

一种简单的破解 .Net 程序的方法

----(使用 windbg + sos)

做过程序破解的同学都知道,一般的破解分为静态和动态两种,对于动态的破解,大家比较常用 ollydbg,其虽然非常强悍,但貌似对 .net 程序支持不是很好,呵呵也许是我使用不够熟练吧。

对于 .net 程序,我们知道,它类似于 Java,编译后只是编译成了一种中间码(IL),其高于汇编但低于高等语言如 C#,Java 之类。其运行时候需要一个运行时,然后在需要的时候才进行 JIT,根据当前系统平台将 IL 转成汇编码。普通的保护也就是做些混淆,加上强签名,再加上 SuppressIldasmAttribute 属性禁止 Ildasm,相应的破解方法也比较简单,使用[修改过的 iildasm](#)就可以将相应的 IL 导出来,然后修改 IL 代码,再使用 ilasm 将之编译回去,破解也就完成了。

不过呢,目前也有一些比较好的 .net 程序保护工具可以将 IL 隐藏,如国内的 Maxtocode。今天下午我就遇到了一个该类型的保护程序。我使用 iildasm 打开后查看到所有的方法内容 IL 都是下边这样:

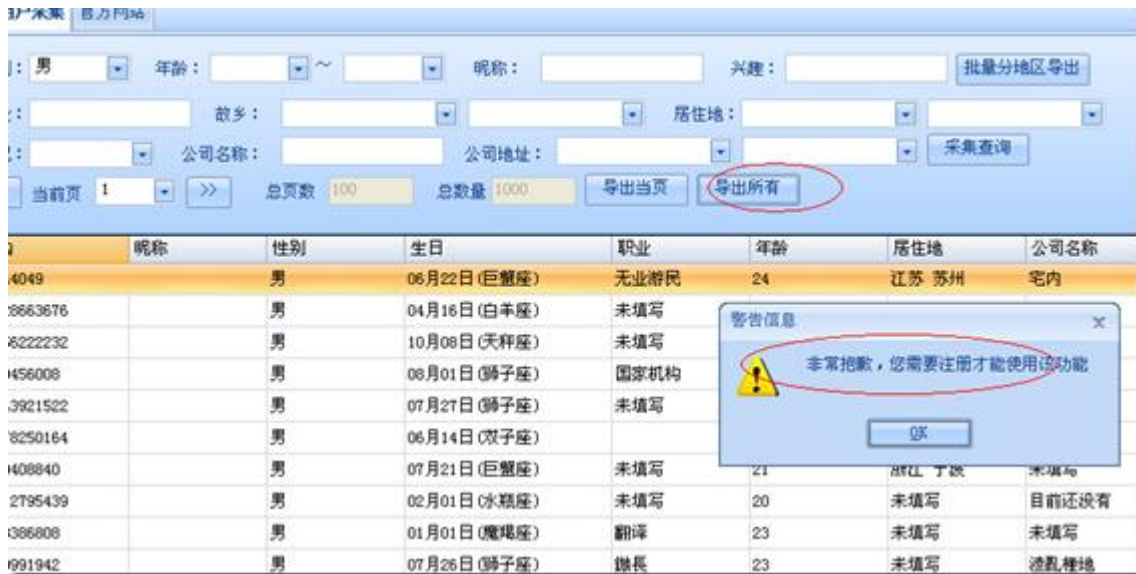
```
.method public hidebysig static void  MethodName(class
[System.Windows.Forms]System.Windows.Forms.Control  ctrl,
                                                                    string text) cil managed
noinlineing
{
    // 代码大小      4 (0x4)
    .maxstack  4
    IL_0000:  nop
    IL_0001:  nop
    IL_0002:  nop
    IL_0003:  ret
} // end of method CallCtrlWithThreadSafetyEx::SetText
```

这样的话即使将它们导出也无意义，即使我们使用反射动态的获取它们的 IL，也会得到我上边的结果，显然这样的保护做的已经是非常好了。

那么我们就没有办法了么？正当我觉得山穷水尽的时候，突然想起来为什么不使用 windbg 呢。我们知道任何的保护它都不应该破坏程序的正常逻辑和行为，那么 .net 的 JIT 机制是不是也应该还是有效呢？如果有效的话，那么我们是否可以通过 windbg 的 !u addr 命令来反编译得到代码呢，说干就干，我立刻挂上目标程序，果不其然，!dumpil 得到的依然是上边说到的 IL 内容，但 !U 却能得到正确的完整的逻辑。

