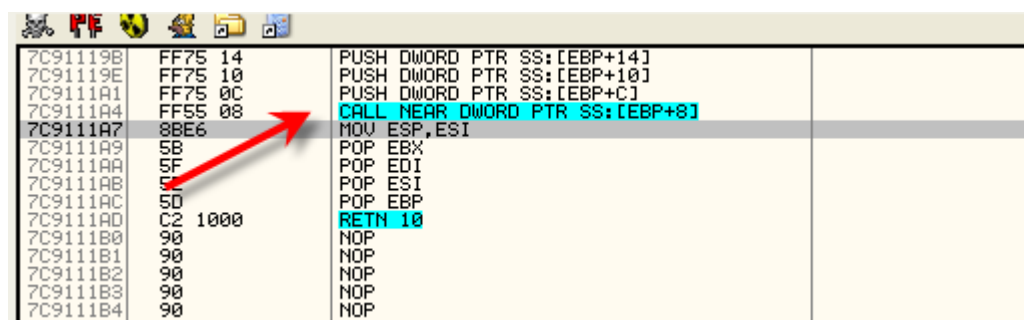
我们可以通过以下方式:重启 OD,其断在系统断点处,我们给 AntiDebugDll.dll 这个模块的代码段设置内存访问断点,我们运行起来,就会断在该 DLL 的入口点处。



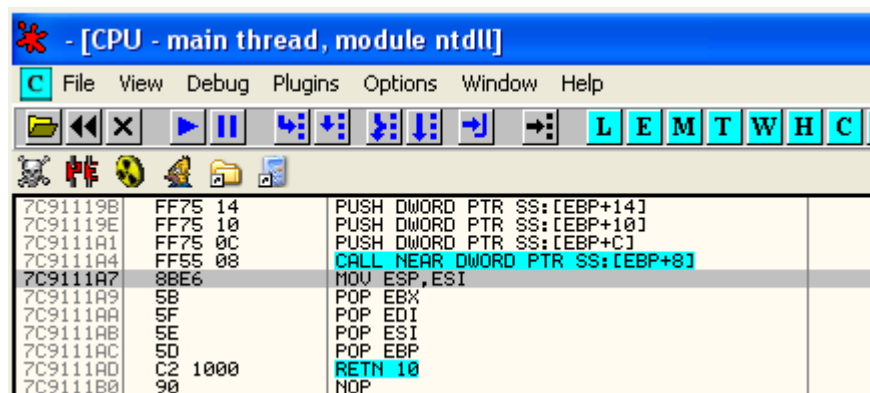
现在我们需要寻找一个点,这个点需要满足如下条件:新进程被创建,AntiDebugDll.dll 的代码还未执行。我们可以选一个调用 AntiDebugDll.dll 入口点之前的地址,然后让其一直在该地址处循环,不往下执行。我们来看一看此时的堆栈情况。