既然已经得到的汇编代码，那就好办了，使用 windbg 的 ed 命令修改一下关键的指令，立刻工作。哈哈，太完美了，下边我就将我的完整破解过程列出来，以防以后忘记。

1. 我想使用该软件（名称就不透漏了，呵呵）导出搜索到的所有 qq 账户，但却由于没有注册而报错。



2. 祭出神器 windbg，选择 attach，选择目标程序，照例执行 .loadby sos mscorwks，然后想办法找到 button 点击对应处理方法的 methoddesc 地址，对于该程序，由于其使用的是模态对话框，所以方法一定在此中断，所以我们只需要简单的使用 !runaway, ~0s, !clrstack 几个命令就可以找到方法的名字。

```

1:dec      0 days 0:00:00.000
):031> ~0s
eax=7b454b50 ebx=0158d9c4 ecx=0012eb34 edx=015f6284 esi=019339c4 edi=00000000
eip=7c92e514 esp=0012eae0 ebp=0012eb78 iopl=0         nv up ei pl zr na pe nc
:s=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
!dll!KiFastSystemCallRet:
'c92e514 c3                ret
):000> !clrstack
VS Thread Id: 0x1c0c (0)
ISP      EIP
!012eaec 7c92e514 [InlinedCallFrame: 0012eaec] System.Windows.Forms.UnsafeNativeMethods.WaitMes
!012eae8 7b08374f System.Windows.Forms.Application+ComponentManager.System.Windows.Forms.Unsafe
!012eb88 7b0831a5 System.Windows.Forms.Application+ThreadContext.RunMessageLoopInner(Int32, Sys
!012ec00 7b082fe3 System.Windows.Forms.Application+ThreadContext.RunMessageLoop(Int32, System.W
!012ec30 7b23b53c System.Windows.Forms.Fora.ShowDialog(System.Windows.Forms.IWin32Window)
!012ed10 054a2370 DevComponents.DotNetBar.I.I(System.Windows.Forms.IWin32Window, System.String,
!012ed40 0fd5f8b2 DevComponents.DotNetBar.MessageBoxEx.I(System.Windows.Forms.IWin32Window, Sys
!012ed64 0fd5f82f DevComponents.DotNetBar.MessageBoxEx.Show(System.String, System.String, Syst
!012ed70 0fd5f7db QQCollector.MessageExUtil.ShowWarning(System.String)
!012ed78 0fd5f5f2 QQCollector.CtrlQQSearch.iRwC6WPFy(System.Object, System.EventArgs)
!012ed94 7b062c9a System.Windows.Forms.Control.OnClick(System.EventArgs)
!012eda8 0fd5954a DevComponents.DotNetBar.ButtonX.OnClick(System.EventArgs)
!012ee18 0fd593d9 DevComponents.DotNetBar.ButtonX.OnMouseUp(System.Windows.Forms.MouseEventArgs
!012ee3c 7b0e5b63 System.Windows.Forms.Control.VmMouseUp(System.Windows.Forms.Message ByRef, S
!012eeac 7b070246 System.Windows.Forms.Control.VndProc(System.Windows.Forms.Message ByRef)
!012eeb0 0553ad82 [InlinedCallFrame: 0012eeb0]

```

3. 看，我们找到的名字是 QQCollector.CtrlQQSearch.iRwC6WPFy，哈哈，显然已经经过混淆了。没事，关系不大。使用命令!name2ee *!QQCollector.CtrlQQSearch.iRwC6WPFy，!dumpmd 就可以最终找到汇编码地址。

```

!dumle: /90c2000 (mscorlib.dll)
-----
!dumle: 00bb2c4c (sortkey.nlp)
-----
!dumle: 00bb2010 (sorttbls.nlp)
-----
!dumle: 00bb239c (prcp.nlp)
-----
!dumle: 00bb28b0 (mscorlib.resources.dll)
-----
!dumle: 00b92c3c (WHC.QQCollector.exe)
!ken: 0x060001b5
!thodDesc: 04789548
!me: QQCollector.CtrlQQSearch.iRwC6WPFy(System.Object, System.EventArgs)
!TTED Code Address: 0fd5f5c8
-----
!dumle: 7a72a000 (System.dll)
-----
!dumle: 648ea000 (System.Configuration.dll)
-----

```

4. 使用!u 查看该 button 的点击事件对应方法的逻辑。

```

:000> !u 0fd5f5c8
JIT generated code
!Collector.CtrlQQSearch.iRu6WPFy(System.Object, System.EventArgs)
!u 0fd5f5c8, size 1bb
>> 0fd5f5c8 57          push     edi
fd5f5c9 56          push     esi
fd5f5ca 53          push     ebx
fd5f5cb 55          push     ebp
fd5f5cc 50          push     eax
fd5f5cd 8bd9       mov     ebx,ecx
fd5f5cf 8b0d001f4d02  mov     ecx,dword ptr [!Unloaded_Ed20.dll!+0xc24d1eff (024d1f00)] (Object: QQCollector.GlobalControl)
fd5f5d5 3909       cmp     dword ptr [ecx],ecx
fd5f5d7 e8847461fc  call   <Unloaded_Ed20.dll!+0xc376a5f (0c376a60) (QQCollector.GlobalControl.CheckRegister(), mdToken: 0600043)
fd5f5dc 85c0       test    eax,eax
fd5f5de 751a       jne    <Unloaded_Ed20.dll!+0xfd5f5f9 (0fd5f5fa)
fd5f5e0 b95e710000  mov     ecx,offset <Unloaded_Ed20.dll!+0x715d (0000715e)
fd5f5e5 e8d67725f1  call   <Unloaded_Ed20.dll!+0xfb6dbf (00fb6dc0) (enllcYB689tEV55umh.Ullaopt2jHvTNnEAKa.OPHM0IOSt(Int32), mdToken: 0600043)
fd5f5ea 8bc8       mov     ecx,eax
fd5f5ec ff159c694601  call   dword ptr [!Unloaded_Ed20.dll!+0x146699b (0146699c)] (QQCollector.MessageExUtil.ShowWarning(System.S
fd5f5f2 59        pop     ecx
fd5f5f3 5d        pop     ebp
fd5f5f4 5b        pop     ebx
fd5f5f5 5e        pop     esi
fd5f5f6 5f        pop     edi
fd5f5f7 c20400    ret     4
fd5f5fa b98c41b50a  mov     ecx,offset <Unloaded_Ed20.dll!+0xab5418b (0ab5418c) (NT: QQCollector.FraExportSetting)
fd5f5ff e8985d126a  call   mscorwks!JIT_NewCrossContext (79e8539c)
fd5f604 8be8       mov     ebp,eax
fd5f606 8bcd       mov     ecx,ebp
fd5f608 e87f22e7fd  call   <Unloaded_Ed20.dll!+0xdbd188b (0dbd188c) (QQCollector.FraExportSetting..ctor(), mdToken: 06000386)
fd5f60d 8bd1       mov     ebx,ecx

```

5. 我们会发现 CheckRegister，显然，这里必有猫腻。往下看，果不其然，出现了跳转语句 jne，我们试试把它改为 je 试试。先用 dd 命令查看内存的值，75 对应的就是 jne，74 就是 je

```

0fd5f75d e8966b6569  call   mscorlib_ni+0x21
0fd5f762 8d5604     lea    edx,[esi+4]
0fd5f765 e8e14f116a  call   mscorwks!JIT_Wr
0fd5f76a b87818bd0d  mov    eax,offset <Unlo
0fd5f76f 89460c     mov    dword ptr [esi+0
0fd5f772 8bd7       mov    edx,edi
0fd5f774 8bce       mov    ecx,esi
0fd5f776 e889366869  call   mscorlib_ni+0x32
0fd5f77b 59        pop    ecx
0fd5f77c 5d        pop    ebp
0fd5f77d 5b        pop    ebx
0fd5f77e 5e        pop    esi
0fd5f77f 5f        pop    edi
0fd5f780 c20400    ret    4
0:000> !dd 0fd5f5de 14
No export dd found
0:000> dd 0fd5f5de 14
0fd5f5de 5eb91a75 e8000071 f12577d6 15ffc88b
0:000> ed 0fd5f5de 5eb91a74