这里我们可以看到返回地址为 7C9111A7,我们定位到该返回地址处。

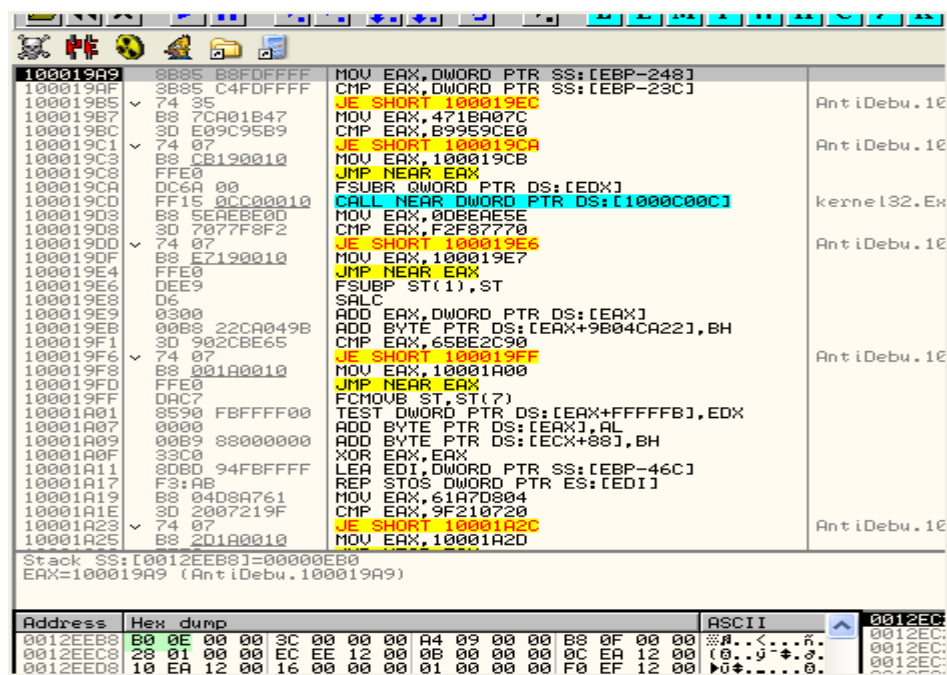


这里 7C9111A4 这个 CALL 就是跳往 AntiDebugDll.dll 的入口点,当前程序的领空属于 NTDLL.DLL,也就是说当新创建的进程由挂起状态恢复为运行状态后,就会运行到该 CALL 处,我们要想办法让其在这里一直循环而不进去。



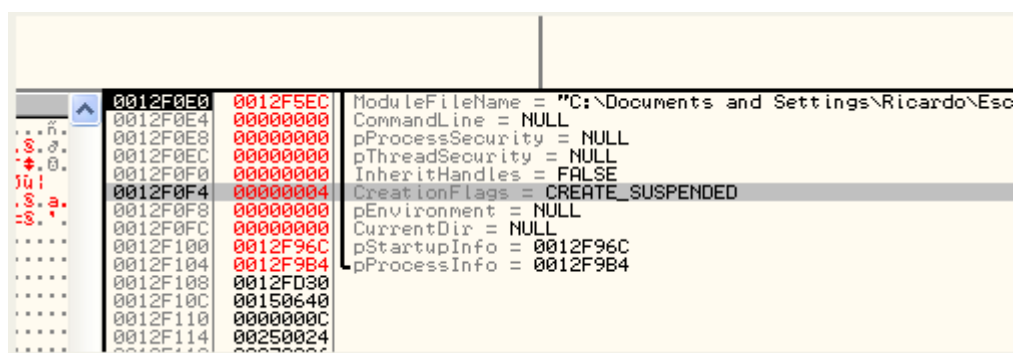
这里我们需要做的就是替换 7C9111A4 地址处的原始字节,该地址处的原始字节为 FF55。

好了思路就是这样的啊,我们之前的两个硬件断点还没有删除吧?我们删除掉内存访问断点,接着直接运行起来,断在了 PID 的比较处。



这里我们将父进程 PID 修改为 explorer.exe 的 PID,继续运行

随即就断在了 CreateProcessA 的调用处。



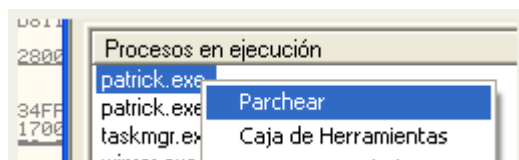
这里我们将 CreateFlags 的值修改为 0x4。

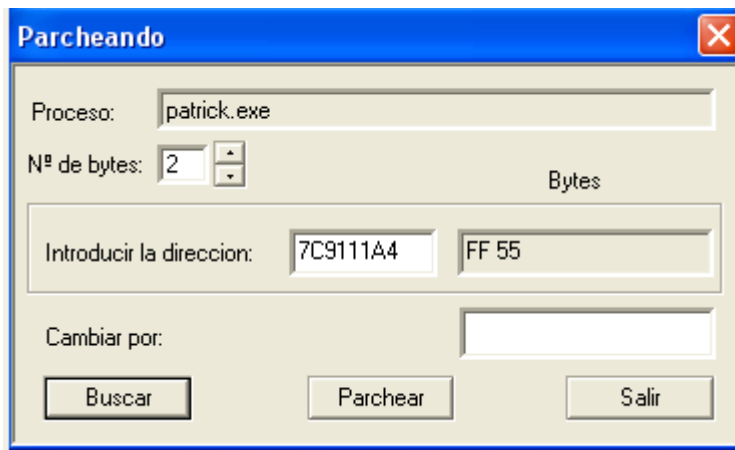
按 F8 键,这样新进程就创建了。

Procesos en ejecucion	ID Proceso	II
patrick.exe	00000C9C	C
patrick.exe	000009A4	C
taskmgr.exe	00000E88	C
winrar.exe	000002B8	C