```

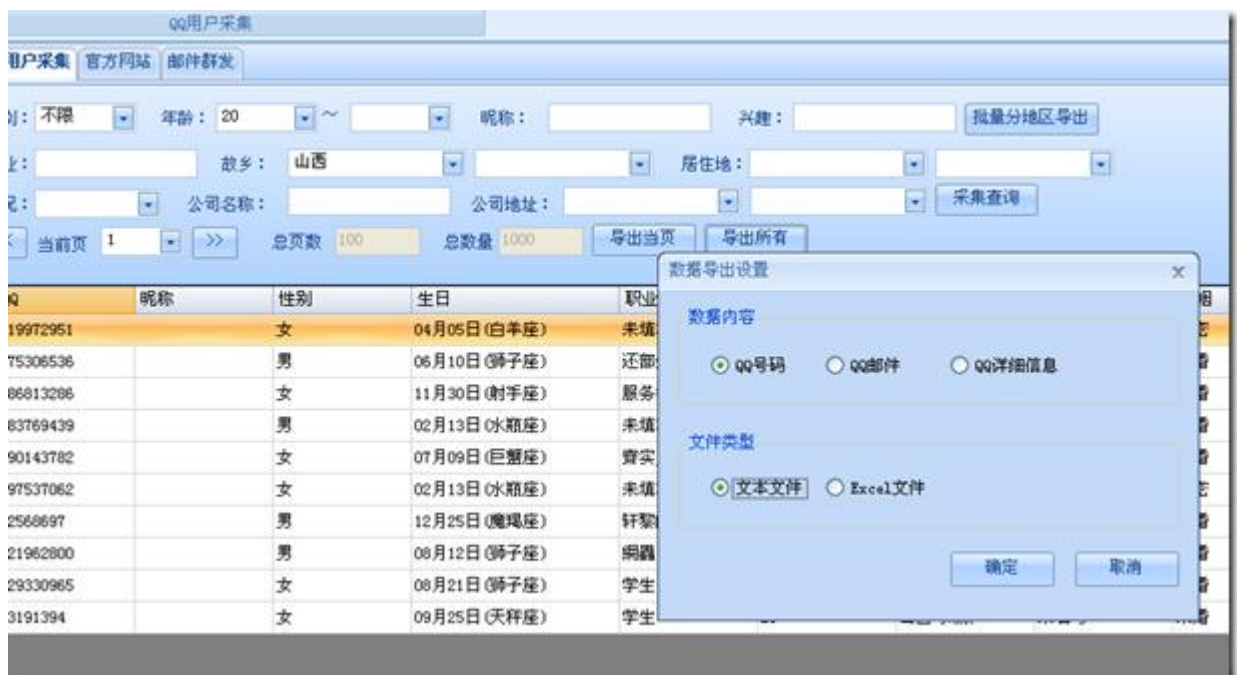
6. 我们再来确认下汇编码：

```

0> !u 0fd5f5c8
al JIT generated code
llector.CtrlQQSearch.iRwC6WPFy(System.Object, System.EventArgs)
n 0fd5f5c8, size 1bb
0fd5f5c8 57          push     edi
f5c9 56          push     esi
f5ca 53          push     ebx
f5cb 55          push     ebp
f5cc 50          push     eax
f5cd 8bd9        mov     ebx,ecx
f5cf 8b0d001f4d02  mov     ecx,dword ptr [<Unloaded_Ed20.dll>+0x24d1ef
f5d5 3909        cmp     dword ptr [ecx],ecx
f5d7 e8847461fc   call   <Unloaded_Ed20.dll>+0xc376a5f (0c376a60) (Q
f5dc 85c0        test    eax,ecx
f5de 741a        je     <Unloaded_Ed20.dll>+0xfd5f5f9 (0fd5f5fa)
f5e0 b95e710000  mov     ecx,offset <Unloaded_Ed20.dll>+0x715d (0000
f5e5 e8d67725f1  call   <Unloaded_Ed20.dll>+0xfb6dbf (00fb6dc0) (en
f5ea 8bc8        mov     ecx,eax
f5ec ff159c694601  call   dword ptr [<Unloaded_Ed20.dll>+0x146699b (0
f5f2 59          pop     ecx

```

7. 咦，真的变成 je 了，好神奇啊（要用小新的声调来读哦）。哈哈，在 windbg 中输入 g，然后再点击一次这个 button 试试看，看是不是出来了呢？



后记：

本文仅作为一个抛砖引玉的砖块，所以可能存在一些错误或不够清楚的地方，还请海涵。上边的破解过程其实有一个缺陷，那就是由于我们只是动态的修改内

存，所以不太好弄一个注册机或者内存补丁类的东西来发布，只能在自己机器上来破解。另外，在本次破解后我就想，天哪，这样破解的话那该怎样预防呢？还请高人指点。

再，常修改的汇编机器码可猛击[此处](#)。

原文地址：<http://www.xioxu.com/?p=326010> 欢迎转载。