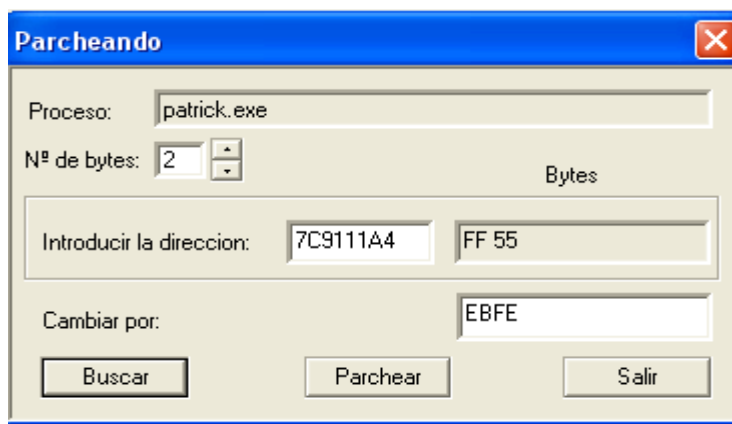
这里我们可以在 PUPE 中看到这两个进程。

这里我们选择新创建的进程进行 Patch,我们选中新创建的进程,单击鼠标右键选择 Parchear(Patch)按钮。



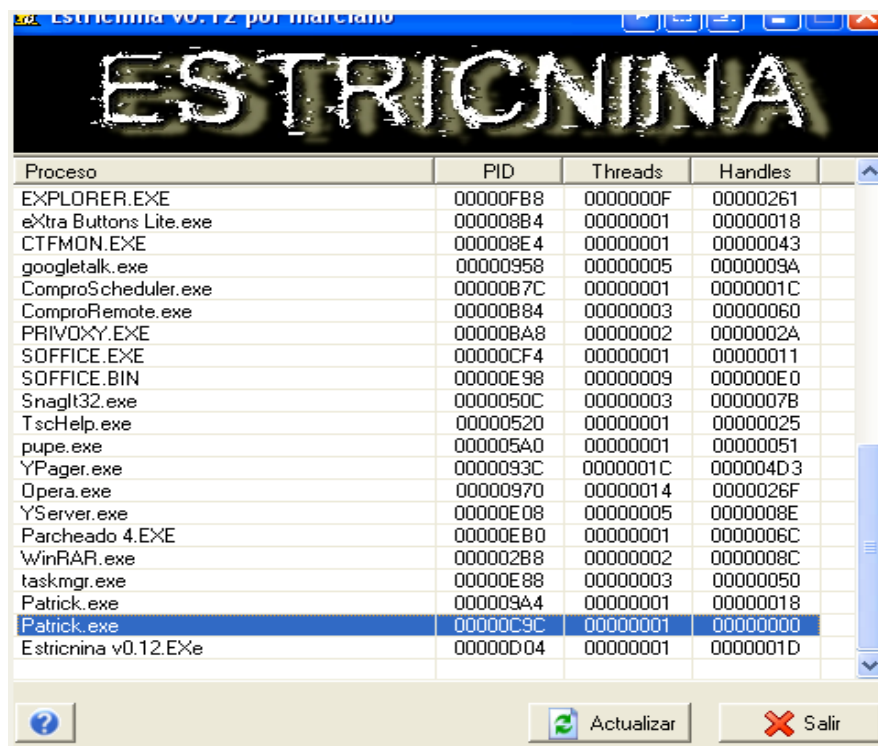


这里我们将 7C9111A4 地址处的前两个字节修改为 EB FE。

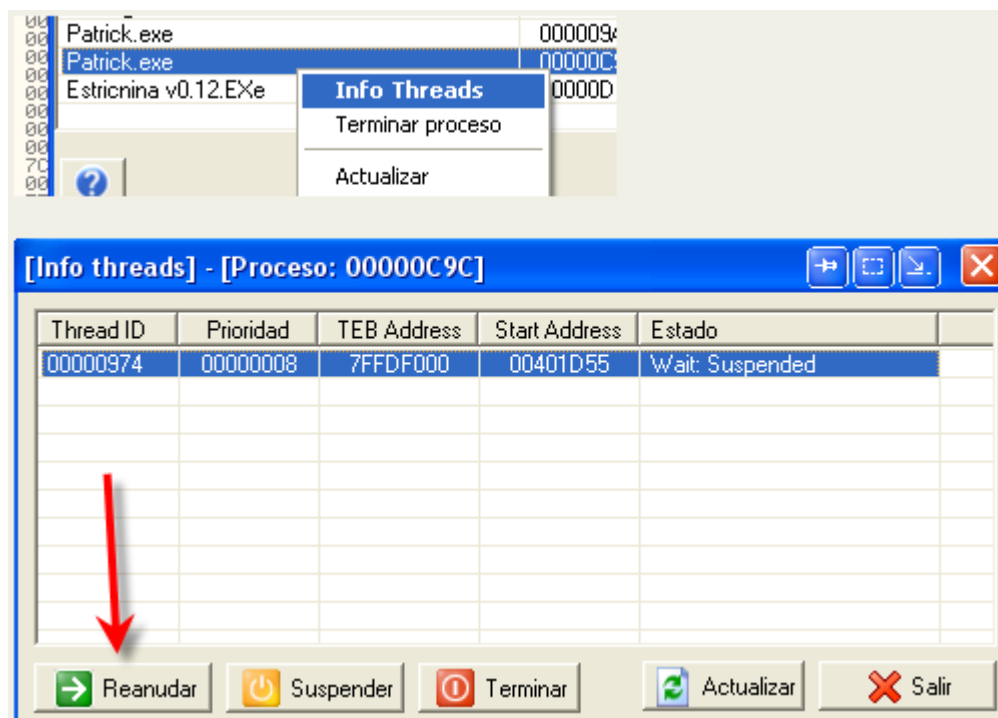


这里我们填写上 7C9111A4(PS:注意你本机的地址),单击 Buscar(获取原始字节)按钮,接着填写上要替换的字节码 EBFE,最后单击 PARCHEAR(Patch)按钮。这样就 Patch 完毕了。现在我们需要将该进程由挂起状态恢复到运行状态。

下面我要用到 Estricnina_v0.12 这款工具。



这里我们定位到新创建的进程,单击鼠标右键选择 Info Threads。

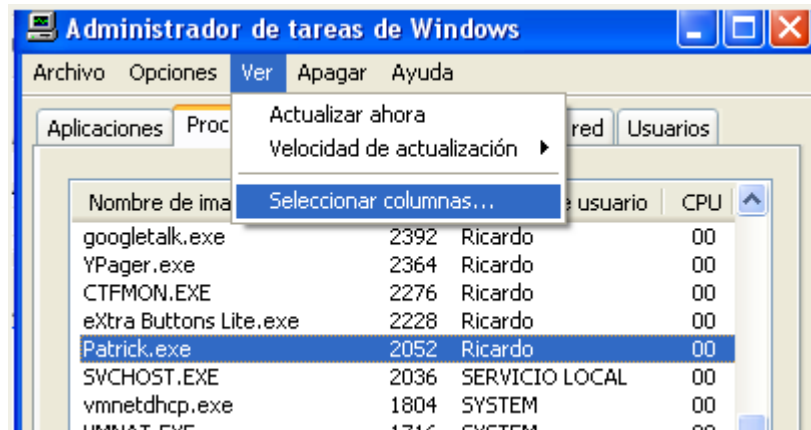


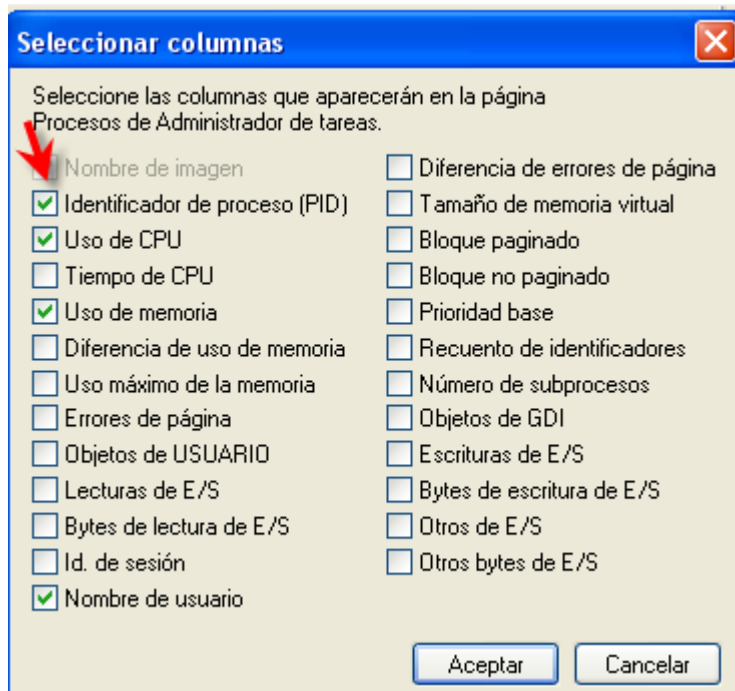
单击 Reanudar(Resume)按钮。

这样挂起的线程就被恢复了,并且一直在 7C9111A4 地址处循环。

接下来我们来附加该进程。

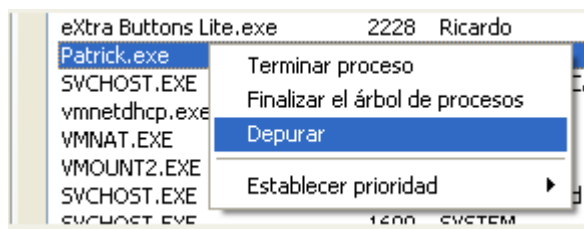
首先我们打开任务管理器,将显示 PID 的列勾选上。



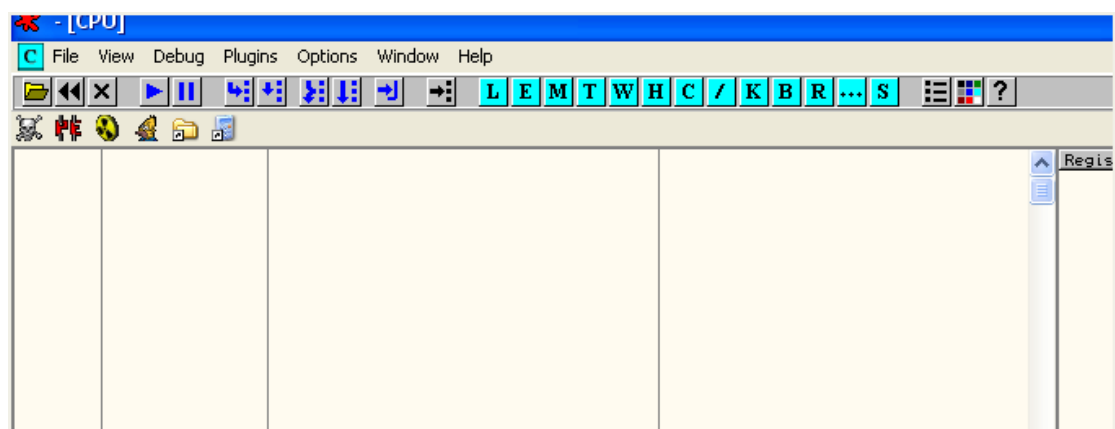


这里将任务管理器的 PID 选项勾选上,显示的是 10 进制,我们转换为 16 进制即可,这样我们就定位到新创建的进程了。

Procesos en ejecución	ID Proceso	ID
patrick.exe	00000804	000
patrick.exe	00000140	000
pupe.exe	00000FEC	000



接下来我们将 OD 设置为即时调试器(JIT),我们选择新创建的 Patrick.exe 这个进程单击鼠标右键选择 Depurar(调试)。这样 OD 的即时调试功能就生效了。



这里我们可以看到 OD 中是空白的,但是并没有发生警告,不用担心,我们直接单击工具栏中的 T 按钮查看下线程。

Ident	Entry	Data block	Last error	Status	Priority	User time	System time
00000974	00000000	7FFDF000	ERROR_SUCCESS (00000000)	Paused	32 + 0	141.9531 s	0.0468 s
00000D9C	7C96077B	7FFDE000	ERROR_SUCCESS (00000000)	Paused	32 + 0	0.0000 s	0.0000 s

这里我们可以看到有两个线程,其中一个是由于我们 Patch 的循环引起的,我们任选一个双击,如果不是循环,那么就证明是另一个。

Address	Disassembly	Comment
7C9111A4	JMP SHORT 7C9111A4	
7C9111A6	OR BYTE PTR DS:[EBX+5E5F5BE6],CL	
7C9111AC	POP EBP	
7C9111AD	RETN 10	
7C9111B0	NOP	
7C9111B1	NOP	
7C9111B2	NOP	
7C9111B3	NOP	

我们可以看到该处处于循环中。

Address	Hex dump	ASCII
7C9111A4	EB FE 08 8B E6 5B 5F 5E 5D C2 10 00 90 90 90 90	00000000 00000000 00000000 00000000
7C9111A6	90 8B FF 55 8B EC 56 57 64 A1 18 00 00 00 8B 80	00000000 00000000 00000000 00000000
7C9111C4	B0 01 00 00 85 F6 8B 7D 0C 0F 85 10 F2 00 00 85	00000000 00000000 00000000 00000000
7C9111D4	FF 0F 85 11 F2 00 00 80 3D 04 C0 98 7C 00 0F 85	00000000 00000000 00000000 00000000
7C9111E4	04 F2 00 00 8B 45 08 83 48 10 10 5F 5E 5D C2 08	00000000 00000000 00000000 00000000
7C9111F4	00 90 90 90 90 90 8B FF 55 8B EC 83 EC 54 56 64	00000000 00000000 00000000 00000000

我们为 7C9111A4 这一行设置一个断点,接着将该地址处的前两个字节恢复为原始字节 FF55。

Address	Disassembly	Comment
7C9111A4	CALL NEAR DWORD PTR SS:[EBP+8]	ntdll.<ModuleEntryPoint>
7C9111A7	MOV ESP,ESI	
7C9111A9	POP EBX	
7C9111AA	POP EDI	
7C9111AB	POP ESI	
7C9111AC	POP EBP	
7C9111AD	RETN 10	
7C9111B0	NOP	
7C9111B1	NOP	

这样我们将附加成功了,我们可以查看下模块列表窗口。

Base	Size	Entry	Name	File version	Path
00400000	00067000	00401055	Patrick		C:\Documents and Settings\Ricardo\Escritorio\CONCURSO 87\carpeta sin t\uitulo\Patrickori\Patrick.exe
7C800000	00101000	7C800436	kernel32	5.1.2600.2180	(C:\WINDOWS\system32\kernel32.dll)
7C910000	00066000	7C923156	ntdll	5.1.2600.2180	(C:\WINDOWS\system32\ntdll.dll)

此时并没有模块列表中并没有出现 AntiDebugDll.dll 这个模块,我们会断在 7C9111A4 处,如果还是没有出现 AntiDebugDll.dll,我继续运行,直到 AntiDebugDll.dll 这个模块为止。

Base	Size	Entry	Name	File version	Path
00400000	00067000	00401055	Patrick		C:\Documents and Settings\Ricardo\Escritorio\CONCURSO 87\carpeta sin t\uitulo\Patrickori\Patrick.exe
10000000	00014000	10006404	AntiDebu		C:\Documents and Settings\Ricardo\Escritorio\CONCURSO 87\carpeta sin t\uitulo\Patrickori\AntiDebugDll.dll
76800000	0002E000	76802B69	WINMM	5.1.2600.2180	(C:\WINDOWS\system32\WINMM.dll)
77010000	00090000	7701F3B0	USER32	5.1.2600.2622	(C:\WINDOWS\system32\USER32.dll)
770A0000	000AC000	770A70D4	ADVAPI32	5.1.2600.2180	(C:\WINDOWS\system32\ADVAPI32.dll)
77E50000	00091000	77E56284	RPCRT4	5.1.2600.2180	(C:\WINDOWS\system32\RPCRT4.dll)
77EF0000	00047000	77EF6594	GDI32	5.1.2600.2019	(C:\WINDOWS\system32\GDI32.dll)
7C800000	00101000	7C800436	kernel32	5.1.2600.2180	(C:\WINDOWS\system32\kernel32.dll)
7C910000	00066000	7C923156	ntdll	5.1.2600.2180	(C:\WINDOWS\system32\ntdll.dll)

现在我们可以看到模块列表中出现了 AntiDebugDll.dll,好,现在我们对其代码段设置内存访问断点。

00310000	00041000			Map	R	R	\Device\HarddiskVolume
00370000	00041000			Map	R	R	\Device\HarddiskVolume
00400000	00001000	Patrick		Map	R	RWE	
00401000	00005000	Patrick	.text	code	Imag	R	
00406000	00002000	Patrick	.rdata	imports	Imag	R	
00408000	00001000	Patrick	.data	data	Imag	R	
00409000	00005000	Patrick	.rsrc	resources	Imag	R	
10000000	00001000	AntiDebu	PE header	PE header	Imag	R	
10001000	00008000	AntiDebu	.text	code			
1000C000	00002000	AntiDebu	.rdata	import			
1000E000	00004000	AntiDebu	.data	data			
10012000	00002000	AntiDebu	.reloc	reloc			
76B00000	00001000	WINMM		PE he			
76B01000	0001F000	WINMM	.text	code,			
76B20000	00002000	WINMM	.data	data			
76B22000	0000A000	WINMM	.rsrc	resou			
76B2C000	00002000	WINMM	.reloc	reloc			
77D10000	00001000	USER32		PE he			
77D11000	00005F00	USER32	.text	code,			
77D70000	00002000	USER32	.data	data			
77D72000	00002B00	USER32	.rsrc	resou			
77D9D000	00003000	USER32	.reloc	reloc			
77DA0000	00001000	ADVAPI32		PE he			
77DA1000	00007500	ADVAPI32	.text	code,			
77E16000	00005000	ADVAPI32	.data	data			
77E1B000	00002C00	ADVAPI32	.rsrc	resou			
77E47000	00005000	ADVAPI32	.reloc	reloc			
77E50000	00001000	RPCRT4		PE he			
77E51000	00008200	RPCRT4	.text	code,			
77ED3000	00007000	RPCRT4	.orpc	code			
77EDA000	00001000	RPCRT4	.data	data			

Actualize	
View in Disassembler	Enter
Dump in CPU	
Dump	
Search	Ctrl+B
Search next	Ctrl+L
Set break-on-access	F2
Set memory breakpoint on access	
Set memory breakpoint on write	
Set access	
Create breakpoint	

现在删除掉 7C9111A4 处的断点,运行起来。

100064D4	55	PUSH EBP	
100064D5	8BEC	MOV EBP,ESP	
100064D7	53	PUSH EBX	
100064D8	8B5D 08	MOV EBX,DWORD PTR SS:[EBP+8]	
100064DB	56	PUSH ESI	
100064DC	8B75 0C	MOV ESI,DWORD PTR SS:[EBP+C]	
100064DF	85F6	TEST ESI,ESI	
100064E1	57	PUSH EDI	
100064E2	8B7D 10	MOV EDI,DWORD PTR SS:[EBP+10]	
100064E5	75 09	JNZ SHORT 100064F0	AntiDebu.100064F0
100064E7	833D 400F0110	CMP DWORD PTR DS:[10010F40],0	
100064EE	EB 26	JMP SHORT 10006516	AntiDebu.10006516
100064F0	83FE 01	CMP ESI,1	
100064F3	74 05	JE SHORT 100064FA	AntiDebu.100064FA
100064F5	83FE 02	CMP ESI,2	
100064F8	75 22	JNZ SHORT 10006510	AntiDebu.10006510
100064FA	A1 3C170110	MOV EAX,DWORD PTR DS:[1001173C]	
100064FF	85C0	TEST EAX,EAX	
10006501	74 09	JE SHORT 1000650C	AntiDebu.1000650C
10006503	57	PUSH ESI	

这里我们就断在了 AntiDebugDll.dll 的入口点处了。下面我们给 CreateMutexA 设置一个断点。

7C911334 52 9B D9 3C 24 74 50 66 81 3C 24 7F 00
7C911344 00 40 01 00 7C 00 FF 00 0F 00 0F 70 00

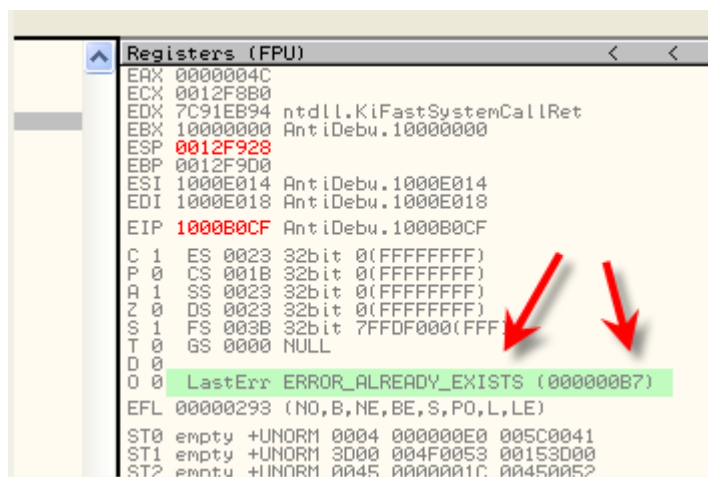
Command B

Memory breakpoint when executing [100064D4]

运行起来。

0012F918	1000B0CF	CALL to CreateMutexA from AntiDebu.1000B0C9
0012F91C	00000000	pSecurity = NULL
0012F920	00000000	InitialOwner = FALSE
0012F924	1000C32C	MutexName = "MYFIRSTINSTANCE"
0012F928	10008457	RETURN to AntiDebu.10008457
0012F92C	0012FD30	
0012F930	00000A28	

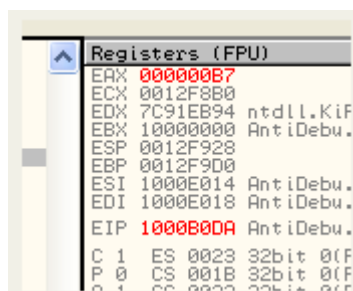
这里我们可以看到将要创建的互斥体为 MYFIRSTINSTANCE,这里由于该互斥体已经存在了,所以我们直接运行到函数返回处。



这里我们可以看到 LastErr 显示为 0xB7,即 ERROR_ALREADY_EXISTS,表示互斥体已经存在。

1000B0C5	6A 00	PUSH 0	
1000B0C7	6A 00	PUSH 0	
1000B0C9	FF15 14C00010	CALL NEAR DWORD PTR DS:[1000C014]	kernel32.CreateMutexA
1000B0CF	A3 D00E0110	MOV DWORD PTR DS:[10010ED0],EAX	
1000B0D4	FF15 10C00010	CALL NEAR DWORD PTR DS:[1000C010]	ntdll.RtlGetLastWin32Error
1000B0DA	68 50B10010	PUSH 1000B150	
1000B0DF	A3 CC0E0110	MOV DWORD PTR DS:[10010ECC],EAX	
1000B0E4	E8 95B0FFFF	CALL 1000617E	AntiDebu.1000617E

接下来这里就调用 GetLastError 这个 API 函数获取错误码,判断错误码是否为 0xB7。



如果为 0xB7 的话表示互斥体已经存在,该进程是并非第一次创建,如果不为 0xB7 的话,就表示该进程是首次创建。这里错误码为 0xB7,表示该进程并非首次创建。

好了,本章就这里。